داده‌کاوی برای برنامه‌نویسان

هنر باستانیِ شمارش

تألیف: ران زاچارسکی (guidetodatamining.com)
ترجمه: مسعود کاویانی (chistio.ir)

NAGHOOS PUBLICATION

سرشناسه	: زاخارسکی , رون
	Zacharski , Ron
عنوان و نام پدیدآور	: داده کاوی برای برنامه نویسان: هنر باستانی شمارش / تالیف رون زاخارسکی ; ترجمه مسعود کاویانی
مشخصات نشر	: تهران: ناقوس، ۱۳۹۸.
[ظاهری	: ٤٤٨ ص.: مصور، جدول.,نمودار
شابک	: 978-600-473-225-3
وضعیت فهرست‌نویسی	: فیپا
یادداشت	: عنوان اصلی.A Prorammers Gui to Data Mining: the Ancient Art of the Numeriati
عنوان دیگر	: هنر باستانی شمارش
موضوع	: داده کاوی
موضوع	: Data mining
شناسه افزوده	: کاویانی , مسعود , ۱۳٦۹- , مترجم
شناسه افزوده	: Kaviani , masoud
رده بندی کنگره	: QA۷٦ /۹
رده بندی دیویی	: ۳۱۲ / ۰۰٦
شمارهٔ کتابشناسی ملی	: ٥٥٦۲۲٤۷

برای خرید Online به آدرس زیر مراجعه کنید:
www.naghoospress.ir

انتشارات ناقوس

NAGHOOS PUBLICATION

چاپ اول

نام کتاب	: داده کاوی برای برنامه نویسان: هنر باستانی شمارش
ناشر	: انتشارات ناقوس
نویسنده	: رون زاخارسکی
مترجم	: مسعود کاویانی
چاپ اول	: ۱۳۹۸
تیراژ	: ده جلد (۱۰نسخه)
چاپ و صحافی	: برجسته
حروف‌نگار و صفحه‌آرا	: یاسمن تجملی محدث
قیمت	: ۸۰۰,۰۰۰ ریال
شابک	: ۹۷۸ -٦۰۰-٤۷۳-۲۲٥-۳
ISBN	: 978-600-473-225-3

انتشارات ناقوس
بلوار کشاورز- خیابان ١٦آذر- کوچه پارسی- پلاک۱۰
تلفاکس: ٦٦٤۷۸۹٥۱ - ٦٦٤۷۸۹٥٤

دیباچه

این کتاب با نام اصلیِ
A Programmer's Guide to Data Mining: The Ancient Art of the Numerati
در وب‌سایت guidetodatamining.com توسط Ron Zacharski تحت لایسنس Creative Commons Attribution Noncommercial 3.0 به صورت رایگان تهیه شده است.

ترجمه‌ی این کتاب در وب‌سایت chistio.ir تحت همین لایسنس منتشر شده است. به این معنا که شما می‌توانید کتاب را به صورت رایگان در اختیار داشته و آن را با رعایت لینک به تولید کننده و مترجم، به صورت غیر تجاری، استفاده، توزیع و یا چاپ کنید.

منابع تمامیِ تصاویرِ استفاده شده در این کتاب در وب‌سایت guidetodataminig.com و chistio.ir همراه با ارجاع، موجود است.

در ترجمه‌ی این کتاب سعی شده است تا به اندازه‌ی ممکن قلم نویسنده حفظ شود. با این حال مترجم در چند قسمت به صورت مختصر توضیحاتی جهت شفاف‌تر شدن موضوع برای خوانندگانِ فارسی زبان اضافه کرده است.

(نویسنده) متشکرم از...

پسرم

Adam

همسرم

Roper

(مترجم) متشکرم از...

پدرم، مادرم، خواهرم و البته همسر نازنینم. امیدوارم که بتوانم در ادامه‌ی این مسیر علمی، باعث سرافرازی آن‌ها شوم. همیشه قدردانتان بوده، هستم و خواهم بود...

فهرست مطالب

فصل ۱. مقدمه

مقدماتی درباره‌ی داده‌کاوی و چگونگی مطالعه‌ی این کتاب

زندگی در یک شهرِ کوچکِ آمریکایی در ۱۵۰ سال را پیش تصور کنید. ساکنینِ آن شهر، همگی یکدیگر را می‌شناختند. فرض کنید در این شهر جعبه‌ای از پارچه به یک مغازه می‌رسید. فروشنده متوجه می‌شود که طرحِ یکی از توپ‌های پارچه، نظرِ خانمِ Clancey را جلب می‌کند چون فروشنده از قبل می‌دانست که خانم Clancey از طرح‌های گلدارِ روشن خوشش می‌آید. این فروشنده یک یادآوری در ذهن خودش حک می‌کند تا یادش باشد که دفعه‌ی بعد که خانم Clancey به فروشگاه آمد، این توپِ پارچه را به او نشان دهد. آقای Chow Winkler به آقای Wilson که یک سالن‌دار است می‌گوید که در فکرِ فروشِ تفنگِ یدکیِ خود با مدل «رمینگتن» است. آقای Wilson هم همین اطلاعات را به آقای Bud Barclay می‌دهد، زیرا می‌داند که آقای Barclay دنبال خریدِ یک تفنگِ باکیفیت است. کلانتر Valquez و سربازِ زیردستش می‌دانند که باید مراقب Lee Pye باشند زیرا این شخص آدمی قوی بوده که دوست دارد مست کند و علاوه بر آن، اخلاق درست و حسابی هم ندارد. همان‌طور که متوجه شده‌اید، زندگی در یک شهر کوچک در ۱۰۰ سال پیش تماماً حول و حوش ارتباطات جریان داشت.

در چنین شهری آدم‌ها سلایقِ شما را می‌دانستند، علاوه بر این، آن‌ها از وضعیتِ سلامتیِ شما نیز با خبر بودند. حتی می‌دانستند که شما مجرد هستید یا متاهل. چه خوب چه بد، این یک تجربه‌ی شخصی‌سازی‌شده (personalized) بود و این زندگیِ شخصی‌سازی‌شده در اکثر نقاط جهان نیز صادق بود.

اجازه بدهید ۱۰۰ سال به جلو برویم. نزدیکی سال ۱۹۶۰میلادی. در این‌سال‌ها، روابطِ شخصی‌سازی‌شده کم‌تر رخ می‌داد ولی هنوز هم وجود داشت. در این سال‌ها، یک مشتریِ عادی با مراجعه به مغازه‌ی کتابفروشی ممکن بود با جمله‌ی «کتابِ جدیدِ James Michener موجود است» توسط فروشنده مواجه شود. زیرا فروشنده می‌دانست که معمولاً یک مشتریِ نوعی به کتاب‌های James Michener علاقه‌مند است. و یا اینکه فروشنده معمولاً کتابِ «وجدان محتاط» اثر Barry Goldwater را به مشتریان پیشنهاد می‌داد زیرا می‌دانست که معمولاً مشتریانِ نوعی محافظه‌کار هستند. یا برای مثالِ ساده‌تر، در همین حول و حوشِ سال ۱۹۶۰، یک مشتری برای صرف شام به رستوران می‌آید و پیش‌خدمت به او می‌گوید: «همان همیشگی؟»

امروزه نیز هنوز تکه‌هایی از آن شخصی‌سازی (personalization) وجود دارد. مثلاً فرض کنید من به کافی‌شاپِ محلِ خودم در مِسیلا (شهر Mesilla) می‌روم و پیش‌خدمت می‌گوید: «یک قهوه لاته با یک شاتِ اضافه؟» زیرا پیش‌خدمت می‌داند که این همان چیزی

است که هر روز صبح سفارش می‌دهم. یا برای مثال، سگ خانگیِ خودم را نزدِ آرایشگرِ سگ‌هایم می‌برم و آرایشگر احتیاجی ندارد که از من درباره‌ی نحوه‌ی کوتاه کردنِ موهای سگم سوالی بپرسد. او دیگر می‌داند که من یک مدلِ خاص (مثلاً مدلِ گوش آلمانی بدون زواید اسپرت) برای موهای سگم می‌پسندم.

اما چیزی که واضح است این است که اوضاع در مقایسه با آن شهر کوچکِ ۱۰۰ سالِ پیش تغییر کرده است. خوابارفروشی‌ها و فروشگاه‌های حجیم جایگزین بقالی‌های محلی و دیگر فروشنده‌ها شده‌اند. در آغازِ این تغییرات، انتخاب‌ها محدود بودند. Henry Ford (بنیان‌گذار شرکت فورد) زمانی می‌گفت: «هر مشتری می‌تواند یک ماشین با هر رنگی که دلش می‌خواهد داشته باشد، البته تا زمانی که رنگ انتخابی سیاه باشد». فروشگاه‌های ضبط صوت، ضبط صوت‌های محدودی داشتند. کتاب‌فروشی‌ها یک سری کتاب محدود داشتند. بستنی می‌خواستید؟ انتخاب‌ها بین وانیلی، شکلاتی و یا شاید توت‌فرنگی بود. ماشین لباسشوی می‌خواستید؟ اگر در سال ۱۹۵۰ یک ماشین لباسشویی از یک فروشگاه محلیِ Sears (یک برندِ فروشگاه زنجیره‌ای در آمریکا) می‌خواستید، دو انتخاب وجود داشت: مدل استاندارد با قیمت ۵۵ دلار یا مدل لوکس با قیمت ۹۵ دلار.

به قرن ۲۱خوش آمدید

در قرن ۲۱، آن محدودیت‌های سابق در خرید، به دست فراموشی سپرده شده است. فرض کنید من می‌خواهم یک موسیقی بخرم. سایت iTunes حدود ۱۱ میلیون قطعه موسیقی برای انتخاب دارد. این فروشگاه، ۱۶ میلیارد قطعه تا اکتبر ۲۰۱۱ فروخته است. کم است؟ انتخاب‌های بیشتری می‌خواهم؟ می‌توانم به اپلیکیشن اسپاتیفای (Spotify) که بیش از ۱۵ میلیون قطعه‌ی موسیقی دارد سر بزنم.

می‌خواهم کتابی بخرم؟ وب‌سایت آمازون بیش از ۲ میلیون عنوان کتاب برای انتخاب دارد.

می‌خواهم یک فیلم تماشا کنم؟ انتخاب‌های زیادی وجود دارد.

بیش از ۱۰۰ هزار عنوان در Netflix موجود است

نزدیک ۵۰ هزار عنوان در هولو (Hulu) موجود است

و بیشتر از ۱۰۰ هزار عنوان در آمازون پرایم (AmazonPrime)

یک لپ‌تاپ می‌خواهم؟ وقتی عبارت لپ‌تاپ را در قسمت جستجوی سایت آمازون تایپ می‌کنم، ۳۸۱۱ نتیجه پیدا می‌کنم.

تایپ می‌کنم: «پلوپز» و بیشتر از ۱۰۰۰ انتخاب جلوی روی من است.

در آینده‌ی نزدیک، انتخاب‌های بیشتری هم به وجود می‌آید – میلیاردها قطعه‌ی موسیقی به صورت آنلاین – انواع مختلفی از فیلم‌ها – وسائلی که می‌توانند با پرینترهای ۳

بعدی، سفارشی‌سازی شوند.

پیدا کردن چیزهای مرتبط

مسئله، در اصل پیدا کردن چیزهای مرتبط است. در میان ۱۱ میلیون قطعه‌ی موسیقی در آی‌تونز (iTunes)، احتمالاً فقط تعدادی قطعه وجود دارد که من قطعاً عاشقشان هستم، ولی چگونه آن‌ها را پیدا کنم؟ می‌خواهم یک فیلم را امشب از سایت Netflix ببینم، کدام را انتخاب کنم؟ می‌خواهم فیلمی توسط P2P دانلود کنم، ولی کدام فیلم را؟ و مسئله پیچیده‌تر هم می‌شود. در هر دقیقه چند ترابایت رسانه (شامل ویدیو، تصویر، اسناد متنی و...) به شبکه (مثلاً شبکه‌ی اینترنت) اضافه می‌شود. در هر دقیقه ۱۰۰ فایل جدید در یوزنت (usenet) در دسترس هستند. در هر دقیقه، ۲۴ ساعت ویدیو در یوتیوب (youtube) بارگزاری می‌شود. در هر ساعت ۱۸۰ کتابِ جدید منتشر می‌شود. هر روز انتخاب‌ها برای خریدن اجناس بیشتر و بیشتر می‌شود. در این اقیانوسِ امکانات، پیدا کردن چیزهای مرتبط هر روز سخت و سخت‌تر می‌شود.

اگر شما یک تولید کننده‌ی محتوای رسانه‌ای مثل موسیقی باشید - مثلاً Zee Avi، خواننده‌ی مالزیایی - خطر این نیست که کسی موسیقیِ شما را به صورت غیرقانونی دانلود کند. **خطر، گمنامی است**.

ولی چگونه این چیزها را پیدا کنیم؟

سالیانِ قبل، در آن شهر کوچک، دوستانمان برای پیدا کردنِ چیزهای مختلف، ما را یاری می‌کردند. آن توپِ پارچه که برای ما بسیار متناسب بود. آن رمانِ جدید در کتابفروشی. یک LP ۳/۱ ۳۳ (یک نوع دیسک قدیمی) جدید در مغازه‌ی موسیقی. و حتماً می‌دانید که حتی امروزه هم ما به دوستانمان برای پیدا کردن چیزهای مرتبط، اتکا می‌کنیم.

تجربه‌ها هستند که به ما کمک می‌کنند تا چیزهای مختلف را پیدا کنیم. در سالیان گذشته، گزارش‌های مصرف‌کنندگان، می‌توانست تمامی ماشین‌های لباسشویی را که فروخته شده بودند، ارزیابی کند – تمامِ ۲۰ تای آن‌ها را – یا تمامیِ پلوپزهای فروخته شده را – تمامیِ ۱۰ تای آن‌ها را و با این کار پیشنهاد مناسبی ارائه دهد. اما امروزه، صدها نوع

پلوپزِ مختلف در سایت آمازون وجود دارد و احتمال کمی دارد که تنها یک **منبعِ تجربه**، بتواند تمامی آن‌ها را ارزیابی کرده و رتبه‌بندی کند. سالیان پیش، Roger Ebert (از منتقدان سینما) تمامی فیلم‌های موجود را به صورت مجازی در اینترنت نقد و بررسی می‌کرد. امروزه اما، نزدیک به ۲۵ هزار فیلم در سال در سراسر جهان تولید می‌شود. علاوه بر این، ما الان به ویدیوهای مختلف، از منابع متفاوت (مثل یوتیوب) دسترسی داریم. Roger Ebert، یا هر متخصصِ دیگری، نمی‌توانند تمامی فیلم‌هایی که امروزه در دسترسِ ما هست را نقد و بررسی کنند.

با کمی بررسی و دقت می‌فهمیم که ما از خودِ همان چیزها هم جهت پیدا کردنِ چیزها یا همان آیتم‌های دیگر استفاده می‌کنیم. برای مثال، من یک ماشین لباسشویی مدل Sears دارم که ۳۰ سال برایم کار کرده است، پس احتمالاً باز هم مدل Sears را برای خریدِ ماشین لباسشوییِ بعدی انتخاب خواهم کرد. یا یک آلبوم از بیتلز (Beatles – یک گروه راک انگلستانی) را دوست داشتم، پس یک آلبومِ دیگر از این گروه را هم می‌خریم چون احتمال می‌دهم آن دومی را هم دوست داشته باشم.

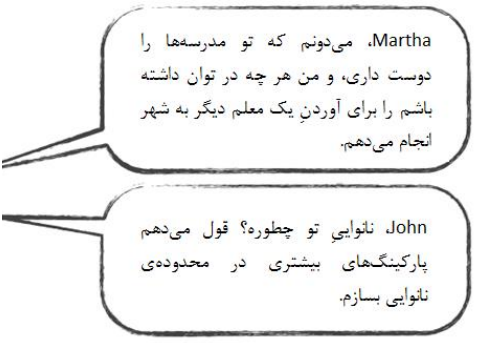
> این روش‌های یافتن آیتم‌های مرتبط – **دوستان، تجربه‌ها** یا **خود آن آیتم‌ها** – امروزه نیز موجود هستند و از آن‌ها استفاده می‌شود ولی در قرنِ حاضر نیاز به کمکِ محاسبات کامپیوتر داریم تا آن‌ها را با قرن ۲۱، یعنی جایی که میلیاردها انتخاب وجود دارد، تطبیق دهیم.

در این کتاب ما به بررسیِ روش‌های تجمیعِ علایق و سلیقه‌های مردم، سابقه‌ی خرید آن‌ها، و داده‌های دیگر می‌پردازیم – با بهره‌گیری از قدرت شبکه‌های اجتماعی (دوستان) – تا بتوانیم چیزهای مرتبط را شناسایی و استخراج کنیم. همچنین روش‌های دیگری که از ویژگی‌های خودِ آن آیتم‌ها استفاده می‌کنند را نیز بررسی می‌کنیم. برای مثال، من گروه فونیکس (Phoenix – یک گروه راک موسیقی) را دوست دارم. سیستم کامپیوتری ممکن است ویژگی‌های گروه Phoenix را درک کند – که مثلاً این گروه از آلاتی مانند راک الکتریک در آهنگ‌هایشان استفاده می‌کنند، پانک راک (نوعی سبک موسیقی راک) را می‌نوازند و از یک هماهنگیِ صوتیِ ظریف بهره می‌برند. سیستم، ممکن است گروه‌های

مشابهی را که این ویژگی‌ها را دارا هستند، به من معرفی کند، مثلاً گروه Strokes (یک گروه راک).

فقط این چیزها نیست...

داده کاوی فقط منحصر به این نیست که یک سری آیتم را به ما پیشنهاد دهد، یا این‌که به بازرگانان و تجار کمک کند که چیزهای بیشتری بفروشند. مثال‌های زیر را در نظر بگیرید:
شهردار یک شهر کوچک در ۱۰۰ سال پیش، همه‌ی افراد آن شهر را می‌شناخت. هنگامی که این شهردار می‌خواست دوباره برای انتخابات آماده شود، دقیقاً می‌دانست که به هر فرد از شهر، برای رای جمع کردن چه بگوید:



پدر من در اتحادیه‌ی کارگران خودرو عضو بود. نزدیک انتخابات، نماینده‌ی اتحادیه به خانه‌ی ما آمد تا به پدرم خاطرنشان کند که به کدام کاندیدا رای دهد:
آهای، Syl، زن و بچه‌هات چطورن؟... حالا اجازه بده بهت بگم چرا بهتره که به Frank Zeidler (کاندیدای سوسیالیست شهرداری) رای بدی...

همین پیام‌های اختصاصیِ سیاسی که به هر فرد به صورت مستقیم گفته می‌شد، به تبلیغات همگن در زمان رشد تلویزیون تبدیل شد. با رشد تلویزیون همه‌ی افراد یک پیامِ دقیقاً یکسان دریافت می‌کردند. یک مثال خوب از این مورد، آگهیِ معروفی بود که Daisy در تلویزیون برای حمایت از Lyndon Johnson (دختر

جوانی که گلبرگ‌های گلی را جدا می‌کرد در حالی که یک بمب اتم در پشت او منفجر می‌شد) انجام داد. حالا امروزه، در حالی که انتخابات با اختلافی کم نسبت به رقیب تعیین کننده شده است و همچنین استفاده‌ی روز افزون از داده‌کاوی انجام می‌شود، شخصی‌سازی دوباره برگشته است. به حقوق زنان علاقه‌مند هستید؟ ممکن است یک تماس تلفنی (توسط ربات) در همین مورد داشته باشید.

کلانترِ آن شهر کوچک می‌داند چه کسانی در شهر دردسرسازند. اما امروزه، تهدیدات مخفی هستند، تروریست‌ها می‌توانند همه‌جا باشند. در ۱۱ اکتبر ۲۰۰۱ دولت آمریکا قانون پاتروییت (Patroit) را تصویب کرد. این قانون، مخفف عبارت «ترکیب و تقویت آمریکا با تامین ابزارهای مناسب و لازم برای رهگیری و انسداد تروریسم» است. بخشی از این لایحه به سرمایه‌گذاران این توانایی را می‌دهد که اطلاعاتِ مختلف را از منابع متفاوت مانند کتابخانه‌ها (مثلاً این‌که چه کتاب‌هایی می‌خوانیم)، هتل‌ها (چه کسی و چه مقداری در کجا اقامت می‌کند)، شرکت‌های کارت‌های اعتباری و عوارض بین جاده‌ای به دست بیاورند. در اکثر موارد، دولت از شرکت‌های خصوصی جهت جمع‌آوری داده‌های ما استفاده می‌کند. شرکت‌هایی مانند Seisint تقریباً از همه‌ی ما داده دارند. داده‌هایی مانند تصاویر مختلفِ ما، جایی که ما زندگی می‌کنیم، اتومبیلمان، درآمد ما، رفتار خرید و همچنین دوستان ما. Seisint ابررایانه‌هایی را دارد که با استفاده از روش‌های داده‌کاوی به پیش‌بینیِ رفتار مردم می‌پردازند. در ضمن، محصول آن‌ها اسم جالبی هم دارد: **ماتریکس**

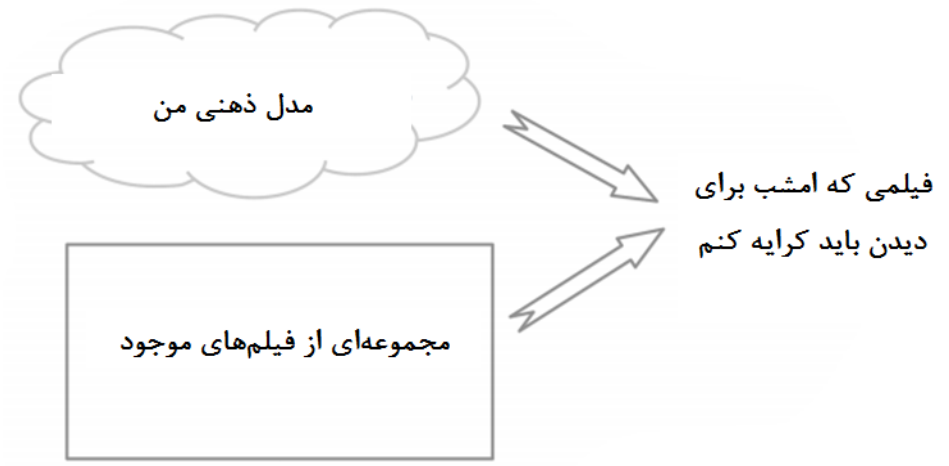
داده‌کاوی، کارهایی که در حال حاضر انجام می‌دهیم را توسعه می‌دهد!

Stephen Baker، کتاب خود را با نام شمارش (The Numerati) این‌گونه آغاز می‌کند:

فرض کنید در یک کافی شاپ هستید، احتمالاً یک کافی‌شاپِ شلوغ مثل این یکی که من الان در آن نشسته‌ام. یک زن جوان در میز سمت راست در حال تایپ کردن با لپ‌تاپِ خود است. شما سرتان را برمی‌گردانید و به صفحه‌ی لپ‌تاپ او نگاه می‌کنید. او در حال گشت‌وگذار در اینترنت است. شما نگاه می‌کنید.

ساعت‌ها می‌گذرد. او یک روزنامه‌ی آنلاین را می‌خواند. متوجه می‌شوید که او در حال خواندنِ ۳ مقاله در مورد چین است. او فیلم‌های جمعه شب را نیز مرور می‌کند و پیش‌نمایشِ فیلم پاندای کونگ‌فوکار را تماشا کرده و بعد از آن بر روی یک تبلیغ نیز کلیک می‌کند، تبلیغی که به او وعده می‌دهد که می‌تواند هم‌کلاسی‌های دورانِ دبیرستان خود را پیدا کرده و به او مرتبط کند. شما نشسته‌اید و یادداشت می‌نویسید. هر دقیقه‌ای که می‌گذرد، درباره‌ی این زن بیشتر می‌فهمید. حالا فرض کنید می‌توانستید ۱۵۰ میلیون نفر را در حال گردش در اینترنت یک‌جا ببینید!

داده‌کاوی بر روی پیدا کردنِ الگوها در داده‌ها تمرکز کرده است. در مقیاسِ کوچک، ما آدم‌ها، متخصصِ پیدا کردنِ مدل‌های ذهنی و اُلگوها هستیم. فرض کنید مثلاً می‌خواهم امشب با همسرم یک فیلم تماشا کنم. طبیعتاً من یک مدلِ ذهنی، از این‌که همسرم از چه فیلمی خوشش می‌آید، دارم. من می‌دانم که او از فیلم‌های خشن خوشش نمی‌آید (به همین دلیل از فیلمِ بلوک ۹ خوشش نیامد). او فیلم‌های چارلی کافمن را دوست دارد. من می‌توانم با استفاده از این مدلِ ذهنی که از سلیقه‌ی او سراغ دارم، پیش‌بینی کنم که او کدام فیلم را می‌پسندد و کدام فیلم را خیر.



دوستی از اروپا به دیدار من آمده است. می‌دانم که او گیاه‌خوار است و با استفاده از این اطلاعات می‌توانم پیش‌بینی کنم که او مفصل ران دوست ندارد. مردم در ساختنِ مدل‌ها و پیش‌بینی‌ها توانا هستند. داده‌کاوی این امکان را بسط و گسترش می‌دهد و ما را در به‌کارگیریِ اطلاعاتِ زیاد، توانمند می‌سازد – مانند۱۵۰ میلیون نفری که آقای Baker در نقل‌وقولِ بالا گفت. داده‌کاوی به سرویس موسیقی پاندورا (Pandora - یک نرم‌افزار پخش آنلاینِ موسیقی) این امکان را می‌دهد که یک کانالِ موسیقی را متناسب با سلیقه‌ی موسیقیاییِ مخصوصِ خودتان، به شما پیشنهاد دهد. همچنین به Netflix توانایی سفارشی‌سازی برای توصیه‌ی فیلم به کاربرانی مانند شما را می‌دهد.

تِراماینینگ چیزِ خیلی عجیبی نیست

اواخر قرن ۲۰، یک مجموعه داده (dataset) با ۱ میلیون کلمه، مجموعه‌ی داده بزرگی به حساب می‌آمد. آن زمان که من یک دانشجوی تحصیلات تکمیلی در سال ۱۹۹۰ بودم (آره، من سنم خیلی زیاده!)، یکسال به عنوان یک برنامه‌نویس در شرکت Geek New Testment کار می‌کردم. فقط در حدود ۲۰۰ هزار کلمه برای آنالیز موجود بود، اما برای همین تعداد هم حافظه‌ی اصلی کم می‌آورد و ما مجبور بودیم نتایج را به صورت تکه‌تکه به دیسکِ مغناطیسی انتقال دهیم. برای همین من مجبور بودم دوباره درخواست دیسک دهم.

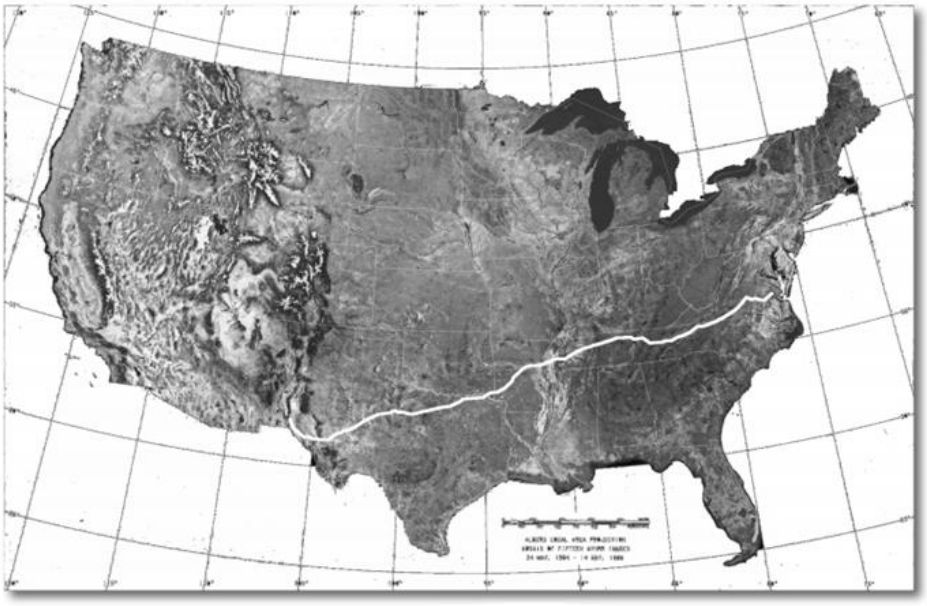
کتابی که از نتایجِ این کار تولید شد با نام Analytical Greek New Testament به وسیله‌ی Timothy و Barbara Friberg در سایت آمازون موجود است. من یکی از سه برنامه‌نویسِ آن پروژه بودم که در دانشگاه Minnesota انجام شد.

امروزه، داده‌کاوی بر روی داده‌هایی در حد ترابایت خیلی هم غیر معقول نیست. گوگل بیش‌از ۵ پتابایت (در حدود ۵ هزار ترابایت) داده‌های وب را در اختیار دارد. در سال ۲۰۰۶، گوگل یک مجموعه داده شامل یک تریلیون کلمه را برای استفاده توسط جامعه‌ی علمی انتشار داد. آژانس امنیت ملی، اطلاعاتِ تماسِ چند تریلیون ارتباط تلفنی را نزد خود دارد. شرکت آکسیوم (Acxiom)، شرکتی است که اطلاعاتِ مختلف را از ۲۰۰ میلیون آمریکایی جمع آوری می‌کند (اطلاعاتی مانند، خرید توسط کارت‌های اعتباری، تماس‌های تلفنی، اطلاعات پزشکی، خرید و فروش و انتقال اتومبیل و غیره). این شرکت بیش از یک پتابایت داده در اختیار دارد.

کانتینری که سروری یک پتابایتی را حمل می‌کند

رابرت اهارو (Robert O'Harrow)، نویسنده‌ی کتاب «هیچ جایی برای مخفی شدن وجود ندارد»، برای این‌که به ما کمک کند که بفهمیم یک پتابایت چقدر اطلاعات دارد می‌گوید: یک پتابایت از اطلاعات، مانند این است که کتابِ مقدس (عهدین – یعنی عهد قدیم و جدید شامل ۶۶ کتاب) را به اندازه ۵۰۰۰۰ مایل (تقریبا ۸۰۰۰۰ کیلومتر) پشت سر هم قرار دهیم. من معمولاً از نیومکزیکو تا ویرجینیا به اندازه‌ی ۲۰۰۰ مایل رانندگی

می‌کنم. هنگامی که می‌خواهم تصور کنم کتاب مقدس در تمامی این مسیر به اندازه‌ی ۲۰۰۰ مایل قرار گرفته باشد، به نظرم مقدار بسیار زیاد و غیرقابل باوری از داده می‌رسد.



فاصله بین ویرجینیا تا نیومکزیکو

کتابخانه‌ی گنکره، تقریباً ۲۰ ترابایت داده‌ی متنی (text data) دارد. شما می‌توانید تمامیِ مجموعه‌ی کتابخانه کنگره را در یک هارد چند هزار دلاری ذخیره کنید! برعکس، وال‌مارت (یک فروشگاه زنجیره‌ی بزرگ آمریکایی) بیش‌از ۵۷۰ ترابایت داده در اختیار دارد. تمام داده‌ها یک‌جا نیستند – به طور مداوم عملیات داده‌کاوی بر روی آن‌ها انجام می‌شود، رابطه‌های جدید کشف و ایجاد شده، و در نتیجه الگوها شناسایی می‌شوند. این همان **تِراماینینگ (Tera-mining)** است.

در این کتاب، ما با مجموعه داده‌های کوچک سروکار داریم. چیز خوبی هم است. با استفاده از مجموعه داده‌های کوچکتر، الگوریتمِ ما احتیاج ندارد هفته‌ها اجرا شود و تازه بعد از آن متوجه شویم چند خطای منطقی داریم. بزرگ‌ترین مجموعه‌ی داده‌ای که ما استفاده خواهیم کرد کمتر از ۱۰۰ مگابایت است. کمترین آن هم فقط چند ده خط داده است.

ساختار این کتاب

این کتاب یک مسیر آموزشِ عملی مبتنی بر انجام (learn-by-doing)، را دنبال می‌کند. به جای این‌که به صورت منفعل کتاب را بخوانید، تشویقتان می‌کنم تمرین‌ها را انجام دهید و تجربه‌ی کار کردن با کدهای زبان پایتون (python) را که برایتان آماده کرده‌ام به دست بیاورید. تجربه کنید، با کدها بازی کنید و روش‌های مختلف همراه با مجموعه داده‌های متفاوت را امتحان کنید. این‌کارها کلید اصلیِ فهم و درک روش‌های مختلفِ آموزش داده شده در این کتاب هستند.

من سعی دارم تعادلی خوب بین کدهای پایتون در زمینه داده‌کاوی و تئوری‌هایی که در پس‌زمینه‌ی تکنیک‌های داده‌کاوی هست برقرار کنم. برای این‌که مغز در هنگام خواندنِ تئوری‌ها، ریاضیات و کدهای پایتون قفل نکند، سعی کردم قسمتِ دیگری از مغزِ شما را با رسم تصاویر و اشکال مختلف به کار بیندازم.

Peter Norvig، مدیر تحقیقات شرکت گوگل، این گفته را در دوره‌ی آموزشیِ عالیِ خود در سایت Udacity در مورد طراحی یک نرم‌افزارِ کامپیوتری گفته است:

> من به شما راه‌حلِ خودم را نشان خواهم داد. حتماً توجه دارید که بیشتر از یک راه‌حل برای حلِ یک مسئله وجود دارد. و منظورِ من این نیست که راه‌حلی که من ارائه می‌دهم تنها راه‌حل و بهترین راه‌حل است. من راه‌حلِ خودم را می‌گویم تا به شما کمک کنم یک حالت و چند تکنیک برای نوشتن برنامه‌های کامپیوتری را یاد بگیرید. اگر جور دیگری مسئله‌ها را حل کردید، خیلی هم خوب است. خوش به حالتان.
> تمام چیزی که یاد می‌گیرید در ذهن شما اتفاق می‌افتد. نه در ذهن من. در نتیجه مهم است که شما بین کُدی که خودتان نوشته‌اید و کُدی که من نوشته‌ام، ارتباط پیدا کنید، به این صورت که پاسخِ درست را با نوشتنِ پاسخ خودتان به دست آورده و بعد از آن می‌توانید کُد من را امتحان کنید. با این کار می‌توانید نکاتی را پیدا کرده و تکنیک‌هایی را یاد بگیرید که بعداً از آن استفاده کنید.

بیش‌تر از این نمی‌توانم موافق این موضوع باشم!

این کتاب یک کتاب کامل و جامع در مورد روش‌ها و تکنیک‌ها داده‌کاوی نیست. کتاب‌هایی مانند، «مقدمه‌ای بر داده‌کاوی، اثر Pang-Ning Tan، Michael Steinbach و Vipin Kumar» در بازار موجود هستند که بسیار بهتر روش‌های مختلفِ داده‌کاوی را پوشش داده و ریاضیاتِ پایه برای فهمِ و درکِ روش‌های مختلف را نیز به صورت عمیق‌تر توضیح داده‌اند. ولی این کتاب – همین کتابی که الان دارید می‌خوانید – می‌خواهد یک مقدمه‌ی سریع، واقع‌گرایانه، کاربردی و دم‌دستی را ارائه کند تا بتواند ساختاری پایه در مورد داده‌کاوی برایتان ایجاد کند. بعداً، می‌توانید یک کتابِ جامع‌تر برای پرکردنِ خلاهایی که دارید را مطالعه کنید.

یکی از قسمت‌های مفیدِ این کتاب، کُدهای زبان پایتون در کنار متون است. فکر کنم گنجاندنِ این دو در کنار هم، باعث تسهیلِ یادگیریِ خواننده برای درک مفاهیم کلیدی می‌شود، ولی این کار باعث می‌شود که خواننده فقط به بررسی کُدها و اسکریپت‌های پایتون نپرداخته و متون و تئوری‌ها را نیز یاد بگیرد.

با خواندن و تمام‌کردنِ این کتاب چه چیزی یاد خواهید گرفت

وقتی که این کتاب را تمام کنید، قادر خواهید بود یک سیستم توصیه‌گر (recommender system) برای وب‌سایت‌ها توسط پایتون یا هر زبانِ دیگری پیاده‌سازی کنید. برای مثال، وقتی یک محصول در آمازون را مشاهده می‌کنید، یا یک آهنگ در پاندورا گوش می‌دهید، شما با مجموعه‌ای از پیشنهادها مواجه می‌شوید (مثلاً آن‌جاهایی که نوشته است: «همچنین محصولات زیر را احتمالاً دوست دارید»). یاد می‌گیرید که چگونه این دست سیستم‌ها را توسعه دهید. به علاوه، این کتاب یک سری کلماتِ ضروری و کلیدی را به شما یاد خواهد داد تا بتوانید در تیم‌های توسعه بر روی مسائلِ حوزه‌ی داده‌کاوی کار کنید.

به عنوانِ قسمتی از این هدف، کتابِ جاری کمک می‌کند تا پرده از راز سیستم‌های پیشنهاددهنده، سیستم‌های شناسایی تروریسم و سیستم‌های داده‌کاویِ دیگر، بردارد. حداقل می‌توانید ایده‌ای از این‌که این سیستم‌ها چگونه کار می‌کنند داشته باشید.

اصلاً چرا؟ چرا چیزهای موجود در این کتاب مهم هستند؟

چرا باید وقت‌تان را صرف خواندن و تمرینِ این کتابِ داده‌کاوی کنید؟ در ابتدای این فصل، مثال‌هایی در مورد اهمیت داده‌کاوی آوردم که خلاصه‌ی آن را در زیر می‌آورم.

چیزهای مختلفِ زیادی در دور و برمان وجود دارد (فیلم‌ها، موسیقی‌ها، کتاب‌ها، پلوپزها). تنوع و تعداد این چیزها در حال رشد است. مسئله این است که ما در میان این همه چیز، به دنبال آیتم‌های مرتبط با خودمان می‌گردیم. از تمام این فیلم‌هایی که موجود است، کدام را باید ببینم؟ کتاب بعدی که باید بخوانم کدام است؟ مسئله‌ی پیدا کردن و کشف آیتم‌های مرتبط همان چیزی است که مقصدِ داده‌کاوی به شما می‌رود. اکثر وب‌سایت‌ها یک قسمت یا ماژولی برای پیدا کردنِ این چیزها دارند. علاوه بر فیلم‌ها، موسیقی، کتاب‌ها و البته پلوپزها، احتمالاً به دنبال پیشنهادهایی جهت پیدا کردنِ دوستانی برای دنبال‌کردن (follow) در شبکه‌های اجتماعی هستید. نظرتان در مورد یک روزنامه‌ی شخصی چیست؟ یا نظرتان درباره‌ی روزنامه‌ای که فقط اخباری را به شما نمایش دهد که بیشتر به آن‌ها علاقه دارید، چیست؟ اگر یک برنامه‌نویس و به طور خاص یک توسعه‌دهنده‌ی وب باشید، داده‌کاوی و روش‌های آن به دردتان می‌خورد.

خب، الان فهمیدید که چرا باید زمانتان را صرف یادگیریِ داده‌کاوی کنید. ولی حالا چرا این کتاب؟ ببینید، کتاب‌های مختلفی وجود دارند که به صورت غیر فنی در مورد داده‌کاوی توضیح داده‌اند و توانسته‌اند یک شِمای کلی از داده‌کاوی را ترسیم کنند. این کتاب‌ها را می‌توانید خیلی سریع و لذت‌بخش بخوانید. این کتاب‌ها ارزان هستند و می‌توانید آن‌ها را قبل از خواب هم مطالعه کنید. یک مثالِ عالی از این کتاب‌ها، کتاب «شمارش» (The Numerati) اثر Stephen Baker است که پیشنهاد خود من هم این است که این کتاب را بخوانید. من نسخه‌ی صوتیِ این کتاب را در حال رانندگی از ویرجینیا به نیومکزیکو گوش دادم. انصافاً جذاب بود. این در حالی است که کتاب‌های دانشگاهی در حوزه‌ی داده‌کاوی نیز وجود دارند. این دست کتاب‌ها معمولاً کامل و جامع هستند و می‌توانند یک دید عمیق و تحلیلی در مورد تئوری‌ها و کاربردهای داده‌کاوی به شما بدهند. این کتاب‌ها را هم پیشنهاد می‌دهم. ولی من این کتاب را نوشتم تا جای خالیِ موجود را پر کنم. این کتاب برای عاشقانِ برنامه‌نویسی طراحی شده است.

$$s(i,j)=\frac{\sum_{u\in U}(R_{u,i}-\overline{R}_u)(R_{u,j}-\overline{R}_u)}{\sqrt{\sum_{u\in U}(R_{u,i}-\overline{R}_u)^2}\sqrt{\sum_{u\in U}(R_{u,j}-\overline{R}_u)^2}}$$

این کتاب برای این طراحی شده است که پشت کامپیوتر مطالعه شود، و خواننده در تمرین‌ها شرکت کرده و درگیر کدها شود.

یک نکته!

این کتاب یک سری فرمول و معادلات ریاضی هم دارد و من سعی دارم این فرمول‌ها و معادلات را به گونه‌ای توضیح دهم که برای یک برنامه‌نویسِ عادی قابل فهم باشد. برنامه‌نویسانی که زخم‌هایی را که از ریاضیات در دانشگاه خورده‌اند، فراموش کرده‌اند!

اگر این صحبت‌ها در مورد استفاده از این کتاب، شما را قانع نکرد، باید بگویم که این کتاب مجانی هم هست. مجانی هم در خرید و هم در اشتراک‌گذاریِ آن با دیگران!

منظور از «هنر باستانی شمارش» در عنوان این کتاب چیست؟

ماه ژون سال ۲۰۱۰ بود که دنبال عنوانی برای این کتاب می‌گشتم. عناوین هوشمندانه را دوست دارم، اما متاسفانه، استعدادی در این زمینه ندارم. اخیراً مقاله‌ای با عنوان «غواصی در به هم ریختگی‌های زبانی: طبقه‌بندیِ جغرافیاییِ زبان عربی» (بله، یک مقاله در حوزه‌ی داده‌کاوی) نوشتم. من این عنوانِ هوشمندانه را دوست داشتم، هوشمندانه به خاطر این‌که این عنوان به صورت بسیار خوبی متناسب با محتوای مقاله بود، ولی خوب، باید به همسرم هم می‌گفتم تا او هم با این عنوان کنار بیاید. یک مقاله‌ی دیگر نیز با کمکِ شخصی دیگر نوشتم و عنوانش این بود: «حال و حوصله و چگونگی آن: خارج از تئوری و وارد معرکه». نویسنده‌ی همکار من، Marjorie McShane با عنوانش کنار آمد. به هر حال، برگردیم به ماهِ جونِ ۲۰۱۰، تمامی ایده‌های هوشمندانه‌ام خیلی مبهم بود. اینقدر مبهم که نمی‌توانستید سرنخی از کتاب را با خواندنِ عنوانِ آن بگیرید. عنوان «آموزش داده‌کاوی برای برنامه‌نویسان» را در قسمتی از عنوان قرار دادم. معتقدم این تکه به صورت مختصر توضیح محتوای این کتاب را می‌دهد – قصد دارم این کتاب یک راهنما برای برنامه‌نویسانی باشد که الان دارند کارِ برنامه‌نویسی می‌کنند. احتمالاً معنای عبارت بعد از نقل و قول، شما را شگفت‌زده کرده باشد:

> **راهنمای داده‌کاوی برای برنامه‌نویسان:**
> **هنر باستانی شمارش**
> **A Programmer's Guide to Data Mining:**
> **The Ancient Art of the Numerati**

کلمه‌ی «شمارش» (Numerati) – که می‌توانیم آن را به «شماره‌ها» و یا حتی «عددی» نیز ترجمه کنیم - توسط Stephen Baker ابداع شد. هر کدام از ما، حجم بسیار زیادی از داده‌های دیجیتالی را هر روز تولید می‌کنیم. خریدهای با کارت‌های اعتباری، نوشته‌های توییتر، نوشته‌های گوالا (Gowalla – یک شبکه‌ی اجتماعی مبتنی بر مکان)، فعالیت‌های

فوراسکویرز (Foursquare – یک نرم‌افزار جستجوی مکان)، تماس‌های تلفنی، پیام‌های ایمیل، پیامک‌ها و غیره.

فرض کن بیدار شده‌ای. ماتریکس، می‌داند که تو در ساعت ۷:۱۰ سوار مترو ایستگاه Foggy Bottom شده و از مترو ایستگاه Westside در ساعت ۷:۳۲ خارج شده‌ای. ماتریکس، می‌داند یک قهوه لاته و یک بیسکوییتِ تمشک‌آبی در استارباکس در تاریخ پنجم ماه سفارش داده‌ای. در ۷:۴۵ شروع به حرکت کرده‌ای. از گوالا (Gowalla) برای ثبت موقعیتِ خود سر کار در ساعت ۸:۰۵ استفاده می‌کنی. یک مجموعه‌ی ۱۳ تایی دی‌وی‌دی برنامه تمرینی تناسب اندام خانگی P90X Extreme همراه با یک بارفیکس در ساعت ۹:۳۵ از سایت آمازون سفارش داده‌ای. در «فلافل طلایی» ناهار خوردی.

Stephen Baker می‌نویسد:

> تنها قومی که می‌توانند داده‌هایی که ما می‌سازیم را درک کنند، ریاضی‌دان‌های خبره، دانشمندان علومِ کامپیوتر و مهندسان هستند. این شماره‌ها چه چیزهایی درباره‌ی ما، با هدایت کردنِ ما به دنیای گیج‌کننده و مرکب اعداد می‌آموزند؟ اول این‌که آن‌ها می‌خواهند ما را پیدا کنند. فرض کنید شما یک خریدار بالقوه از فروشگاه SUV در حومه‌ی شمالی نیویورک هستید، یا یک شخص مذهبی که به کلیسا می‌رود یا عضو گروه ضد سقط جنین در آلبوکرک (از شهرهای نیومکزیکو در آمریکا) هستید. ممکن است یک برنامه‌نویس جاوا باشید که آماده نقل مکان به حیدرآباد (در هند) است، یا یک عشق جاز، یک متولد برج قوس، که به دنبال پیاده‌روی در روستا و بازدید شومینه‌های استکهلم است، یا – خدا کمکان کند – ممکن است علاقه داشته باشید که یک کمربند انتحاری ببندید و داخل یک اتوبوس شوید. هر چه هستید – که البته هر کدام از ما مجموعه‌ای از چیزهای مختلف هستیم – شرکت‌ها و دولت‌ها می‌خواهند شما را شناسایی و ردیابی کنند.
>
> **Baker**

همان‌طور که احتمالاً حدس زده‌اید، من این عبارتِ «شمارش (Numerati)» و طرز تشریح آن توسط Stephen Baker را دوست دارم.

فصل ۲. سیستم‌های توصیه‌گر

سیستم‌های توصیه‌گر یا همان سیستم‌های پیشنهاددهنده

پایش مشارکتی (Collaborative Filtering) یا همان پایش با همکاری

من چیزی را که تو دوست داشته باشی، دوست دارم

شروعِ کندوکاو در داده‌کاوی را با نگاهی بر سیستم‌های توصیه‌گر (recommendation systems) آغاز می‌کنیم. سیستم‌های توصیه‌گر همه‌جا هستند – از آمازون گرفته:

تا سایت last.fm که کنسرت یا موسیقی‌های مختلف را پیشنهاد می‌دهد:

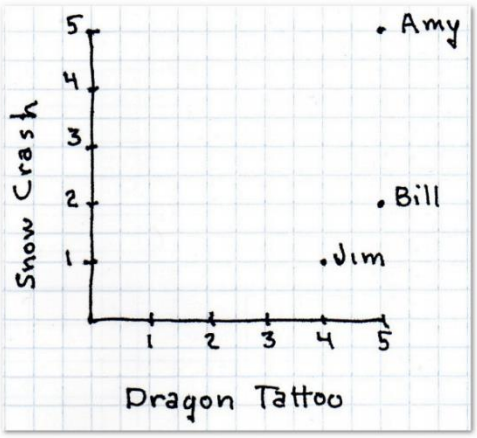
در مثالِ آمازون که در بالا گفته شد، آمازون دو تِکه از اطلاعات را برای ساختِ یک پیشنهاد جدید با هم ترکیب می‌کند. اولین تِکه، از آن‌جا گرفته می‌شود که مثلاً من کتاب The Lotus Sutra ترجمه‌ی Gene Reeves را می‌بینیم. و تکه‌ی دوم، از آن‌جا گرفته می‌شود که برای مثال مشتریانی که همین ترجمه را دیده‌اند، ترجمه‌های دیگری را نیز دیده باشند (مثلاً ترجمه‌های مختلف را در مرورگر باز کرده باشند).

روشی که در این فصل در مورد آن صحبت می‌کنیم، پایشِ مشارکتی (collaborative filtering) یا همان پایش با همکاری نام دارد. دلیل این‌که از کلمه‌ی **مشارکتی** در این عبارت استفاده شده است به خاطر این موضوع است که پیشنهادهای ارائه شده در این روش، بر اساسِ **افراد** طراحی می‌شود – در واقع، افراد با یکدیگر همکاری می‌کنند تا بتوانند یک سیستمِ توصیه‌گر بسازند. اگر بخواهیم با یک مثال، مسئله را روشن‌تر کنیم، مثالی مانند مسئله‌ی زیر می‌تواند کمک‌کننده باشد:

فرض کنید، وظیفه‌ی من این است که یک کتاب به شما معرفی کنم. من در میان کاربرانِ سایت جستجو کرده تا بتوانم کسی که علایقش در زمینه‌ی کتاب مشابه شما باشد را پیدا کنم. به محضِ این‌که آن شخصِ مشابه را پیدا کردم، می‌توانم ببینم آن شخص به چه کتابی

علاقه دارد و آن کتاب را به شما پیشنهاد دهم – مثلاً شاید The Windup Girl اثر Paolo Bacigalupi.

ولی چگونه یک فرد مشابه را پیدا کنم؟



خب! پس اولین قدم پیدا کردنِ یک فردِ مشابه است. شکل روبرو یک مثالِ دو بُعدی (2D) است. فرض کنید، کاربران، کتاب‌ها را در یک سیستمِ ۵ ستاره‌ای امتیاز می‌دهند (یعنی می‌توانند از ۱ تا ۵ به یک کتاب امتیاز دهند) – امتیاز صفر به معنیِ این است که کتاب واقعاً افتضاح بوده و امتیاز ۵ به معنیِ عالی بودنِ آن کتاب است. چون اولِ این پاراگراف گفتم که ما مسئله را در یک مثالِ **دو بُعدی (2D)** ساده‌سازی کردیم، پس فعلاً **دو کتاب** را مورد بررسی قرار می‌دهیم: کتابِ Snow Crash اثر Neal Stephenson و کتابِ The Girl with the Dragon Tattoo اثر Steig Larsson.

ابتدا، تعداد ۳ کاربر که هر کدام به این دو کتاب امتیازهای مختلفی داده‌اند را در جدول زیر نمایش می‌دهیم (Amy، Bill و Jim کاربران ما هستند):

	Snow Crash	Girl with the Dragon Tattoo
Amy	5☆	5☆
Bill	2☆	5☆
Jim	1☆	4☆

حالا می‌خواهم یک کتاب به خانم X (یک خانمِ فرضی) پیشنهاد دهم. این خانمِ X کتابِ Snow Crash را با امتیاز ۴ و کتابِ The Girl with the Dragon Tattoo را با امتیاز ۲، ارزیابی کرده است. اولین کار این است که شخصی را پیدا کنیم که بیشترین شباهت به این

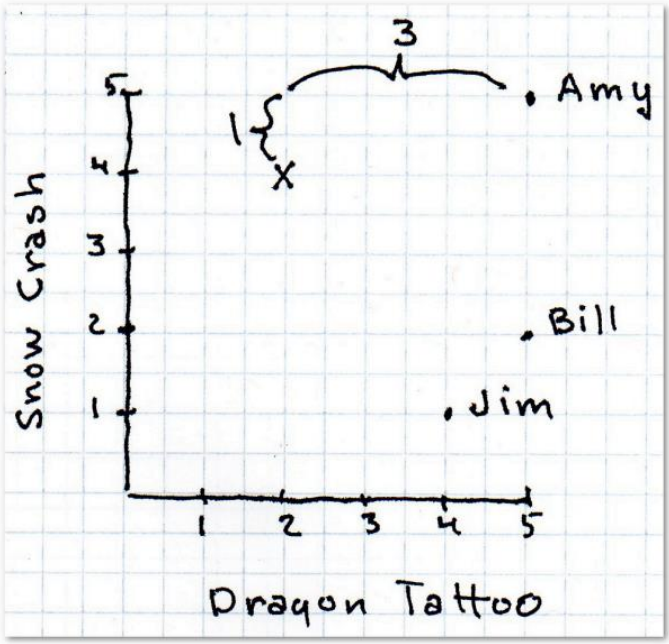
خانم داشته باشد. یا بهتر است بگوییم به این خانم نزدیک‌تر باشد. من این کار را با **محاسبه‌ی فاصله** انجام می‌دهم.

فاصله‌ی مَنهَتَن (Manhattan Distance)

ساده‌ترین معیارِ محاسبه‌ی فاصله، فاصله‌ی منهتن است که به فاصله‌ی «راننده‌ی تاکسی» نیز شهرت دارد. در این مثال که دو بُعد داشت، هر کدام از افرادِ امتیازدهنده، با یک نقطه‌ی (x, y) نمایش داده می‌شوند. برای مشخص کردنِ افرادِ خاص، از یک زیرنویس برای x و y استفاده می‌کنم. پس برای مثال (x_1, y_1) می‌تواند Amy (در جدولِ بالا) باشد و یا (x_2, y_2) می‌تواند خانم X باشد.
خب، برویم سراغِ منهتن. فاصله‌ی منهتن به صورت زیر محاسبه می‌شود:

$$|\mathbf{x1} - \mathbf{x2}| + |\mathbf{y1} - \mathbf{y2}|$$

(یعنی قدر مطلقِ اختلافِ xها و قدر مطلق اختلاف yها را باید با هم جمع می‌کنیم). سرانجام با استفاده از همین فرمول، فاصله‌ی منهتن بین Amy و خانم X برابر ۴ است. چیزی مانند شکل زیر:



در جدول زیر، فاصله‌ی خانمِ X با افراد مختلفِ جدول بالا یعنی Amy، Bill و Jim نمایش داده شده است:

	فاصله نسبت به خانم X
Amy	4
Bill	5
Jim	5

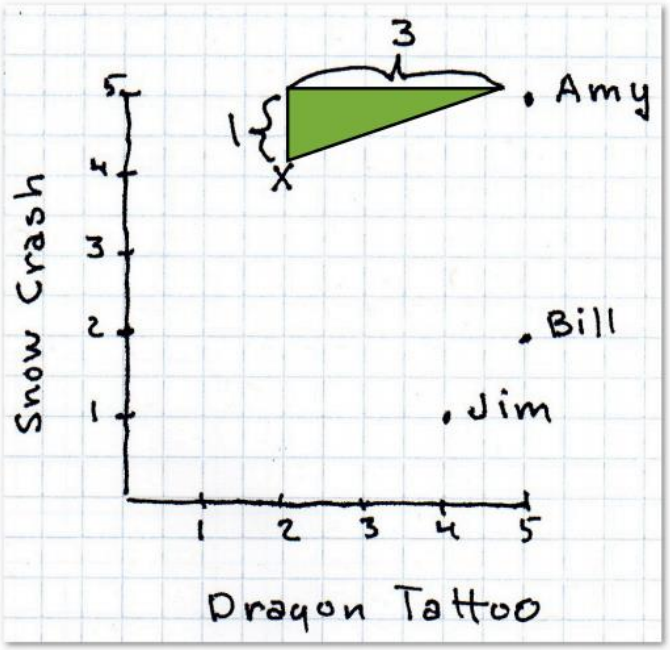
در این جدول مشاهده می‌کنید که Amy بیشترین شباهت را به خانمِ X دارد یعنی به خانم X نزدیک‌تر است. حالا ما می‌توانیم به سابقه‌ی Amy نگاه کنیم و ببینیم برای مثال، Amy به کتاب The Windup Girl اثر Paolo Bacigalupi، امتیاز ۵ را داده است، پس این کتاب را به خانمِ X هم پیشنهاد می‌دهیم.

فاصله‌ی اقلیدسی (Euclidean Distance)

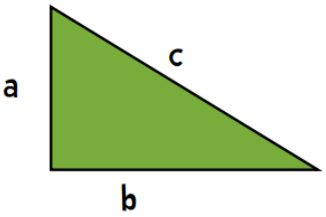
یکی از مزیت‌های فاصله‌ی منهتن، سرعت بالای محاسبه‌ی آن است. اگر ما مثلاً سایت فیس‌بوک باشیم و بخواهیم یکی از افرادِ مشابهِ Danny در شهرِ Kalamazoo را از بین یک میلیون کاربر پیدا کنیم، سرعت محاسبات برایمان مهم می‌شود.

تئوریِ فیثاغورث

احتمالاً تئوریِ فیثاغورث را از ریاضیاتِ دوران دبیرستان به یاد دارید. در این‌جا به جای محاسبه‌ی فاصله‌ی منهتن بین Amy و خانم X که برابر ۴ می‌شد، می‌خواهیم خط مستقیم (مانند پرواز مگس) را محاسبه کنیم:



تئوری فیثاغورث به ما می‌گوید که چگونه این فاصله را محاسبه کنیم:



$$c = \sqrt{a^2 + b^2}$$

خط مستقیم، مانند پرواز مگس، همان فاصله‌ی اقلیدسی است (c در شکلِ بالا). فرمول زیر را مشاهده کنید:

$$\sqrt{(\mathbf{x_1} - \mathbf{x_2})^2 + (\mathbf{y_1} - \mathbf{y_2})^2}$$

یادتان می‌آید که x_1 عددی بود که نشان می‌داد شخصِ شماره‌ی ۱، چقدر کتاب Dragon Tattoo را دوست داشت و x_2 نشان می‌داد که چقدر شخصِ شماره‌ی ۲، این کتاب را دوست داشت. به همین ترتیب y_1 بیان‌گر این است که شخصِ شماره‌ی ۱ چقدر کتاب Snow Crash را دوست دارد و y_2 بیان‌گر این است که شخص شماره‌ی ۲ چقدر این کتاب را دوست دارد.

Amy به هر دو کتابِ Snow Crash و Dragon Tattoo امتیازِ ۵ داده بود. خانم X به این کتاب‌ها به ترتیب امتیاز ۲ و ۴ داده است. پس فاصله ی اقلیدسی بین آن‌ها به صورت زیر محاسبه می شود:

$$\sqrt{(\mathbf{5} - \mathbf{2})^2 + (\mathbf{5} - \mathbf{4})^2} = \sqrt{\mathbf{10}} = \mathbf{3.16}$$

با محاسبه‌ی بقیه‌ی فاصله‌ها به جدول زیر می‌رسیم:

	فاصله نسبت به خانم X
Amy	3.16
Bill	3.61
Jim	3.61

تفکرِ چند بُعدی (N-Dimensional Thinking)

اجازه بدهید کمی از نگاه کردن به مسئله‌ی دو کتاب (به صورت دوبُعدی) خارج شویم و بتوانیم به چیزی که کمی پیچیده‌تر است برسیم. تصور کنید ما برای یک سرویسِ پخش آنلاین موسیقی کار می‌کنیم و می‌خواهیم گروه‌های موسیقی را به کاربرانِ مختلف پیشنهاد دهیم و به این طریق کاربران را به استفاده از این تجربه‌ها ترغیب کنیم. برای این‌کار کاربران می‌توانند گروه‌های موسیقی را در یک سیستمِ امتیازدهی بین ۱ تا ۵ امتیاز دهند (این امتیاز می‌تواند به صورت ۰,۵ امتیاز هم باشد – مثلا ۲,۵ امتیاز). در جدول زیر ۸ کاربر به همراه امتیازِ آن‌ها به ۸ گروه موسیقی نمایش داده شده است (ستون‌ها مانند Angelica کاربران را تشکیل می‌دهند و سطرها مانند Blues Traveler گروه‌های موسیقی هستند):

	Angelica	Bill	Chan	Dan	Hailey	Jordyn	Sam	Veronica
Blues Traveler	3.5	2	5	3	-	-	5	3
Broken Bells	2	3.5	1	4	4	4.5	2	-
Deadmau5	-	4	1	4.5	1	4	-	-
Norah Jones	4.5	-	3	-	4	5	3	5
Phoenix	5	2	5	3	-	5	5	4
Slightly Stoopid	1.5	3.5	1	4.5	-	4.5	4	2.5
The Strokes	2.5	-	-	4	4	4	5	3
Vampire Weekend	2	3	-	2	1	4	-	-

خط تیره به معنیِ این است که یک کاربر، به یک گروهِ موسیقی امتیاز نداده است. فعلاً، می‌خواهیم فاصله را بر مبنای گروه‌های موسیقی که هر دو نفر به آن‌ها امتیاز داده باشند محاسبه کنیم. پس برای مثال، اگر بخواهیم فاصله را بین Angelica و Bill محاسبه کنیم، باید امتیازهای گروه‌های Blues Traveler، Broken Bells، Phoenix، Slightly Stoopid

و Vampire Weekend که هر دوی این افراد به آن‌ها امتیاز داده‌اند، را در محاسباتِ خود اعمال کنیم. پس فاصله‌ی منهتن بین Angelica و Bill به صورت زیر محاسبه می‌شود:

	Angelica	Bill	Difference
Blues Traveler	3.5	2	1.5
Broken Bells	2	3.5	1.5
Deadmau5	-	4	
Norah Jones	4.5	-	
Phoenix	5	2	3
Slightly Stoopid	1.5	3.5	2
The Strokes	2.5	-	-
Vampire Weekend	2	3	1
Manhattan Distance:			9

فاصله‌ی منهتن (Manhattan Distance در سطر پایانی)، حاصل جمع تمامی تفریق‌ها (ستون Diffrence) بین امتیازاتِ متناظرِ Bill و Angelica بود (۱,۵ + ۱,۵ + ۳ + ۲ + ۱) که برابر با ۹ شده است.

محاسبه‌ی فاصله‌ی اقلیدسی نیز به همین صورت است (فقط آن گروه‌های موسیقی را در محاسبات در نظر می‌گیریم که هر دو – Angelica و Bill – به آن‌ها امتیاز داده‌اند):

	Angelica	Bill	Difference	Difference^2
Blues Traveler	3.5	2	1.5	2.25
Broken Bells	2	3.5	1.5	2.25
Deadmau5	-	4		
Norah Jones	4.5	-		
Phoenix	5	2	3	9
Slightly Stoopid	1.5	3.5	2	4
The Strokes	2.5	-	-	
Vampire Weekend	2	3	1	1
Sum of squares				18.5
Euclidean Distance				4.3

(ستون Diffrence که بیان‌گر تفریق‌های گروه‌های مشترکِ Angelica و Bill است. و ستون $Diffrence^2$ همان ستونِ قبلی است که به توان ۲ رسیده. از این ستون برای محاسبه‌ی «جمع مربعات – Sum of Squares در سطر یکی مانده به آخر و همچنین محاسبه‌ی فاصله‌ی اقلیدسی – Euclidean Distance در سطر آخر استفاده می‌کنیم)

اگر بخواهیم شکل بالا را به صورت ریاضی بنویسیم، چیزی مانند زیر می‌شود:

$$\sqrt{(3.5-2)^2+(2-3.5)^2+(5-2)^2+(1.5-3.5)^2+(2-3)^2} = \sqrt{18.5} = 4.3$$

گرفتید چی شد؟

حالا مثال زیر را خودتان انجام دهید...

	Angelica	Bill	Chan	Dan	Hailey	Jordyn	Sam	Veronica
Blues Traveler	3.5	2	5	3	-	-	5	3
Broken Bells	2	3.5	1	4	4	4.5	2	-
Deadmau5	-	4	1	4.5	1	4	-	-
Norah Jones	4.5	-	3	-	4	5	3	5
Phoenix	5	2	5	3	-	5	5	4
Slightly Stoopid	1.5	3.5	1	4.5	-	4.5	4	2.5
The Strokes	2.5	-	-	4	4	4	5	3
Vampire Weekend	2	3	-	2	1	4	-	-

مدادتان را تیز کنید

با توجه به جدولِ بالا،

فاصله‌ی اقلیدسی بین Hailey و Veronica را محاسبه کنید:

فاصله‌ی اقلیدسی بین Hailey و Jordyn را محاسبه کنید:

مدادتان را تیز کنید – راه‌حل

با توجه به جدولِ بالا،

فاصله‌ی اقلیدسی بین Hailey و Veronica را محاسبه کنید:

$$\sqrt{(4-5)^2 + (4-3)^2} = \sqrt{2} = 1.414$$

فاصله‌ی اقلیدسی بین Hailey و Jordyn را محاسبه کنید:

$$\sqrt{(4-4.5)^2 + (1-4)^2 + (4-5)^2 + (4-4)^2 + (1-4)^2} = \sqrt{19.25} = 4.87$$

یک عیب اساسی

به نظر می‌رسد که در هنگام استفاده از این معیارهای فاصله به یک مشکل برمی‌خوریم. هنگامی که فاصله‌ی بین Hailey و Veronica را محاسبه کردیم، متوجه شدیم که آن‌ها فقط در دو گروه موسیقی با هم مشترک هستند (Norah Jones و Ther Strokes) در حالی‌که وقتی فاصله‌ی بین Hailey و Jordyn را محاسبه کردیم، متوجه شدیم آن‌ها ۵ گروه را به صورت مشترک امتیازدهی کرده‌اند. با این اوصاف به نظر می‌رسد که که معیار محاسبه‌ی ما دچار مشکلِ چولگی (skew) شده است، و این اِشکال از آن‌جایی نشات می‌گیرد که فاصله‌ی بین Haily و Veronica در دو بُعد مورد محاسبه قرار گرفته است (به خاطر این‌که آن‌ها فقط ۲ گروه موسیقی را به صورت مشترک امتیاز داده بودند) در حالی‌که فاصله‌ی بین haily و Jordyn در ۵ بعد، مورد محاسبه قرار گرفته است. فاصله‌های منهتن و اقلیدسی هنگامی که هیچ داده‌ی گم شده یا مفقودی موجود نباشد، بهترین کارایی را دارند ولی در این‌جا در واقع **داده‌های مفقود شده** داشتیم. مواجهه با داده‌های گم شده یا مفقود شده یک حوزه‌ی تحقیقاتیِ فعال در بین محققان است. بعداً در همین کتاب در مورد برخورد و مواجهه با این مسئله صحبت خواهیم کرد. ولی برای الان فقط از این مشکل مطلع باشید و در گوشه‌ی ذهنتان به آن توجه کنید.

یک عمومی‌سازی (Generalization)

می‌توانیم فاصله‌ی منهتن و اقلیدسی را عمومی‌سازی کنیم. به این عمومی‌سازی، **معیار فاصله‌ی مینکوسفکی (Minkowski)** گفته می‌شود:

$$\mathbf{d}(\mathbf{x},\mathbf{y}) = \left(\sum_{k=1}^{n} |x_k - y_k|^r\right)^{\frac{1}{r}}$$

که در آن اگر:

r برابر ۱ باشد: فرمول تبدیل به فاصله‌ی منهتن می‌شود
r برابر ۲ باشد: فرمول تبدیل به فاصله‌ی اقلیدسی می‌شود
r برابر بی‌نهایت باشد: فرمول تبدیل به فاصله سوپرمم (supermum) می‌شود

اووووه ریاضیات!

وقتی که با فرمولی مانند فرمول بالا مواجه می‌شوید، چند راه جلوی پای شماست. یکی از راه‌ها این است که هنگامی که به یک فرمول ریاضی رسیدید به آن نگاه کنید – نورون‌های مغز فعال می‌شوند و می‌گویند: این یک فرمول ریاضی است – و سریعاً از روی آن رد شوید تا به متنِ بعدی برسید. باید قبول کنم که من هم زمانی یکی از پرش‌کنندگان بودم (و منِ مترجم هم!). ولی راهِ دیگر این است که فرمول را نگاه کنید، مکث کنید، و آن‌را کالبدشکافی و واکاوی کنید. در بسیاری از موارد، فرمول را خیلی قابل درک خواهید یافت.

اجازه بدهید همین الان این یکی را کالبدشکافی کنیم. در همان فرمول بالا هنگامی که r برابر ۱ باشد، فرمول تبدیل به فاصله‌ی منهتن می‌شود:

$$d(x, y) = \sum_{k=1}^{n} |x_k - y_k|$$

در موردِ مثالِ سیستمِ موسیقی و امتیازها که در این فصل از آن استفاده می‌کنیم، x و y نماینده‌ی دو شخص هستند و d(x, y) بیان‌گر فاصله‌ی میان آن‌هاست. n تعداد گروه‌های موسیقی‌ای است که این دو شخص به صورت مشترک به آن‌ها امتیاز داده‌اند. محاسبات را در چند پاراگراف بالاتر انجام دادیم:

	Angelica	Bill	Difference
Blues Traveler	3.5	2	1.5
Broken Bells	2	3.5	1.5
Deadmau5	-	4	
Norah Jones	4.5	-	
Phoenix	5	2	3
Slightly Stoopid	1.5	3.5	2
The Strokes	2.5	-	-
Vampire Weekend	2	3	1
Manhattan Distance:			9

در ستون اختلاف (Difference) میزان قدر مطلق اختلاف (تفریق) بین امتیازات، مشخص شده است و در آخر جمعِ آن‌ها را محاسبه کردیم که عدد ۹ به دست آمد.
هنگامی که r برابر ۲ باشد، به فاصله‌ی اقلیدسی می‌رسیم:

$$\mathbf{d(x,y)} = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

قاعده‌ی کلی در مورد فرمولِ مینکوفسکی به این صورت است:
هر چه r بزرگ‌تر باشد، اختلافِ (حاصل تفریقِ) بیشتر در یک بُعد، تاثیر بیشتری بر نتیجه‌ی نهایی دارد.

نمایش داده‌ها در زبان پایتون

روش‌های مختلفی برای نمایش داده‌های جدول بالا (گروه‌های موسیقی و امتیاز کاربران به آن‌ها) در پایتون وجود دارد. من می‌خوام از **دیکشنری‌ها** در پایتون استفاده کنم (دیکشنری یکی از انواع متغیرها در پایتون هستند). این دیکشنری‌ها، به اسم آرایه‌های انجمنی یا جدولِ درهم‌ساز (hash table) نیز شناخته می‌شوند.

> **یادآوری**
> **تمامی کدها در سایت guidetodatamining.com و همچنین سایت chistio.ir موجود است.**

```
users = {
    'Angelica': {
        'Blues Traveler': 3.5,
        'Broken Bells': 2.0,
        'Norah Jones': 4.5,
        'Phoenix': 5.0,
        'Slightly Stoopid': 1.5,
        'The Strokes': 2.5,
        'Vampire Weekend': 2.0,
        },
    'Bill': {
        'Blues Traveler': 2.0,
```

```
        'Broken Bells': 3.5,
        'Deadmau5': 4.0,
        'Phoenix': 2.0,
        'Slightly Stoopid': 3.5,
        'Vampire Weekend': 3.0,
        },
    'Chan': {
        'Blues Traveler': 5.0,
        'Broken Bells': 1.0,
        'Deadmau5': 1.0,
        'Norah Jones': 3.0,
        'Phoenix': 5,
        'Slightly Stoopid': 1.0,
        },
    'Dan': {
        'Blues Traveler': 3.0,
        'Broken Bells': 4.0,
        'Deadmau5': 4.5,
        'Phoenix': 3.0,
        'Slightly Stoopid': 4.5,
        'The Strokes': 4.0,
        'Vampire Weekend': 2.0,
        },
    'Hailey': {
        'Broken Bells': 4.0,
        'Deadmau5': 1.0,
        'Norah Jones': 4.0,
        'The Strokes': 4.0,
        'Vampire Weekend': 1.0,
        },
    'Jordyn': {
        'Broken Bells': 4.5,
        'Deadmau5': 4.0,
        'Norah Jones': 5.0,
        'Phoenix': 5.0,
        'Slightly Stoopid': 4.5,
        'The Strokes': 4.0,
        'Vampire Weekend': 4.0,
        },
    'Sam': {
        'Blues Traveler': 5.0,
        'Broken Bells': 2.0,
        'Norah Jones': 3.0,
        'Phoenix': 5.0,
        'Slightly Stoopid': 4.0,
        'The Strokes': 5.0,
```

```
        },
    'Veronica': {
        'Blues Traveler': 3.0,
        'Norah Jones': 5.0,
        'Phoenix': 4.0,
        'Slightly Stoopid': 2.5,
        'The Strokes': 3.0,
        },
    }
```

می‌توانیم امتیازاتِ یک شخص به گروه‌های موسیقی را به صورت زیر به دست بیاوریم:

```
>>> users["Veronica"]
{"Blues Traveler": 3.0, "Norah Jones": 5.0, "Phoenix": 4.0,
"Slightly Stoopid": 2.5, "The Strokes": 3.0}
```

کد پایتون برای محاسبه‌ی فاصله‌ی منهتن

می‌خواهم تابعی مانند تابعِ زیر بنویسم تا فاصله‌ی منهتن را محاسبه کند:

```
def manhattan(rating1, rating2):
    """Computes the Manhattan distance. Both rating1 and
 rating2 are
    dictionaries of the form
    {'The Strokes': 3.0, 'Slightly Stoopid': 2.5 ..."""

    distance = 0
    for key in rating1:
        if key in rating2:
            distance += abs(rating1[key] - rating2[key])
    return distance
```

برای تستِ این تابع می‌توانید به این صورت عمل کنید:

```
>>> manhattan(users['Hailey'], users['Veronica'])
2.0
>>> manhattan(users['Hailey'], users['Jordyn'])
7.5
>>>
```

حالا به سراغ تابعی می‌رویم که بتواند نزدیک‌ترین فرد را به یک فردِ دیگر (با توجه به امتیازات داده شده به گروه‌های موسیقی) پیدا کند. البته در واقع این تابع یک لیستِ مرتب‌شده را برمی‌گرداند که اولین خانه‌ی آن، نزدیک‌ترین فرد به فردِ مورد نظر است:

```
def computeNearestNeighbor(username, users):
    """creates a sorted list of users based on their dis
tance to
    username"""

    distances = []
    for user in users:
        if user != username:
            distance = manhattan(users[user], users[user
name])
            distances.append((distance, user))

    # sort based on distance -- closest first

    distances.sort()
    return distances
```

و یک تست از این تابع:

```
>>> computeNearestNeighbor("Hailey", users)
[(2.0, ''Veronica'), (4.0, 'Chan'),(4.0, 'Sam'), (4.5, 'Dan'), (5.0,
'Angelica'), (5.5, 'Bill'), (7.5, 'Jordyn')]
```

در آخر، تمامیِ کدها را کنار هم قرار می‌دهیم تا پیشنهادها را آماده کنیم. بیایید یک پیشنهاد برای Hailey بسازیم. می‌خواهم نزدیک‌ترین همسایه را به او پیدا کنم – که در این مورد Veronica می‌شود. بعد از آن به دنبال گروه‌هایی خواهم گشت که Veronica به آن‌ها امتیاز داده است ولی Haily خیر. همچنین، فرض می‌کنم که Haily، گروه‌های موسیقی را مانند Veronica (یا خیلی شبیه به او) امتیاز می‌دهد. برای مثال، Hailey به گروه موسیقیِ Phoenix امتیازی نداده است ولی Veronica به این گروه امتیاز ۴ داده‌است. پس فرض می‌کنیم که Hailey هم احتمالاً این گروه موسیقی را مانند Veronica دوست دارد.

در زیر تابعی که برای این پیشنهاد نوشتم را مشاهده می‌کنید:

```
def recommend(username, users):
    """Give list of recommendations"""

    # first find nearest neighbor

    nearest = computeNearestNeighbor(username, users)[0]
[1]
    recommendations = []

    # now find bands neighbor rated that user didn't

    neighborRatings = users[nearest]
    userRatings = users[username]
    for artist in neighborRatings:
        if not artist in userRatings:
            recommendations.append((artist, neighborRati
ngs[artist]))

    # using the fn sorted for variety - sort is more eff
icient

    return sorted(recommendations, key=lambda artistTupl
e: \
                  artistTuple[1], reverse=True)
```

و حالا برای ایجادِ یک یا چند پیشنهاد به Hailey می‌توانید مانند شکل زیر عمل کنید:

```
>>> recommend('Hailey', users)
[('Phoenix', 4.0), ('Blues Traveler', 3.0), ('Slightly Stoopid', 2.5)]
```

که نتیجه مطابق انتظار ما بود. همان‌طور که در شکلِ بالا مشاهده می‌کنیم، نزدیک‌ترین همسایه به Haily، Veronica است و Veronica به گروه موسیقی Phoenix امتیاز ۴ داده است. اجازه بدهید چند مورد دیگر را امتحان کنیم:

```
>>> recommend('Chan', users)
[('The Strokes', 4.0), ('Vampire Weekend', 1.0)]

>>> recommend('Sam', users)
[('Deadmau5', 1.0)]
```

پس فکر می‌کنیم که Chan گروه موسیقیِ Strokes را دوست خواهد داشت. همچنین پیش‌بینی می‌کنیم که Sam، گروه موسیقی Deadmau5 را دوست نخواهد داشت.

```
>>> recommend('Angelica', users)
[]
```

خب! برای Angelica نتیجه خالی بود. در واقع چیزی برای پیشنهاد به او نداشتیم. اجازه بدهید ببینیم مشکل از کجاست:

```
>>> computeNearestNeighbor('Angelica', users)
[(3.5, 'Veronica'), (4.5, 'Chan'), (5.0, 'Hailey'), (8.0, 'Sam'), (9.0,
'Bill'), (9.0, 'Dan'), (9.5, 'Jordyn')]
```

نزدیک‌ترین همسایه به Angelica، Veronica است. هنگامی که به امتیازاتِ این دو نگاه می‌کنیم – شکل زیر:

	Angelica	Bill	Chan	Dan	Hailey	Jordyn	Sam	Veronica
Blues Traveler	3.5	2	5	3	-	-	5	3
Broken Bells	2	3.5	1	4	4	4.5	2	-
Deadmau5	-	4	1	4.5	1	4	-	-
Norah Jones	4.5	-	3	-	4	5	3	5
Phoenix	5	2	5	3	-	5	5	4
Slightly Stoopid	1.5	3.5	1	4.5	-	4.5	4	2.5
The Strokes	2.5	-	-	4	4	4	5	3
Vampire Weekend	2	3	-	2	1	4	-	-

مشاهده می‌کنیم که Angelica به تمامیِ گروه‌های موسیقی‌ای امتیاز داده است که Veronica هم امتیاز داده. در واقع ما امتیازِ جدیدی در این بین نداشتیم، پس پیشنهادی هم برای او در میان گزینه‌های ما نبود.

در قسمت‌های آینده، نشان خواهیم داد چگونه می‌توان سیستم را تقویت کرد تا به یک همچین مشکلاتی برنخورد.

تمرین:

۱- تابعی برای فاصله‌ی مینکوفسکی با زبان پایتون پیاده‌سازی کنید:

۲- تابع computeNearestNeighbor را در بالا تغییر دهید به گونه‌ای که از فاصله‌ی مینکوفسکی استفاده کند:

تمرین – راه‌حل:

۱- تابعی برای فاصله‌ی مینکوفسکی با زبان پایتون پیاده‌سازی کنید:

```
def minkowski(rating1, rating2, r):
    """Computes the Minkowski distance.
    Both rating1 and rating2 are dictionaries of the f
orm
    {'The Strokes': 3.0, 'Slightly Stoopid': 2.5}"""

    distance = 0
    commonRatings = False
    for key in rating1:
        if key in rating2:
            distance += pow(abs(rating1[key] - rating2
[key]), r)
            commonRatings = True
    if commonRatings:
        return pow(distance, 1 / r)
    else:
        return 0  # Indicates no ratings in common
```

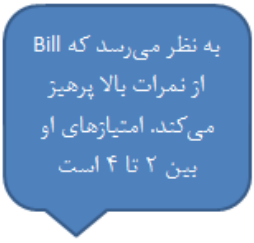
۲- تابع computeNearestNeighbor را در بالا تغییر دهید به گونه‌ای که از فاصله‌ی مینکوفسکی استفاده کند:

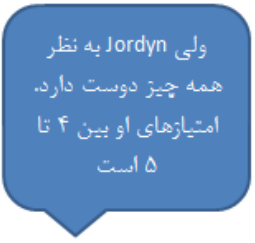
```
distance = minkowski(users[user], users[username], 2)
```

عددِ ۲ در ورودیِ تابع بالا، به معنای این‌است که به دنبال فاصله‌ی اقلیدسی هستیم.

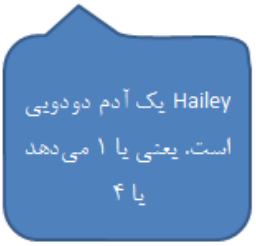
یک شرمساری برای کاربران

اجازه بدهید کمی دقیق‌تر به امتیازاتِ کاربران نگاه کنیم. مشاهده می‌کنیم که کاربران، رفتارهای بسیار متفاوتی برای امتیاز دادن به گروه‌های موسیقی در پایگاه‌داده‌ی ما دارند:





	Angelica	Bill	Chan	Dan	Hailey	Jordyn	Sam	Veronica
Blues Traveler	3.5	2	5	3	-	-	5	3
Broken Bells	2	3.5	1	4	4	4.5	2	-
Deadmau5	-	4	1	4.5	1	4	-	-
Norah Jones	4.5	-	3	-	4	5	3	5
Phoenix	5	2	5	3	-	5	5	4
Slightly Stoopid	1.5	3.5	1	4.5	-	4.5	4	2.5
The Strokes	2.5	-	-	4	4	4	5	3
Vampire Weekend	2	3	-	2	1	4	-	-



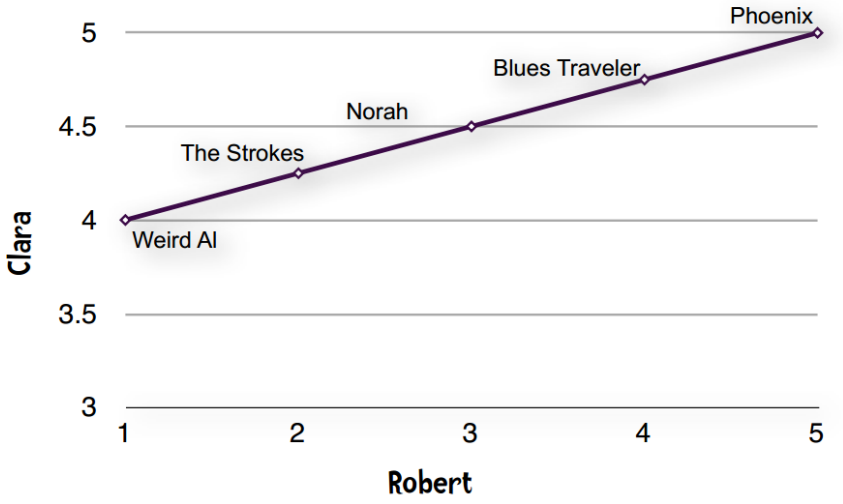
خب پس چگونه ما Hailey را با Jordan مقایسه کنیم؟ آیا اگر Hailey به یک گروه موسیقی امتیاز ۴ داده باشد، مانند این است که Jordyn به آن گروه امتیاز ۴ داده است؟ یا خیر، مانند این است که Jordyn به آن گروه امتیاز ۵ داده است؟ به نظر من، اگر Hailey به گروهی امتیاز ۴ داده باشد مانند این است که Jordyn به آن گروه امتیاز ۵ داده است (در واقع امتیاز ۴ برای Hailey به نوعی معادلِ امتیاز ۵ برای Jordyn است). این تنوع می‌تواند منجر به ایجاد مشکلاتی در یک سیستمِ توصیه‌گر شود.

ضریب همبستگی پیرسون (Pearson Correlation Coefficient)

یکی از راه‌های غلبه بر این مشکل (تنوع در امتیازدهی)، استفاده از ضریب همبستگیِ پیرسون است. اجازه بدهید اول یک ایده‌ی کلی را مطرح کنیم. داده‌های زیر را در نظر بگیرید (این داده‌ها با داده‌های قبلی متفاوت هستند) (در شکل زیر سطرها، اشخاص هستند و ستون‌ها بیان‌گر گروه‌ها یا هنرمندان موسیقی):

	Blues Traveler	Norah Jones	Phoenix	The Strokes	Weird Al
Clara	4.75	4.5	5	4.25	4
Robert	4	3	5	2	1

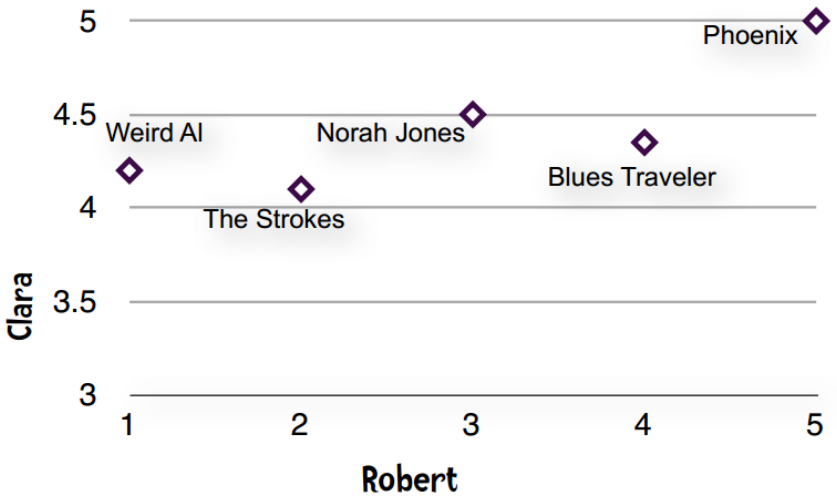
این مثالی از «**تورم درجه — grade inflation**» در حوزه‌ی داده‌کاوی است. در جدول بالا، کمترین امتیازِ Clara، 4 است — در واقع تمامی امتیازاتی که او به گروه‌های موسیقی داده‌است، بین ۴ و ۵ بوده‌اند. اگر جدول بالا را بر روی یک گراف رسم کنیم، چیزی مانند شکل زیر می‌شود:



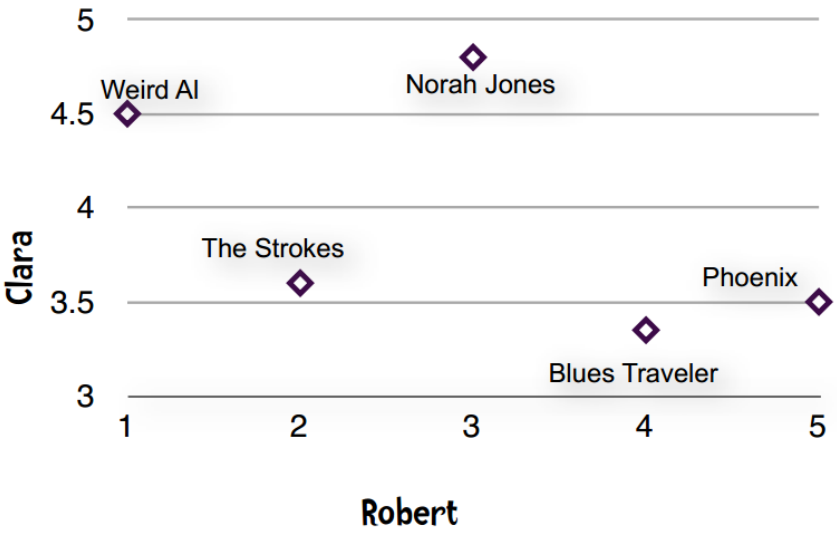
خط مستقیم = تفاهم کامل!!!

واقعیت این است که خط مستقیمی که در گراف بالا رسم شده است، نشان‌دهنده‌ی تفاهمِ کامل بین Clara و Robert است. هر دوی آن‌ها، گروه موسیقیِ Phoenix را به عنوانِ بهترین امتیازِ خود محاسبه کرده‌اند، بعد از آن گروه Blues Traveler و بعد Norah Jones و الی آخر. هر چقدر که Clara و Robert کمتر با هم در امتیازدهی به گروه‌های موسیقی تفاهم داشته باشند، داده‌ها در گراف بالا، کمتر در یک خط راست قرار می‌گیرند.

برای مثال، شکل زیر یک تفاهم تقریبی را بین دو کاربر نشان می‌دهد:



و شکل زیر تفاهمِ کمتر از شکل‌های قبلی را نمایش می‌دهد:



پس اگر بخواهیم بر اساس نمودارها صحبت کنیم، تفاهمِ کامل با یک خط مستقیم نشان داده می‌شود. **ضریب همبستگی پیرسون**، معیاری برای محاسبه‌ی همبستگی بین دو متغیر است (در مثال ما، همبستگی بین Angelica و Bill). ضریب همبستگیِ پیرسون عددی در بازه‌ی ۱- تا ۱ (شامل خود ۱- و ۱ هم می‌شود) است. **عددِ ۱ به معنای تفاهم و همبستگی کامل** بین دو متغیر است و **۱- به معنای عدم تفاهم کامل**. برای این‌که یک درک کلی از ماجرا داشته باشید، در نمودارهای بالا، نمودار اولی که یک خط مستقیم داشت، ضریب همبستگی پیرسون در آن برابر ۱ بود. نمودار بعدی (دومی) دارای ضریب هبستگیِ پیرسون برابر ۰٫۹۱ بود و آخری (سومی) هم ضریب همبستگی پیرسون ۰٫۸۱ را داشت. بنابراین از این معیار می‌توانیم برای پیدا کردنِ افرادی که به فرد مورد نظر ما شبیه‌تر هستند، استفاده کنیم.

فرمول اصلی برای ضریب همبستگی پیرسون به صورت زیر است:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\ \sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

دوباره ریاضیات!

یک اعتراف شخصی. من مدرکِ کارشناسیِ هنرهای زیبا در رشته موسیقی دارم. هنگامی که دروسی درباره رقص باله، رقص مدرن و طراحی لباس بر می‌داشتم حتی یک درس ریاضی هم به عنوان یک دانشجو نداشتم. قبل از آن، وارد یک دبیرستان پسرانه‌ی تجارت شدم و دروسی در حوزه‌ی لوله‌کشی و تعمیر خودرو را برداشتم، ولی بازهم هیچ درسی در مورد ریاضیات (البته به جز دروس پایه) برنداشتم. پس یا به علت این پیش‌زمینه و یا به علت ساختار داخلیِ ذهنی، هنگامی که کتابی شامل فرمول‌هایی شبیه به فرمول بالا را می‌خواندم، تمایل داشتم که از روی فرمول‌ها پرش کنم و به جملات پایین آن‌ها برسم. اگر شما هم شبیه به من هستید باید اصرار کنم که با این حالت در خود مبارزه کنید و به فرمول نگاه کنید و

پرش نکنید. بسیاری از فرمول‌هایی که با یک نگاه جزئی، بسیار پیچیده به نظر می‌رسند، در واقع کاملاً قابل درک و فهم هستند.

جدای از پیچدگی ظاهری، مشکل فرمول بالا، این است که الگوریتمی که می‌خواهد فرمولِ بالا را پیاده‌سازی کند، نیاز دارد تا چند بار از روی داده‌ها رد شود (اصطلاحاً به این الگورتیم‌ها multi pass می‌گویند). خوشبختانه برای ما آدم‌های الگوریتمی، یک فرمولِ جایگزین وجود دارد که تقریبی از همان پیرسون است:

$$r = \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}}{\sqrt{\sum_{i=1}^{n} x_i{}^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}} \sqrt{\sum_{i=1}^{n} y_i{}^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}}}$$

(یادتان باشد که نباید از فرمول‌ها پرش کنید!) این فرمول، علاوه بر این‌که در ابتدا بسیار پیچیده به نظر می‌رسد، مهمتر از آن، از لحاظِ عددی **غیرپایدار (unstable)** است، به این معنی که اگر چیزی باعثِ خطا شود، در این اصلاحِ فرمول، تشدید خواهد شد. ولی مزیت بزرگی که این فرمول دارد این است که می‌توانیم آن‌را با یک بار رد شدن از روی داده‌ها (به اصطلاح single pass) پیاده‌سازی کنیم. ولی اول اجازه دهید این فرمول را کالبدشکافی کرده و با مثالی که چند پاراگراف بالاتر گفته بودیم، تشریح کنیم:

	Blues Traveler	Norah Jones	Phoenix	The Strokes	Weird Al
Clara	4.75	4.5	5	4.25	4
Robert	4	3	5	2	1

برای شروع، اجازه دهید فرمول زیر را محاسبه کنیم:

$$\sum_{i=1}^{n} x_i y_i$$

این فرمول، در صورتِ کسرِ فرمول اول دیده می‌شد. در این‌جا برای مثال x و y به ترتیب نمایان‌گر Clara و Robert هستند.

	Blues Traveler	Norah Jones	Phoenix	The Strokes	Weird Al
Clara	4.75	4.5	5	4.25	4
Robert	4	3	5	2	1

برای هر کدام از گروه‌های موسیقی، امتیازات داده‌شده توسط Clara و Robert را به صورت نظیر به نظیر با هم ضرب می‌کنیم و حاصلِ این ضرب‌ها را با هم جمع می‌کنیم. چیزی مانند شکل زیر:

$$(4.75 \times 4) + (4.5 \times 3) + (5 \times 5) + (4.25 \times 2) + (4 \times 1)$$
$$=19 + 13.5 + 25 + 8.5 + 4 = 70$$

خوب شد. حالا اجازه دهید بقیه‌ی فرمول‌ها در صورت کسر را محاسبه کنیم:

$$\frac{\sum_{i=1}^{n} x_i \ \sum_{i=1}^{n} y_i}{n}$$

در این فرمول، عبارت

$$\sum_{i=1}^{n} x_i$$

جمع امتیازاتی است که Clara به گروه‌های موسیقی داده‌است، که برابر ۲۲٫۵ می‌شود. جمع امتیازات Robert نیز ۱۵ شده است و آن‌ها هر کدام به ۵ گروهِ موسیقی، امتیاز داده‌اند.

$$\frac{22.5 \times 15}{5} = 67.5$$

پس صورت کسر به صورت زیر محاسبه می‌شود:

$$70 - 67.5 = 2.5$$

خب! حالا اجازه دهید مخرج کسر را محاسبه کنیم:

$$\sqrt{\sum_{i=1}^{n} x_i^2 - \frac{(\sum_{i=1}^{n} x_i)^2}{n}}$$

اول، مانندِ زیر محاسبات را انجام می‌دهیم:

	Blues Traveler	Norah Jones	Phoenix	The Strokes	Weird Al
Clara	4.75	4.5	5	4.25	4
Robert	4	3	5	2	1

$$\sum_{i=1}^{n} x_i^2 = (4.75)^2 + (4.5)^2 + (5)^2 + (4.25)^2 + (4)^2 = 101.875$$

الان ما امتیازاتِ داده‌شده توسط Clara را با هم جمع کردیم که ۲۲,۵ شد. مجذور آن برابر ۵۰۶,۲۵ می شود. حالا آن‌را تقسیم بر تعداد گروه‌هایی که هر دوی این اشخاص به آن‌ها امتیاز داده‌اند (که ۵ گروه بودند) می‌کنیم و نتیجه برابر **۱۰۱,۲۵** می‌شود.
حالا همه‌ی این‌ها را با توجه به فرمول قبلی محاسبه می‌کنیم:

$$\sqrt{101.875 - 101.25} = \sqrt{0.625} = 0.79057$$

همین‌کار را برای Robert نیز باید انجام دهیم:

$$\sqrt{\sum_{i=1}^{n} y_i^2 - \frac{(\sum_{i=1}^{n} y_i)^2}{n}} = \sqrt{55 - 45} = 3.162277$$

حالا همه‌ی اعداد را در فرمول اصلی می‌گذاریم:

$$r = \frac{2.5}{.79057(3.162277)} = \frac{2.5}{2.5} = 1.00$$

که نتیجه برابر ۱ می‌شود و این عددِ ۱ به معنی این است که تفاهم کاملی بین Clara و Robert وجود دارد.

حالا کمی استراحت کنید، تا ادامه دهیم!!

آخرین فرمول – شباهت کسینوسی (Cosine Similarity)

می‌خواهیم آخرین فرمول، که بسیار در حوزه‌ی پردازش متن معروف است را معرفی کنم. این فرمول همچنین در پایشِ مشارکتی (collaborative filtering) نیز استفاده می‌شود – به این فرمول **شباهت کسینوسی** می‌گویند. برای این‌که ببینیم چه موقع باید از این فرمول استفاده کنیم، اجازه دهید مثالِ قبلی را کمی تغییر دهیم. ما تعداد دفعاتی را که یک شخص، موسیقیِ خاصی را پخش می‌کند نگه می‌داریم و از این اطلاعات برای سیستمِ پیشنهاده‌دهنده استفاده می‌کنیم.

	تعداد دفعات پخش		
موسیقی / شخص	The Decemberists The King is Dead	Radiohead The King of Limbs	Katy Perry E.T.
Ann	10	5	32
Ben	15	25	1
Sally	12	6	27

فقط با نگاهِ چشمی به جدولِ بالا (و با استفاده از هر کدام از فرمول‌های گفته شده در این فصل) می‌توانیم بفهمیم که Sally در موردِ سلیقه‌ی موسیقیایی بیشتر شبیهِ Ann است تا Ben.

خب، مشکل کجاست؟

من حدود چهارهزار قطعه‌ی موسیقی در iTunes دارم. شکل زیر، تصویری از تعدادی از این قطعات است که بر اساس میزانِ پخش مرتب شده‌اند:

✓	Name	Time	Artist	Album	Genre	Plays ▾
✓	Moonlight Sonata	7:38	Marcus Miller	Silver Rain	Jazz+Funk	25
✓	Blast!	5:43	Marcus Miller	Marcus	Jazz	20
✓	Art Isn't Real (City of Sin)	2:48	Deer Tick	War Elephant	Alt-Country	19
✓	Between the Lines	4:35	Sara Bareilles	Little Voice	Folk	19
✓	Stay Around A Little Longer (Feat. B.B. King)	5:00	BUDDY GUY	Living Proof	Blues	18
✓	My Companjera	3:22	Gogol Bordello	Trans-Continental...	Alternative...	18
✓	Rebellious Love	3:57	Gogol Bordello	Trans-Continental...	Alternative...	18
✓	Immigraniada (We Comin' Rougher)	3:46	Gogol Bordello	Trans-Continental...	Alternative...	18
✓	Love Song	4:19	Sara Bareilles	Little Voice	Folk	18

بهترین موسیقیِ من، آهنگ Moonlight Sonata اثر Marcus Miller است که ۲۵ مرتبه پخش شده‌است. این احتمال وجود دارد که شما این آهنگ را اصلاً پخش نکرده باشید. در واقع، احتمالِ زیادی وجود دارد که شما هیچ کدام از آهنگ‌های مورد علاقه‌ی من را اصلاً گوش نکرده باشید. به علاوه‌ی این‌که، بیش از ۱۵ میلیون قطعه‌ی موسیقی در iTunes موجود است و من فقط چهارهزارتای آن‌ها را دارم. بنابراین داده‌ها برای یک فرد در این سایت، تُنُک (sparse) است (یعنی تعداد کمی از آن دارای عدد هستند و تعداد زیادی از آن‌ها صفر هستند) و به این دلیل است که این داده‌ها تعداد نسبتاً کمی ویژگی‌های غیر صفر دارند (منظور از ویژگی‌ها – attributes - در این‌جا، تعداد دفعات پخش قطعه‌ی موسیقی است). هنگامی که ما دو شخص را با توجه به تعداد دفعات پخشِ قطعاتِ موسیقی در میان ۱۵ میلیون قطعه‌ی موسیقیِ موجود مقایسه می‌کنیم، در بیشتر مواقع آن‌ها هیچ نقطه‌ی اشتراکی ندارند و در واقع بیشترین اشتراک آن‌ها صفرها هستند. بااین‌حال، نمی‌خواهیم این صفرها در محاسباتمان نقش داشته باشند.

مورد مشابه می‌تواند هنگامی باشد که بخواهیم اسنادِ متنی (documents) را با استفاده از کلمات، با هم مقایسه کنیم. فرض کنید، ما یک کتاب خاص را دوست داریم، مثلاً کتاب «پیشگامان فضا» اثر Carey Rockwell، و می‌خواهیم یک کتابِ مشابه به آن پیدا کنیم. یکی از راه‌ها این است که از تعداد تکرار کلمات استفاده کنیم. در این‌جا، ویژگی‌ها، کلماتِ یکتا هستند و مقدارِ این ویژگی‌ها، تعدادِ تکرارِ آن‌ها در کتاب هستند. ۶,۱۳ درصد از کلمات در کتابِ «پیشگامان فضا»، کلمه‌ی «این – the» هستند، ۰,۸۹ درصد کلمه‌ی «Tom» هست و ۰,۲۵ درصد کلمه‌ی «space». من می‌توانم شباهتِ این کتاب را به بقیه‌ی کتاب‌ها با استفاده از این تعداد تکرار کلمات محاسبه کنم. ولی، مشکلِ تُنُک بودن

(sparseness) در داده‌ها، این‌جا هم اتفاق می‌افتد. تعداد ۶۶۲۹ کلمه‌ی یکتا در کتاب «پیشگامان فضا» وجود دارد. بیش‌از ۱ میلیون کلمه‌ی یکتا در زبان انگلیسی موجود است. پس اگر ویژگی‌های ما کلمات انگلیسی باشند، تقریباً تعداد کمی ویژگیِ غیر صفر در کتاب «پیشگامان فضا» یا هر کتاب دیگری وجود دارد. **دوباره تکرار می‌کنم که هر معیار شباهتی، بهتر است که وابسته به صفرهای مشترک نباشند.**

شباهت کسینوسی، اشتراک‌های ۰-۰ را نادیده می‌گیرد. این معیار به صورت زیر تعریف می‌شود:

$$\cos(x, y) = \frac{x \,.\, y}{||x|| \times ||y||}$$

در این فرمول، . (نقطه یا همان dot) به معنای ضرب داخلی است و ||x||، **اندازه‌ی بردار x** است. اندازه بردار به صورت زیر محاسبه می‌شود:

$$\sqrt{\sum_{i=1}^{n} {x_i}^2}$$

اجازه بدهید برای این معیار با همان مثال بالا، یک امتحان انجام دهیم. داده‌های زیر را مانند بالا در نظر بگیرد:

	Blues Traveler	Norah Jones	Phoenix	The Strokes	Weird Al
Clara	4.75	4.5	5	4.25	4
Robert	4	3	5	2	1

در این‌جا دو بردار به صورت زیر هستند:

x = (4.75,4.5,5,4.25,4)
y = (4,3,5,2,1)

پس:

$$||x|| = \sqrt{4.75^2 + 4.5^2 + 5^2 + 4.25^2 + 4^2} = 10.09$$
$$||y|| = \sqrt{4^2 + 3^2 + 5^2 + 2^2 + 1^2} = 7.416$$

ضرب داخلی به صورت زیر است:

$$x \cdot y = (4.75 \times 4) + (4.5 \times 3) + (5 \times 5) + (4.25 \times 2) + (4 \times 1) = 70$$

و در نهایت، شباهت کسینوسی به صورت زیر محاسبه می‌شود:

$$\cos(x, y) = \frac{70}{10.093 \times 7.416} = 0.935$$

شباهت کسینوسی مقداری است که در بازه‌ی ۱ به عنوان شباهتِ مثبت حداکثر تا ۱- به عنوان شباهت منفیِ حداکثری قرار دارد.

مدادتان را تیز کنید

شباهت کسینوسی را بین Angelica و Veronica محاسبه کنید (خط تیره‌ها را صفر در نظر بگیرید):

	Blues Traveler	Broken Bells	Deadmau 5	Norah Jones	Phoenix	Slightly Stoopid	The Strokes	Vampire Weekend
Angelica	3.5	2	-	4.5	5	1.5	2.5	2
Veronica	3	-	-	5	4	2.5	3	-

مدادتان را تیز کنید – راه‌حل

شباهت کسینوسی را بین Angelica و Veronica محاسبه کنید (خط تیره‌ها را صفر در نظر بگیرید):

	Blues Traveler	Broken Bells	Deadmau 5	Norah Jones	Phoenix	Slightly Stoopid	The Strokes	Vampire Weekend
Angelica	3.5	2	-	4.5	5	1.5	2.5	2
Veronica	3	-	-	5	4	2.5	3	-

x = (3.5,2,0,4.5,5,1.5,2.5,2)

y = (3,0,0,5,4,2.5,3,0)

$$||x|| = \sqrt{3.5^2 + 2^2 + 0^2 + 4.5^2 + 5^2 + 1.5^2 + 2.5^2 + 2^2} = 8.602$$
$$||y|| = \sqrt{3^2 + 0^2 + 0^2 + 5^2 + 4^2 + 2.5^2 + 3^2 + 0^2} = 8.078$$

ضرب داخلی هم به صورت زیر محاسبه می‌شود:

x.y = (3.5 × 3) + (2 × 0) + (0 × 0) + (4.5 × 5) + (5 × 4) + (1.5 × 2.5) + (2.5 × 3) + (2 × 0) = 64.25

شباهت کسینوسی هم به صورت زیر محاسبه می‌شود:

$$cos(x, y) = \frac{64.25}{8.602 \times 8.078} = 0.9246$$

از کدام معیار شباهت استفاده کنیم؟

پاسخ به این سوال را در سراسر این کتاب بررسی خواهیم کرد ولی برای الان، چند نکته‌ی کوچک می‌آوریم:

پس اگر داده‌ها غلیظ باشند (یعنی تُنُک نباشند، به این معنی که تقریباً تمامی ویژگی‌ها مقادیرِ غیر صفر داشته باشند) معیارهای منهتن و اقلیدسی گزینه‌های معقولی برای استفاده به نظر می‌رسند. ولی اگر داده‌ها غلیظ نباشند (یعنی تنک باشند) چه؟ یک سیستم امتیازدهیِ گسترده‌ی موسیقی را تصور کنید. سه نفر در این سایت هستند و هر کدام ۱۰۰ قطعه از موسیقی‌های ما را امتیازدهی کرده‌اند:

آقای Jake: عاشق آهنگ‌های کانتری

Linda و Eric: عاشق آهنگ‌های راک دهه‌ی ۶۰ میلادی

Linda و Eric سلیقه‌ی موسیقیاییِ شبیه به هم دارند. در واقع، در میان امتیازات آن‌ها، ۲۰ قطعه‌ی موسیقیِ مشترک مشاهده می‌شود و تفاوتِ امتیازیِ آن‌ها در میان این ۲۰ قطعه (در بازه‌ی ۱ تا ۵) به طور میانگین فقط ۰٫۵ امتیاز است!! فاصله‌ی منهتن بین این دو نفر برابر ۱۰ = ۰٫۵ × ۲۰ می‌شود. فاصله‌ی اقلیدسی برای این دو به صورت زیر محاسبه می‌گردد:

$$\boldsymbol{d = \sqrt{(0.5)^2 \times 20} = 2.236}$$

این در حالی است که Linda و Eric فقط یک قطعه‌ی موسیقی به صورت مشترک داشته‌اند: قطعه‌ی «چه روز زیبایی» اثر Chris Cagle. در این‌جا Lind فکر کرد که این موسیقیِ جالبی است و امتیاز ۳ را به آن داد، Jake فکر کرد که این قطعه‌ی موسیقی خارق‌العاده است و امتیاز ۵ را به آن داد. پس فاصله‌ی منهتن بین Jake و Linda برابر ۲ و فاصله‌ی اقلیدسی نیز به صورت زیر محاسبه می‌شود:

$$d = \sqrt{(3-5)^2} = 2$$

پس هر دو معیار منهتن و اقلیدسی نشان دادند که Jake به Linda نزدیک‌تر است تا Eric. نتیجه می‌گیریم که در این مورد هر دوی این معیارها (منهتن و اقلیدسی) نتایج ضعیفی را تولید کردند.

> اهان، ایده‌ای دارم که احتملاً بتونه مشکل را حل کنه.
> الان، افرادِ مختلف، قطعات موسیقی را در بازه‌ی ۱ تا ۵ امتیازدهی کرده‌اند. حالا من آن قطعاتی را که افراد به آن‌ها امتیازی نداده‌اند را ۰ در نظر می‌گیرم. به این صورت، مشکل داده‌های تُنُک را حل می‌کنیم چون با این کار، هر کدام از ویژگی‌ها در تمام موجودیت‌ها دارای مقدار هستند.

فکر خوبیه، ولی کاربردی نیست. برای این‌که ببینیم چرا کاربردی نیست، احتیاج داریم تا چند شخصیت به نمایش‌نامه‌ی خودمان (همان قطعات موسیقی و امتیاز کاربران به آن‌ها) اضافه کنیم: Cooper و Kelsey. حالا، Jake، Cooper و Kelsey سلیقه‌ی موسیقیاییِ بسیار شبیه به هم دارند. فرض کنید Jake ۲۵ قطعه‌ی موسیقی را در وب‌سایت ما امتیازدهی کرده‌است.

Cooper، ۲۶ قطعه‌ی موسیقی را امتیازدهی کرده‌است که ۲۵ تای آن‌ها همان قطعاتی است که Jake امتیازدهی کرده است. آن‌ها موسیقی‌های شبیه به هم را دوست دارند و میانگینِ فاصله‌ی امتیازات آن‌ها ۰,۲۵ است!!

Cooper

Kelsey هم عاشق وب‌سایت ماست و هم عاشق آهنگ‌ها، بنابراین ۱۵۰ قطعه‌ی موسیقی را امتیازدهی کرده‌است. ۲۵ تا از این ۱۵۰ قطعه‌ی موسیقی، همان‌هایی هستند که Cooper و Jake به آن‌ها نیز امتیاز داده‌اند. میانگین فاصله‌ی بین Kelsey و Jake نیز ۰,۲۵ است!!

حسِ درونیِ ما می‌گوید که Cooper و Kelsey به یک اندازه به Jake شباهت دارند.

حالا تصور کنید، تغییری که در فاصله‌ی منهتن و اقلیدسی دادیم را اعمال کنیم. این تغییر به این معنی بود که به هر قطعه‌ی موسیقی که یک شخص به آن امتیاز نداده‌است، امتیاز ۰ را اختصاص دهیم.

Kelsey

با این حساب، Cooper به نسبت Kelsey خیلی بیشتر به Jake نزدیک‌تر است.

ولی چرا؟

برای پاسخ به چراییِ این موضوع، اجازه دهید به این مثال ساده‌شده نگاهی بیندازیم (عدد ۰ به معنای این است که یک فرد به یک قطعه‌ی موسیقی امتیاز نداده‌است)

Song:	1	2	3	4	5	6	7	8	9	10
Jake	0	0	0	4.5	5	4.5	0	0	0	0
Cooper	0	0	4	5	5	5	0	0	0	0
Kelsey	5	4	4	5	5	5	5	5	4	4

با نگاهی به قطعاتِ موسیقی‌ای که همه‌ی این افراد به آن‌ها امتیاز داده‌اند (قطعات موسیقیِ شماره‌ی ۴ و ۵ و ۶)، به نظر می‌رسد که Cooper و Kelsey به صورت یکسان به Jake شباهت دارند. با این‌حال، فاصله‌ی منهتن با احتساب صفرهای قرار داده‌شده، چیز دیگری می‌گوید:

$$d_{Cooper,Jake} = (4-0)+(5-4.5)+(5-5)+5-4.5) = 4+0.5+0+0.5 = 5$$

$$d_{Kelsey,Jake} = (5-0)+(4-0)+(4-0)+(5-4.5)+(5-5)+(5-4.5)+(5-0)$$
$$+(5-0)+(4-0)+(4-0)$$
$$= 5+4+4+0.5+0+.5+5+5+4+4 = 32$$

مشکل این‌جاست که مقادیرِ صفر، تمایل دارند که بر هر معیاری از فاصله، سایه بیفکنند. پس این راه حلِ اضافه کردنِ صفر، بهتر از همان فرمولِ اصلی نیست. یک راه‌حل دیگری که برخی افراد از آن استفاده می‌کنند – به نوعی – محاسبه‌ی یک «میانگینِ» فاصله است که در آن برای محاسبه‌ی فاصله از قطعات موسیقی‌ای که افراد به صورت مشترک به آن‌ها امتیاز داده‌اند تقسیم بر تعدادِ قطعات موسیقی که به صورت مشترک امتیاز داده‌اند، استفاده می‌کنند.

دوباره تکرار می‌کنم، فاصله‌های منهتن و اقلیدسی در مواقعی که داده‌ها غلیظ (با تعداد صفر کم) باشد، بسیار عالی عمل می‌کنند، ولی اگر داده‌ها تُنُک باشند، بهتر است از معیار شباهت کسینوسی استفاده کنیم.

یک چیز عجیب

فرض کنید، می‌خواهیم پیشنهادهایی برای Amy آماده کنیم. Amy عاشق گروه‌های موسیقی Phoenix، Passion Pit و Vampire Weekend است. نزدیک‌ترین فرد به او، Bob است که او هم عاشق Phoenix، Passion Pit و Vampire Weekend است. پدر Bob نوازنده‌ی آکاردئون (نوعی ساز موسیقی) در گروه موسیقی Walpter Ostanek است که امسال در مسابقه‌ی گرامی (Grammy – یک مسابقه‌ی موسیقی) در گروه پولکا (نوعی موسیقی-رقص لهستانی) برنده شده‌اند. به دلیلِ تعهدات خانوادگی، Bob، به گروه موسیقی Walpter Ostanek امتیازِ ۵ می‌دهد. براساس سیستمِ پیشنهاددهنده‌ی فعلی، فکر می‌کنیم که Amy حتماً، گروهِ موسیقیِ Walpter Ostanek (که پدر Bob در آن نوازنده است) را دوست خواهد داشت. ولی عقل سلیم به ما می‌گویند که احتمالاً این‌طور نیست.

یا مثلا پروفسور Billy Bo Olivera که عاشق مطالعه‌ی کتاب‌های داده‌کاوی و علمی تخیلی هست را تصور کنید. نزدیک‌ترین فرد به او من هستم که مانند او کتاب‌های داده‌کاوی و علمی تخیلی را دوست دارم. در کنار این‌ها، من پودل‌ها (poodles - نوعی سگ) را هم دوست دارم و کتابِ «زندگیِ پنهانیِ پودل‌های استاندارد» را با امتیازِ بالا، امتیازدهی کرده‌ام. سیستمِ پیشنهاد دهنده‌ی فعلی، احتمالاً این کتاب را هم به پروفسور پیشنهاد خواهد داد.

مشکل از آن‌جایی ناشی می‌شود که ما فقط به **یک فرد** به عنوان «شبیه‌ترین فرد» اتکا کرده‌ایم. هر گونه تناقض یا تغییر جهتِ ناگهانی که آن فرد داشته باشد، به عنوان پیشنهاد در سیستم لحاظ می‌شود. یکی از راه‌هایی که می‌توان بر این تناقضات فائق آمد، این است که پیشنهادهایمان را نه بر اساس یک نفر، بلکه بر اساس **چندین نفر** از شبیه‌ترین افراد به شخص مورد نظر، پایه‌گذاری کنیم. برای این‌کار می‌توانیم از رهیافتِ **k نزدیک‌ترین همسایه (k-nearest neighot)** استفاده کنیم.

K نزدیک‌ترین همسایه (K-Nearest Neighbor)

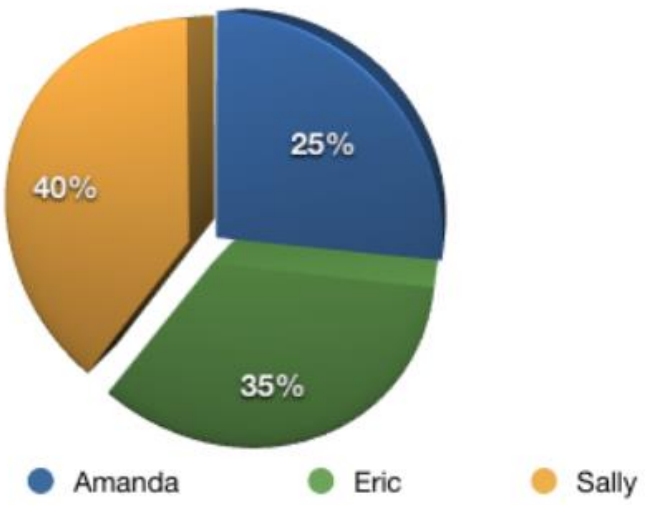
در روش k نزدیک‌ترین همسایه برای پایشِ مشارکتی (collaborative filtering) از تعدادِ k فرد، که بیشتر از همه به فردِ مورد نظرِ ما شبیه باشند، جهت شناساییِ پیشنهادهای مناسب استفاده می‌کنیم. بهترین مقدار برای k به مسئله بستگی دارد – بایستی آزمایش‌های مختلفی را برای به دست آوردنِ بهترین k انجام دهید. در زیر یک مثال می‌آوریم تا ایده‌ی اولیه‌ی این روش را درک کنید.

فرض کنید می‌خواهم یک سری پیشنهاد برای Ann آماده کنم. برای این‌کار از روش k نزدیک‌ترین همسایه با k=3 استفاده می‌کنم. در جدول زیر، ۳ همسایه‌ی نزدیک (ستون Person) و مقدار پیرسونِ آن‌ها (ستون Pearson) نسبت به Ann، آورده شده است:

Person	Pearson
Sally	0.8
Eric	0.7
Amanda	0.5

0.8 + 0.7 + 0.5 = 2.0

هر کدام از این ۳ شخص، می‌خواهند بر نتیجه‌ی پیشنهادها اثر بگذارند. سوال این است که چگونه تعیین کنم که هر کدام از این افراد، چه مقدار تاثیری بر ارائه‌ی پیشنهاد به Ann داشته باشند. اگر یک نمودارِ دایره‌ای تاثیرات (Pie of Influence™) وجود داشته باشد، هر کدام از قسمت‌ها باید چه اندازه‌ای باشند؟ در جدولِ بالا، مقادیر پیرسون (Pearson) را با هم جمع کردم که برابر ۲ شد. در همان جدول، سهم Sally برابر ۰٫۸ تقسیم بر ۲ (برابر ۴۰٪) می‌شود. سهم Eric برابر ۰٫۷ تقسیم بر ۲ (برابر ۳۵٪) و سهم Amanda نیز برابر ۲۵٪ می‌شود.



حالا فرض کنید Amanda، Eric و Sally گروه موسیقیِ Grey Wardens را به صورت زیر امتیاز داده باشند (ستون Person همان افراد هستند و ستون Grey Wardens Rating، امتیازی است که هر شخص به این گروه موسیقی داده است):

Person	Grey Wardens Rating
Amanda	4.5
Eric	5
Sally	3.5

پس با احتسابِ ضریب تاثیری که در بالا حساب شد چیزی مانند جدول زیر برای تاثیرِ هر کدام از افراد به دست می‌آید (در ستون Influence، می‌توانید تاثیر هر کدام را مشاهده کنید):

Person	Grey Wardens Rating	Influence
Amanda	4.5	25.00%
Eric	5	35.00%
Sally	3.5	40.00%

$$\text{Projected rating} = (4.5 \times 0.25) + (5 \times 0.35) + (3.5 \times 0.4)$$

$$= 4.275$$

مدادتان را تیز کنید

فرض کنید من، از داده‌هایی مانند داده‌های جدول بالا استفاده کنم، ولی این‌بار در روش k نزدیک‌ترین همسایه (knn)، k را برابر ۲ قرار دهم. تاثیر امتیازات برای گروه Grey Wardens چقدر می‌شود؟

Person	Pearson
Sally	0.8
Eric	0.7
Amanda	0.5

Person	Grey Wardens Rating
Amanda	4.5
Eric	5
Sally	3.5

مدادتان را تیز کنید

فرض کنید من، از داده‌هایی مانند داده‌های جدول بالا استفاده کنم، ولی این‌بار در روش k نزدیک‌ترین همسایه (knn)، k را برابر ۲ قرار دهم. تاثیر امتیازات برای گروه Grey Wardens چقدر می‌شود؟

Person	Pearson
Sally	0.8
Eric	0.7
Amanda	0.5

Person	Grey Wardens Rating
Amanda	4.5
Eric	5
Sally	3.5

تاثیر امتیازات = سهم Sally + سهم Eric

= (3.5 x (0.8 / 1.5)) + (5 x (0.7 / 1.5))

= (3.5 x .5333) + (5 x 0.4667)

= 1.867 + 2.333

= 4.2

یک کلاس (Class) توصیه‌گر با زبان پایتون

این چند بخشی را که پیش از این در فصلِ جاری پوشش دادیم، در این‌جا در یک کلاسِ پایتون با هم ترکیب کرده‌ام. اگرچه کمی طولانی شده‌است ولی تمامیِ کد را این‌جا می‌آورم (کدها را همچنین می‌توانید در سایت guidetodatamining.com و وب‌سایت chistio.ir نیز دانلود کنید).

```
import codecs
from math import sqrt

users = {"Angelica": {"Blues Traveler": 3.5, "Broken Bells": 2.0,
                      "Norah Jones": 4.5, "Phoenix": 5.0,
                      "Slightly Stoopid": 1.5,
                      "The Strokes": 2.5, "Vampire Weekend": 2.0},

         "Bill":{"Blues Traveler": 2.0, "Broken Bells": 3.5,
                 "Deadmau5": 4.0, "Phoenix": 2.0,
                 "Slightly Stoopid": 3.5, "Vampire Weekend": 3.0},

         "Chan": {"Blues Traveler": 5.0, "Broken Bells": 1.0,
                  "Deadmau5": 1.0, "Norah Jones": 3.0, "Phoenix": 5,
                  "Slightly Stoopid": 1.0},

         "Dan": {"Blues Traveler": 3.0, "Broken Bells": 4.0,
                 "Deadmau5": 4.5, "Phoenix": 3.0,
                 "Slightly Stoopid": 4.5, "The Strokes": 4.0,
                 "Vampire Weekend": 2.0},

         "Hailey": {"Broken Bells": 4.0, "Deadmau5": 1.0,
                    "Norah Jones": 4.0, "The Strokes": 4.0,
                    "Vampire Weekend": 1.0},

         "Jordyn":  {"Broken Bells": 4.5, "Deadmau5": 4.0,
                     "Norah Jones": 5.0, "Phoenix": 5.0,
                     "Slightly Stoopid": 4.5, "The Strokes": 4.0,
                     "Vampire Weekend": 4.0},

         "Sam": {"Blues Traveler": 5.0, "Broken Bells": 2.0,
                 "Norah Jones": 3.0, "Phoenix": 5.0,
                 "Slightly Stoopid": 4.0, "The Strokes": 5.0},

         "Veronica": {"Blues Traveler": 3.0, "Norah Jones": 5.0,
                      "Phoenix": 4.0, "Slightly Stoopid": 2.5,
                      "The Strokes": 3.0}
        }


class recommender:

    def __init__(self, data, k=1, metric='pearson', n=5):
        """ initialize recommender
        currently, if data is dictionary the recommender is initialized
        to it.
        For all other data types of data, no initialization occurs
        k is the k value for k nearest neighbor
        metric is which distance formula to use
        n is the maximum number of recommendations to make"""
        self.k = k
        self.n = n
        self.username2id = {}
        self.userid2name = {}
        self.productid2name = {}
        # for some reason I want to save the name of the metric
        self.metric = metric
```

```
        if self.metric == 'pearson':
            self.fn = self.pearson
        #
        # if data is dictionary set recommender data to it
        #
        if type(data).__name__ == 'dict':
            self.data = data

    def convertProductID2name(self, id):
        """Given product id number return product name"""
        if id in self.productid2name:
            return self.productid2name[id]
        else:
            return id


    def userRatings(self, id, n):
        """Return n top ratings for user with id"""
        print ("Ratings for " + self.userid2name[id])
        ratings = self.data[id]
        print(len(ratings))
        ratings = list(ratings.items())
        ratings = [(self.convertProductID2name(k), v)
                   for (k, v) in ratings]
        # finally sort and return
        ratings.sort(key=lambda artistTuple: artistTuple[1],
                     reverse = True)
        ratings = ratings[:n]
        for rating in ratings:
            print("%s\t%i" % (rating[0], rating[1]))



    def loadBookDB(self, path=''):
        """loads the BX book dataset. Path is where the BX files are
        located"""
        self.data = {}
        i = 0
        #
        # First load book ratings into self.data
        #
        f = codecs.open(path + "BX-Book-Ratings.csv", 'r', 'utf8')
        for line in f:
            i += 1
            #separate line into fields
            fields = line.split(';')
            user = fields[0].strip('"')
            book = fields[1].strip('"')
            rating = int(fields[2].strip().strip('"'))
            if user in self.data:
                currentRatings = self.data[user]
            else:
                currentRatings = {}
            currentRatings[book] = rating
            self.data[user] = currentRatings
        f.close()
        #
        # Now load books into self.productid2name
        # Books contains isbn, title, and author among other fields
```

```
        #
        f = codecs.open(path + "BX-Books.csv", 'r', 'utf8')
        for line in f:
            i += 1
            #separate line into fields
            fields = line.split(';')
            isbn = fields[0].strip('"')
            title = fields[1].strip('"')
            author = fields[2].strip().strip('"')
            title = title + ' by ' + author
            self.productid2name[isbn] = title
        f.close()
        #
        #  Now load user info into both self.userid2name and
        #  self.username2id
        #
        f = codecs.open(path + "BX-Users.csv", 'r', 'utf8')
        for line in f:
            i += 1
            #print(line)
            #separate line into fields
            fields = line.split(';')
            userid = fields[0].strip('"')
            location = fields[1].strip('"')
            if len(fields) > 3:
                age = fields[2].strip().strip('"')
            else:
                age = 'NULL'
            if age != 'NULL':
                value = location + '  (age: ' + age + ')'
            else:
                value = location
            self.userid2name[userid] = value
            self.username2id[location] = userid
        f.close()
        print(i)


    def pearson(self, rating1, rating2):
        sum_xy = 0
        sum x = 0
        sum y = 0
        sum x2 = 0
        sum_y2 = 0
        n = 0
        for key in rating1:
            if key in rating2:
                n += 1
                x = rating1[key]
                y = rating2[key]
                sum_xy += x * y
                sum_x += x
                sum y += y
                sum x2 += pow(x, 2)
                sum y2 += pow(y, 2)
        if n == 0:
            return 0
        # now compute denominator
        denominator = (sqrt(sum x2 - pow(sum x, 2) / n)
                       * sqrt(sum_y2 - pow(sum_y, 2) / n))
```

```
        if denominator == 0:
            return 0
        else:
            return (sum_xy - (sum_x * sum_y) / n) / denominator


    def computeNearestNeighbor(self, username):
        """creates a sorted list of users based on their distance to
        username"""
        distances = []
        for instance in self.data:
            if instance != username:
                distance = self.fn(self.data[username],
                                   self.data[instance])
                distances.append((instance, distance))
        # sort based on distance -- closest first
        distances.sort(key=lambda artistTuple: artistTuple[1],
                       reverse=True)
        return distances

    def recommend(self, user):
       """Give list of recommendations"""
       recommendations = {}
       # first get list of users  ordered by nearness
       nearest = self.computeNearestNeighbor(user)
       #
       # now get the ratings for the user
       #
       userRatings = self.data[user]
       #
       # determine the total distance
       totalDistance = 0.0
       for i in range(self.k):
          totalDistance += nearest[i][1]
       # now iterate through the k nearest neighbors
       # accumulating their ratings
       for i in range(self.k):
          # compute slice of pie
          weight = nearest[i][1] / totalDistance
          # get the name of the person
          name = nearest[i][0]
          # get the ratings for this person
          neighborRatings = self.data[name]
          # get the name of the person
          # now find bands neighbor rated that user didn't
          for artist in neighborRatings:
             if not artist in userRatings:
                if artist not in recommendations:
                   recommendations[artist] = (neighborRatings[artist]
                                              * weight)
                else:
                   recommendations[artist] = (recommendations[artist]
                                              + neighborRatings[artist]
                                              * weight)
       # now make list from dictionary
       recommendations = list(recommendations.items())
       recommendations = [(self.convertProductID2name(k), v)
                          for (k, v) in recommendations]
       # finally sort and return
       recommendations.sort(key=lambda artistTuple: artistTuple[1],
```

```
                                reverse = True)
        # Return the first n items
        return recommendations[:self.n]
```

یک مجموعه‌ی داده‌ی جدید

خب! حالا وقت آن است که نگاهی به یک مجموعه داده‌ی واقعی‌تر بیندازیم. Cai-Nicolas Zeigler نزدیک به یک میلیون مورد از امتیازدهی به کتاب‌ها را از سایت مبادله‌ی کتاب (Book Crossing) جمع‌آوری کرده است. این امتیازات از ۲۷۸۸۵۸ کاربر که مجموعاً ۲۷۱۳۷۹ کتاب را امتیازدهی کرده بودند، به دست آمده‌است. این داده‌های ناشناس شده (anonymized) در وب‌سایت http://www2.informatik.uni-freiburg.de/~cziegler/BX/ به صورت خروجیِ SQL و همچنین فایل متنیِ CSV در دسترس است. در هنگام بارگزاریِ این داده‌ها در زبانِ پایتون، مشکلاتی به دلیل اینکودینگ (encoding) کاراکترها داشتم. نسخه‌ی اصلاح‌شده‌ی CSV این داده‌ها در وب‌سایت خودم guidetodatamining.com و همچنین chistio.ir موجود است.

فایل CSV شامل سه جدول می‌شود:

جدول BX-Users، که مانند اسمش، حاوی اطلاعاتِ کاربران است. در این جدول، یک ستون user-id برای شناسه‌ی کاربران، یک ستون برای مکانِ جغرافیایی کاربران (مثلا شهر Albuquerque یا NM) و یک ستون مربوط به سن کاربران موجود است. نام کاربران به دلیل ناشناس‌سازی (anonymizition) حذف شده است.

جدول BX-Books که حاوی اطلاعات مربوط به کتاب‌ها است. کتاب‌ها با شناسه‌ی ISBN، عنوان کتاب، نویسنده، سال انتشار و انتشارات آن، در این جدول شناخته می‌شوند.

جدول BX-Book-Ratings که شامل user-id یا همان شناسه‌ی هر کاربر، ISBN کتاب و یک امتیاز بین ۰ تا ۱۰ است که مشخص می‌کند یک کاربر چه امتیازی به یک کتاب داده است.

تابع loadBookDB در کلاس recommender، داده‌ها را از این فایل‌ها بارگزاری می‌کند.

حالا می‌خواهم این مجموعه داده‌ی مربوط به کتاب‌ها را بارگزاری کنم. آرگومانی که به loadBookDB داده می‌شود، مسیر فایل‌های BX book است که در این‌جا /Users/raz/Downloads/BX-Dump است.

```
>>> r.loadBookDB('/Users/raz/Downloads/BX-Dump/')
1700018
```

نکته

این مجموعه‌ی داده، یک مجموعه‌ی داده‌ی حجیم است و ممکن است بارگزاریِ آن کمی زمان ببرد. در هکینتاشِ من (۲٫۵ گیگاهرتز، Core i7 860 با ۸گیگابایت RAM، ۲۴ ثانیه بارگزاریِ مجموعه‌ی داده و ۳۰ ثانیه اجرای پرس‌وجو (query) زمان برد.

حالا، می‌توانم پیشنهاداتی را برای کاربرِ شماره‌ی ۱۷۱۱۱۸ که فردی از تورنتو است، به دست بیاورم:

```
>>> r.recommend('171118')
[("The Godmother's Web by Elizabeth Ann Scarborough", 10.0), ("The Irrational
Season (The Crosswicks Journal, Book 3) by Madeleine L'Engle", 10.0), ("The
Godmother's Apprentice by Elizabeth Ann Scarborough", 10.0), ("A Swiftly
Tilting Planet by Madeleine L'Engle", 10.0), ('The Girl Who Loved Tom Gordon by
Stephen King', 9.0), ('The Godmother by Elizabeth Ann Scarborough', 8.0)]
```

```
>>> r.userRatings('171118', 5)
Ratings for toronto, ontario, canada
2421
The Careful Writer by Theodore M. Bernstein   10
Wonderful Life: The Burgess Shale and the Nature of History by Stephen Jay
Gould  10
Pride and Prejudice (World's Classics) by Jane Austen      10
The Wandering Fire (The Fionavar Tapestry, Book 2) by Guy Gavriel Kay   10
Flowering trees and shrubs: The botanical paintings of Esther Heins by Judith
Leet   10
```

پروژه‌ها

طبیعتاً نمی‌توانید مباحثِ این فصل را کاملاً درک کنید، مگراینکه با کدها سر و کله بزنید. در ادامه چند پیشنهاد آورده‌ام که احتمالاً بتوانید با انجام آن‌ها به درکِ بهتری از روش‌های ارائه شده در این فصل برسید.

۱. فاصله‌ی منهتن و فاصله‌ی اقلیدسی را پیاده‌سازی کنید و نتایجِ این روش‌ها را با یکدیگر مقایسه کنید.

۲. وب‌سایت این کتاب فایلی حاویِ امتیازِ کاربران به ۲۵ فیلم دارد. تابعی بنویسید که این داده‌ها را به الگوریتمِ طبقه‌بندی بارگزاری کند. تابع recommend که در بالا توضیح داده شد، باید بتواند فیلم‌هایی را به شخصِ خاصی پیشنهاد دهد.

فصل ۳. پایش بر اساس اقلام

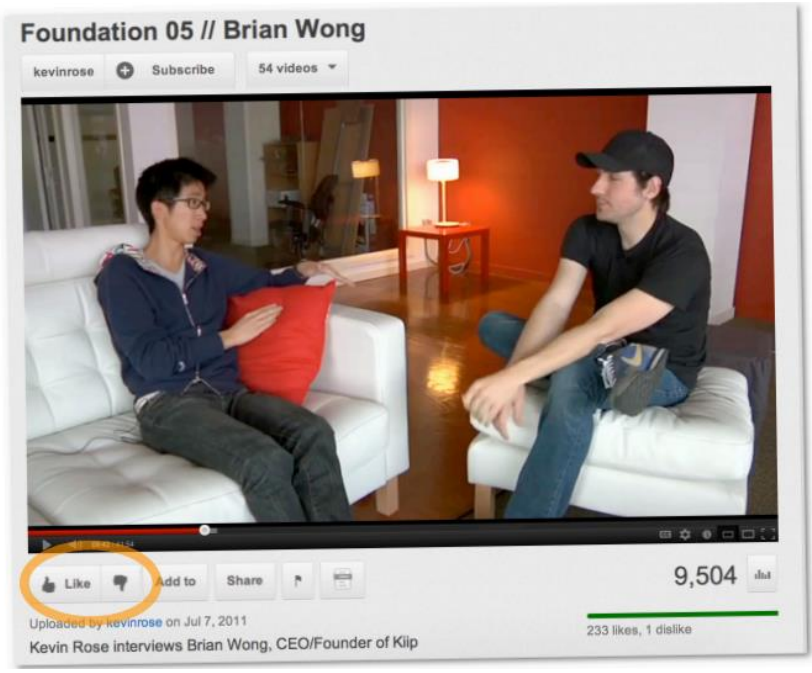
امتیازدهیِ ضمنی (Implicit rating) و پایش براساس اقلام (Item based filtering)

در فصل ۲، مقدمات پایش مشارکتی (collaborative filtering) و سیستم‌های توصیه‌گر (recommendation systems) را فرا گرفتیم. الگوریتم‌های تشریح شده در آن فصل، الگوریتم‌های همه‌کاره بودند و می‌توانستند با داده‌های مختلفی مورد استفاده قرار بگیرند. کاربران، اقلامِ مختلف را در بازه‌ی ۰ تا ۵ یا ۰ تا ۱۰ امتیازدهی می‌کردند و الگوریتم‌ها، کاربران دیگری که به صورتِ مشابهی آیتم‌ها را امتیازدهی کرده بودند، شناسایی می‌کردند. همان‌طور که قبلاً گفته شد، در برخی از موارد، کاربران یک همچین نتایج خوبی را مشاهده نمی‌کنند و به جای آن ممکن است اقلامی را ببینند که بیشترین امتیاز یا کمترین امتیاز را کسب کرده‌اند. این استراتژیِ همه یا هیچ، ممکن است در بعضی مواقع به نتایجِ غیرپایدار و با کیفیتی پایین منجر شود. در این فصل، روش‌هایی را امتحان می‌کنیم که به وسیله‌ی آن‌ها بتوانیم پایشِ مشارکتی را بهبود بخشیده و پیشنهاداتِ دقیق‌تری را با روش‌های کارآمد تولید کنیم.

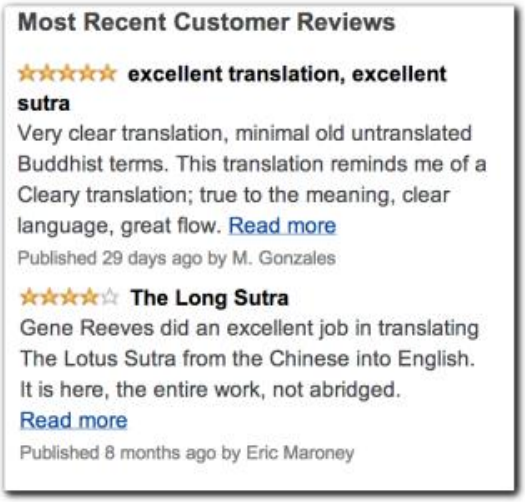
امتیازدهی صریح (Explicit Rating)

یکی از راه‌های تمایز علاقه‌مندی‌های کاربران، این‌است که ببینیم آیا آن‌ها صریح هستند یا ضمنی. منظور از صریح بودنِ امتیازها، این است که کاربر به صورتِ صریح به یکی از اقلام، امتیاز داده باشد. یکی از مثال‌های امتیازِ صریح، موقعی است که کاربران، دکمه‌های تایید

(مثلاً با نشانه‌ی انگشت شصت به سمت بالا) یا عدم تایید (مثلاً با نشانه‌ی انگشت شصت به سمت پایین) را در وب‌سایت‌هایی مانند پاندورا (Pandora) یا یوتیوب کلیک می‌کنند.



و یا سیستمِ ستاره‌دهی در سایت آمازون:



امتیازدهی ضمنی (Implicit Rating)

برای انجامِ عملیاتِ امتیازدهیِ ضمنی، از کاربر تقاضا نمی‌کنیم تا به صورت مستقیم امتیازدهی را انجام دهد – فقط رفتار او را مشاهده می‌کنیم. مثالی از این حالت (یعنی امتیازدهی ضمنی) ردیابیِ کلیک‌های یک کاربر (جایی که کاربر کلیک می‌کند) در وب‌سایت New York Times است.

بعد از این‌که مشاهده کردیم هر کاربر در طول چند هفته، کجاها را کلیک کرد، می‌توانیم یک پروفایلِ قابلِ قبول از کاربر توسعه دهیم – مثلاً این‌که، این کاربر ورزش را دوست ندارد ولی به نظر می‌رسد به اخبارِ حوزه‌ی تکنولوژی علاقه‌مند باشد. اگر کاربر بر روی مقاله‌ای با عنوانِ «سریع‌ترین راهِ کاهش وزن شناخته شده توسط مربیانِ حرفه‌ای» کلیک کرده باشد و همچنین بر روی مقاله‌ی «آرام و مطمئن: چگونه وزن کم کنیم و آن را همان‌طور


Egypt Judges Vow to Oppose Expansion of Morsi's Power

By DAVID D. KIRKPATRICK
3:18 P.M. ET

CAIRO — Egyptian judges and prosecutors struck back on Saturday against an attempt by President Mohamed Morsi to place his decrees above judicial review, vowing to challenge his edict in court and reportedly striking in Alexandria.

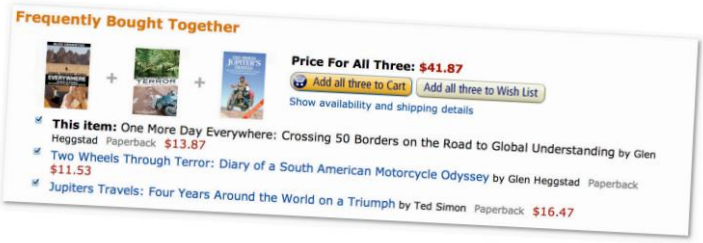
Clashes Break Out After Morsi Seizes New Power in Egypt

نگه‌داریم» نیز کلیک کرده باشد، احتمالاً به دنبال کم کردن وزن است. اگر این کاربر بر روی یک تبلیغ از آی‌فون کلیک کرد، احتمالاً به آن محصول علاقه دارد (به این کار – یعنی کلیک کردن یک کاربر بر روی یک تبلیغ - در اصلاح «click through» گفته می‌شود).

تصور کنید که چه اطلاعاتی را می‌توان از کلیک‌کردنِ کاربران بر روی محصولات، در سایت آمازون به دست آورد. در صفحه‌ی اصلیِ وب‌سایت آمازون که برای شما شخصی‌سازی شده است، این اطلاعات به نمایش درمی‌آید:

در این مثال، آمازون کلیک‌های کاربران را نگه‌داری و پیگیری می‌کند. آمازون می‌داند که برای مثال، فردی که کتاب « Jupiter's Travels: Four years around the world on a Triumph» را دیده‌است، دی‌وی‌دیِ «Long Way Round» را هم مشاهده کرده است. دی‌وی‌دیِ «Long Way Round» تاریخچه‌ی بازیگری به نام Ewan McGregor است که با همسرش به وسیله‌ی موتورسیکلت، به سراسر جهان سفر می‌کرد. همان‌طور که در شکلِ بالا از سایت آمازون مشخص است، این اطلاعات در قسمتِ **«مشتریانی که این محصول را مشاهده‌کرده‌اند همچنین محصولات زیر را هم دیده‌اند (Customers who viewed this also viewed)»** مورد استفاده قرار می‌گیرد.

مورد دیگری از **امتیازدهیِ ضمنی** چیزهایی است که یک کاربر خرید می‌کند. آمازون، حواسش به این اطلاعات هم هست و از آن‌ها برای پیشنهادهایی که به کاربر می‌دهد استفاده می‌کند: مانند «اقلامی که معمولاً با هم خریده شده‌اند (Frequently Bought Together)» و یا «مشتریان که این قلم را دیده‌اند، همچنین اقلام زیر را خریده‌اند (Customers Who Viewed This Item Also Bought)»:



ممکن است فکر کنید که قسمتِ **«اقلامی که معمولاً با هم خریده شده‌اند (Frequently Bought Together)»**، به پیشنهادهای غیرمتعارفی منجر خواهد شد ولی این‌طور نیست و این روش به صورتی خارق‌العاده‌ای خوب کار می‌کند.

تصور کنید که یک برنامه‌ی کامپیوتری، با مشاهده و نظارت بر رفتار شما در سایت iTunes، چه اطلاعاتی را می‌تواند کسب کند:

Name	Time	Artist	Plays
Anchor	3:24	Zee Avi	52
My Companjera	3:22	Gogol Bordello	27
Wake Up Everybody	4:25	John Legend & the...	17
Milestone Moon	3:40	Zee Avi	17
...			

اول، من آهنگی را به سایت iTunes اضافه کردم. این حداقل به این معناست که من این آهنگ را دوست داشتم. همچنین اطلاعاتی راجع به تعداد دفعات پخش (ستونِ Plays) موجود است. در شکلِ بالا، به آهنگِ Anchor از Zee Avi، ۵۲ مرتبه گوش کرده‌ام. این بدین معناست که من این آهنگ را دوست داشتم (و در واقع هم همین‌طور بوده است). اگر برای مدتی یک قطعه‌ی موسیقی را در کتابخانه‌ی دیجیتالم داشتم و فقط یک‌بار به آن گوش می‌کردم، احتمالاً به این معنا بود که من این آهنگ را دوست ندارم.

ورزش ذهنی

فکر نمی‌کنید که امتیاز دادنِ صریحِ یک کاربر به یک قلم جنس، دقیق‌تر باشد؟ یا برعکس، چیزی که یک کاربر می‌خرد و یا کاری که انجام می‌دهد (برای مثال، تعداد دفعات پخش)، می‌تواند بیشتر نشان‌دهنده‌ی علاقه‌ی یک کاربر باشد؟ کدام یک دقیق‌تر است؟

مثالی از امتیازدهی صریح:

بیوگرافی افراد در سایت match.com:

من یک گیاه‌خوار هستم. از نوشابه‌ی خاصی خوشم می‌آید، به قدم زدن در جنگل هم علاقه دارم. همچنین خواندن کتاب‌های چخوف در کنار آتش، فیلم‌های فرانسوی، آخرهفته‌ها در موزه‌ی هنر و موسیقی‌های «شومان (نوازنده‌ی پیانو)» را هم دوست دارم.

مثالی از امتیازدهی ضمنی (غیر صریح):

چیزهایی که در جیب Jim پیدا می‌کنیم!

در جیب او، قبض‌های مختلفی را مشاهده می‌کنیم:

۱۲ بسته آبجو، فست فود واتابرگر، بستنی‌های Ben and Jerry، پیتزا و دونات‌ها، قبض کرایه‌ی DVD مربوط به فیلم انتقام‌جویان، اقامت‌گاه شیطان: مجازات و اونگ باک ۳

مشکلات امتیازدهیِ صریح

مشکل شماره‌ی ۱: مردم تنبل هستند و اقلام را امتیازدهی نمی‌کنند

اول اینکه، کاربران به خود زحمت نمی‌دهند تا اقلام را امتیازدهی کنند. فرض می‌کنم که اکثر شما، اقلامِ متعددی را از سایت آمازون خریداری کرده‌اید. مانند من. در ماهِ گذشته، من یک میکروهلیکوپتر، یک هارد ۱ ترابایتی، یک مبدل USB به SATA، یک بسته ویتامین، دو کتابِ کیندل (Murder City و Ready Player One) و کتاب‌های کاغذی به نام‌های «هیچ جایی برای مخفی شدن نیست»، «۸ هفته برای رسیدن به سلامتی» از دکتر ویل، «ضد سرطان: یک سبک جدید زندگی» و کتاب «باز انجام (Rework)» را خریده‌ام. همه با هم ۱۲ قلم کالا می‌شود. ولی چندتا از آن‌ها را امتیازدهی کرده‌ام؟ هیچ کدام را! تصورِ من این است که شما هم همین‌طور هستید. شما به چیزهایی که می‌خرید امتیاز نمی‌دهید.

من درد زانو دارم. دوست دارم کوهنوردی کنم و به همین دلیل چند عدد عصای دستی مخصوص راهپیمایی دارم. بعضی از آن‌ها که ارزان هم هستند را از سایت آمازون خریده‌ام که مشکلاتِ زیادی هم داشتند. سال پیش برای سه روز جهتِ شرکت در جشنواره‌ی موسیقی به شهر «آستن (Austin)» رفتم. مشکل درد زانوهایم به خاطر پروازهای متعدد تشدید شد و سرانجام به فروشگاهِ REI رفتم تا یک عصای دستیِ تقریبا گران‌قیمت با برند REI خریداری کنم. در عرض کمتر از یک روز پیاده‌روی در سطح چمن در پارکِ شهر، این عصای دستی شکست. من عصاهای دستی ارزان قیمتِ ۱۰ دلاری داشتم که با استفاده‌ی ممتد در کوهستان در میان سخره‌ها نمی‌شکست، ولی این مدلِ گران‌قیمت، بر روی چمن‌های مسطح شکست. در زمانِ جشنواره، در حالی که عصبانی بودم، با خودم گفتم که بعداً حتماً عصاهای دستیِ برندِ REI را امتیازدهی می‌کنم و یک خلاصه و نظر راجع به آن‌ها می‌نویسم. آیا من این کار را انجام دادم؟ خیر! من، خیلی خیلی تنبل هستم. حتی در این شرایطِ بسیار بد، آن‌ها را امتیازدهی نکردم. فکر می‌کنم که آدم‌های تنبل مثل من زیاد هستند. مردم به طور کل، خیلی تنبل یا بی‌انگیزه هستند که بخواهند محصولات را امتیازدهی کنند.

مشکل شماره‌ی ۲: مردم ممکن است دروغ بگویند و یا این‌که اطلاعات ناقص بدهند

فرض کنیم بلاخره فردی بر مشکلِ تنبلی غلبه کرده باشد و یک محصول را امتیازدهی کند. این شخص ممکن است دروغ بگوید. مردم ممکن است به صورتِ مستقیم دروغ بگویند – با بی‌دقتی امتیاز دهند یا سهواً دروغ بگویند – که باعث می‌شود اطلاعات به صورت ناقص ایجاد شود. مثلاً Ben با Ann به سینما می‌روند تا یک فیلم تایلندی به نام « Uncle Boonmee» را ببینند. آن‌ها، با دوستِ Ben که Dan نام دارد و همچنین دوستِ Dan که Clara نام دارد به سینما می‌روند. Ben بعد از دیدنِ فیلم، فکر می‌کند که این بدترین فیلمی است که تا به حال دیده است. ولی بقیه‌ی همراهانِ Ben عاشق این فیلم شده بودند و بعد از فیلم در رستوران در مورد آن صحبت می‌کردند و می‌خندیدند. طبیعتاً خیلی غیرعادی نیست که Ben، رایِ خود را در مورد این فیلم در سایت‌های رتبه‌بندیِ فیلم، بالاتر از تصور خودش بزند.

مشکل شماره‌ی ۳: مردم امتیازهایی که داده‌اند را به‌هنگام (آپدیت) نمی‌کنند

فرض کنید من بعد از نوشتنِ این فصل از کتاب، انگیزه پیدا کردم تا محصولاتی که از آمازون خریدم را در این سایت امتیازدهی کنم. آن هارد دیسکِ ۱ ترابایتی خوب کار می‌کند – سرعتِ خوبی دارد و خیلی بی‌صداست. من به این هارد دیسک پنج ستاره می‌دهم. آن میکروهلیکوپتر نیز عالی است. پرواز آن راحت است و بسیار سرگرم‌کننده هست و همچنین از چندین تصادف و برخورد، جان سالم به در برده. یک پنج ستاره هم به این میکروهلیکوپتر می‌دهم. یک ماه می‌گذرد. هارد دیسک خراب می‌شود و در نتیجه من تمام فیلم‌ها و موسیقی‌هایی که دانلود کرده بودم را از دست می‌دهم – و این خیلی آزاردهنده است. آن میکروهلیکوپتر ناگهان کار نمی‌کند – انگار موتورش خراب شده. الان که فکر می‌کنم، هر دوی این محصولات، بی‌خود هستند. ولی به احتمالِ زیاد، به سایت آمازون نخواهم رفت و این یعنی امتیازی که قبلاً به محصولات داده‌ام را به‌هنگام (آپدیت) نمی‌کنم (دوباره تنبلی!). مردم هنوز فکر می‌کنند که نظرِ من، همان ۵ ستاره به هر کدام از این‌هاست!

Mary را در نظر بگیرید که یک دانشجو است. به دلایلی، او دوست دارد تا در سایت آمازون به محصولات امتیاز دهد. ده سال پیش، او آلبومِ موسیقیِ مورد علاقه‌اش را با ۵ ستاره امتیاز داده است. آلبوم «Giggling and Laughing: Silly Songs for Kids» و آلبوم «Sesame Songs: Sing Yourself Silly»! اخیراً نیز او چند آلبوم موسیقی را ۵ ستاره، امتیاز داده‌است. آلبوم‌های «Wolfgang Amadeus Phoenix» و « :The Twilight Saha

Eclipse Sourdtrack». بر اساس این امتیازاتِ اخیر، Mary به یک دانشجوی دیگر به نام Jen به عنوانِ نزدیک‌ترین همسایه، انتخاب شده است. همان‌طور که حدس می‌زنید، کار عجیبیست اگر آلبومِ «Giggling and Laughing: Silly Songs for Kids» را به Jen پیشنهاد دهیم. این مشکل نیز، یک نوعِ دیگر از مسئله‌ی به‌هنگام نبودن امتیازات و سلایق است.

ورزش ذهنی:
به نظر شما، مشکلاتِ امتیازدهی به صورت ضمنی (غیر صریح) چیست؟
(راهنمایی: در مورد خریدهایتان از سایتی مانند آمازون فکر کنید)

در چند پاراگرافِ بالاتر، لیستی از خریدهای خودم در یک ماهِ گذشته از سایت آمازون را گذاشتم. فرض کنید من دو تا از این اجناس را برای شخص دیگری خریده باشم. مثلاً کتابِ «ضد سرطان» را برای پسر عمویم و کتابِ «باز انجام (Rework)» را برای پسرم خریدم. برای این‌که نشان دهم چرا این موضوع می‌تواند منجر به مشکل شود، اجازه بدهید مثالی متقاعدکننده‌تر برایتان بزنم. برای این‌کار باید به کمی عقب‌تر برگردیم و سابقه‌ی خریدهای من را نگاه کنیم. من چند وزنه‌ی ورزشی (kettlebells) را همراهِ کتاب «شروع کار با وزنه: رازهای سوپرمن‌های شوروی» به عنوان هدیه برای پسرم خریدم. همراه این دو قلم جنس، یک سگِ پشمالوی جدید برای همسرم خریدم، چون سگِ قبلی ما در ۱۴ سالگی فوت کرد. اگر فرض کنیم سابقه‌ی خرید می‌تواند معیاری برای این باشد که مردم چه چیزهایی دوست دارند، احتمالاً در این مثال باید فرض کنیم، افرادی که وزنه‌ی ورزشی دوست دارند، همچنین به حیواناتِ اسباب‌بازی، میکروهلیکوپترها، کتاب‌های ضدسرطان و نیز کتابِ «قهرمان شماره‌ی یک» نیز علاقه دارند. سابقه‌ی خریدهای آمازون، نمی‌تواند بین خریدهایی که برای خودم انجام می‌دهم و خریدهایی که به عنوان هدیه برای دیگران انجام داده‌ام، تمایزی قائل شود. آقای Stephen Baker، یک مثالِ مرتبط در این زمینه دارد:

تصور کنید در قدمِ اول، یک کامپیوتر بایستی بفهمد که یک بلوزِ سفیدِ خاص، به دردِ یک نوزادِ دختر می‌خورد. مهمترین کار این است که یک پروفایل برای شخصی که این بلوز را خریداری کرده است درست کنیم. فرض کنید این شخص، همسر من است. او به سایت Macy (سایتی جهت خرید لباس) می‌رود و چهار یا پنج قلم جنس می‌خرد. لباس‌های

مختلفی مانند شلوار و بلوز و کمربند. تمامیِ این اقلام، متناسب با یک شخصی مانند او هستند. سپس در هنگامِ اتمامِ خرید، یادش می‌آیند که باید برای تولدِ برادرزاده‌ی شانزده ساله‌اش، هدیه‌ای بخرد. آخرین باری که برادرزاده‌اش را دیده بودیم، یک لباس مشکلی با نوشته‌های زیادی بر روی آن پوشیده بود. بیشترِ این نوشته‌ها، نوشته‌های خشمگینی بود! خودِ او به ما گفته بود که خود را یک گُت (goth – یک نژاد آلمانی) می‌داند. برای همین همسر من، به یک قسمتِ دیگر رفت و یک دست‌بند با میخ‌های تیز برداشت!!!

Baker 2008.60-61.

اگر بخواهیم پروفایل یک کاربر – برای مثال همین همسر آقای Stephen Baker – را بسازیم، این دست‌بند با میخ‌های تیز، مشکل‌ساز خواهد بود.

برای مثالِ آخر، فرض کنید یک زن و شوهر، از یک اکانتِ Netflix استفاده کنند. مرد از فیلم‌های اکشن، که دارای انفجار و هلیکوپتر و این‌ها است خوشش می‌آید. اما خانم، فیلم‌های فکری و رمانتیک-کمدی را دوست دارد. اگر ما فقط به تاریخچه‌ی مشاهده‌ی فیلم‌ها نگاه کنیم، با پروفایلی مواجه خواهیم شد که انگار یک شخص، به دو دسته‌ی کاملاً متفاوت از فیلم‌ها، علاقه دارد.

یادتان بیاید که قبلا گفتم کتابِ «ضد سرطان: یک سبک جدید برای زندگی» را به عنوان هدیه برای پسرعمویم خریده بودم. اگر سابقه‌ی خریدِ من را کمی بیشتر واکاوی کنیم، می‌بینیم که من قبلاً این کتاب را باز هم خریده بودم. در واقع، چندین کپی از از سه کتاب را خریدم. حال فردی می‌تواند این‌گونه فکر کند که خریدهای متوالیِ من، به خاطر گم شدنِ کتاب و یا به خاطر این‌که فراموش کرده‌ام که این کتاب‌ها را قبلاً خریداری کرده‌ام، نبوده است. منطقی‌ترین استدلال، این است که این شخص با خود بگوید که من این کتاب‌ها را اینقدر دوست داشتم که به عنوان هدیه برای چندین نفر خریده و فرستاده‌ام. پس متوجه می‌شویم که می‌توانیم اطلاعاتِ بسیار زیادی را از سابقه‌ی خرید افراد به دست آوریم.

ورزش ذهنی
چه استفاده‌ای (به عنوان داده‌های ضمنی) می‌توان از مشاهده‌ی رفتارِ یک فرد با یک کامپیوتر کرد؟

داده‌های ضمنی:
صفحات وب:
کلیک کردن بر روی یک لینک از یک صفحه
زمانی سپری شده در هنگام خواندنِ یک صفحه
مشاهده‌ی چندین باره‌ی یک صفحه
ارجاع داده‌ی یک صفحه به صفحات دیگر
اینکه یک شخص چه صفحاتی را در سایت Hulu (وب‌سایت پخش ویدیو) دیده‌است

پخش‌کننده‌ی موسیقی:
این‌که یک شخص کدام موسیقی‌ها را پخش کرده است
از کدام موسیقی‌ها بدون گوش دادن عبور کرده است
تعداد دفعاتی که یک موسیقیِ تکراری را گوش داده است

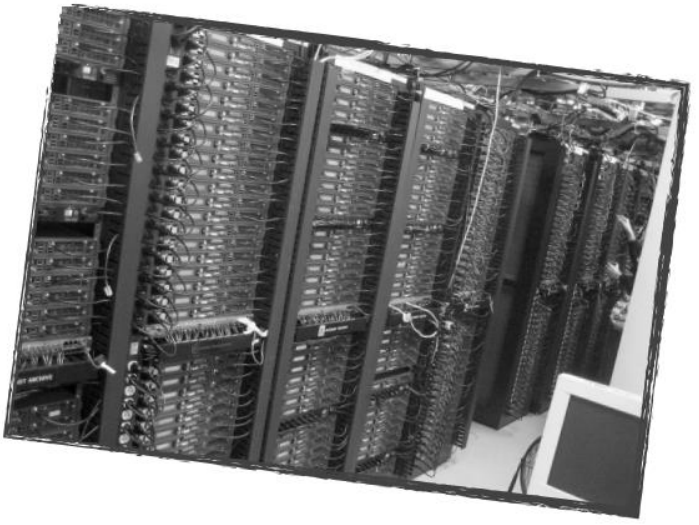
البته این‌ها فقط مثال‌های سطحی از داده‌های ضمنی بودند!

توجه داشته باشید که الگوریتم‌های معرفی شده در فصلِ دوم، بدون توجه به صریح یا ضمنی بودنِ داده‌ها، می‌توانند مورد استفاده قرار گیرند.

مشکل موفقیت

فرض کنید، شما یک سرویس آنلاین موسیقی دارید که دارای یک سیستم پیشنهاددهنده‌ی داخلی است. مشکل کجا می‌تواند اتفاق بیوفتد؟
فرض کنید این سیستم یک میلیون کاربر دارد. هر بار که بخواهید یک پیشنهاد برای یک شخص آماده کنید، بایستی فاصله‌ی این شخص را با ۹۹۹۹۹۹ نفرِ دیگر مقایسه کنید. حالا

فرض کنید که بخواهیم چندین پیشنهاد را در هر ثانیه به افرادِ مختلف بدهیم. به این صورت محاسبات بسیار پیچیده و سخت خواهد شد. به جز در حالتی که خرجِ زیادی برای خریدِ سرورها (servers) انجام دهید، سیستم کُند خواهد شد. به بیانِ رسمی‌تر، تاخیر (latency)، می‌تواند یک مشکلِ اساسی در سیستم‌های پیشنهاددهنده مبتنی بر همسایگی (neighbor-based) باشد. البته که خوشبخانه یک راه حل وجود دارد.



یک مزرعه‌ی سرور (server farm)

پایشِ مبتنی بر کاربر (User-based Filtering)

تا به این‌جای کار، ما پایشِ مبتی بر کاربران به صورت مشارکتی را مد نظر قرار می‌دادیم. برای این‌کار بایستی یک کاربر را با تمامیِ کاربرانِ دیگر مقایسه کنیم تا بتوانیم نزدیک‌ترین افراد را شناسایی کنیم. دو مشکل اصلی در این روش وجود دارد:

۱. **مقیاس‌پذیری (scalability).** همان‌طور که گفتیم، با زیاد شدنِ کاربران، محاسبات نیز زیاد می‌شود. روش‌های مبتنی بر کاربر (user-based)، ممکن است با هزاران کاربر خوب کار کنند، ولی با کاربران میلیونی، مقیاس‌پذیریِ آن‌ها به مشکل برمی‌خورد.

۲. تنک بودن (sparsity). بسیاری از سیستم‌های پیشنهاددهنده، تعداد زیادی کاربر و محصول دارند. ولی هر کاربر به صورت میانگین، درصدِ کمی از محصولات را امتیازدهی می‌کند. برای مثال، سایت آمازون، میلیون‌ها کتاب دارد ولی به طور میانگین، کاربران، تعداد انگشت‌شماری از این کتاب‌ها را امتیازدهی کرده‌اند. به همین دلیل، الگوریتم‌هایی که در فصل دوم گفته شد، ممکن است نتوانند هیچ همسایه‌ی نزدیکی را برای یک شخص پیدا کنند.

به خاطرِ وجود این دو مشکل، بهتر است از **پایش مبتنی بر اقلام (item-based filtering)** استفاده کنیم.

پایش مبتنی بر اقلام (Item-Based Filtering)

فرض کنید من الگوریتمی در اختیار دارم که می‌تواند محصولاتی که بیشتر شبیه به هم هستند را شناسایی کند. برای مثال، یک همچین الگوریتمی می‌تواند به این موضوع پی‌ببرد که آلبوم Wolfgang Amadeus Phoenix به آلبوم Manners شباهت دارد. بر همین اساس، اگر کاربری، آلبومِ Wolfgang Amadeus Phoenix را با امتیازِ بالایی امتیازدهی کرد، می‌توانیم آلبوم Manners را به او پیشنهاد دهیم. توجه کنید که این روش، با روشِ مبتنی بر کاربر (user-based) متفاوت است. در روشِ مبتنی بر کاربر، ما یک کاربر را در نظر داشتیم و به دنبال پیدا کردنِ شبیه‌ترین شخص به او در میان کاربرانِ دیگر بودیم تا بتوانیم توسطِ امتیازاتِ دیگران، پیشنهادهایی را برای این کاربر پیدا کنیم. ولی در پایشِ مبتنی بر اقلام، قبل از زمانِ پیشنهاد، شبیه‌ترین اقلام را به یکدیگر پیدا می‌کنیم و آن‌ها را با امتیازاتِ کاربران ترکیب کرده تا بتوانیم پیشنهادها یا توصیه‌هایی را شکل دهیم.

می‌توانی مثالی از این موضوع به من بگویی؟

فرض کنید سایتِ موسیقیِ ما، دارای m کاربر و n گروه موسیقی باشد که در این سایت، کاربران، گروه‌های موسیقی را امتیازدهی می‌کنند. در شکلِ زیر این امتیازدهی مشخص است. مانند قبل، سطرها بیان‌گرِ کاربران بوده و ستون‌ها نشان‌دهنده‌ی گروه‌های موسیقی هستند.

	Users	...	Phoenix	...	Passion Pit	...	n
1	Tamera Young		5				
2	Jasmine Abbey				4		
3	Arturo Alvarez		1		2		
...	...						
u	Cecilia De La Cueva		5		5		
...	...						
m-1	Jessica Nguyen		4		5		
m	Jordyn Zamora		4				

می‌خواهیم شباهتِ ببین گروه موسیقیِ Phoenix را با گروه موسیقیی Passion Pit مقایسه کنیم. برای انجام این کار از کاربرانی که به هر دو گروهِ موسیقی امتیاز داده‌اند استفاده می‌کنیم (مستطیل‌های آبی رنگ). همان‌طور که متوجه شدید، در روشِ پایش مبتنی بر اقلام (item-based filtering)، شباهتِ بینِ ستون‌ها – در این‌جا گروه‌های موسیقی – را با یکدیگر مشخص می‌کنیم.

پایش مبتنی بر کاربر (user-based filtering)، پایش مبتنی بر حافظه به صورت مشارکتی (memory based collaborative filtering) نیز نامیده می‌شود. چرا؟ به خاطر این‌که ما به ذخیره‌سازیِ تمامی امتیازات احتیاج داریم تا بتوانیم یک پیشنهاد ارائه دهیم

پایش مبتنی بر اقلام (item-based filtering)، پایش مبتنی بر مدل به صورت مشارکتی (model-based collaborative filtering) نیز نامیده می‌شود. چرا؟ به خاطر این‌که احتیاجی به ذخیره‌سازیِ تمامیِ امتیازات نیست. ما یک مدل می‌سازیم که نشان‌دهنده‌ی نزدیکی و شباهتِ یک قلم جنس به تمامیِ اجناس دیگر است.

شباهت کسینوسیِ تعدیل شده (Adjusted Cosine Similarity)

برای محاسبه‌ی شباهتِ بین اقلام، ما از معیارِ شباهتِ کسینوسی که در فصل دوم معرفی شد، استفاده می‌کنیم. همان‌طور که خاطرتان هست، در مورد تورم درجه (grade inflation) نیز صحبت کرده‌ایم. تورم درجه به این معناست که کاربری امتیاز بالاتری را نسبت به چیزی که از او انتظار داریم، به محصولی می‌دهد. برای جبران این تورم، ما میانگین امتیازاتِ کاربر را از هر کدام از امتیازات کم می‌کنیم. این کار، به ما مقدار شباهتِ کسینوسیِ تعدیل‌شده را می‌دهد.

$$s(i,j) = \frac{\sum_{u \in U}\left(R_{u,i} - \overline{R_u}\right)\left(R_{u,j} - \overline{R_u}\right)}{\sqrt{\sum_{u \in U}\left(R_{u,i} - \overline{R_u}\right)^2}\ \sqrt{\sum_{u \in U}\left(R_{u,j} - \overline{R_u}\right)^2}}$$

در فرمولِ بالا، U، مجموعه‌ی تمامیِ کاربرانی است که هر دو آیتمِ **i** و **j** را امتیازدهی کرده‌اند.

فرمول بالا از مقاله‌ی اصلیِ پایش مشارکتی استخراج شده است. این مقاله با عنوان «الگوریتم‌های توصیه‌گر جهت پایش مبتنی بر اقلام به صورت مشارکتی» نوشته‌ی Badrul Sarwar، George Karypis، Joseph Konstan و Jon Reidl در (http://www.grouplens.org/papers/pdf/www10_sarwar.pdf) انتشار داده شده است.

در فرمولِ بالا

$$\left(R_{u,j} - \overline{R_u}\right)$$

این‌گونه تفسیر می‌شود: امتیازِ R توسط کاربرِ u به آیتمِ i، منهای میانگین تمامیِ امتیازاتی که کاربر u (همین کاربر) به تمامیِ اقلام داده است. که در واقع امتیازِ نرمال‌شده را به ما می‌دهد. در فرمولِ بالا، برای هر s(i, j)، شباهتِ اقلامِ i و j را با هم به دست می‌آوریم. در فرمولِ اصلی، صورتِ کسر می‌گوید: برای هر کاربر که هر دو محصولِ i و j را امتیازدهی کرده‌است، این امتیاز را در، امتیازِ نرمال‌شده‌ی آن دو قلم جنس، ضرب می‌کنیم، و نتایج را با یکدیگر جمع خواهیم کرد. در مخرجِ کسر، ما جمع مربعاتِ تمامیِ امتیازاتِ نرمال‌شده را برای قلمِ i محاسبه کرده و سپس جذرِ این نتایج را می‌گیریم. همین‌کار را هم برای قلمِ j انجام می‌دهیم و نتایج را در یکدیگر ضرب می‌کنیم تا مخرج کسر به دست آید. (اقلام همان آیتم‌ها هستند)

برای این‌که بتوانیم شباهتِ کسینوسیِ تعدیل‌شده را تصور کنیم، از داده‌های زیر کمک می‌گیریم. در این داده‌ها، پنج دانشجو (سطرها)، پنج گروه موسیقی (ستون‌ها) را امتیازدهی کرده‌اند.

Users	میانگین امتیازات	Kacey Musgraves	Imagine Dragons	Daft Punk	Lorde	Fall Out Boy
David			3	5	4	1
Matt			3	4	4	1
Ben		4	3		3	1
Chris		4	4	4	3	1
Torri		5	4	5		3

اولین کاری که باید برای محاسبه‌ی معیارِ شباهتِ کسینوسیِ تعدیل‌شده انجام دهیم این است که میانگینِ امتیازاتِ هر کاربر را محاسبه کنیم.

Users	میانگین امتیازات	Kacey Musgraves	Imagine Dragons	Daft Punk	Lorde	Fall Out Boy
David	3.25		3	5	4	1
Matt	3.0		3	4	4	1
Ben	2.75	4	3		3	1
Chris	3.2	4	4	4	3	1
Tori	4.25	5	4	5		3

حالا برای هر جفت گروهِ موسیقی (هنرمندان)، شباهت بینِ آن‌ها را محاسبه می‌کنیم. اجازه دهید با Kacey Musgraves و Imaging Dragons شروع کنیم. در جدول بالا، دورِ کاربرانی که به هر دوی این گروه‌های موسیقی امتیاز داده‌اند خطِ آبی کشیده‌ام. پس مقدارِ شباهت کسینوسیِ تعدیل‌شده برای این دو گروهِ موسیقی به صورت زیر محاسبه می‌شود:

$$s(\boldsymbol{Musgraves, Dragons}) = \frac{\sum_{u \in U}(R_{u,Musgraves} - \overline{R_u})(R_{u,Dragons} - \overline{R_u})}{\sqrt{\sum_{u \in U}(R_{u,Musgraves} - \overline{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,Dragons} - \overline{R_u})^2}}$$

$$= \frac{(4-2.75)(3-2.75)+(4-3.2)(4-3.2)+(5-4.25)(4-4.25)}{\sqrt{(4-2.75)^2+(4-3.2)^2+(5-4.25)^2}\sqrt{(3-2.75)^2+(4-3.2)^2+(4-4.25)^2}}$$

$$= \frac{0.7650}{\sqrt{2.765}\sqrt{0.765}} = \frac{0.7650}{(1.6628)(0.8746)} = \frac{0.7650}{1.4543} = 0.5260$$

پس مقدار شباهت بین Kacey Musgraves و Imagine Dragons برابر ۰٫۵۲۶۰ است. من شباهتِ گروه‌های موسیقیِ دیگر را نیز با یکدیگر محاسبه کرده‌ام و در جدول زیر نمایش داده‌ام:

	Fall Out Boy	Lorde	Daft Punk	Imagine Dragons
Kacey Musgraves	-0.9549		1.0000	0.5260
Imagine Dragons	-0.3378		0.0075	
Daft Punk	-0.9570			
Lorde	-0.6934			
Fall Out Boy				

مدادتان را تیز کنید
بقیه‌ی مقادیر را در جدول بالا محاسبه کنید

مدادتان را تیز کنید – راه حل
بقیه‌ی مقادیر را در جدول بالا محاسبه کنید

	Fall Out Boy	Lorde	Daft Punk	Imagine Dragons
Kacey Musgraves	-0.9549	0.3210	1.0000	0.5260
Imagine Dragons	-0.3378	-0.2525	0.0075	
Daft Punk	-0.9570	0.7841		
Lorde	-0.6934			

برای محاسبه‌ی این مقادیر، یک اسکریپت کوچک پایتون نوشتم:

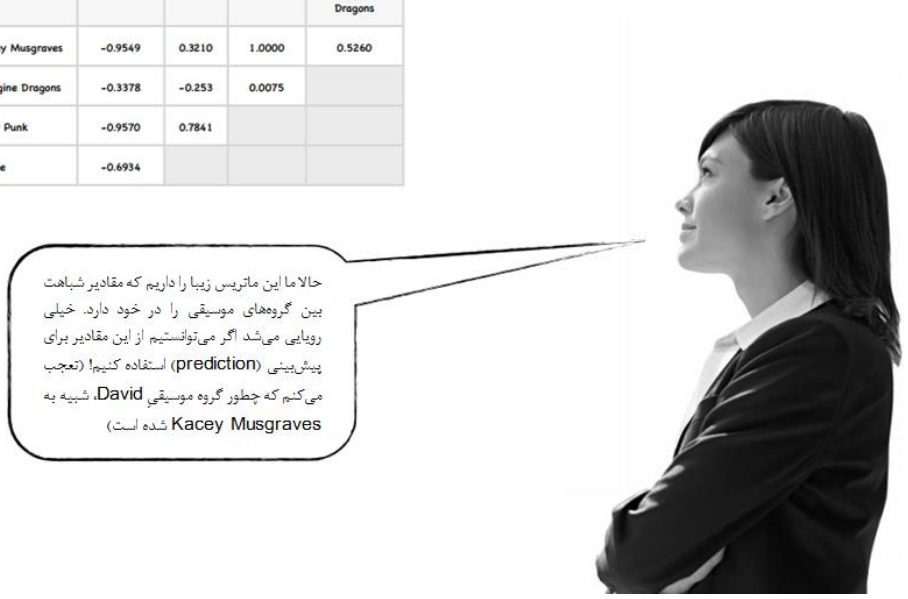
```
def computeSimilarity(band1, band2, userRatings):
    averages = {}
    for (key, ratings) in userRatings.items():
        averages[key] = (float(sum(ratings.values()))
            / len(ratings.values()))
    num = 0 # numerator
    dem1 = 0 # first half of denominator
    dem2 = 0
    for (user, ratings) in userRatings.items():
        if band1 in ratings and band2 in ratings:
            avg = averages[user]
            num += (ratings[band1] - avg) * (ratings[band
2] - avg)
            dem1 += (ratings[band1] - avg)**2
            dem2 += (ratings[band2] - avg)**2
    return num / (sqrt(dem1) * sqrt(dem2))
```

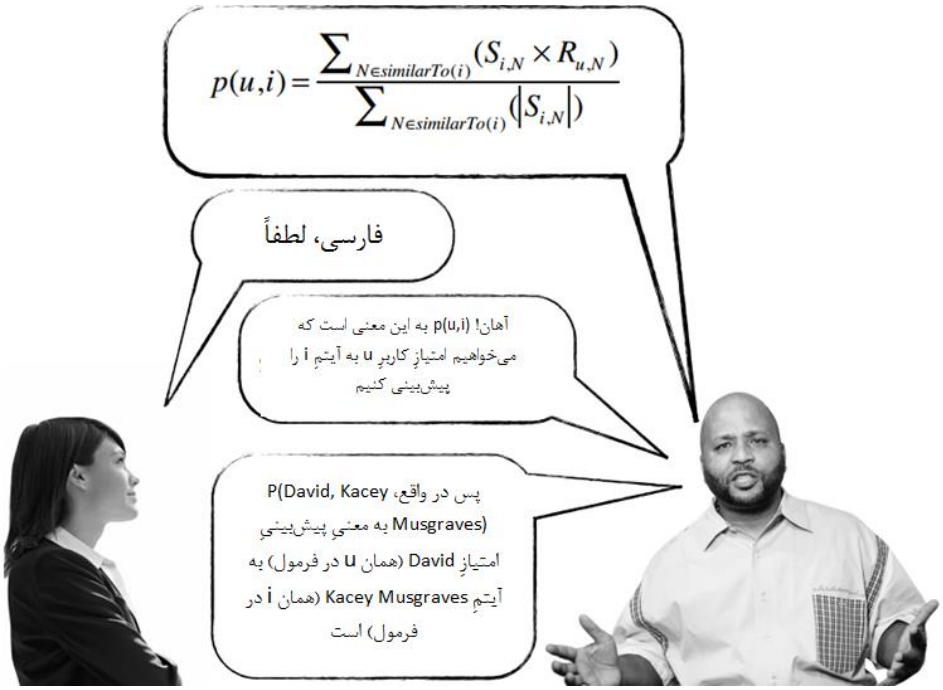
در زیر فرمت **userRatings** آورده شده است:

```
users3 = {"David": {"Imagine Dragons": 3, "Daft Punk":
 5,
"Lorde": 4, "Fall Out Boy": 1},
"Matt": {"Imagine Dragons": 3, "Daft Punk": 4,
"Lorde": 4, "Fall Out Boy": 1},
"Ben": {"Kacey Musgraves": 4, "Imagine Dragons": 3,
"Lorde": 3, "Fall Out Boy": 1},
"Chris": {"Kacey Musgraves": 4, "Imagine Dragons": 4,
"Daft Punk": 4, "Lorde": 3, "Fall Out Boy": 1},
```

```
"Tori": {"Kacey Musgraves": 5, "Imagine Dragons": 4,
"Daft Punk": 5, "Fall Out Boy": 3}}
```

	Fall Out Boy	Lorde	Daft Punk	Imagine Dragons
Kacey Musgraves	-0.9549	0.3210	1.0000	0.5260
Imagine Dragons	-0.3378	-0.253	0.0075	
Daft Punk	-0.9570	0.7841		
Lorde	-0.6934			




$$p(u,i)=\frac{\sum_{N\in similarTo(i)}(S_{i,N}\times R_{u,N})}{\sum_{N\in similarTo(i)}(|S_{i,N}|)}$$
فارسی، لطفاً
آهان! p(u,i) به این معنی است که می‌خواهیم امتیازِ کاربرِ u به آیتمِ i را پیش‌بینی کنیم
پس در واقع، P(David, Kacey Musgraves) به معنیِ پیش‌بینیِ امتیازِ David (همان u در فرمول) به آیتمِ Kacey Musgraves (همان i در فرمول) است

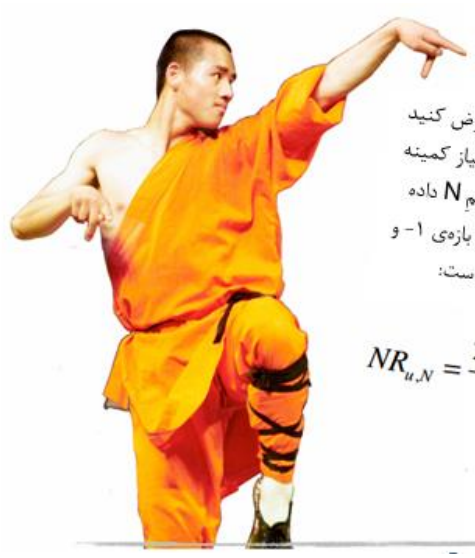

در این‌جا N، هر کدام از اقلامی است که شخصِ u آن را امتیازدهی کرده و شبیه به قلمِ i است. منظورمان از کلمه‌ی «شبیه» این‌جا، این است که یک امتیازِ شباهت بین N و i در ماتریس ما وجود دارد

$R_{u,N}$ امتیازی است که کاربر u به آیتمِ N داده است

$S_{i,N}$ در واقع شباهت بین i و N است (که از ماتریس شباهت به دست می‌آید)

$$p(u,i) = \frac{\sum_{N \in similarTo(i)} (S_{i,N} \times R_{u,N})}{\sum_{N \in similarTo(i)} (|S_{i,N}|)}$$

$R_{u,N}$ به معنی این است که تلاش می‌کنیم تا پیش‌بینی کنیم که کاربر u چقدر آیتمِ i را دوست داشته است (در واقع کاربر u چه امتیازی به آیتمِ i می‌دهد)



امتیازاتی که به موسیقی‌ها می‌دادیم، بین ۱ تا ۵ بود. فرض کنید که Max_R امتیاز بیشینه (در مثال ما ۵) و Min_R امتیاز کمینه (۱) باشد. $R_{u,N}$ امتیاز جاری است که کاربر u به آیتمِ N داده است. $NR_{u,N}$ امتیاز نرمال‌شده است (امتیاز جدید در بازه‌ی ۱- و ۱). فرمول نرمال‌سازی امتیازات به صورت زیر است:

$$NR_{u,N} = \frac{2(R_{u,N} - Min_R) - (Max_R - Min_R)}{(Max_R - Min_R)}$$

فرمول غیرنرمال‌سازی (denormalize) یعنی فرمولی که داده‌ها را از بازه‌ی نرمال شده به بازه‌ی اصلی (در مثالِ ما ۱ تا ۵) تبدیل می‌کند به صورت زیر است:

$$R_{u,N} = \frac{1}{2}((NR_{u,N} + 1) \times (Max_R - Min_R)) + Min_R$$

فرض کنید یک نفر، Fall Out Boy را امتیازِ ۲ داده است. پس داده‌ی نرمال‌شده به صورت زیر است:

$$NR_{u,N} = \frac{2(R_{u,N} - Min_R) - (Max_R - Min_R)}{(Max_R - Min_R)} = \frac{2(2-1)-(5-1)}{(5-1)} = \frac{-2}{4} = -0.5$$

و حالا اگر بخواهیم همین داده را غیرنرمال‌سازی (denormalize) کنیم، به صورت زیر عمل می‌کنیم:

$$R_{u,N} = \frac{1}{2}((NR_{u,N}+1)\times(Max_R - Min_R)) + Min_R$$

$$= \frac{1}{2}((-0.5+1)\times 4)+1 = \frac{1}{2}(2)+1 = 1+1 = 2$$

خب! الان عملیات نرمال‌سازی را در مُشتِ خودمان داریم.

اولین کار این است که امتیازات David را نرمال‌سازی کنیم:

امتیازات David به گروه‌های موسیقی

Artist	R	NR
Imagine Dragons	3	0
Daft Punk	5	1
Lorde	4	0.5
Fall Out Boy	1	-1

در فصل بعدی، بیشتر در مورد نرمال‌سازی (normalization) یاد خواهیم گرفت!

ماتریس شباهت

	Fall Out Boy	Lorde	Daft Punk	Imagine Dragons
Kacey Musgraves	-0.9549	0.3210	1.0000	0.5260
Imagine Dragons	-0.3378	-0.2525	0.0075	
Daft Punk	-0.9570	0.7841		
Lorde	-0.6934			

David به گروه‌های Imagine Dragons، Daft Punk، Lorde و Fall Out Boy امتیاز داده است، پس ما این گروه‌ها را در محاسباتمان اعمال می‌کنیم تا ببینیم که او، چقدر Kacaey Musgraves را دوست دارد

البته که از امتیازاتِ نرمال‌شده استفاده می‌کنیم

$$p(u,i)=\frac{\sum_{N\in similarTo(i)}(S_{i,N}\times NR_{u,N})}{\sum_{N\in similarTo(i)}(|S_{i,N}|)}=$$

Imagine Dragons Daft Punk Lorde Fall Out Boy

$$\frac{(.5260\times 0)+(1.00\times 1)+(.321\times 0.5)+(-.955\times -1)}{0.5260+1.000+0.321+0.955}$$

$$=\frac{0+1+0.1605+0.955}{2.802}=\frac{2.1105}{2.802}=0.753$$

پس در این‌جا پیش‌بینی کردیم که David، به گروه Kacey Musgraves امتیاز ۰,۷۵۳ (در بازه‌ی ۱- تا ۱) می‌دهد. حالا بایستی این عدد را غیرنرمال‌سازی (denormalize) کنیم:

$$R_{u,N}=\frac{1}{2}((NR_{u,N}+1)\times(Max_R-Min_R))+Min_R$$

$$=\frac{1}{2}((0.753+1)\times 4)+1=\frac{1}{2}(7.012)+1=3.506+1=4.506$$

پس با توجه به فرمول‌های بالا، پیش‌بینی کردیم که David به Kacey Musgraves امتیاز ۴,۵۰۶ می‌دهد.

شباهت کسینوسیِ تعدیل شده یک مدلِ پایشِ مشارکتیِ مبتنی بر مدل (Model-Based) است. همان‌طور که قبلاً گفته شد، یکی از مزیت‌های این روش نسبت به روش مبتنی بر حافظه (memory-based) این است که این روش، مقیاس‌پذیریِ بهتری دارد.

برای مجموعه‌داده‌های بزرگ‌تر، روش‌های مبتنی بر مدل، نیاز دارند تا سریع‌تر بوده و حافظه‌ی کمتری مصرف کنند.

معمولاً افراد، مقیاس‌های امتیازی را متفاوت استفاده می‌کنند. مثلاً من ممکن است گروه موسیقی‌ای را که دوست ندارم با امتیاز ۳ و آن گروه موسیقی‌ای را که دوستش دارم با امتیاز ۴، امتیازدهی کنم. اما شما ممکن است به گروه موسیقی‌ای که به آن علاقه ندارید، امتیاز ۱ داده و به گروهی که به آن علاقه دارید امتیاز ۵ بدهید. شباهت کسینوسیِ تعدیل شده، این مشکل را با تفریقِ میانگین امتیاز کاربر از هر کدام از امتیازها، حل می‌کند.

الگوریتم شیبِ یک (Slope One)

یکی دیگر از الگوریتم‌های موجود، برای انجام عملیاتِ پایش مبتنی بر اقلام به صورت مشارکتی (iterm-based collaborative filtering) الگوریتمِ شیبِ یک (Slope One) است. مزیتِ اصلیِ این روش، سادگی و بنابراین پیاده‌سازیِ آسانِ آن است. الگوریتم شیبِ یک، در مقاله‌ی «پیش‌بینی‌های شیبِ یک برای پایش مبتنی بر امتیاز به صورت مشارکتی و برخط» توسط Daniel Lemire و Anna Machlachlan در http://www.daniel-lemire.com/fr/abstracts/SDM2005.html نوشته شده است. این مقاله، یکی از مقالات عالی در این زمینه است و حتماً ارزش خواندن را دارد.

ایده‌ی اصلی به این صورت است: فرض کنید Amy امتیاز ۳ را به گروه موسیقیِ PSY داده‌است و امتیاز ۴ را به Whitney Houston. این در حالی‌است که Ben امتیاز ۴ را به PSY داده. حالا می‌خواهیم پیش‌بینی کنیم که Ben چه امتیازی را به Whitney Houston می‌دهد. جدولی که می‌توان برای نمایشِ این مسئله رسم کرد، چیزی مانندِ جدول زیر است (سطرها، افراد هستند و ستون‌ها گروه‌های موسیقی):

	PSY	Whitney Houston
Amy	3	4
Ben	4	?

برای این‌که بتوانیم حدس بزنیم که Ben احتمالاً چه امتیازی به Whitney Houston خواهد داد، به صورت زیر استدلال می‌کنیم:

Amy به Whitney Houston یک امتیاز بهتر از PSY داده‌است. حالا می‌توانیم پیش‌بینی کنیم که Ben نیز، یک امتیاز بیشتر از امتیازی که به PSY داده‌است، به Whitney Houston خواهد داد. پس چون Ben به PSY، امتیاز ۴ داده‌است، احتمالاً به Whitney Houston امتیاز ۵ (یکی بیشتر) را خواهد داد.

در واقع چندین نوع از **الگوریتمِ شیبِ یک** موجود است. در ادامه، الگوریتمِ **شیبِ یکِ وزن‌دار (weighted slop one)** را توضیح خواهم داد. یادتان باشد که مزیتِ اصلیِ این روش، سادگیِ آن است. چیزی که قرار است توضیح دهم، ممکن است پیچیده به نظر برسد، ولی به من اعتماد کنید، چیزهایی که می‌گوییم برایتان روشن و واضح خواهند شد. می‌توانید الگوریتمِ شیبِ یک را به دو مرحله تقسیم کنید. **اول**، قبل از این‌که زمان محاسبه فرا برسد (در حالتِ دسته‌ای، مثلاً در نصف شب یا هر زمان دیگر)، به سراغ محاسبه‌ی انحراف (deviation) بین هر جفت از اقلام می‌رویم. در مثال بالا، این مرحله به این معنی است که بفهمیم Whitney Houston یکی بیشتر از PSY امتیازدهی شده است (این کار توسط Amy انجام شده). حالا یک پایگاه‌داده از انحرافات در میان اقلام داریم. در **مرحله‌ی دوم**، عملیاتِ پیش‌بینی را انجام می‌دهیم. کاربری مانند Ben می‌آید. تا به حال، آهنگی از Whitney Houston نشنیده است و می‌خواهیم پیش‌بینی کنیم که Ben، چه امتیازی به این گروه می‌دهد. با استفاده از تمام گروه‌های موسیقی‌ای که Ben امتیازدهی کرده است، همراه با پایگاه‌داده‌ی انحراف‌هایی که در مرحله‌ی اول ساخته بودیم، پیش‌بینی را انجام می‌دهیم.

The Broad Brush Picture

قسمت اول (که قبل از محاسبات اصلی مثلاً در نیمه شب انجام می‌شد): محاسبه‌ی انحراف‌ها بین هر جفت از آیتم‌ها (اقلام)

قسمت دوم: محاسبه‌ی انحراف‌ها برای پیش‌بینی

قسمت اول: محاسبه‌ی انحراف (deviation)

اجازه دهید، مثالِ قبلی را کمی پیچیده‌تر کنیم. برای این کار دو کاربرِ جدید و یک گروهِ موسیقی (یا همان هنرمندان) جدید به جدول اضافه می‌کنیم:

	Taylor Swift	PSY	Whitney Houston
Amy	4	3	4
Ben	5	2	?
Clara	?	3.5	4
Daisy	5	?	3

قدم اول، محاسبه‌ی انحراف‌ها بود. میانگینِ انحراف، بین قلمِ i با توجه به قلم j برابر است با:

$$dev_{i,j} = \sum_{u \in S_{i,j}(X)} \frac{u_i - u_j}{card(S_{i,j}(X))}$$

که در آن card(S) تعداد اِلمان‌هایی است که در S موجود است و X برابرِ تعدادِ تمامیِ امتیازات است. پس $card(S_{i,j}(x))$ برابر تعداد افرادی است که هر دو قلم i و j را امتیازدهی کرده‌اند. برای مثال، اجازه دهید انحرافِ PSY را نسبت به Taylor Swift اندازه

بگیریم. در این مورد، $card(S_{i,j}(x))$ برابر ۲ است – دو شخص وجود دارند که هر دوی این گروه‌های موسیقی (Taylor Swift و PSY) را امتیازدهی کرده‌اند. عبارت $u_j - u_i$ در صورتِ کسر، برابر است با **امتیازِ کاربر برای Taylor Swift** منهای **امتیاز کاربر برای PSY**. پس انحراف به صورت کلی، این‌طور محاسبه می‌شود:

$$dev_{swift,psy} = \frac{(4-3)}{2} + \frac{(5-2)}{2} = \frac{1}{2} + \frac{3}{2} = 2$$

پس انحراف از PSY تا Taylor Swift برابر ۲ است، به این معنی که به طور میانگین، کاربران ۲ امتیاز بیشتر به Taylor Swift نسبت به PSY داده‌اند. حالا انحراف از Taylor Swift تا PSY چقدر است؟

$$dev_{psy,swift} = \frac{(3-4)}{2} + \frac{(2-5)}{2} = -\frac{1}{2} + -\frac{3}{2} = -2$$

مدادتان را تیز کنید

بقیه‌ی مقادیر را در جدول زیر محاسبه و تکمیل کنید:

	Taylor Swift	PSY	Whitney Houston
Taylor Swift	0	2	
PSY	-2	0	
Whitney Houston			0

مدادتان را تیز کنید – راه حل

بقیه‌ی مقادیر را در جدول زیر محاسبه و تکمیل کنید:

	Taylor Swift	PSY	Whitney Houston
Taylor Swift	0	2	
PSY	-2	0	
Whitney Houston			0

Taylor Swift با توجه به Whitney Houston:

$$dev_{swift,houston} = \frac{(4-4)}{2} + \frac{(5-3)}{2} = \frac{0}{2} + \frac{2}{2} = 1$$

PSY با توجه به Whitney Houston:

$$dev_{psy,houston} = \frac{(3-4)}{2} + \frac{(3.5-4)}{2} = \frac{-1}{2} + \frac{-.5}{2} = -.75$$

	Taylor Swift	PSY	Whitney Houston
Taylor Swift	0	2	1
PSY	-2	0	-0.75
Whitney Houston	-1	0.75	0

ورزش ذهنی

تصور کنید یک وب‌سایت پخش آنلاین موسیقی با یک میلیون کاربر و ۲۰۰ هزار امتیاز به هنرمندان داریم. اگر یک کاربرِ جدید به وب‌سایت اضافه شود و این کاربر ۱۰ هنرمند را امتیازدهی کند، آیا بایستی دوباره الگوریتم را برای بدست آوردن ۲۰۰ هزار ضربدر ۲۰۰ هزار انحراف، اجرا کنیم؟ یا راه حل ساده‌تری هم موجود است؟
(برای جواب با ما همراه باشید!)

ورزش ذهنی – راه حل
تصور کنید یک وب‌سایت پخش آنلاین موسیقی با یک میلیون کاربر و ۲۰۰ هزار امتیاز به هنرمندان داریم. اگر یک کاربرِ جدید به وب‌سایت اضافه شود و این کاربر ۱۰ هنرمند را امتیازدهی کند، آیا بایستی دوباره الگوریتم را برای بدست آوردن ۲۰۰ هزار ضربدر ۲۰۰ هزار انحراف، اجرا کنیم؟ یا راه‌حل ساده‌تری هم موجود است؟

احتیاجی نیست الگوریتم را دوباره، برای تمامیِ مجموعه‌ی داده اجرا کنید. زیباییِ این روش در این‌جاست که برای هر جفت از اقلام، ما فقط باید انحراف در بین این دو و همچنین تعداد افرادی که که هر دوی این اقلام را امتیازدهی کرده‌اند، نگه‌داری کنیم.
برای مثال، فرض کنید من انحراف بین **Taylor Swift** را با توجه به **PSY** را دارم که برابر ۲ می‌شود و این انحراف، بر اساس امتیازِ ۹ نفر از کاربران به دست آمده‌است. یک کاربر جدید می‌آید و به **Taylor Swift** امتیاز ۵ و به **PSY** امتیازِ ۱ می‌دهد. حالا، انحراف را به صورت زیر به‌روزرسانی می‌کنیم:

$$((9 \times 2) + 4) / 10 = 2.2$$

مرحله‌ی ۲: پیش‌بینی با استفاده از الگوریتمِ شیبِ یکِ وزن‌دار (Weighted Slope One)

خب، حالا ما یک مجموعه‌ی بزرگی از انحراف‌ها در اختیار داریم. اما چگونه می‌توانیم از این مجموعه برای ساخت پیش‌بینی‌ها استفاده کنیم؟ همان‌طور که گفتم، ما از الگوریتم شیبِ یکِ وزن‌دار (Weighted Slope One) یا همان P^{wS1} استفاده می‌کنیم. فرمول محاسبه‌ی این الگوریتم به صورت زیر است:

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u)-\{j\}} (dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}}$$

که در آن:

$$c_{j,i} = card(S_{j,i}(\chi))$$

در این فرمول، $P^{wS1}(u)_j$ به این معنی است که پیش‌بینیِ ما، از شیبِ یکِ وزن‌دار، در حالتی‌که کاربرِ u به آیتمِ j امتیاز داده است، استفاده می‌کند. پس برای مثال، $P^{wS1}(Ben)_{Whitney\ Houston}$ به معنیِ پیش‌بینیِ ما از امتیازی است که Ben به Whitney Houston خواهد داد.

فرض کنیم می‌خواهم به همین سوال پاسخ بدهم: Ben چه امتیازی به Whitney Houston خواهد داد؟

اجازه بدهید صورتِ کسر را در فرمول واکاوی کنیم:

$$\sum_{i \in S(u) - \{j\}}$$

این به ما می‌گوید: برای هر کدام از گروه‌های موسیقی (یا همان هنرمندان) که Ben به آن‌ها امتیاز داده‌است (به جز Whitney Houston که عبارت {j} در فرمولِ بالاست).

کلِ صورتِ کسر در فرمول، به این معناست که: برای هر کدام از گروه‌های موسیقیِ i که Ben به آن‌ها امتیاز داده‌است (به جز Whitney Houston) i به دنبالِ انحراف از Whitney Houston تا آن گروهِ موسیقی هستم و نتیجه‌ی آن را با امتیازِ Ben به گروه موسیقیِ i جمع می‌کنیم. سپس آن را در کاردینالیتیِ آن دو (cardinality – تعداد افرادی که به هر دو گروه موسیقی (Whiney Houston و i) امتیاز داده‌اند) ضرب می‌کنیم.

اجازه بدهید با مثالِ مرحله به مرحله پیش برویم:

اول، امتیازات Ben به همراه انحراف‌ها را که از قبل داشتیم:

	Taylor Swift	PSY	Whitney Houston
Ben	5	2	?

	Taylor Swift	PSY	Whitney Houston
Taylor Swift	0	2	1
PSY	-2	0	-0.75
Whitney Houston	-1	0.75	0

۱. Ben به Taylor Swift امتیاز ۵ داده‌است – این همان $\boldsymbol{u_i}$ است.
۲. انحراف از Whitney Houston با توجه به Taylor Swift برابر ۱- است – این همان $\boldsymbol{dev_{j,i}}$ است
۳. پس $\boldsymbol{dev_{j,i} + u_i}$ برابر ۴ می‌شود
۴. به چند صفحه قبل نگاه کنید، جایی که نشان دادیم دو کاربر (Amy و Daisy) وجود داشتند که هم Taylor Swiftو هم Whitney Houston را امتیازدهی کرده بودند. پس $\boldsymbol{c_{j,i}}$ برابر ۲ می‌شود.
۵. پس $\boldsymbol{(dev_{j,i} + u_i)\, c_{j,i} = 4 \times 2 = 8}$
۶. Ben به PSY امتیاز ۲ داده‌است
۷. انحراف از Whitney Houston نسبت به PSY برابر ۰,۷۵ است
۸. پس $\boldsymbol{dev_{j,i} + u_i}$ برابر ۲,۷۵ می‌شود
۹. دو کاربر هستند که هر دوی Whitney Houston و PSY را امتیازدهی کرده‌اند. پس:

$$\boldsymbol{(dev_{j,i} + u_i)c_{j,i} = 2.75 \times 2 = 5.5}$$

۱۰. جمعِ مرحله‌ی ۵ و ۹ را حساب می‌کنیم که می‌شود ۱۳,۵ (برای صورت کسر)

حالا برای مخرج کسر:
۱۱. با واکاویِ مخرجِ کسر، می‌فهمیم که معنیِ آن این است که: برای هر کدام از گروه‌های موسیقی که Ben به آن‌ها امتیاز داده‌است، جمعِ کاردینالیتی را برای آن گروه‌های موسیقی به دست می‌آوریم (یعنی چند نفر آن گروهِ موسیقی و Whitney Houston را با هم امتیازدهی کرده‌اند). پس Ben، به Taylor Swift امتیاز داده است و کاردینالیتیِ Taylow Swift و Whitney Houston برابر ۲ است (یعنی تعداد افرادی که به هر دوی Taylor Swift و Whitney Houston امتیاز داده‌اند). Ben به PSY امتیاز داده‌است و کاردینالیتیِ PSY هم برابر ۲ است. پس مخرجِ کسر برابر ۴ می‌شود.
۱۲. بنابراین پیش‌بینیِ این‌که Ben چه امتیازی به Whitney Houston می‌دهد به این صورت محاسبه می‌شود:

$$\frac{13.5}{4} = 3.375$$

حالا نوبت کد پایتون است

می‌خواهم کلاسِ نوشته شده در فصلِ ۲ را توسعه دهم. برای این‌که خیلی طولانی نشود، کدهای مربوط به کلاسِ recommender را این‌جا تکرار نمی‌کنم – فقط به آن ارجاع می‌دهم (و یادتان باشد که می‌توانید کدها را از سایت guidtodatamining.com و سایت chistio.ir دانلود کنید). حتماً به یاد می‌آورید که فرمتِ داده‌های آماده شده برای آن کلاس، به این صورت بود:

```
users2 = {"Amy": {"Taylor Swift": 4, "PSY": 3, "Whitney
Houston": 4},
"Ben": {"Taylor Swift": 5, "PSY": 2},
"Clara": {"PSY": 3.5, "Whitney Houston": 4},
"Daisy": {"Taylor Swift": 5, "Whitney Houston": 3}}
```

اول، محاسبه‌ی انحراف‌ها (deviations).

دوباره این‌جا فرمولِ محاسبه‌ی انحراف‌ها را می‌آورم:

$$dev_{i,j} = \sum_{u \in S_{i,j}(X)} \frac{u_i - u_j}{card(S_{i,j}(X))}$$

پس داده‌ی ورودی به تابعِ computeDeviations باید به فرمتِ users2 که در بالا نمایش‌داده شد، باشد. خروجی باید یک حالتی از داده‌های زیر شود:

	Taylor Swift	PSY	Whitney Houston
Taylor Swift	0	2 (2)	1 (2)
PSY	-2 (2)	0	-0.75 (2)
Whitney Houston	-1 (2)	0.75 (2)	0

اعدادِ داخل پرانتز، تعدادِ تکرار (فرکانس) هستند (یعنی، تعداد کاربرانی که هر دوی این گروه‌های موسیقی را امتیازدهی کرده‌اند). پس برای هر جفت گروهِ موسیقی، بایستی انحراف و تعداد تکرار (فرکانس) را ذخیره کنیم.
سودوکد برای تابعِ ما، چیزی شبیه به کدِ زیر است:

```
def computeDeviations(self):
   for each i in bands:
      for each j in bands:
         if i ≠ j:
            compute dev(j,i)
```

سودوکدِ بالا، زیباست ولی همان‌طور که می‌بینید، یک ناهماهنگی بین فرمتِ داده‌های ورودیِ این سودوکد و داده‌های واقعی، وجود دارد (users2 را در مثال بالا ببینید). به عنوانِ یک برنامه‌نویس، دو راه جلوی پایمان است، یا این‌که فرمت داده‌ها را تغییر دهیم، یا بایستی یک بازنگری در سودوکد داشته باشیم. من راه دوم را انتخاب می‌کنم. سودوکدِ بازنگری شده چیزی شبیه به سودوکدِ زیر می‌شود:

```
def computeDeviations(self):
   for each person in the data:
      get their ratings
      for each item & rating in that set of ratings:
         for each item2 & rating2 in that set of ratings:
            add the difference between the ratings to our computation
```

حالا بیایید با هم، تابع مورد نیاز را مرحله به مرحله بسازیم:

مرحله‌ی ۱:

```
def computeDeviations(self):
    # for each person in the data:
    # get their ratings
    for ratings in self.data.values():
```

دیکشنری‌های پایتون (که به جداول درهم‌ساز – hash tables نیز شهرت دارند) جفت‌های کلید/مقدار (key/value) هستند. در کدِ بالا، self.data یک دیکشنری در پایتون است. تابعِ values، مقادیر را از دیکشنری (self.data) استخراج می‌کند. داده‌ی ما چیزی شبیه به کد زیر است:

```
users2 = {"Amy": {"Taylor Swift": 4, "PSY": 3, "Whitney
Houston": 4},
"Ben": {"Taylor Swift": 5, "PSY": 2},
"Clara": {"PSY": 3.5, "Whitney Houston": 4},
"Daisy": {"Taylor Swift": 5, "Whitney Houston": 3}}
```

پس در اولین دورِ حلقه داریم:

```
ratings = {"Taylor Swift": 4, "PSY": 3, "Whitney Houston
": 4}
```

مرحله‌ی ۲:

```
def computeDeviations(self):
    # for each person in the data:
```

```
    # get their ratings
    for ratings in self.data.values():
        #for each item & rating in that set of ratings:
        for (item, rating) in ratings.items():
            self.frequencies.setdefault(item, {})
            self.deviations.setdefault(item, {})
```

در تابع init از کلاس recommender، من self.frequencies و self.deviations را مقداردهیِ اولیه می‌کنم تا به دیکشنری تبدیل شوند.

```
def __init__(self, data, k=1, metric='pearson', n=5):
    ...
    #
    # The following two variables are used for Slope One
    #
    self.frequencies = {}
    self.deviations = {}
```

تابعِ setdefault در دیکشنری‌های پایتون، ۲ آرگومان می‌گیرد. یک کلید و یک مقدار اولیه. این تابع به این صورت عمل می‌کند: اگر کلید در دیکشنری موجود نبود، این کلید را با مقدار اولیه‌ای که به عنوان آرگومان دوم به این تابع داده‌ایم، مقداردهیِ اولیه می‌کند. در غیر این صورت، مقدارِ فعلیِ کلید در آن دیکشنری را برمی‌گرداند.

مرحله‌ی ۳:

```
def computeDeviations(self):
    # for each person in the data:
    # get their ratings
    for ratings in self.data.values():
        # for each item & rating in that set of ratings:
        for (item, rating) in ratings.items():
            self.frequencies.setdefault(item, {})
            " " " " "
            self.deviations.setdefault(item, {})
            # for each item2 & rating2 in that set of ratin
gs:
            for (item2, rating2) in ratings.items():
                if item != item2:
```

```
                    # add the difference between the rating
s
                    # to our computation
                    self.frequencies[item].setdefault(item2
, 0)
                    self.deviations[item].setdefault(item2,
 0.0)
                    self.frequencies[item][item2] += 1
                    self.deviations[item][item2] += rating
- rating2
```

کدِ بالا، اختلافِ دو امتیاز را محاسبه می‌کند و آن را با self.deviations جمع می‌کند. دوباره، داده‌ها چیزی شبیه به کد زیر است:

```
{"Taylor Swift": 4, "PSY": 3, "Whitney Houston": 4}
```

وقتی که در حلقه‌ی خارجی هستیم که مقدار آیتمِ موردِ نظرِ ما برابر item = “Tayor Swift” و امتیاز مورد نظر ما برابر ۴ (rating=4) است، و در حلقه‌ی داخلی که آیتمِ دومِ ما (item2) برابر PSY (یعنی item2 = “PSY”) و امتیاز دوم (rating2) برابر با ۳ (= rating2 3) است، آخرین خط کدِ بالا، عددِ ۱ را به self.deviation[“Teylor Swift”][“PSY”] اضافه می‌کند.

مرحله‌ی ۴:

سرانجام، نیاز داریم تا بر روی self.deviations چند دور (iterate) بزنیم تا بتوانیم هر کدام از انحراف‌ها را تقسیم بر تعداد تکرار آن‌ها (فرکانس) بکنیم.

```
def computeDeviations(self):
    # for each person in the data:
    # get their ratings
    for ratings in self.data.values():
        # for each item & rating in that set of ratings:
        for (item, rating) in ratings.items():
            self.frequencies.setdefault(item, {})
            self.deviations.setdefault(item, {})
            # for each item2 & rating2 in that set of ra
tings:
            for (item2, rating2) in ratings.items():
```

```
                if item != item2:
                    # add the difference between the rat
ings
                    # to our computation
                    self.frequencies[item].setdefault(it
em2, 0)
                    self.deviations[item].setdefault(ite
m2, 0.0)
                    self.frequencies[item][item2] += 1
                    self.deviations[item][item2] += rati
ng - rating2
    for (item, ratings) in self.deviations.items():
        for item2 in ratings:
            ratings[item2] /= self.frequencies[item][ite
m2]
```

تمام شد! در واقع ما فرمولِ زیر را در ۱۸ خط کد (با کامنت‌ها) پیاده‌سازی کردیم:

$$dev_{i,j} = \sum_{u \in S_{i,j}(X)} \frac{u_i - u_j}{card(S_{i,j}(X))}$$

هنگامی که این تابع را با داده‌هایی که برای این مثال استفاده کرده‌ام، اجرا می‌کنم:

```
users2 = {"Amy": {"Taylor Swift": 4, "PSY": 3, "Whitney
Houston": 4},
"Ben": {"Taylor Swift": 5, "PSY": 2},
"Clara": {"PSY": 3.5, "Whitney Houston": 4},
"Daisy": {"Taylor Swift": 5, "Whitney Houston": 3}}
```

نتیجه‌ی زیر حاصل می‌شود:

```
>>> r = recommender(users2)
>>> r.computeDeviations()
>>> r.deviations
{'PSY': {'Taylor Swift': -2.0, 'Whitney Houston': -0.75}, 'Taylor
Swift': {'PSY': 2.0, 'Whitney Houston': 1.0}, 'Whitney Houston':
{'PSY': 0.75, 'Taylor Swift': -1.0}}
```

که همان‌چیزی است که در محاسبه‌ی دستی نیز به دست آمد:

	Taylor Swift	PSY	Whitney Houston
Taylor Swift	0	2	1
PSY	-2	0	-0.75
Whitney Houston	-1	0.75	0

یک تشکر هم بکنیم از Bryan O'Sullivan و وبلاگِ خوبش (serpentine.com/blog) که پیاده‌سازیِ پایتون را برای **الگوریتمِ شیبِ یک** در این وبلاگ قرار داده بود. کدهایی که این‌جا گفتم، بر اساس کارهای ایشان بوده‌است.

الگوریتمِ شیبِ یکِ وزن‌دارد (Weighted Slope 1): قسمت توصیه‌گر

حالا وقت آن است که قسمت توصیه‌گر را بنویسیم:

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u)-\{j\}} (dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}}$$

سوالی که مطرح می‌شود این است که آیا می‌توانیم آن ۱۸ خط کد را که برای محاسبه‌ی انحراف (computeDeviations) نوشتیم، کمترش کنیم؟ اجازه بدهید، فرمول بالا را تفسیر کنیم و آن را به زبانِ فارسی یا سودوکد بنویسیم. اول شما امتحان کنید:

مدادتان را تیز کنید
فرمول را به صورت سودوکد بنویسید:

مدادتان را تیز کنید – راه حل
راه‌حل من به این صورت است
می‌خواهم یک پیشنهاد برای یک کاربرِ خاص درست کنم. پیشنهادهای آن کاربر را به فرم زیر دارم:

سودوکد:
برای هر userItem (آیتم‌های کاربر) و هر userRating (امتیازات کاربر) در قسمت پیشنهادهای کاربر:
برای هر diffItem (اختلاف آیتم‌ها) که آن کاربر امتیازدهی نکرده باشد (item2 ≠ item):
اختلاف diffItem را با توجه به userItem به userRating برای userItem اضافه کنید. آن را در تعداد افرادی که userItem و diffItem را امتیازدهی کرده‌اند، ضرب کنید
آن را با مقدار جاری diffItem جمع کنید
همچنین جمع تعداد افرادی که diffItem را امتیازدهی کرده‌اند، نگه دارید

> **در آخر برای هر کدام از diffItemها که در مجموعه‌ی نتایجِ ما هست، جمعِ تمام آن آیتم را تقسیم بر تمام فرکانس‌های آن آیتم کنید و نتایج را برگردانید**

و این هم راه حل من به عنوان یک کد پایتون:

```
def slopeOneRecommendations(self, userRatings):
    recommendations = {}
    frequencies = {}
    # for every item and rating in the user's recommenda
tions
    for (userItem, userRating) in userRatings.items():
        # for every item in our dataset that the user di
dn't rate
        for (diffItem, diffRatings) in self.deviations.i
tems():
            if diffItem not in userRatings and userItem
in self.deviations[diffItem]:
                freq = self.frequencies[diffItem][userIt
em]
                recommendations.setdefault(diffItem, 0.0
)
                frequencies.setdefault(diffItem, 0)
                # add to the running sum representing th
e numerator
                # of the formula
                recommendations[diffItem] += (diffRating
s[userItem] + userRating) * freq
                # keep a running sum of the frequency of
 diffitem
                frequencies[diffItem] += freq
    recommendations = [(self.convertProductID2name(k),
        v / frequencies[k])
        for (k, v) in recommendations.items()]
    # finally sort and return
    recommendations.sort(key=lambda artistTuple: artistT
uple[1], reverse = True)
    return recommendations
```

و در زیر، یک تستِ ساده برای **الگوریتمِ شیبِ یک** به صورت کامل:

```
>>> r = recommender(users2)
>>> r.computeDeviations()
>>> g = users2['Ben']
>>> r.slopeOneRecommendations(g)
[('Whitney Houston', 3.375)]
```

نتایاجِ حاصل شده، با چیزی که به صورت دستی محاسبه کردیم، انطباق دارد. پس قسمتِ پیشنهاددهنده (recommendation) در این الگوریتم، در ۱۸ خط انجام شد. در نتیجه توسطِ ۳۶ خط کدِ پایتون، الگوریتمِ شیبِ یک را پیاده‌سازی کردیم. با این مثال فهمیدیم که با استفاده از زبان پایتون، می‌توانید کدهای نسبتاً فشرده را تولید کنید.

مجموعه‌ی داده‌ی MovieLens

اجازه بدهید، الگوریتم شیبِ یک را بر روی یک مجموعه‌ی داده‌ی دیگر، امتحان کنیم. مجموعه‌ی داده‌ی MoviewLens که توسط گروه تحقیقاتیِ GroupLens در دانشگاهِ مینه‌سوتا جمع‌آوری شده، شاملِ امتیازاتِ کاربران به فیلم‌های مختلف است. این مجموعه‌ی داده در سایت www.grouplens.org جهت دانلود در سه سایز مختلف در دسترسِ عموم قرار داده شده است. برای یک آزمایشِ ساده، من از کوچکترین مجموعه‌ی آن استفاده می‌کنم که شامل صد هزار امتیاز (بین ۱ تا ۵) از ۹۴۳ کاربر در مورد ۱۶۸۲ فیلم است. یک تابع نوشتم که به وسیله‌ی آن می‌توانیم داده‌ها را به کلاسِ پیشنهادگر (recommender) معرفی کنیم.

حال بیایید امتحان کنیم:

اول، داده‌ها را به شیِ توصیه‌گر (recommender object) در پایتون، بارگزاری می‌کنم:

```
>>> r = recommender(0)
>>> r.loadMovieLens('/Users/raz/Downloads/ml-100k/')
102625
```

از اطلاعاتِ کاربرِ شماره‌ی ۱ استفاده خواهم کرد. فقط جهتِ بررسی داده‌ها، به ۵۰ عدد از اقلامی که کاربرِ شماره‌ی ۱ به آن‌ها امتیاز داده‌است، نگاه می‌کنیم:

```
>>> r.showUserTopItems('1', 50)
When Harry Met Sally... (1989)     5
Jean de Florette (1986) 5
Godfather, The (1972)   5
Big Night (1996)  5
Manon of the Spring (Manon des sources) (1986) 5
Sling Blade (1996)      5
Breaking the Waves (1996)     5
Terminator 2: Judgment Day (1991)   5
Searching for Bobby Fischer (1993)  5
```

کاربرِ شماره‌ی ۱ به همه‌ی این ۵۰ فیلم، امتیاز ۵ داده‌است!
حالا اولین مرحله‌ی **الگوریتمِ شیبِ یک** را انجام می‌دهم: محاسبه‌ی انحراف‌ها:

```
>>> r.computeDeviations()
```

و در آخر، بیایید یک سری پیشنهاد برای کاربر شماره‌ی ۱ بسازیم:

```
>>> r.slopeOneRecommendations(r.data['1'])
[('Entertaining Angels: The Dorothy Day Story (1996)', 6.375), ('Aiqing
wansui (1994)', 5.849056603773585), ('Boys, Les (1997)',
5.644970414201183), ("Someone Else's America (1995)",
5.391304347826087), ('Santa with Muscles (1996)', 5.380952380952381),
('Great Day in Harlem, A (1994)', 5.275862068965517),  ...
```

و یک سری پیشنهاد برای کاربر شماره‌ی ۲۵:

```
>>> r.slopeOneRecommendations(r.data['25'])
[('Aiqing wansui (1994)', 5.674418604651163), ('Boys, Les (1997)',
5.523076923076923), ('Star Kid (1997)', 5.25), ('Santa with Muscles
(1996)',
```

پروژه‌ها

۱. ببینید که چطور الگوریتمِ شیبِ یک، فیلم‌های مختلف را به شما پیشنهاد می‌دهد. برای این‌کار، ۱۰ فیلم را امتیازدهی کنید (در همین مجموعه‌ی داده‌ی MovieLens). آیا الگوریتم، فیلم‌هایی را که دوست دارید به شما پیشنهاد می‌دهد؟

۲. فرمول شباهتِ کسینوسیِ تعدیل‌شده (adjusted cosine similarity) را پیاده‌سازی کنید. کاراییِ آن را با روشِ شیبِ یک مقایسه کنید.

۳. (سخت‌تر از همه) وقتی که می‌خواستم این روش را بر روی داده‌های سایت مبادله‌ی کتاب (Book Crossing) اجرا کنم، حافظه‌ی اصلی (RAM) کم آوردم! من یک حافظه‌ی اصلیِ ۸ گیگابایتی دارم. یادتان بیاید که ۲۷۰ هزار کتاب داشتیم که امتیازدهی شده بودند. پس به یک دیکشنری با ابعاد ۲۷۰۰۰۰ در ۲۷۰۰۰۰ برای ذخیره‌ی انحراف‌ها، احتیاج داریم. یعنی تقریباً ۷۳ میلیارد خانه‌ی در دیکشنری! یک همچین دیکشنری‌ای برای داده‌های MovieLens چقدر تنک (sparse) است؟ کُد را به گونه‌ای تغییر دهید تا بتوانیم مجموعه داده‌های بزرگتری را نیز پردازش کنیم.

برای اتمامِ فصل ۳ به شما تبریک می‌گوییم!!

انصافاً عملیات‌هایی سخت و طاقت‌فرسا در این فصل کم نداشتیم. واکاویِ کدهای به ظاهر پیچیده برای درکِ هر چه بیشترِ آن‌ها و پیاده‌سازیِ آن‌ها کار راحتی نبود.

فصل ٤. طبقه‌بندی

پایش بر اساس محتوا و طبقه‌بندی (Content Based Filtering & Classification)

طبقه‌بندی (Classification) بر اساس ویژگی‌های اقلام

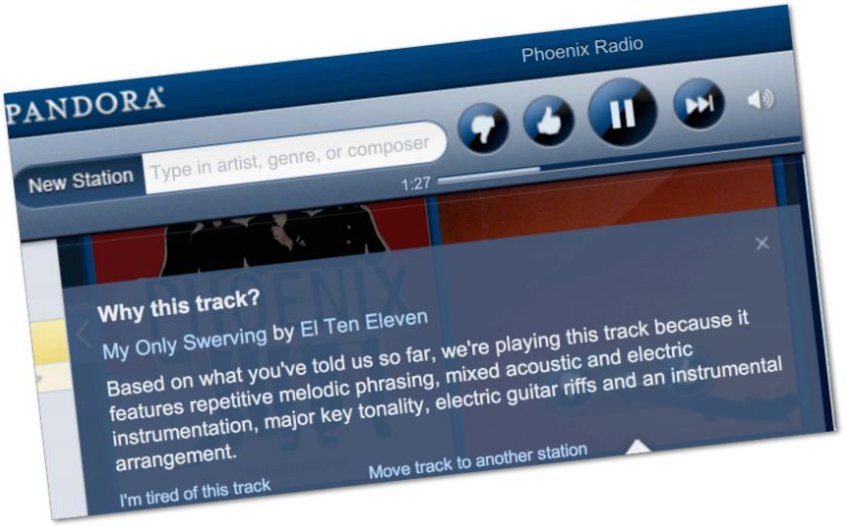
در فصل‌های قبلی، در مورد ایجاد پیشنهادات و توصیه ها برای کاربران، از طریق پایش و فیلترینگِ مشارکتی که پایشِ اجتماعی – social filtering نیز نامیده می‌شود صحبت کردیم. در پایش مشارکتی (collaborative filtering) ما قدرتِ اجتماعِ مردم را به نوعی مهار کردیم تا بتوانیم توسط آن، پیشنهادهایی را برای کاربران آماده کنیم. فرض کنید شما آلبومِ Wolfgang Amadeus Phoenix را خریده‌اید. ما می‌دانیم که بسیاری از مشتریان ما که این آلبومِ موسیقی را خریده‌اند، همچنین آلبوم Contra اثر Vampire Weekend را نیز خریداری کرده‌اند. پس ما آلبوم Contra را نیز به شما توصیه می کنیم. من یک قسمت از سریال Doctor Who را می‌بینم و سایت Netflix، سریال Quantum Leap را به من برای دیدن، پیشنهاد می‌دهد. سایت Netflix به این دلیل این کار را انجام می‌دهد که، بسیاری از کاربرانی که سریال Doctor who را در این سایت دیده‌اند، همچنین سریال Quantum Leap را هم دیده‌اند. در فصل‌های قبلی، در موردِ برخی از چالش‌های پایشِ مشارکتی صحبت کردیم. این چالش‌ها شاملِ مشکلاتی با تنک بودن داده‌ها (data sparsity) و مقیاس‌پذیر بودنِ سیستم می‌شد. مشکلِ دیگر این است که سیستم‌های پیشنهاددهنده یا همان سیستم‌های توصیه‌گر که مبتنی بر پایشِ مشارکتی کار می‌کنند، بیشتر تمایل دارند تا اقلامی را به کاربران پیشنهاد دهند که در حال حاضر، مشهور هستند – در واقع یک انحراف (bias) به سمت شهرت وجود دارد. به عنوان مثال که البته مثالی

زیاد خوبی هم نیست، تصور کنید که یک گروهِ موسیقیِ تازه‌کار، اولین آلبوم خود را انتشار می‌دهد. از آن‌جایی که این گروهِ موسیقی و این آلبوم، توسطِ هیچ کاربری، تا به حال امتیازدهی نشده است، هیچ‌وقت به کسی توسطِ سیستم، پیشنهاد نمی‌شود.

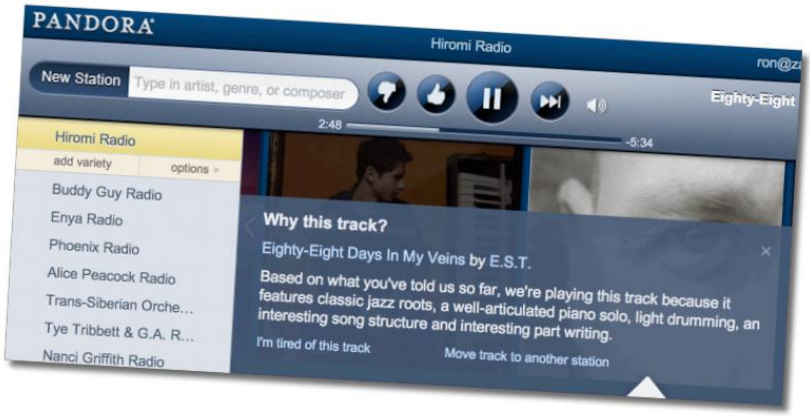
«این قبیل سیستم‌های پیشنهاددهنده، می‌توانند پولدارها را پولدارتر کنند (برای محصولات مشهور) و برعکس (برای محصولات کم‌تر شناخته‌شده)»

Daniel Fleder & Kartik Hosanagar. 2009. "Blockbusters Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity" Management Science vol 55

در این فصل، ما راهکارِ متفاوتی را دنبال می‌کنیم. سایت پخشِ آنلاینِ موسیقیِ پاندورا (Pandora) را تصور کنید. در این سایت، همان‌طور که خیلی از شما می‌دانید، می‌توانید ایستگاه‌های رادیوییِ مختلفی را ایجاد و تنظیم کنید. در هر ایستگاهِ رادیویی، شما با گوش دادن به یک گروه موسیقی یا یک هنرمند، یک نوع بذرپاشیِ اولیه را انجام می‌دهید (seeding) و بعد از آن پاندورا، آهنگ‌هایی را پخش خواهد کرد که به آن گروه موسیقی یا هنرمندِ اولیه، شبیه باشند. مثلاً من می‌توانم یک ایستگاه درست کرده و آن را با گوش دادن به گروهِ Phoenix شروع کنم. حالا این سایت، بعد از این، خودش شروع به پخش موسیقی‌هایی می‌کند که احتمال می‌دهد شبیه به Phoenix باشد. برای مثال، یک آهنگ از El Ten Eleven را پخش می‌کند. سایت پاندورا، این کار را با روشِ پایش مشارکتی انجام نمی‌دهد. در واقع، آهنگی از El Ten Eleven را پخش خواهد کرد، به خاطر این‌که الگوریتم‌های به کار گرفته‌شده در این سایت به این جمع‌بندی رسیده‌اند که El Ten Eleven از لحاظ موسیقیایی شبیه به Phoenix است. می‌توانید از پاندورا بپرسید که چرا این موسیقی را پخش کرده است (چیزی مانند شکل زیر):



همان‌طور که در تصویر می‌بینید (و خود پاندورا هم دلیلِ این رفتار را گفته است)، پاندورا آهنگِ My Only Swerving از El Ten Eleven را انتخاب و در ایستگاهِ موسیقیِ Phoenix پخش کرده است، و برای این‌کارِ خود دلایلی داشته است. این دلایل را زیر آهنگ در قسمتِ «Why this track?» نوشته است: «بر اساس چیزهایی که تا به حال شما به ما گفته اید، این قطعه‌ی موسیقی را پخش می‌کنیم، به خاطر تکرار ملودی‌ها، آکوستیکِ مرکب و تنظیم الکترونیک، تونالیتیِ کلید اصلی، ریف های گیتار الکتریک و چیدمانِ ابزاری (instrumental arrangement)». در ایستگاهِ موسیقیِ Hiromi، پاندورا یک موسیقی از E.S.T پخش کرد، به خاطر این که: «ریشه‌ی جاز کلاسیک داشت، یک تک نوازی زیبای پیانو داشت، ضرب‌زنی آرامی داشت و هچنین نوشته و ساختار جذابی داشت».



پاندورا، سیستم پیشنهادگرِ خود را بر پایه‌ی پروژه‌ای به نامِ «**پروژه‌ی ژنوم موسیقی (Music Genome Project)**» گذاشته است. آن‌ها، چندین موسیقی‌دان حرفه‌ای با پیش‌زمینه‌ی خوب در تئوری و تحلیلِ آثار موسیقی را به کار گرفتند تا بتوانند ویژگی‌ها را برای انواعِ موسیقی‌ها به دست آورند (آن‌ها به هر کدام از این ویژگی‌ها «ژن» می‌گویند). این تحلیلگران، بیش تر از ۱۵۰ ساعت، آموزش می‌بینند. هنگامی که آموزش دیدند، به صورت میانگین بین ۲۰ تا ۳۰ دقیقه برای هر قطعه‌ی موسیقی وقت صرف می‌کنند تا بتوانند ویژگی‌ها یا همان ژن ها را برای آن قطعه‌ی موسیقی مشخص کنند. بسیاری از این ژن ها، فنی هستند (نمونه‌ای از برگه‌های این تحلیل‌گران را مشاهده می‌کنید):

(در تصویر بالا دو آهنگِ EL Ten Eleven و My Only Swerving را مشاهده می‌کنیم که هر کدام ۵ ویژگی یعنی ۵ ژن دارند. مثلاً ویژگیِ **تعداد ضربه در دقیقه (Beats per Minute)** برای آهنگ EI Ten Eleven برابر ۱۱۰ نوشته شده است و به همین ترتیب ویژگی‌های دیگر نیز موجود هستند)

تحلیل گران، این مقادیر را برای بیش‌از ۴۰۰ ژن فراهم می‌کنند. این فرآیند، کاری به شدت طاقت‌فرسا است و تقریباً ۱۵ هزار قطعه‌ی موسیقی، در هر ماه اضافه می‌شود.

> نکته: الگوریتم‌های پاندورا، به صورت اختصاصی در اختیار پاندورا قرار دارد و من نمی‌دانم که این الگوریتم‌ها چگونه کار می‌کنند. چیزی که گفته شد، در واقع نحوه‌ی کارکردِ پاندورا نبود، بلکه به نوعی توضیحی درباره‌ی چگونگیِ ساختنِ یک همچین سیستمی شبیه به پاندورا در موسیقی بود.

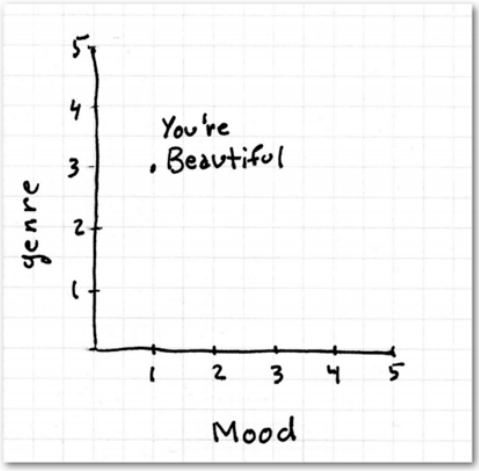
اهمیتِ انتخابِ مقادیرِ مناسب

فرض کنید که پاندورا از این دو ژن (ویژگی) برای موسیقی‌های خود استفاده کند: **ژانر (genre)** و **حال‌وهوا (mood)**. مقادیرِ این ژن‌ها می‌تواند چیزی مانند شکل زیر شود:

Mood	
Melancholy	1
joyful	2
passion	3
angry	4
unknown	5

genre	
Country	1
Jazz	2
Rock	3
Soul	4
Rap	5

پس هنگامی که مقدار ژانر (genre) برابر ۴ باشد به معنیِ سبکِ «روحی (Soul)» است و اگر مقدار حال‌وهوا (mood) برابر ۳ باشد، به معنای سبک «شور و اشتیاق (passion)» است. حالا فرض کنید، من موسیقیِ راک با حال و هوای «ملایم (melancholy)» داشته باشم. برای مثال، آهنگِ You’re Beautiful اثر James Blunt در یک فضای ۲ بُعدی (2D)، چیزی شبیه به شکل زیر برای این آهنگ به وجود می‌آید:



یک حقیقت:
در نظرسنجیِ Rolling Stone در مورد بدترین آهنگ‌ها تا به حال، قطعه‌ی You’re Beautiful در رده‌ی ۷# قرار گرفت!

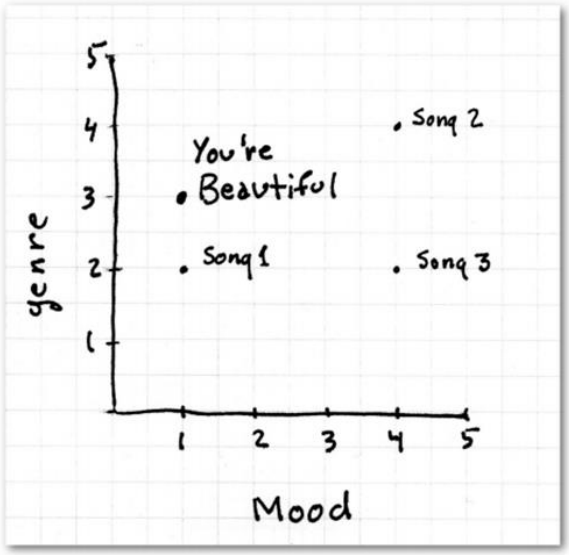
فرض کنید، Tex، عاشقِ آهنگِ You’re Beautiful است و ما می‌خواهیم، پیشنهادِ یک آهنگِ جدید را به او بدهیم.

اجازه بدهید، به این مجموعه‌ی داده، چند موسیقیِ دیگر اضافه کنیم. موسیقیِ شماره‌ی ۱ «جاز (jazz) » و «ملایم (melancholy)» است. موسیقیِ شماره‌ی ۲ یک موسیقیِ «روحی (soul)» و «خشمگین (angry)» است و در نهایت موسیقیِ شماره‌ی ۳، یک موسیقیِ «جاز (jazz)» و «خشمگین (angry)» است. در شکل زیر این موسیقی‌ها را نمایش داده‌ایم:

موسیقی ۱ = song 1

موسیقی ۲ = song 2

موسیقی ۳ = song 3



به نظر آهنگ ۱ (song 1) نزدیک‌تر از همه هست

امیدوارم فهمیده باشید که یک اشتباهِ مهلک، در طراحی شِمای موسیقی‌ها داشته‌ایم. اجازه بدهید یک بار دیگر، مقادیرِ ممکن برای متغیرها را نگاه کنیم:

Mood	
melancholy	1
joyful	2
passion	3
angry	4
unknown	5

genre	
Country	1
Jazz	2
Rock	3
Soul	4
Rap	5

اگر سعی کنیم از هر معیارِ فاصله (که در فصل ۲ در موردش صحبت کردیم) استفاده کنیم، با استفاده از شِمایی که در شکل بالا داریم، در واقع منظورمان این است که آهنگ های «جاز (jazz)» به آهنگ‌های «راک (rock)» نزدیک‌تر هستند تا آهنگ‌های «روحی (soul)»

(فاصله‌ی بین «جاز (jazz)» و «راک (rock)» برابر ۱ است و فاصله‌ی بین «جاز (jazz)» و «روحی (soul)» برابر ۲ است). یا برای مثال، آهنگ‌های «ملایم (melancholy)» به آهنگ‌های «شاد (joyful)» نزدیک‌تر هستند تا آهنگ‌های «خشمگین (angry)». حتی اگر مقادیر را مانند زیر بازچینی کنیم، مشکل همچنان باقی می‌ماند.

Mood	
melancholy	1
angry	2
passion	3
joyful	4
unknown	5

genre	
Country	1
Jazz	2
Soul	3
Rap	4
Rock	5

بازچینی، مشکل را حل نمی‌کند. فرقی نمی‌کند که چگونه مقادیر را بازچینی کنیم، اگر به این طریق عمل کنیم، الگوریتم جوابِ خوبی نمی‌دهد. در واقع این حالت به ما یادآوری می‌کند که ویژگی‌هایمان را به خوبی انتخاب نکرده ایم. ما ویژگی‌هایی را می‌خواهیم که مقادیرِ آن ها، در یک بازه‌ی معنی‌دار قرار بگیرند. برای حلِ این مشکل می‌توانیم به سادگی، ویژگیِ «ژانر (genre)» را به ۵ ویژگیِ جدا از هم تبدیل کنیم – یکی برای سبکِ موسیقیِ «کانتری (country)»، یکی برای «جاز (jazz)» و همین‌طور الی آخر.

تمامِ این ژانرها، می‌توانند بین ۱ تا ۵ باشند. – برای مثال، می‌توانیم بفهمیم که یک قطعه‌ی موسیقی چقدر در ژانرِ «کانتری (country)» قرار می‌گیرد – ۱ به معنای این است که این قطعه‌ی موسیقی اصلاً در

ژانرِ «کانتری (country)» قرار نمی‌گیرد و ۵ به معنای این است که این قطعه‌ی موسیقی کلاً ژانرِ «کانتری (country)» دارد. حالا، این مقیاس، یک معنیِ خاصی می‌دهد. مثلاً الان اگر بخواهیم یک قطعه‌ی موسیقی، شبیه به قطعه‌ی دیگر، که مقدار ژانرِ «کانتری (country)» برای آن ۵ شده است، پیدا کنیم، قطعه‌ای که مقدارِ «کانتری (country)» بودن برای آن برابر ۴ باشد، طبیعتاً نزدیک تر از قطعه‌ای است که مقدارِ «کانتری (country)» بودن برای آن برابرِ ۱ باشد.

این دقیقاً همان روشی است که پاندورا، مجموعه‌ی ژن‌ها یا همان ویژگی‌های خود را می‌سازد. ژن‌های مختلف در پاندورا، اکثراً در بازه‌ی ۱ تا ۵ هستند (البته با قدم‌های ۰,۵ تایی). ژن‌ها در دسته‌ها قرار می‌گیرند. برای مثال، یک دسته‌بندیِ کیفیِ موسیقیایی وجود دارد که شاملِ ژن‌هایی برای انواع موسیقی‌های «راک بلوز (Blues Rock)»، «راک محلی (Folk Rock)» و «پاپ راک (Pop Rock)» در میان بقیه‌ی موسیقی‌ها می‌شود. دسته‌بندیِ دیگر، به وسیله‌ی ژانرهایی مانندِ «آکاردئون (accordion)»، «ریف‌های گیتار الکتریک کثیف (Dirty Electric Guitar Riffs)» و «استفاده از ارگان‌های صدایی کثیف (Use of Dirty Sounding Organs)» نواخته می‌شود. با استفاده از این ژن‌ها (ویژگی‌ها) که هر کدام از آن ها، به خوبی مقادیری بین ۱ تا ۵ دارند، پاندورا، هر قطعه‌ی موسیقی را به یک بُردار (vector) با اندازه‌ی ۴۰۰ تبدیل کرده است (یعنی هر قطعه‌ی موسیقی، یک نقطه در یک فضای ۴۰۰ بُعدی است). حالا پاندورا، می‌تواند پیشنهادات خود را بسازد. این پیشنهادها شاملِ تصمیم به پخشِ یک موسیقی در یک ایستگاه رادیوییِ ساخته شده توسط کاربر است). این پیشنهاد ها بر اساس توابع فاصله‌ی استاندارد که در فصل ۲ و ۳ درمورد آن‌ها صحبت کردیم، هستند.

یک مثال ساده

اجازه بدهید یک مجموعه داده‌ی ساده تولید کنیم تا با این روش بیشتر آشنا شویم. فرض کنید، هفت ویژگی داریم که هر کدام در بازه‌ی ۱ تا ۵ هستند (با افزایش ۰,۵تایی). البته قبول دارم که خیلی منطقی و کامل نیست:

۱. piano (پیانو) = اگر در آهنگ از پیانو استفاده نشده باشد، عدد ۱ و اگر در سراسر آهنگ کاملاً و به صورتِ موثر از پیانو استفاده شده بود عدد ۵ در این ویژگی قرار می‌گیرد

۲. **vocals (وکال ها)** = اگر در آهنگ هیچ وکالی نبود، عدد ۱ و اگر در سراسر آهنگ به صورت موثر از وکال استفاده شده بود، عدد ۵ به این ویژگی تعلق می‌گیرد

۳. **driving beat (ضربه درایوینگ)** = ترکیبی از تمپوی ممتد، و این‌که چطور درام‌ها و باس، ضربه‌های موسیقی را هدایت می‌کنند

۴. **blues Infulence (تاثیر بلوز)**

۵. **dirty elec. guitar (گیتار الکتریک کثیف)**

۶. **backup vocals (وکال‌های پشتیبان)**

۷. **rap inf. (تاثیر رپ)**

حالا با استفاده از این ویژگی‌ها، می‌خواهم ۱۰ قطعه‌ی موسیقی را امتیازدهی کنم (قطعاتِ موسیقی در سطرها قرار دارند و ویژگی‌ها در ستون‌ها هستند):

	Piano	Vocals	Driving beat	Blues infl.	Dirty elec. Guitar	Backup vocals	Rap infl.
Dr. Dog/ Fate	2.5	4	3.5	3	5	4	1
Phoenix/ Lisztomania	2	5	5	3	2	1	1
Heartless Bastards / Out at Sea	1	5	4	2	4	1	1
Todd Snider/ Don't Tempt Me	4	5	4	4	1	5	1
The Black Keys/ Magic Potion	1	4	5	3.5	5	1	1
Glee Cast/ Jessie's Girl	1	5	3.5	3	4	5	1
Black Eyed Peas/ Rock that Body	2	5	5	1	2	2	4
La Roux/ Bulletproof	5	5	4	2	1	1	1
Mike Posner/ Cooler than me	2.5	4	4	1	1	1	1
Lady Gaga/ Alejandro	1	5	3	2	1	2	1

از آن‌جایی که هر کدام از قطعاتِ موسیقی، شاملِ مجموعه‌ای از اعداد هستند، می‌توانیم از هر تابعِ فاصله‌ای برای محاسبه‌ی فاصله بین این قطعاتِ موسیقی استفاده کنیم. برای مثال،

فاصله‌ی منهتن (Manhatan distance) بین قطعه‌ی Dr. Dog's Fate و قطعه‌ی Phoenix's Lisztomania به صورت زیر محاسبه می‌شود:

Dr. Dog/ Fate	2.5	4	3.5	3	5	4	1
Phoenix/ Lisztomania	2	5	5	3	2	1	1
Distance	**0.5**	**1**	**1.5**	**0**	**3**	**3**	**0**

جمع سطرِ آخر – **Distance یعنی جمع فاصله‌ها در هر ویژگی**، برابر فاصله‌ی منهتن (در این‌جا برابر ۹) می‌شود.

مدادتان را تیز کنید

دارم سعی می‌کنم بفهمم که کدام قطعات موسیقی، به آهنگِ Jessie's Girl اثر Glee با توجه به فاصله‌ی اقلیدسی، نزدیک تر هستند. آیا می‌توانید جدول زیر را تکمیل کنید و بگویید کدام یک به این آهنگ، نزدیک‌تر است.

	فاصله تا آهنگ Jessie's Girl اثر Glee's
Dr. Dog/ Fate	??
Phoenix/ Lisztomania	4.822
Heartless Bastards / Out at Sea	4.153
Todd Snider/ Don't Tempt Me	4.387
The Black Keys/ Magic Potion	4.528
Glee Cast/ Jessie's Girl	0
Black Eyed Peas/ Rock that Body	5.408
La Roux/ Bulletproof	6.500
Mike Posner/ Cooler than me	5.701
Lady Gaga/ Alejandro	??

مدادتان را تیز کنید – راه‌حل

	فاصله تا آهنگ Jessie's Girl اثر Glee's
Dr. Dog/ Fate	2.291
Lady Gaga/ Alejandro	4.387

به یاد بیاورید که فاصله‌ی اقلیدسی (Euclidean distance) بین دو شیِ **x** و **y** که دارای **n** ویژگی (**n** بُعد) بودند، به صورت زیر محاسبه می‌شد:

$$d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}$$

پس فاصله‌ی اقلیدسی بین Glee و Lady Gaga به صورت زیر محاسبه می‌شود:

	piano	vocals	beat	blues	guitar	backup	rap	SUM	SQRT
Glee	1	5	3.5	3	4	5	1		
Lady G	1	5	3	2	1	2	1		
(x-y)	0	0	0.5	1	3	3	0		
$(x-y)^2$	0	0	0.25	1	9	9	0	19.25	4.387

برویم به سراغ کدهای پایتون

به یاد بیاورید که داده‌های ما برای پایش اجتماعی (social filtering) در فصلِ قبلی، به فرمت زیر بود:

```
users = {"Angelica": {"Blues Traveler": 3.5, "Broken Bells": 2.0,
                      "Norah Jones": 4.5, "Phoenix": 5.0,
                      "Slightly Stoopid": 1.5, "The Strokes": 2.5,
                      "Vampire Weekend": 2.0},
         "Bill":     {"Blues Traveler": 2.0, "Broken Bells": 3.5,
                      "Deadmau5": 4.0, "Phoenix": 2.0,
                      "Slightly Stoopid": 3.5, "Vampire Weekend": 3.0}}
```

داده‌های فعلی را نیز می‌توانیم به همان شکل نمایش دهیم:

```
music = {"Dr Dog/Fate": {"piano": 2.5, "vocals": 4, "beat": 3.5,
                         "blues": 3, "guitar": 5, "backup vocals": 4,
                         "rap": 1},
         "Phoenix/Lisztomania": {"piano": 2, "vocals": 5, "beat": 5,
                                 "blues": 3, "guitar": 2,
                                 "backup vocals": 1, "rap": 1},
         "Heartless Bastards/Out at Sea": {"piano": 1, "vocals": 5,
                                           "beat": 4, "blues": 2,
                                           "guitar": 4,
                                           "backup vocals": 1,
                                           "rap": 1},
         "Todd Snider/Don't Tempt Me": {"piano": 4, "vocals": 5,
                                        "beat": 4, "blues": 4,
                                        "guitar": 1,
                                        "backup vocals": 5, "rap": 1},
         "The Black Keys/Magic Potion":{"piano": 1, "vocals": 4,
                                           "beat": 5, "blues": 3.5,
                                           "guitar": 5,
                                           "backup vocals": 1,
                                           "rap": 1},
         "Glee Cast/Jessie's Girl": {"piano": 1, "vocals": 5,
                                     "beat": 3.5, "blues": 3,
                                     "guitar":4, "backup vocals": 5,
                                     "rap": 1},
         "La Roux/Bulletproof": {"piano": 5, "vocals": 5, "beat": 4,
```

حالا فرض کنید که من دوستی دارم که می‌گوید بـه قطعـه‌ی Black Keys Magic Potion علاقه‌مند است. من می‌توانم نزدیـک‌تـرین همسـایه‌ی آن را توسـط تـابعِ فاصـله‌ی منهـتن (Manhattan) که با پایتون نوشته‌ام محاسبه کنم:

```
>>> computeNearestNeighbor('The Black Keys/Magic Potion', music)
[(4.5, 'Heartless Bastards/Out at Sea'), (5.5, 'Phoenix/Lisztomania'),
(6.5, 'Dr Dog/Fate'), (8.0, "Glee Cast/Jessie's Girl"), (9.0, 'Mike
Posner'), (9.5, 'Lady Gaga/Alejandro'), (11.5, 'Black Eyed Peas/Rock
That Body'), (11.5, 'La Roux/Bulletproof'), (13.5, "Todd Snider/Don't
Tempt Me")]
```

و با استفاده از ایـن روش، مـی‌تـوانم بـه او قطعـه‌ی Heartless Bastard’s Out at Sea را پیشنهاد دهم که به نظر پیشنهاد خوبی هم هست.

نکته

کدهای این مثال، مانند تمامیِ دیگر کدهای این کتاب، در وب‌سایت guidtodatamining.com و وب‌سایت chistio.ir موجود است.

پاسخ به سوالِ «چرا؟»

هنگامی که پاندورا، چیزی را به شما پیشنهاد می‌دهد، برایتان توضیح می‌دهد که چرا ممکن است آن را دوست داشته باشید:

ما هم می‌توانیم این کار را بکنیم. آن دوستمان را به یاد بیاورید که قطعه‌ی Black Keys Magic Potion را دوست داشت. ما به او قطعه‌ی Heartless Bastards Out at Sea را پیشنهاد دادیم. حال سوال این جاست که کدام ویژگی‌ها در آن پیشنهاد تاثیرگذار بودند. بیایید، دو بردارِ ویژگی (برای هر قطعه‌ی موسیقی یک بردار) را با یکدیگر مقایسه کنیم (سطرها نمایان‌گرِ قطعات موسیقی هستند و ستون‌ها بیان‌گر ویژگی‌ها):

	Piano	Vocals	Driving beat	Blues infl.	Dirty elec. Guitar	Backup vocals	Rap infl.
Black Keys Magic Potion	1	5	4	2	4	1	1
Heartless Bastards Out at Sea	1	4	5	3.5	5	1	1
difference	0	1	1	1.5	1	0	0

ویژگی‌هایی که بین این دو قطعه‌ی موسیقی مشترک بود، پیانو (piano)، وجود وکال‌های پشتیبان (backup vocals) و تاثیرِ رپ (rap inf.) بود – که همه‌ی آن‌ها فاصله‌ی صفر با هم داشتند، یعنی در هر دو بردار برابر بودند. با این‌حال، این ویژگی‌ها در هر کدام از قطعاتِ موسیقی، در کمترین حالتِ خود قرار داشتند: یعنی هر دو قطعه‌ی موسیقی، بدون پیانو، بدون حضور وکال‌های پشتیبان و بدون تاثیر رپ بودند. پس احتمالاً خیلی جالب نیست که مثلاً بگوییم: «ما فکر می‌کنیم که شما این قطعه‌ی موسیقی را به دلیلِ نداشتنِ وکال های پیشتیبان، دوست داشتید». در عوض، **تمرکزِ خودمان را بر روی اشتراک های این دو قطعه‌ی موسیقی در اعدادِ بزرگ‌تر می‌گذاریم** – یعنی جاهایی که اعدادِ موجود در بردارها نزدیک به ۵ باشند.

برای مثال فکر می‌کنیم شما قطعه‌ی Heartless Bastards Out at Sea را به خاطرِ داشتنِ ضربه‌های هدایت‌شونده و وکال‌ها و گیتار الکتریک کثیف دوست داشته‌اید.

به خاطرِ این که مجموعه‌ی داده‌ی ما دارای ویژگی‌های کمی هست، و البته خیلی هم متعادل نیست، بقیه‌ی پیشنهادها خیلی جالب و متقاعدکننده به نظر نمی‌رسند:

```
>>> computeNearestNeighbor("Phoenix/Lisztomania", music)

[(5, 'Heartless Bastards/Out at Sea'), (5.5, 'Mike Posner'), (5.5, 'The
Black Keys/Magic Potion'), (6, 'Black Eyed Peas/Rock That Body'), (6,
'La Roux/Bulletproof'), (6, 'Lady Gaga/Alejandro'), (8.5, "Glee Cast/
Jessie's Girl"), (9.0, 'Dr Dog/Fate'), (9, "Todd Snider/Don't Tempt
Me")]

>>> computeNearestNeighbor("Lady Gaga/Alejandro", music)

[(5, 'Heartless Bastards/Out at Sea'), (5.5, 'Mike Posner'), (6, 'La
Roux/Bulletproof'), (6, 'Phoenix/Lisztomania'), (7.5, "Glee Cast/
Jessie's Girl"), (8, 'Black Eyed Peas/Rock That Body'), (9, "Todd
Snider/Don't Tempt Me"), (9.5, 'The Black Keys/Magic Potion'), (10.0,
'Dr Dog/Fate')]
```

مثلاً پیشنهاد Lady Gaga به نظر پیشنهاد خوبی نیست.

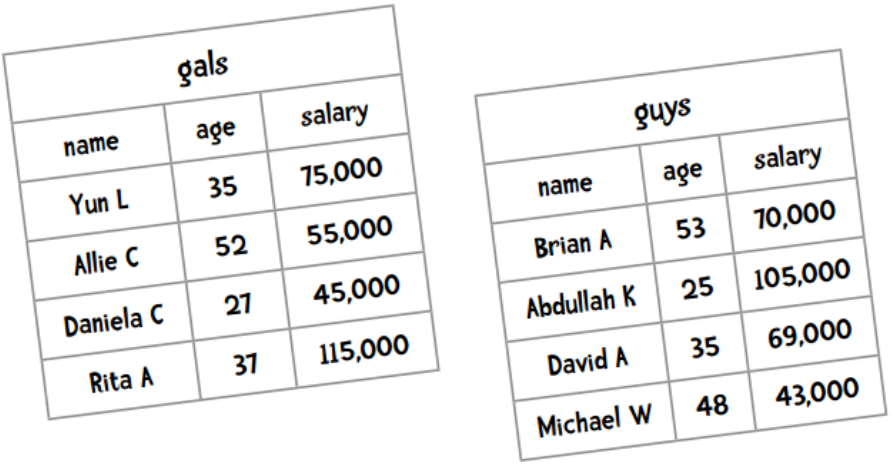
مشکل در مقیاس‌های متفاوت

فرض کنید می‌خواهم یک ویژگیِ دیگر به مجموعه‌ی داده‌هایم اضافه کنم. این بار، تعداد ضربه در دقیقه (beats per minute یا bpm) را اضافه می‌کنم. به نظر ویژگیِ به درد بخوری می‌رسد – مثلاً ممکن است ضربه‌های سریع در موسیقی‌ها را دوست داشته باشم و یا آهنگ‌های رمانتیک‌تر را ترجیح دهم. حالا با این تغییر، داده‌های من به شکل زیر در می‌آیند (مانند قبل، سطرها بیان‌گر قطعات موسیقی هستند و ستون‌ها بیان‌گر ویژگی‌ها):

	Piano	Vocals	Driving beat	Blues infl.	Dirty elec. Guitar	Backup vocals	Rap infl.	bpm
Dr. Dog/ Fate	2.5	4	3.5	3	5	4	1	140
Phoenix/ Lisztomania	2	5	5	3	2	1	1	110
Heartless Bastards / Out at Sea	1	5	4	2	4	1	1	130
The Black Keys/ Magic Potion	1	4	5	3.5	5	1	1	88
Glee Cast/ Jessie's Girl	1	5	3.5	3	4	5	1	120
Bad Plus/ Smells like Teen Spirit	5	1	2	1	1	1	1	90

بدون استفاده از bpm، نزدیک‌ترین همسایه به Black Keys’ Magic Potion، قطعه‌ی Heartless Bastards’ Out to Sea و دورترین قطعه به آن، Bad Plus’s Smells Like Teen Spirit است. ولی هنگامی که ما ویژگیِ bpm را اضافه می‌کنیم، با استفاده از همان تابعِ فاصله ، نتایج کاملاً به هم می‌ریزد و خراب می‌شود – در واقع bpm به نوعی، بر روی محاسبات ما حکم‌رانی و سیطره پیدا می‌کند. حالا، قطعه‌ی Bad Plus به Black Keys خیلی نزدیک‌تر است و دلیل آن این است که ویژگیِ bpm آن‌ها به یکدیگر خیلی نزدیک است.

مثال دیگری را در نظر بگیرید. فرض کنید من یک سایت، مخصوص قرار ملاقات دارم و اعتقاد دارم که بهترین ویژگی‌ها برای این که دو نفر را به هم معرفی کنیم یکی سن (age) است و دیگری حقوق (salary):



gals

name	age	salary
Yun L	35	75,000
Allie C	52	55,000
Daniela C	27	45,000
Rita A	37	115,000

guys

name	age	salary
Brian A	53	70,000
Abdullah K	25	105,000
David A	35	69,000
Michael W	48	43,000

(جدول gals، جنسیت مونث هستند و جدول guys جنسیت مذکر)

در این جا بازه‌ی سن، بین ۲۵ تا ۵۳ سال (یعنی با اختلاف ۲۸ سال) است ولی بازه‌ی حقوق در بین ۴۳ هزار تا ۱۱۵ هزار، یعنی با اختلاف ۷۲ هزار واحد قرار دارد. به خاطرِ تفاوتِ بسیار زیادِ این بازه‌ها، ویژگیِ حقوق (salary) بر تمامیِ محاسبات حکم‌رانی و سیطره پیدا می‌کند. اگر بخواهیم به صورت چشمی، دو نفر را به هم مرتبط کنیم، احتمالاً David را به Yun مرتبط خواهیم کرد چون آن‌ها در یک رده‌ی سنی هستند و حقوقشان هم تقریباً به هم نزدیک است. با این حال، اگر همین کار را با استفاده از هر کدام از فرمول‌های فاصله که تا به این‌جا گفتیم (مثل منهتن یا اقلیدسی)، انجام دهیم، Brian با ۵۳ سال سن، شخصی است که به Yun با ۳۵ سال سن پیشنهاد می‌شود که این قطعاً برای سایتِ تازه تاسیسِ من خوب نیست.

در واقع، بازه‌ها و مقیاس‌های متفاوت بین ویژگی‌ها، یک مشکل بزرگ برای هر سیستم پیشنهاددهنده‌ای است.

نرمال‌سازی

وحشت نکنید!

یک نفس عمیق بکشید و آسوده باشید...

راه‌حل، نرمال‌سازی (normalization) است!
برای این‌که این مشکل و انحراف را حل کنیم، احتیاج داریم تا داده‌ها را **نرمال‌سازی (normalize)** یا **استانداردسازی (standardize)** کنیم. یک راه‌حلِ رایج در نرمال‌سازی این است که مقادیرِ هر ویژگی را در بازه‌ی صفر یا یک داشته باشیم.
برای مثال، ویژگیِ حقوق (salary) را در وب‌سایتِ قرار مقالات تصور کنید. کمترین میزانِ حقوق برابر ۴۳۰۰۰ و بیشترینِ آن برابر ۱۱۵۰۰۰ بود. در واقع یک بازه‌ی ۷۲۰۰۰ تایی وجود داشت. برای تبدیلِ هر مقدار به بازه‌ی ۰ تا ۱، آن مقدار را از کمترین مقدارِ موجود در ویژگیِ متناظرِ خود کم می‌کنیم و نتیجه را بر بازه‌ی موجود تقسیم می‌کنیم.

مونث

name	salary	normalized salary
Yun L	75,000	0.444
Allie C	55,000	0.167
Daniela C	45,000	0.028
Rita A	115,000	1.0

پس مقدار نرمال‌شده برای Yum برابر است با:

(75,000 - 43,000) / 72,000 = 0.444

بستگی به داده ها و مجموعه‌ی داده ای که در اختیار دارید، این روشِ ساده می‌تواند به خوبی کار کند.
(در ستون **normalized salary** در شکل سمت چپ، می‌توانید بقیه‌ی حقوق‌های نرمال‌شده را مشاهده کنید)
اگر دروس آماری را بخوانید با روش‌های دقیق‌تر برای استانداردسازی داده‌ها آشنا خواهید شد. برای مثال، می‌توانیم از مقداری به نام «**امتیاز استاندارد (standard score)**» استفاده کنیم که به صورت زیر محاسبه می‌شود:
می‌توانیم یک مقدار را به وسیله‌ی «**امتیاز استاندار**» که به **z-score** نیز معروف است، استانداردسازی کنیم. **این مقدار به ما می‌گوید که یک عدد در بین اعدادِ دیگر، به چه مقدار از میانگینِ آن‌ها انحراف دارد.**

هر عدد میانگین امتیاز استاندارد

$$\frac{\text{(each value)} - \text{(mean)}}{\text{(standard deviation)}} = \text{Standard Score}$$

انحراف استاندارد

انحراف استاندارد (standard deviation) یا همان انحراف معیار برابر است با:

$$sd = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{card(x)}}$$

که در آن card(x) مقدار کاردینالیتی برای x است – یعنی چند تا مقدار در آن است.

یک نکته: اگر با آمار مشکلی دارید، می‌توانید کتابِ «The Manga Guide to Statistics» را مطالعه کنید.

name	salary
Yun L	75,000
Allie C	55,000
Daniela C	45,000
Rita A	115,000
Brian A	70,000
Abdullah K	105,000
David A	69,000
Michael W	43,000

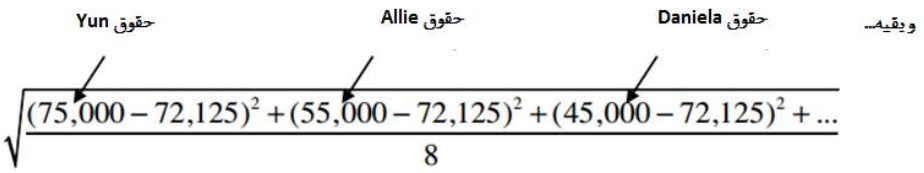
داده‌های مربوط به وب‌سایتِ قرار ملاقات در چند صفحه‌ی قبل را به یاد آورید.

در این داده‌ها، جمع تمامیِ حقوق‌ها (ستون salary)، برابر ۵۷۷۰۰۰ شده‌است. از آن‌جایی که ۸ نفر در این داده‌ها حضور داشتند، میانگینِ حقوق برابر ۷۲۱۲۵ می‌شود.

حالا، اجازه دهید، انحرافِ استاندارد (sd) را برای حقوق به دست بیاوریم (فرمول که مانند زیر بود):

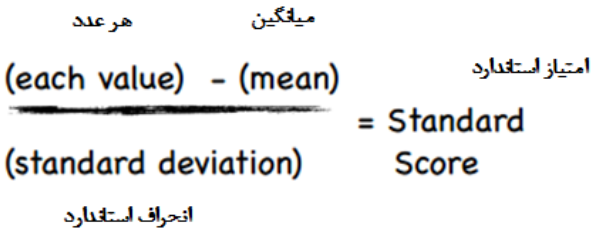
$$sd = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{card(x)}}$$

پس برای حقوق داریم:



$$=\sqrt{\frac{8{,}265{,}625+293{,}265{,}625+735{,}765{,}625+\ldots}{8}}=\sqrt{602{,}395{,}375}$$
$$=24{,}543.01$$

دوباره به یاد بیاورید که امتیازِ استاندارد (z-score) به صورت زیر محاسبه می‌شد:



پس امتیازِ استاندارد (z-score) برای فردی مثل Yun، برابر است با:

$$\frac{75000-72125}{24543.01}=\frac{2875}{24543.01}=0.117$$

مدادتان را تیز کنید

برای افراد زیر، امتیازِ استاندارد (z-score) را محاسبه کنید:

name	salary	Standard Score
Yun L	75,000	0.117
Allie C	55,000	
Daniela C	45,000	
Rita A	115,000	

مدادتان را تیز کنید – راه حل

برای افراد زیر، امتیازِ استاندارد (z-score) را محاسبه کنید:

name	salary	Standard Score
Yun L	75,000	0.117
Allie C	55,000	-0.698
Daniela C	45,000	-1.105
Rita A	115,000	1.747

Allie:
(55,000 - 72,125) / 24,543.01
= -0.698

Daniela:
(45,000 - 72,125) / 24,543.01
= -1.105

Rita:
(115,000 - 72,125) / 24,543.01
= 1.747

مشکلِ استفاده از امتیازِ استاندارد

مشکل اصلی در استفاده از امتیازِ استاندارد یا همان z-score، این است که این امتیاز، از داده های پرت، تاثیر زیادی می‌گیرد. برای مثال، اگر در یک فروشگاهی مانند LargeMart، 100 نفر کارگر حضور داشته باشند و هر کدام ۱۰ دلار در ساعت حقوق بگیرند، اما مدیرعامل ۶ میلیون دلار در سال حقوق داشته باشد، حق‌الزحمه‌ی میانگین برای هر ساعت به صورت زیر محاسبه می‌شود:

$$(100 * \$10 + 6{,}000{,}000 / (40 * 52)) / 101$$

$$= (1000 + 2885) / 101 = \$38/hr.$$

که برای فروشگاهی مانند LargeMart رقمِ کمی نیست. همان طور که مشاهده می‌کنید، میانگین، کاملاً تحتِ تاثیرِ داده‌ی پرت (همان حقوقِ مدیرعامل) قرار گرفت.
به خاطر این مشکل که با میانگین (mean) پیدا می‌کنیم، معمولاً فرمولِ امتیازِ استاندارد را تغییر می‌دهیم.

امتیازِ استانداردِ اصلاح‌شده (Modified Standard Score)

برای محاسبه‌ی امتیازِ استاندردِ اصلاح شده، میانگین (mean) را در فرمولِ بالا به میانه (median) تغییر می‌دهیم. همچنین انحراف استاندارد (در مخرج) را با «انحرافِ استانداردِ مطلق (asd)» جایگزین می‌کنیم:

$$asd = \frac{1}{card(x)} \sum_i |x_i - \mu|$$

که در این فرمول μ همان میانه (median) است.

پس امتیازِ استانداردِ اصلاح شده به صورت زیر محاسبه می‌شود:

$$\frac{\text{(each value) - (median)}}{\text{(absolute standard deviation)}}$$

برای محاسبه‌ی **میانه (median)**، اعداد را به ترتیب از کوچک‌ترین تا بزرگ‌ترین می‌چینیم، سپس عددِ میانیِ آن را انتخاب می‌کنیم. اگر تعدادِ اعداد زوج بود، میانه (media) برابرِ میانگینِ دو عددِ وسط می‌شود.
خب، حالا بیایید با مثال یاد بگیریم. در جدولِ زیر، مقادیرِ حقوق را از کم به زیاد، مرتب کرده‌ام.

Name	Salary
Michael W	43,000
Daniela C	45,000
Allie C	55,000
David A	69,000
Brian A	70,000
Yun L	75,000
Abdullah K	105,000
Rita A	115,000

آن جاهایی که تعداد اعداد زوج می‌شود، میانه (median) برابرِ میانگینِ دو عدد وسط است:

$$median = \frac{(69{,}000 + 70{,}000)}{2} = 69{,}500$$

و انحراف استانداردِ مطلق (asd) به صورت زیر محاسبه می‌شود:

$$asd = \frac{1}{card(x)} \sum_i |x_i - \mu|$$

$$asd = \frac{1}{8}(|43{,}000 - 69{,}500| + |45{,}000 - 69{,}500| + |55{,}000 - 69{,}500)| + ...)$$

$$= \frac{1}{8}(26{,}500 + 24{,}500 + 14{,}500 + 500 + ...)$$

$$= \frac{1}{8}(153{,}000) = 19{,}125$$

حالا بیایید امتیازِ استانداردِ اصلاح شده (mss) را برای Yun محاسبه کنیم.

Modified Standard Score:

$$\frac{\text{(each value) - (median)}}{\text{(absolute standard deviation)}}$$

$$mss = \frac{(75{,}000 - 69{,}500)}{19{,}125} = \frac{5{,}500}{19{,}125} = 0.2876$$

مدادتان را تیز کنید

در جدولِ زیر، تعداد دفعاتی که یک قطعه‌ی موسیقی (ستون track) را پخش کرده‌ام در ستون play count، نمایش داده شده است. آیا می‌توانید مقادیر را با اسـتفاده از امتیـازِ استانداردِ اصلاح‌شده، استانداردسازی کنید؟

track	play count	modified standard score
Power/Marcus Miller	21	
I Breathe In, I Breathe Out/ Chris Cagle	15	
Blessed / Jill Scott	12	
Europa/Santana	3	
Santa Fe/ Beirut	7	

مدادتان را تیز کنید – راه حل

قدم ۱: محاسبه‌ی میانه (median)

مقادیر را به ترتیب صعودی (۳, ۷, ۱۲, ۱۵, ۲۱) قرار داده و مقدار میانی را برمی‌دارم که برابر ۱۲ می‌شود.

قدم ۲: محاسبه‌ی انحراف استاندارد مطلق (asd)

$$asd = \frac{1}{5}(|3-12|+|7-12|+|12-12|+|15-12|+|21-12|)$$

$$= \frac{1}{5}(9+5+0+3+9) = \frac{1}{5}(26) = 5.2$$

قدم ۳: محاسبه‌ی امتیازات استاندارد اصلاح‌شده برای هر کدام از نمونه‌ها

Power / Marcus Miller: (21 - 12) / 5.2 = 9/5.2 = 1.7307692

I Breathe In, I Breathe Out / Chris Cagle: (15 - 12) / 5.2 = 3/5.2 = 0.5769231

Blessed / Jill Scott: (12 - 12) / 5.2 = 0

Europa / Santana: (3 - 12) / 5.2 = -9 / 5.2 = -1.7307692

Santa Fe / Beirut: (7 - 12) / 5.2 = - 5 / 5.2 = -0.961538

نرمال‌سازی را انجام بدهیم یا نه؟

نرمال سازی هنگامی جواب می‌دهد که بازه و مقیاسِ ویژگی‌ها (ابعاد) نسبت به یکدیگر بسیار متغیر باشند. در مثالِ موسیقی که قبل‌تر در همین فصل گفتیم، تعدادی از ویژگی‌ها وجود داشتند که در بازه‌ی ۱ تا ۵ قرار داشتند و این در حالی است که ویژگیِ «ضربه در دقیقه (beats per minute – bpm)» به صورت بلقوه در بازه‌ای بین ۶۰ تا ۱۸۰ قرار می‌گیرد. در مثالِ سایتِ قرار ملاقات هم، عدمِ تطابقِ بازه، بین ویژگیِ «سن (age)» با ویژگیِ «حقوق (salary)» مشهود بود.

تصور کنید من رویای پولدار شدن دارم و به دنبال منزلی در شهرِ سانتافه (Santa Fe) در نیومکزیکو هستم. جدولِ زیر، چند خانه در بازار را نشان می‌دهد.

قیمت	تعداد اتاق	تعداد سرویس‌ها	متراژ به فوت
$1,045,000	2	2.0	1,860
$1,895,000	3	4.0	2,907
$3,300,000	6	7.0	10,180
$6,800,000	5	6.0	8,653
$2,250,000	3	2.0	1,030

این‌جا هم مشکل، نمایان است. از آن‌جایی که بازه و مقیاسِ یکی از ویژگی‌ها (در این‌جا ویژگیِ «قیمت») خیلی بیشتر از بقیه‌ی ویژگی‌ها هست، این ویژگی بر هر معیارِ فاصله‌ای سایه می‌اندازد. و اینکه مثلاً خانه، دو اتاق داشته باشد یا بیست اتاق، تاثیر زیادی بر فاصله‌ی بین دو خانه نخواهد گذاشت.

پس هنگامی از نرمال‌سازی استفاده می‌کنیم که:

۱. روشی که مورد استفاده قرار می‌دهیم، به دنبال محاسبه‌ی فاصله‌ی بین دو رکورد (دو سطر) بر اساسِ مقادیرِ ویژگی‌های آن‌ها باشد

۲. مقیاس یا بازه‌ی ویژگی‌های مختلف در آن مجموعه‌ی داده، با یکدیگر تفاوت داشته باشند (مخصوصاً هنگامی که این تفاوت بسیار فاحش باشد – برای مثال، بازه‌ی «قیمت» در مقایسه با بازه‌ی «تعداد اتاق» در یک منزل مانند مجموعه‌ی داده‌ی بالا)

حالا فرض کنید شخصی، علاقه‌ی خود را با استفاده از آیکون های موافقت/مخالفت (انگشت شصت به سمت بالا/انگشت شصت به سمت پایین) در یک وب‌سایت خبری، برای اخبار مختلف نشان دهد. در زیر لیستی از علایق کاربر به اخبار را می‌بینید (عدد ۱ به معنیِ موافقت و عدد ۰ به معنی مخالفت است):

Bill = {0, 0, 0, 1, 1, 1, 1, 0, 1, 0 ... }

واضح است که احتیاجی به نرمال سازیِ داده ها وجود ندارد. در موردِ سایتِ پاندورا (همان سایتِ پخش موسیقی) چه؟ تمامیِ متغیرها (ویژگی‌ها)، در بازه‌ی ۱ تا ۵ قرار می‌گرفتند. آیا

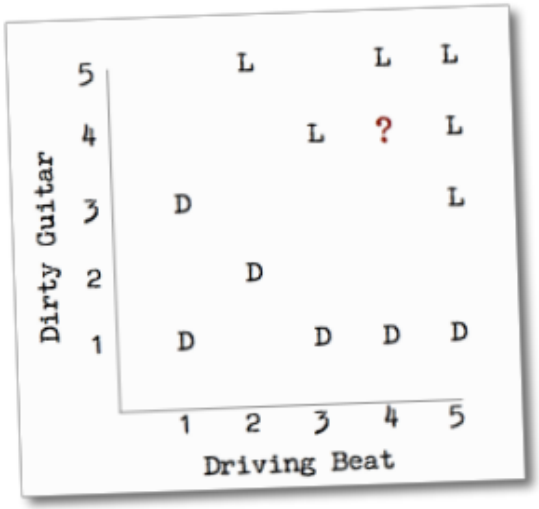
آن‌ها نیز نیاز به نرمال‌سازی دارند؟ اگر داده‌ها را نرمال‌سازی کنیم احتمالاً دقتِ الگوریتمِ مورد استفاده، آسیبی نمی‌بیند، اما حواستان به هزینه‌ی محاسباتی، برای نرمال سازی باشد. در این مورد، می‌توانیم به صورتِ تجربی و چشمی، نتایجِ داده‌های نرمال‌شده و غیرنرمال شده را مشاهده کرده و بهترین راه را انتخاب کنیم. بعداً در همین فصل موردی را خواهیم دید که نرمال‌سازی، دقتِ الگوریتم را کاهش می‌دهد.

برگردیم به مثالِ پاندورا (Pandora – سایت پخش آنلاین موسیقی)

در مثالِ پاندورا، هر موسیقی را به وسیله‌ی چندین ویژگی (attributes) نمایش می‌دادیم. اگر یک کاربر، یک ایستگاه رادیوی برای گروهِ موسیقیِ Green Day بسازد، ما تصمیم می‌گیریم که چه چیزی را بر اساس نزدیک ترین همسایه به این گروهِ موسیقی، پخش کنیم. پاندورا به کاربر اجازه می‌دهد که به یک قطعه‌ی موسیقیِ خاص، با علامت های «انگشت شصت بالا/انگشت شصت پایین» امتیاز دهد. مسئله این‌جاست که چگونه از این اطلاعاتی که کاربر در اختیار ما گذاشته استفاده کنیم؟

فرض کنید من از ۲ ویژگی، برای نمایشِ هر قطعه‌ی موسیقی استفاده کنم: مقدارِ «گیتار کثیف (Dirty Guittar)» و مقدار «ضربه‌های هدایت‌شونده (Driving Beat)» که هر کدام از آن‌ها می‌توانند مقداری در بازه‌ی ۱ تا ۵ داشته باشند. کاربری به ۵ قطعه‌ی موسیقی، امتیازِ «علامت مثبت (انگشت شصت بالا)» می‌دهد که نشان دهنده‌ی این است که او، این آهنگ‌ها را دوست داشته است و در شکل زیر با کاراکترِ «L» مشخص شده است. همچنین این شخص به ۵ قطعه‌ی موسیقی امتیازِ «علامت منفی (انگشت شصت پایین)» می‌دهد به این معنی که این آهنگ‌ها را دوست نداشته است که در شکل زیر با کاراکترِ «D» مشخص شده است.

به شکل زیر نگاه کنید. به نظرِ شما، کاربر، قطعه‌ی موسیقی‌ای را که با علامتِ سوال یعنی «?» نشان داده شده است، دوست دارد یا خیر؟



حدس می‌زنم که بگویید، بله، او این قطعه را دوست خواهد داشت. این گفته بر این اساس است که «?» به «L»ها در شکل نزدیک‌تر است تا «D»ها. ما ادامه‌ای این فصل را به این ایده اختصاص می‌دهیم. واضح‌ترین راه‌حل این است که نزدیک‌ترین همسایه را به «?» پیدا کنیم و پیش‌بینیِ خود را بر این اساس بگذاریم که این قطعه‌ی موسیقی هم، هم طبقه‌ی نزدیک‌ترین قطعه به او است. در شکلِ بالا، نزدیک‌ترین گزینه به «?»، «L» است، پس برای همین حدس می‌زنیم (پیش‌بینی می‌کنیم) که «?»، نیز مقدار «L» را خواهد داشت یعنی کاربر این قطعه‌ی موسیقی را نیز دوست دارد.

کد پایتون برای طبقه‌بندیِ نزدیک‌ترین همسایه

اجازه بدهید از مجموعه‌ی داده‌ای که قبلاً به آن اشاره کردم، استفاده کنم – ۱۰ قطعه‌ی موسیقی که هر کدام ۷ ویژگی دارند (سطرها، قطعات موسیقی هستند و ستون‌ها، ویژگی‌های ما را تشکیل می‌دهند):

	Piano	Vocals	Driving beat	Blues infl.	Dirty elec. Guitar	Backup vocals	Rap infl.
Dr. Dog/ Fate	2.5	4	3.5	3	5	4	1
Phoenix/ Lisztomania	2	5	5	3	2	1	1
Heartless Bastards / Out at Sea	1	5	4	2	4	1	1
Todd Snider/ Don't Tempt Me	4	5	4	4	1	5	1
The Black Keys/ Magic Potion	1	4	5	3.5	5	1	1
Glee Cast/ Jessie's Girl	1	5	3.5	3	4	5	1
Black Eyed Peas/ Rock that Body	2	5	5	1	2	2	4
La Roux/ Bulletproof	5	5	4	2	1	1	1
Mike Posner/ Cooler than me	2.5	4	4	1	1	1	1
Lady Gaga/ Alejandro	1	5	3	2	1	2	1

قبلاً در همین فصل، کدِ پایتون برای داده‌های زیر را توسعه دادیم:

```
music = {"Dr Dog/Fate": {"piano": 2.5, "vocals": 4, "beat": 3.5,
                         "blues": 3, "guitar": 5, "backup vocals": 4,
                         "rap": 1},
         "Phoenix/Lisztomania": {"piano": 2, "vocals": 5, "beat": 5,
                                 "blues": 3, "guitar": 2,
                                 "backup vocals": 1, "rap": 1},
         "Heartless Bastards/Out at Sea": {"piano": 1, "vocals": 5,
                                           "beat": 4, "blues": 2,
                                           "guitar": 4,
                                           "backup vocals": 1,
                                           "rap": 1},
         "Todd Snider/Don't Tempt Me": {"piano": 4, "vocals": 5,
                                        "beat": 4, "blues": 4,
                                        "guitar": 1,
                                        "backup vocals": 5, "rap": 1},
```

اگر کد را نگاه کنید، رشته‌های piano، vocals، beat، blues، guitar، backup vocals و rap چندین بار در سراسر این کد تکرار شده اند. حالا اگر من ۱۰۰ هزار قطعه‌ی موسیقی داشته باشم، این رشته‌ها ۱۰۰ هزار بار تکرار خواهند شد. می‌خواهم با استفاده از بردارها (vectors)، این تکرار را حذف کنم:

```
#
#  the item vector represents the attributes: piano, vocals,
#  beat, blues, guitar, backup vocals, rap
#
items = {"Dr Dog/Fate": [2.5, 4, 3.5, 3, 5, 4, 1],
         "Phoenix/Lisztomania": [2, 5, 5, 3, 2, 1, 1],
         "Heartless Bastards/Out at Sea": [1, 5, 4, 2, 4, 1, 1],
         "Todd Snider/Don't Tempt Me": [4, 5, 4, 4, 1, 5, 1],
         "The Black Keys/Magic Potion": [1, 4, 5, 3.5, 5, 1, 1],
         "Glee Cast/Jessie's Girl": [1, 5, 3.5, 3, 4, 5, 1],
         "La Roux/Bulletproof": [5, 5, 4, 2, 1, 1, 1],
         "Mike Posner": [2.5, 4, 4, 1, 1, 1, 1],
         "Black Eyed Peas/Rock That Body": [2, 5, 5, 1, 2, 2, 4],
         "Lady Gaga/Alejandro": [1, 5, 3, 2, 1, 2, 1]}
```

علاوه بر تبدیل ویژگی‌های یک قطعه‌ی موسیقی به صورت برداری، احتیاج دارم تا علامت‌های «موافقت/مخالفت (انگشت شصت به سمت بالا/انگشت شصت به سمت پایین)» را نیز به صورت برداری تبدیل کنم. به خاطر این که هر کاربر، همه‌ی قطعات موسیقی را امتیازدهی نکرده است یعنی داده‌ها تُنُک (sparse) است، از روشِ دیکشنریِ دیکشنری‌ها (dictionary of dictionaries) استفاده خواهم کرد:

```
users = {"Angelica": {"Dr Dog/Fate": "L", "Phoenix/Lisztomania": "L",
                      "Heartless Bastards/Out at Sea": "D",
                      "Todd Snider/Don't Tempt Me": "D",
                      "The Black Keys/Magic Potion": "D",
                      "Glee Cast/Jessie's Girl": "L",
                      "La Roux/Bulletproof": "D",
                      "Mike Posner": "D",
                      "Black Eyed Peas/Rock That Body": "D",
                      "Lady Gaga/Alejandro": "L"},
         "Bill":  {"Dr Dog/Fate": "L", "Phoenix/Lisztomania": "L",
                   "Heartless Bastards/Out at Sea": "L",
                   "Todd Snider/Don't Tempt Me": "D",
                   "The Black Keys/Magic Potion": "L",
                   "Glee Cast/Jessie's Girl": "D",
                   "La Roux/Bulletproof": "D", "Mike Posner": "D",
                   "Black Eyed Peas/Rock That Body": "D",
                   "Lady Gaga/Alejandro": "D"}             }
```

در کدِ بالا، روشی که من برای مخالفت/موافقت انتخاب کردم (L و D) اختیاری است. شما می‌توانید مثلاً ۰ و ۱ بگذارید.

از آن‌جایی که فرمتِ داده‌ها را تغییر دادیم، بایستی تغییری در تابعی که برای محاسبه‌ی فاصله منهتن نوشتیم، بدهم. هچنین تغییری در تابع computeNearestNeighbor نیز می‌دهم:

```
def manhattan(vector1, vector2):
    """Computes the Manhattan distance."""
    distance = 0
    total = 0
    n = len(vector1)
    for i in range(n):
        distance += abs(vector1[i] - vector2[i])
    return distance

def computeNearestNeighbor(itemName, itemVector, items):
    """creates a sorted list of items based on their dis
tance to item"""
    distances = []
    for otherItem in items:
        if otherItem != itemName:
            distance = manhattan(itemVector, items[other
Item])
            distances.append((distance, otherItem))
    # sort based on distance -- closest first
    distances.sort()
```

```
    return distances
```

سرانجام، می‌خواهم یک تابع برای طبقه بندی (classify) بنویسم. می‌خواهم بتوانم پیش‌بینی کنم که چطور یک کاربرِ خاص، یک قطعه‌ی موسیقی را امتیازدهی می‌کند (یعنی آیا این قطعه‌ی موسیقی را دوست دارد یا خیر). این کار با استفاده از ویژگی‌های آن قطعه‌ی موسیقی انجام می‌شود:

```
"Chris Cagle/ I Breathe In. I Breathe Out"  [1, 5, 2.5, 1, 1, 5, 1]
```

(نکته: برای شکل ظاهریِ بهتر در کدهای پایتون در زیر، از «Cagle» برای مشخص کردن این قطعه‌ی موسیقیِ گفته شده در بالا و خواننده‌ی آن استفاده می‌کنم)
اولین کاری که تابع باید انجام دهد، این است که نزدیک ترین همسایه را برای قطعه‌ی موسیقیِ Chris Cagle پیدا کند. بعد از آن باید ببینم، کاربر، آن قطعه‌ی نزدیک‌ترین همسایه را چگونه امتیازدهی کرده است، و بعد از طریق همین کار پیش‌بینی کنیم که یک کاربر چگونه قطعه‌ی موسیقیِ Chris Cagle را امتیازدهی خواهد کرد. تابعِ زیر، تلاش ابتدایی من برای پیاده‌سازی **طبقه‌بندی (classify)** است:

```
def classify(user, itemName, itemVector):
    """Classify the itemName based on user ratings
    Should really have items and users as parameters"""
    # first find nearest neighbor
    nearest = computeNearestNeighbor(itemName, itemVecto
r, items)[0][1]
    rating = users[user][nearest]
    return rating
```

خب، بیایید این تابع را آزمایش کنیم. تعجب می‌کنم که اگر Angelica، آهنگِ I Breathe In, I Breathe Out از Chris Cage را دوست داشته باشد.

```
classify('Angelica', 'Cagle', [1, 5, 2.5, 1, 1, 5, 1])
"L"
```

الان ما پیش‌بینی کردیم که Angelica این قطعه‌ی موسیقی را دوست دارد! چرا این‌طور شد؟

```
computeNearestNeighbor('Angelica', 'Cagle', [1, 5, 2.5, 1, 1, 5, 1])
```

```
[(4.5, 'Lady Gaga/Alejandro'), (6.0, "Glee Cast/Jessie's Girl"), (7.5,
"Todd Snider/Don't Tempt Me"), (8.0, 'Mike Posner'), (9.5, 'Heartless
Bastards/Out at Sea'), (10.5, 'Black Eyed Peas/Rock That Body'), (10.5,
'Dr Dog/Fate'), (10.5, 'La Roux/Bulletproof'), (10.5, 'Phoenix/
Lisztomania'), (14.0, 'The Black Keys/Magic Potion')]
```

ما پیش‌بینی کردیم که Angelica قطعه‌ی I Breathe In, I Breathe Out از Chris Cagle را دوست دارد به خاطر این‌که، این قطعه‌ی موسیقی به قطعه‌ی Alejandro از Lady Gaga نزدیک بود (در واقع نزدیک‌ترین همسایه به این قطعه‌ی موسیقی بود) و بنابراین Angelica این قطعه‌ی موسیقی را دوست داشته است.

کاری که در این‌جا انجام دادیم، ایجاد یک **طبقه‌بند (classifier)** بود – در این‌جا، وظیفه‌ی ما این بود که یک قطعه‌ی موسیقی را برای یک کاربر طوری طبقه‌بندی کنیم که به یکی از دو گروه (دوست داشتن/دوست نداشتن) تعلق بگیرد.

یک طبقه‌بند (classifier)، برنامه‌ای است که از ویژگی‌های یک شی، برای مشخص کردنِ گروه یا کلاسی که آن شی به آن تعلق دارد، استفاده می‌کند.

یک طبقه‌بند، از مجموعه‌ای از اشیا یعنی همان رکوردها که دارای برچسب (label) هستند استفاده می‌کند. این برچسب‌ها نشان‌دهنده‌ی گروه یا کلاسی هستند که شی، به آن تعلق دارد. طبقه‌بند، از آن مجموعه برای طبقه‌بندیِ اشیای جدید و برچسب‌نخورده استفاده می‌کند. بنابراین در مثال ما، ما می‌دانیم که Angelica چه قطعات موسیقی‌ای را دوست دارد (که با برچسبِ «liked» برچسب خورده‌اند) و همچنین قطعاتی را که دوست ندارد را هم می‌دانیم. حالا می‌خواهیم بدانیم که آیا Angelica، قطعات Chris Cagle را دوست دارد یا نه (چون Angelica برای قطعات Chris Cagle امتیازی ثبت نکرده است یعنی برچسبی نزده است).

بعد از آن بررسی می‌کنیم که آیا Angelica این آهنگ (Alegandro) را دوست دارد یا خیر – او این آهنگ را دوست دارد. پس این پیش‌بینی را می‌کنیم که Angelica آهنگِ I Breathe In, I Breath Out از Chris Cagle را دوست دارد.

طبقه‌بندها کاربردهای فراوانی دارند. در زیر به برخی از آن‌ها اشاره می‌کنیم:

طبقه‌بندی برای تحلیل احساسات توییتری

برخی از افراد بر روی طبقه‌بندیِ احساسات (مثبت یا منفی) بر روی داده‌های توییتر کار می‌کنند. این داده ها می‌توانند به روش‌های گوناگون مورد استفاده قرار بگیرند. برای مثال، اگر شرکت Axe یک اسپری ضد تعریقِ جدید تولید کرده باشد، می‌توانند از این طریق (توییت ها و تحلیل احساسات) متوجه شوند که آیا کاربران از آن خوششان آمده است یا خیر. ویژگی‌ها در این مسئله، کلماتِ هر توییت هستند.

تشخیص خودکار افراد در تصاویر

در حال حاضر نرم افزارهایی وجود دارند که می‌توانند دوستانِ شما را در تصاویر شناسایی کرده و برچسب‌زنی کنند. (و همین تکنولوژی برای شناسایی افراد در هنگام پیاده‌روی در خیابان توسط دوربین‌های عمومی مورد استفاده قرار می‌گیرند). روش‌های متفاوتی برای

این کار وجود دارد ولی برخی از آن‌ها، از موقعیت نسبی و اندازه‌ی چشم‌ها، بینی، فک و غیره در افراد مختلف برای این‌کار استفاده می‌کنند.

طبقه‌بندی برای تبلیغات سیاسیِ هدف‌مند

به این روش، میکروهدفمندی (microtargeting) می‌گویند. افراد در دسته بندی‌های مختلفی قرار می‌گیرند. این دسته‌بندی‌ها می‌توانند به این صورت باشند: « Barn Raisers»، «Inner Compass»، «Hearth Keepers» (فرض کنید این‌ها هر کدام یک گروهی از افراد در یک جامعه باشند، مثلاً «اصلاح‌طلب»، «اصول‌گرا» و «اعتدال‌گرا»). حالا مثلا گروهِ «Hearth Keepers»، تمرکزشان بر روی اعضای خانواده و حفظ کردن آن‌ها است (پس مثلاً برای این گروه از افراد، تبلیغِ مختص به خودشان انجام می‌شود)

سلامتی و حفاظت از خود

الان شروعی برای رسیدن به خود است. در این زمانه، می‌توانیم وسائل ساده ای مثل فیت‌بیت (FitBit) و دست‌بندهای Nike را بخریم (این دست‌بندها اطلاعاتی از وضعیت سلامت ما از طریق حسگرهایی به دست می‌آورند). شرکت اینتل و شرکت‌های دیگر در حال کار بر روی خانه‌های هوشمند هستند که در کفِ آن‌ها وسائلی تعبیه شده است که می‌توانند ما را وزن کنند، رفت و آمد ما را زیر نظر بگیرند و در صورت رفتار غیرعادی، به شخص دیگری هشدار دهند (برای مثال کسی حالش بد می‌شود و کف زمین می‌افتد). خبرگانِ این حوزه پیش‌بینی می‌کنند که تا چند سالِ آینده ما کامپیوترهای کوچکی را خواهیم پوشید که می‌توانند هزاران فاکتورِ مختلف از بدن ما را در لحظه، نظارت کنند و سریعاً عملیات طبقه‌بندی را انجام دهند.

تبلیغات هدف‌مند

تبلیغات هدف‌مند چیزی شبیه به میکروهدف‌مندی در سیاست است که بالاتر در مورد آن صحبت کردیم. به جای ایجادِ کمپین‌های تبلیغاتیِ گسترده برای فروشِ ساختمان مجلل مسکونی در شهر وگاس که مالک آن هستم، آیا می‌توانم مشتریانِ بالقوه را شناسایی کرده و تبلیغات را مستقیماً و فقط به آن ها نشان دهم؟ یا حتی بهتر، آیا

> می‌توانیم زیرگروهی از مشتریان بالقوه را شناسایی کنم و تبلیغاتم را مخصوص آن‌ها طراحی کنم؟

این لیست بی‌انتهاست...

- تقسیم کردن افراد به تروریست و غیرتروریست
- طبقه‌بندیِ خودکار ایمیل (ایمیل‌های مهم، ایمیل‌های عادی، هرزنامه یا همان اسپم‌ها)
- پیش‌بینیِ خروجیِ پزشکان
- مشخص کردنِ کلاه‌برداری‌های مالی (برای مثال، دزدی‌های کارت‌های اعتباری)

مسئله‌ی «چه ورزشی؟»

برای این که یک دید کلی در مورد کارهایی که قرار است در چند فصل بعد با هم انجام دهیم را داشته باشید، یک مثال ساده‌تر را با هم مرور می‌کنیم – طبقه‌بندیِ این‌که زنانِ ورزشکار چه ورزشی را انجام می‌دهد، این طبقه بندی تنها دو ویژگی دارد: **وزن (weight)** و **قد (height)**. در جدول زیر یک مجموعه‌ی داده‌ی کوچک را که از وب‌سایت‌های مختلف جمع‌آوری کردم، مشاهده می‌کنید:

(ستون Name، نام ورزشکار است، ستون Sport، ورزشی است که این ورزشکار انجام می‌دهد، ستون Age سن، ستون Height قد و ستون Weight وزن ورزشکار را نشان می‌دهد، همچنین در ستون Sport، مقادیر مختلف ورزش‌ها به این صورت درج شده است: **Gymnastics** ورزش ژیمناستیک، **Basketball** ورزش بسکتبال و **Track** ورزش دو و میدانی یا ماراتن هستند)

Name	Sport	Age	Height	Weight
Asuka Teramoto	Gymnastics	16	54	66
Brittainey Raven	Basketball	22	72	162
Chen Nan	Basketball	30	78	204
Gabby Douglas	Gymnastics	16	49	90
Helalia Johannes	Track	32	65	99
Irina Miketenko	Track	40	63	106
Jennifer Lacy	Basketball	27	75	175
Kara Goucher	Track	34	67	123
Linlin Deng	Gymnastics	16	54	68
Nakia Sanford	Basketball	34	76	200
Nikki Blue	Basketball	26	68	163
Qiushuang Huang	Gymnastics	20	61	95
Rebecca Tunney	Gymnastics	16	58	77
Rene Kalmer	Track	32	70	108
Shanna Crossley	Basketball	26	70	155
Shavonte Zellous	Basketball	24	70	155
Tatyana Petrova	Track	29	63	108
Tiki Gelana	Track	25	65	106
Valeria Straneo	Track	36	66	97
Viktoria Komova	Gymnastics	17	61	76

در این مجموعه، داده‌های ژیمیناستیکِ برخی از شرکت‌کنندگان المپیک‌های ۲۰۱۲ و ۲۰۰۸ را داریم. بازیکنان بسکتبال، برای تیم‌هایی در لیگ WNBA (NBA زنان) بازی می‌کردند. و همچنین زنانِ دونده که توانسته بودند دوی ماراتن در المپیک ۲۰۱۲ را به پایان برسانند. درست است که این مثال، یک مثال بدیهی است، ولی می‌تواند به ما کمک کند تا روش‌هایی را که آموخته‌ایم، بر روی این مجموعه‌ی داده به کار بگیریم.

همان‌طور که می‌بینید، من، ستونِ **سن (age)** را هم در جدول اضافه کردم. با یک نگاهِ ساده به مجموعه‌ی داده‌ی بالا، مشاهده می‌کنید که سن (age) به تنهایی می‌تواند یک فاکتورِ پیش‌بینیِ نسبتاً خوب باشد. سعی کنید رشته‌ی ورزشیِ ورزشکاران زیر را محاسبه کنید:

Candace Parker; Age 26

McKayla Maroney; Age 16

Lisa Jane Weightman; Age 34

Olivera Jevtić; Age 35

پاسخ‌ها

خانم Candace Parker، برای تیم‌های لس‌آنجلس اسپارک در WNBA و برای تیم Ekaterinburg در لیگ UMMC در روسیه، بسکتبال بازی می‌کند. خانم McKayla

Maroney عضو تیم ژیمیناستیک آمریکا بود و یک مدال طلا و یک نقره برنده شده بود. خانم Olivera Kevtic یک دونده‌ی استقامتِ صربستانی است که در رقابت‌های المپیک سال ۲۰۰۸ و ۲۰۱۲ شرکت کرده بود. خانم Lisa Jane Weightman نیز یک دونده‌ی استقامتِ استرالیایی است که او هم در سال ۲۰۰۸ و ۲۰۱۲ در المپیک شرکت کرده بود.
اگر تمرین بالا را انجام داده باشید، الان شما یک طبقه‌بندی (classification) را انجام داده‌اید – در واقع شما کلاسِ اشیا بر اساس ویژگی‌های آن‌ها پیش‌بینی کرده‌اید (در این مثال، پیش‌بینیِ رشته‌ی ورزشی زنان بر اساس ویژگیِ سن (age))

ورزش ذهنی

فرض کنید، من می‌خواهم حدس بزن که بر اساس قد و وزن، یک شخص چه رشته‌ی ورزشی را انجام می‌دهد. مجموعه‌ی داده‌ای که در اختیار دارم کوچک است و فقط دو نفر در آن هستند. خانم Nakia Sanford بازیکن WNBA در تیم Phoenix Mercury است که ۶ پا و ۴ اینچ (معادل تقریباً ۱۹۲ سانتی‌متر) قد و ۲۰۰ پوند (تقریباً معادل ۹۰ کیلوگرم) وزن دارد. خانم Sarah Beale، بازیکنِ خط حمله‌ی راگبی تیم ملی انگلستان است که ۵ پا و ۱۰ اینچ (معادل تقریباً ۱۷۷ سانتی‌متر) قد و ۱۹۰ پوند (تقریباً معادل ۸۶ کیلوگرم) وزن دارد. بر اساس این مجموعه‌ی داده‌ی خیلی کوچک، می‌خواهم خانم Catherine Spencer را طبقه‌بندی کنم. به این معنا که ببینم این خانم، بازیکنِ بسکتبال هست یا بازیکن راگبی؟ او ۵ پا و ۱۰ اینچ (تقریباً معادل ۱۷۷ سانتی‌متر) قد و ۲۰۰ پوند (تقریباً معادل ۹۰ کیلوگرم) وزن دارد. به نظر شما این شخص چه ورزشی را انجام می‌دهد؟

اگر بگویید بازیکنِ راگبی است، پاسخ درستی داده‌اید. خانم Catherine Spencer مهاجم راگبی در تیم ملی انگلستان است. با این‌حال، اگر حدسمان را بر اساس معیار فاصله‌ای مانند معیار فاصله‌ی منهتن (Manhattan distance) بگذاریم، پاسخ اشتباه است. معیار فاصله‌ی منهتن بین Catherine Spencer و خانم Nakia (که بازیکن بسکتبال بود) ۶ است. این در حالی است که فاصله‌ی منهتن بین خانم Catherine و Sarah (که بازیکن راگبی بود) برابر ۱۰ است. پس اگر بخواهیم نزدیک‌ترین فرد را بر اساس این معیار فاصله

انتخاب کنیم، بایستی Nakia را انتخاب کرده و بگوییم که Catherine هم مانند او، یک بازیکن بسکتبال است.
آیا از مثالِ بالا، چیزی آموختیم که بتواند به ما در طبقه‌بندیِ دقیق‌تر کمک کند؟

بله، ما می‌توانیم از امتیازِ استانداردِ اصلاح‌شده (modified standard score) استفاده کنیم:

$$\frac{\text{(each value) - (median)}}{\text{(absolute standard deviation)}}$$

داده‌های آزمون (Test Data)

بیایید ویژگیِ سن (Age) را از میان داده‌هایمان حذف کنیم. در جدول زیر افرادی را می‌بینید که می‌خواهم آن‌ها را طبقه‌بندی کنم:

Name	Sport	Height	Weight
Crystal Langhorne		74	190
Li Shanshan		64	101
Kerri Strug		57	87
Jaycie Phelps		60	97
Kelly Miller		70	140
Zhu Xiaolin		67	123
Lindsay Whalen		69	169
Koko Tsurumi		55	75
Paula Radcliffe		68	120
Erin Thorn		69	144

کد پایتون

این‌بار به جای‌اینکه داده‌ها را مستقیماً در پایتون هاردکُد (hard-coding) کنم، تصمیم گرفتم که آن‌ها را در دو فایل مجزا قرار دهم. فایل‌ها به این صورت هستند:
athletesTrainingSet.txt و athletesTestSet.txt

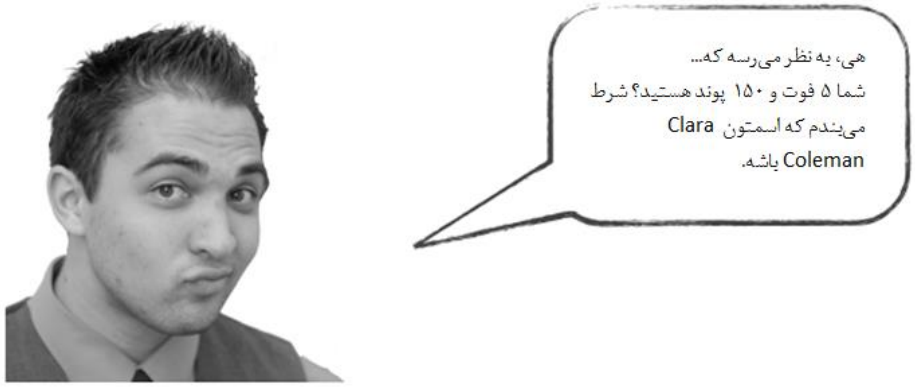
من، از داده‌های فایلِ athletesTrainingSet.txt برای آموزشِ طبقه‌بند استفاده می‌کنم. داده‌های فایلِ athletesTestSet.txt برای آزمودنِ کیفیت طبقه‌بندی که ساخته‌ایم مورد استفاده قرار می‌گیرند. به بیانی دیگر، هر کدام از سطرهای موجود در مجموعه‌ی آزمون (test set) بر اساس طبقه‌بندی که از روی داده‌های آموزش (training set) ساخته‌شده است، به صورت خودکار طبقه‌بندی می‌شوند.

فرمتِ این فایل‌ها به صورت زیر است:

Asuka Teramoto	Gymnastics	54	66
Brittainey Raven	Basketball	72	162
Chen Nan	Basketball	78	204
Gabby Douglas	Gymnastics	49	90

حتماً می‌دانید که فایل‌های داده و کدهای پایتون در سایت guidetodatamining.com و وب‌سایت chistio.ir موجود است.

در این فایل، هر خط یک شی (یک ورزشکار) است که ویژگی‌های آن‌ها با tab از یکدیگر جدا شده‌اند. از طبقه‌بندِ خودم می‌خواهم تا با استفاده از قد و وزنِ یک فرد بتواند پیش‌بینی کند که یک شخص در چه رشته‌ای ورزش می‌کند. پس **دو ستونِ آخر**، ویژگی‌های عددی هستند که من در فرآیندِ طبقه‌بندی از آن‌ها استفاده می‌کنم و ستونِ دوم، همان ستونی هست که کلاسِ یک شی (رشته‌ی ورزشی که آن فرد انجام می‌دهد) را نشان می‌دهد. نامِ ورزش‌کار توسط طبقه‌بند استفاده نمی‌شود. البته من امتحان نکردم که ببینم آیا نامِ یک فرد می‌تواند به پیش‌بینیِ ورزشیِ که آن فرد انجام می‌دهد کمکی کند یا خیر، و که این کار را هم نخواهم کرد.



با این حال، نام می‌تواند به عنوان وسیله‌ای برای توضیح دادنِ تصمیماتِ طبقه‌بندی مفید باشد. مثلاً ما فکر می‌کنیم Amelia Pond یک ژیمیناستیک‌کار است به خاطر این‌که با توجه به قد و وزنش به Gabby Douglas که او هم یک ژیمیناستیک‌کار است، نزدیک‌تر است.»

همان‌طور که گفتم، کد پایتون را طوری می‌نویسم که به یک نمونه‌ی خاص محدود نباشد (برای مثال، فقط به مجموعه‌ی داده‌ی بالا محدود نباشد). برای رسیدن به این هدف، یک سطر به عنوان سرآیند (header) به این مجموعه‌ی داده اضافه می‌کنم. این سطرِ اولیه عملکردِ هر ستون را مشخص می‌کند. در زیر چند سطرِ اول را می‌آورم:

comment	class	num	num
Asuka Teramoto	Gymnastics	54	66
Brittainey Raven	Basketball	72	162

هر ستونی که comment (توضیحات) باشد، به وسیله‌ی طبقه‌بندی که ما می‌نویسیم نادیده گرفته می‌شود. ستونِ class (کلاس یا طبقه) بیان‌گر طبقه‌ی آن شی است و ستونی که با num (عدد) مشخص شده باشد، ویژگی‌های آن شی یا رکورد است.

ورزش ذهنی

فکر می‌کنید، چگونه این داده‌ها را در پایتون نمایش می‌دهیم؟ چند نمونه را در زیر می‌آورم (شما هم می‌توانید نمونه‌ی خود را داشته باشید).

فرمِ دیکشنری:

```
{'Asuka Termoto': ('Gymnastics', [54, 66]),
 'Brittainey Raven': ('Basketball', [72, 162]), ...
```

فرمِ لیستی از لیست‌ها:

```
[['Asuka Termoto', 'Gymnastics', 54, 66],
 ['Brittainey Raven', 'Basketball', 72, 162], ...
```

فرمِ لیستی از تاپل‌ها (tuples):

```
[('Gymnastics', [54, 66], ['Asuka Termoto']),
 ('Basketball', [72, 162], ['Brittainey Raven'],...
```

ورزش ذهنی – راه حل

فرم دیکشنری:

```
{'Asuka Termoto': ('Gymnastics', [54, 66]),
 'Brittainey Raven': ('Basketball', [72, 162]), ...
```

این فرم خیلی خوب نیست. کلیدِ هر دیکشنری، نام ورزشکار است که اصلا در محاسبات از آن استفاده نمی‌شود.

فرمِ لیستی از لیست‌ها:

```
[['Asuka Termoto', 'Gymnastics', 54, 66],
 ['Brittainey Raven', 'Basketball', 72, 162], ...
```

این فرم بد نیست. این فرم در واقع برگردانِ فایل‌های ورودی است و از آن‌جایی که الگوریتم نزدیک‌ترین همسایه می‌خواهد که بر روی داده‌ها تکرار (iterate) داشته باشد، این فرمِ لیست، به درد می‌خورد.

فرم لیستی از تاپل‌ها (tuples):

```
[('Gymnastics', [54, 66], ['Asuka Termoto']),
 ('Basketball', [72, 162], ['Brittainey Raven'],...
```

من این فرم را بیشتر از بقیه دوست دارم، چون این طرزِ نمایش، هر کدام از ویژگی‌ها را در یک لیستِ مخصوص به خودش نگه می‌دارد و یک سطح جداگانه‌ای بین کلاس‌ها، ویژگی‌ها و توضیحات را برای ما انجام می‌دهد. من توضیحات (comment) را هم لیست کردم چون تعداد ستون‌های توضیحات می‌توانند چند تا باشد.

کد پایتون من که قرار است فایل را به فرمت زیر تبدیل کند:

```
[('Gymnastics', [54, 66], ['Asuka Termoto']),
 ('Basketball', [72, 162], ['Brittainey Raven'],...
```

یک همچین شکلی دارد:

```
class Classifier:
    def __init__(self, filename):
        self.medianAndDeviation = []
        # reading the data in from the file
        f = open(filename)
        lines = f.readlines()
        f.close()
        self.format = lines[0].strip().split('\t')
        self.data = []
        for line in lines[1:]:
            fields = line.strip().split('\t')
            ignore = []
            vector = []
            for i in range(len(fields)):
                if self.format[i] == 'num':
                    vector.append(int(fields[i]))
                elif self.format[i] == 'comment':
                    ignore.append(fields[i])
                elif self.format[i] == 'class':
                    classification = fields[i]
            self.data.append((classification, vector, ig
nore))
```

کد بزنید:

قبل از این‌که داده‌ها را توسط امتیازِ استانداردِ اصلاح‌شده، استانداردسازی کنیم، تابعی احتیاج داریم که بتواند میانه (median) و انحرافِ استاندارد مطلق (asd) را برای اعداد در لیست محاسبه کند:

```
>>> heights = [54, 72, 78, 49, 65, 63, 75, 67, 54]
>>> median = classifier.getMedian(heights)
>>> median
65
>>> asd = classifier.getAbsoluteStandardDeviation(heights, median)
>>> asd
8.0
```

آیا می‌توانید این توابع را در پایتون پیاده‌سازی کنید؟
فایلِ testMedianAndASD.py را از سایت guidetodatamining.com و یا chistio.ir دانلود کنید.

خطاهای Assertion و تابع Assert در پایتون

مهم این است که هر قسمت از راه‌حل به دو تکه کد تبدیل شود. یک تکه کُد برای پیاده‌سازیِ آن قسمت و یک تکه کُد برای آزمودن (test). در واقع، یک روش این است که ابتدا آزمون (test) را بنویسید و بعد از آن پیاده‌سازی را داشته باشید (به این کار در اصطلاح Test First Programming می‌گویند). قالبی که من برای این‌کار طراحی کردم یک تابع به اسم unitTest دارد. یک ورژن ساده از این تابع که فقط یک آزمون (test) دارد در زیر نمایش داده شده است:

```
def unitTest():
    list1 = [54, 72, 78, 49, 65, 63, 75, 67, 54]
    classifier = Classifier('athletesTrainingSet.txt')
    m1 = classifier.getMedian(list1)
    assert(round(m1, 3) == 65)
    print("getMedian and getAbsoluteStandardDeviation wo
rk correctly")
```

تابع getMedian بایستی چیزی شبیه به کد زیر باشد:

```
def getMedian(self, alist):
    """return median of alist"""
    """TO BE DONE"""
    return 0
```

پس ابتدا تابعِ getMedian، عدد ۰ را برای هر لیستی برمی‌گرداند. شما بایستی getMedian را طوری کامل کنید که مقدارِ درست را برگرداند. در تابع unitTest، من تابعِ getMedian را با لیست زیر صدا می‌زنم:

```
[54, 72, 78, 49, 65, 63, 75, 67, 54]
```

خطِ assert در تابع unitTest می‌گوید، مقداری که getMedian برمی‌گرداند بایستی برابر ۶۵ باشد. اگر این‌طور باشد، فرآیندِ اجرا به صورت زیر ادامه پیدا می‌کند و عبارتِ:

```
getMedian and getAbsoluteStandardDeviation work correctly
```

چاپ می‌شود. اگر چیزی که برمی‌گرداند برابر ۶۵ نباشد، خطِ زیر به عنوان خطا چاپ می‌شود:

```
File "testMedianAndASD.py", line 78, in unitTest
    assert(round(m1, 3) == 65)
AssertionError
```

اگر کد را از سایت دانلود کرده باشید و بدون تغییر اجرا کنید، این خطا را خواهید دید. اما هنگامی که تابع getMedian و تابع getAbsoluteStandardDeviation را به درستی پیاده‌سازی کنید، این خطا از بین خواهد کرد. (در واقع ما تست را نوشتیم و شما بایستی کد را بنویسید)

«این نکته مهم است که هر قسمتی از جزئیات که می‌خواهید بنویسید، به دو قسمت تبدیل شود. یک قسمت پیاده‌سازی و یک قسمت هم برای آزمودنِ آن. اگر آزمونی مانند این را ننویسید، نمی‌توانید بفهمید که چه موقع آن را به اتمام رسانده‌اید، نمی‌دانید که آیا به درستی موضوع را درک کرده‌اید یا خیر، و نمی‌دانید که تغییراتِ آتی می‌تواند منجر به خراب شدنِ این قسمت از کد شود.» پیتر نورویگ (Peter Norvig)

راه‌حل

یکی از راه‌های پیاده‌سازی این الگوریتم به صورت زیر است:

```
def getMedian(self, alist):
    """return median of alist"""
    if alist == []:
        return []
    blist = sorted(alist)
    length = len(alist)
    if length % 2 == 1:
        # length of list is odd so return middle element
        return blist[int(((length + 1) / 2) - 1)]
    else:
        # length of list is even so compute midpoint
        v1 = blist[int(length / 2)]
        v2 =blist[(int(length / 2) - 1)]
        return (v1 + v2) / 2.0

def getAbsoluteStandardDeviation(self, alist, median):
    """given alist and median return absolute standard d
eviation"""
    sum = 0
    for item in alist:
        sum += abs(item - median)
```

```
    return sum / len(alist)
```

همان‌طور که می‌بینید، تابعِ getMedian در ابتدا و قبل از پیدا کردنِ میانه، لیستِ اعداد را مرتب می‌کند. به خاطرِ این‌که من با مجموعه‌ی داده‌های خیلی بزرگی سر و کار ندارم، فکر می‌کنم این راه‌حلِ مناسبی باشد. اگر بخواهم کُد را بهینه کنم، ممکن است این قسمت را با یک الگوریتمِ انتخاب (selection algorithm) جایگزین کنم.

الان، داده‌ها از فایلِ athletesTrainingSet.txt خوانده می‌شوند و در لیستی به نام data در طبقه‌بند نگه‌داشته می‌شوند. فرمتِ آن چیزی شبیه به کد زیر است:

```
[('Gymnastics', [54, 66], ['Asuka Teramoto']),
 ('Basketball', [72, 162], ['Brittainey Raven']),
 ('Basketball', [78, 204], ['Chen Nan']),
 ('Gymnastics', [49, 90], ['Gabby Douglas']), ...
```

حالا می‌خواهم بُردار را نرمال‌سازی کنم تا لیستی که در data در طبقه‌بند قرار دارد، داده‌های نرمال‌شده را در خود داشته باشند. برای مثال:

```
[('Gymnastics', [-1.93277, -1.21842], ['Asuka Teramoto']),
 ('Basketball', [1.09243, 1.63447], ['Brittainey Raven']),
 ('Basketball', [2.10084, 2.88261], ['Chen Nan']),
 ('Gymnastics', [-2.77311, -0.50520], ['Gabby Douglas']),
 ('Track', [-0.08403, -0.23774], ['Helalia Johannes']),
 ('Track', [-0.42017, -0.02972], ['Irina Miketenko']),
```

برای این کار، چند خطِ زیر را به تابعی که اول نوشتم اضافه می‌کنم:

```
# get length of instance vector
self.vlen = len(self.data[0][1])
# now normalize the data
for i in range(self.vlen):
    self.normalizeColumn(i)
```

در حلقه‌ی for می‌خواهیم داده‌ها را ستون به ستون نرمال‌سازی کنیم. پس برای اولین بار در حلقه، ستونِ **قد (height)** را نرمال‌سازی خواهیم کرد، بعد از آن نوبت به **ستونِ وزن (weight)** می‌رسد.

کد بزنید:
آیا می‌توانید تابعِ normalizeColumn را بنویسید؟
قالبِ نمونه را می‌توانید با نام normalizedColumnTemplate.py از سایت guidetodtamining.com و یا chistio.ir برای نوشتن و آزمون، دانلود کنید.

راه‌حل

این‌جا برایتان تابع normalizeColumn را پیاده‌سازی کردم:

```
def normalizeColumn(self, columnNumber):
    """given a column number, normalize that column in s
elf.data"""
    # first extract values to list
    col = [v[1][columnNumber] for v in self.data]
    median = self.getMedian(col)
    asd = self.getAbsoluteStandardDeviation(col, median)
    #print("Median: %f ASD = %f" % (median, asd))
    self.medianAndDeviation.append((median, asd))
    for v in self.data:
        v[1][columnNumber] = (v[1][columnNumber] - media
n) / asd
```

همان‌طور که می‌بینید، مقادیر میانه (median) و انحراف استانداردِ مطلق (asd) را برای هر ستون در لیست medianAndDeviation قرار دادم. از این اطلاعات هنگامی استفاده خواهم کرد که بخواهم طبقه‌بند، کلاسِ جدید را برای نمونه‌های تازه‌تر، پیش‌بینی کند. برای مثال، می‌خواهم پیش‌بینی کنم که خانم Kelly Miller چه رشته‌ی ورزشی را بازی می‌کند؟ این خانم، **۵ فوت و ۱۰ اینچ قد** و **۱۷۰ پوند وزن** دارد. اولین قدم تبدیلِ قد و وزنِ او به صورت امتیازهای استانداردِ اصلاح‌شده است. بردار ویژگیِ اصلی برای او برابر [70, 140] است.

بعد از انجام عملیات بر روی داده‌های آموزشی، مقادیر موجود در meanAndDeviation به صورت زیر درمی‌آید:

```
[(65.5, 5.95), (107.0, 33.65)]
```

به این معنی که داده‌ها در ستونِ اولِ بردار، میانه‌ای برابر ۶۵,۵ و انحراف استانداردِ مطلق برابر ۵,۹۵ دارند. میانه برای ستونِ دوم برابر ۱۰۷ و انحراف استانداردِ مطلق برای این ستون برابر ۳۳,۶۵ است.
من از این اطلاعات برای تبدیل بردارِ اصلی، به برداری که امتیاز استانداردِ اصلاح‌شده را در خود داشته باشد، استفاده می‌کنم. برای مثال، برای ویژگیِ اول، این عدد به صورت زیر اصلاح می‌شود:

$$mss = \frac{x_i - \tilde{x}}{asd} = \frac{70 - 65.5}{5.95} = \frac{4.5}{5.95} = 0.7563$$

و برای ویژگیِ دوم:

$$mss = \frac{x_i - \tilde{x}}{asd} = \frac{140 - 107}{33.65} = \frac{33}{33.65} = 0.98068$$

تابعِ پایتونی که این کار را انجام دهد نیز، مانند شکل زیر است:

```
def normalizeVector(self, v):
    """We have stored the median and asd for each column
.
    We now use them to normalize vector v"""
    vector = list(v)
    for i in range(len(vector)):
        (median, asd) = self.medianAndDeviation[i]
        vector[i] = (vector[i] - median) / asd
    return vector
```

آخرین تکه کدی که باید برای تکمیل این قسمت زده شود، کُدِ مربوط به پیش‌بینیِ نمونه‌های جدید است – در مثالِ ما، ورزشی که یک ورزشکار انجام می‌دهد. برای این‌که مشخص شود که شخصی مانند Kelly Miller که ۵ فوت و ۱۰ اینچ (یعنی در مجموع ۷۰ اینچ) قد و ۱۷۰ پوند وزن دارد، چه ورزشی را انجام می‌دهد، بایستی تابع زیر را صدا بزنیم:

```
classifier.classify([70, 170])
```

در کدِ من، تابعِ classify فقط یک ساختارِ کلی، برای انجام عملیاتِ نزدیک‌ترین همسایه با تابعِ nearestNeighbor هست:

```
def classify(self, itemVector):
    """Return class we think item Vector is in"""
    return(self.nearestNeighbor(self.normalizeVector(ite
mVector))[1][0])
```

کد بزنید

آیا می‌توانید تابعِ nearestNeighbor را پیاده‌سازی کنید؟ (من، یک تابعِ جدا به اسم manhattanDistance نوشتم)

دوباره، قالبِ مربوط به این مسئله را می‌توانید با اسم classifyTemplate.py از سایت guidetodatamining.com و chistio.ir دریافت کنید.

راه حل

پیاده‌سازیِ تابعِ nearestNeighbor کوتاه است:

```
def manhattan(self, vector1, vector2):
    """Computes the Manhattan distance."""
    return sum(map(lambda v1, v2: abs(v1 - v2), vector1,
 vector2))

def nearestNeighbor(self, itemVector):
    """return nearest neighbor to itemVector"""
    return min([ (self.manhattan(itemVector, item[1]), i
tem)
```

```
                for item in self.data])
```

آقای فیشر (Fisher) یک فرد به یاد ماندنی بود. او انقلابی در علمِ آمار به وجود آورد و ریچارد دایکینز (Richard Dawkins) او را «بزرگ‌ترین زیست‌شناس از زمان داروین» نامیده است.

تمام شد!!!

ما تابع طبقه‌بندیِ نزدیک‌ترین همسایه را با پایتون در حدود ۲۰۰ خط کد نوشتیم.

در کُدِ کامل، که می‌توانید از وب‌سایت ما دانلود کنید، یک تابع به نام test قرار دادم، که به عنوانِ ورودی، یک فایل برای مجموعه‌ی آموزشی و یک فایل برای مجموعه‌ی آزمون از شما می‌گیرد و دقتِ طبقه‌بند را در خروجی چاپ می‌کند. در شکل زیر مشاهده می‌کنید تابع طبقه‌بند، چگونه داده‌های ورزشکارانِ زن را طبقه‌بندی کرده است:

```
>>> test("athletesTrainingSet.txt", "athletesTestSet.txt")
-         Track  Aly Raisman        Gymnastics  62    115
+    Basketball  Crystal Langhorne  Basketball  74    190
+    Basketball  Diana Taurasi      Basketball  72    163
<snip>
-         Track  Hannah Whelan      Gymnastics  63    117
+    Gymnastics  Jaycie Phelps      Gymnastics  60    97
80.00% correct
```

همان‌طور که می‌بینید، طبقه‌بندِ ما ۸۰ درصد دقت دارد. این طبقه‌بند، در مورد بازیکنانِ بسکتبال بسیار عالی عمل کرده‌است ولی چهار خطا در دو و میدانی و ژیمیناستیک دارد.

مجموعه داده‌ی گل‌های زنبق (Iris Dataset)

طبقه‌بندِ ساده‌ای که نوشیم را بر روی مجموعه‌ی داده‌ی گل‌های زنبق (Iris) که بدون شک مشهورترین مجموعه‌ی داده‌ی موجود در داده‌کاوی است، آزمایش کردم. این مجموعه‌ی داده به وسیله‌ی Sir Ronald Fisher در ۱۹۳۰ استفاده می‌شد. مجموعه‌ی داده‌ی گل‌های زنبق متشکل از ۵۰ نمونه‌ی گل برای هر نوع گلِ زنبق است (زنبقِ ستوتا (Iris Setos)، زنبقِ ویرجینیکا (Iris Virginica) و زنبقِ ورسیکالر (Iris Versicolor)). این مجموعه‌ی داده از اندازه‌های دو بخشِ گل زنبق تشکیل شده است: **کاسبرگ (sepal)** یعنی قسمت سبز رنگِ غنچه‌ی گل و **گلبرگ‌ها (petals)**.

تمامیِ مجموعه‌داده‌هایی که در این کتاب به آن‌ها اشاره شده‌است در سایتِ کتاب، یعنی guidetodatamining.com و همچنین وب‌سایت chistio.ir موجود است. این کار به شما این امکان را می‌دهد تا داده‌ها را دانلود کرده و با الگوریتم‌ها، تجربه کسب کنید. مثلاً این‌که بفهمید، آیا نرمال‌سازیِ داده‌ها دقت را افزایش می‌دهد یا برعکس باعث کاهشِ دقت می‌شود؟ آیا داده‌های بیشتر در مجموعه‌ی داده، نتایج را بهبود می‌بخشد یا خیر؟ تغییر معیار به فاصله‌ی اقلیدسی، چه تاثیراتی دارد؟

یادتان باشد که: هر چیزی که یاد بگیرید، در ذهنِ شماست، نه من (و نه منِ مترجم!). هر چقدر بیشتر با کدها و داده‌های اشاره شده در این کتاب تعامل داشته باشید و کار کنید، بیشتر یاد خواهید گرفت.

مجموعه‌ی داده‌ی گل‌های زنبق (Iris) چیزی شبیه به شکل زیر است (گونه‌های مختلفِ گلِ زنبق، چیزی است که طبقه‌بند می‌خواهد پیش‌بینی کند)

Sepal length	Sepal width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	I.setosa
4.9	3.0	1.4	0.2	I setosa

۱۲۰ نمونه گلِ زنبق همراه با برچسب در مجموعه‌ی داده‌های آموزشی (training set) قرار دارد و ۳۰ نمونه هم همراه با برچسب در مجموعه‌ی آزمون (test set) قرار گرفته‌اند (هیچ‌کدام از داده‌های مجموعه‌ی آزمون در مجموعه‌ی آموزشی نیستند).
اجازه دهید ببینیم طبقه‌بندِ ما (که در قسمتِ قبلی ساختیم) چگونه و با چه کیفیتی داده‌های گل‌های زنبق را طبقه‌بندی می‌کند؟

```
>>> test('irisTrainingSet.data', 'irisTestSet.data')

93.33% correct
```

دوباره یک نتیجه‌ی خوب با همان طبقه‌بندِ ساده‌ای که داریم به دست آمده است. حال این نکته جالب است که بدون نرمال‌سازیِ داده‌ها، طبقه‌بند دارای دقتِ ۱۰۰ درصدی شده است (ولی با نرمال‌سازی مانند شکلِ بالا ۹۳,۳۳ درصد دقت داشته‌ایم). این مشکلی که در اثرِ نرمال‌سازی پیش‌می‌آید را در فصلِ بعدی با جزئیاتِ بیشتری بررسی می‌کنیم.

مسئله‌ی مایل به ازای هر گالن

در آخر، طبقه‌بندمان را بر روی یک مجموعه داده‌ی اصلاح‌شده، که به صورت گسترده هم استفاده می‌شود آزمایش کردیم. این مجموعه‌ی داده، **مایل به ازای هر گالن در اتومبیل‌ها (Auto Miles Per Gallon)** از دانشگاه کارنی ملون (Carnegie Melon) است که در ابتدا در سال ۱۹۸۳ در نمایشگاهِ انجمنِ آمارِ آمریکا استفاده شد. فرمتِ داده‌ها برای این مجموعه‌ی داده، چیزی مانندِ شکل زیر است:

mpg	cylinders	c.i.	HP	weight	secs. 0-60	make/model
30	4	68	49	1867	19.5	fiat 128
45	4	90	48	2085	21.7	vw rabbit (diesel)
20	8	307	130	3504	12	chevrolet chevelle malibu

در نسخه‌ی اصلاح‌شده‌ی این مجموعه‌ی داده، می‌خواهیم ستونِ mpg را که یک مجموعه‌ی گسسته است، پیش‌بینی کنیم (مقادیرِ این ستون ۱۰، ۱۵، ۲۰، ۲۵، ۳۰، ۳۵ و ۴۵ است). ویژگی‌های مورد استفاده، **تعداد سیلندر (cylanders)**، **ظرفیت پیستون (displacement) که در ستونِ جدولِ بالا با c.i. مشخص شده است**، **اسب بخار (horsepower) یا همان HP در جدولِ بالا**، **وزن (weight)** و **شتاب (acceleration) یعنی زمان به ثانیه بین ۰ تا ۶۰ کیلومتر (secs. 0-60)** است.

تعداد ۳۴۲ نمونه ماشین، در مجموعه‌ی آموزشی (training set) موجود هستند و تعداد ۵۰ نمونه، در مجموعه‌ی آزمون (test set). اگر به صورت تصادفی، مقدارِ mpg (مایل به ازای هر گالن) را پیش‌بینی کنیم، دقت برابر ۱۲,۵ خواهد بود.

```
>>> test('mpgTrainingSet.txt', 'mpgTestSet.txt')

56.00% correct
```

همان‌طور که می‌بینید، با نرمال‌سازی، به دقتِ ۵۶ درصد رسیدیم، ولی بدون نرمال‌سازی دقت ما برابر ۳۲ درصد می‌شود.

ته‌مانده‌ی فصل

حواستان به نرمال‌سازی باشد!

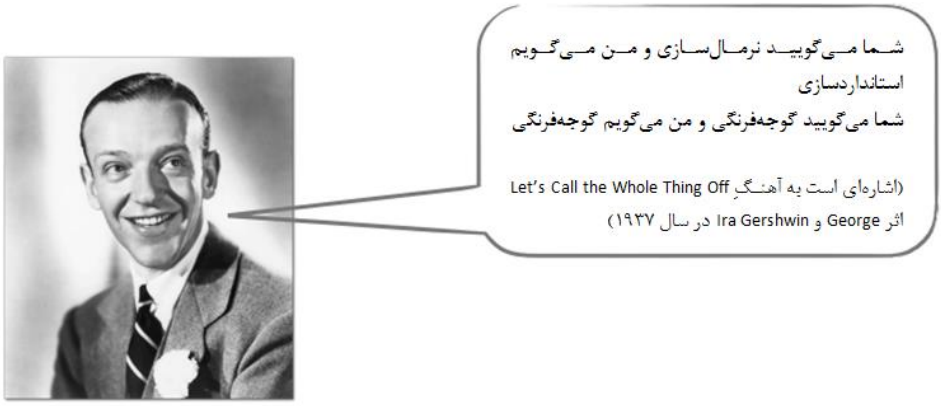
در این فصل، درباره‌ی اهمیتِ نرمال‌سازیِ داده‌ها صحبت کردیم. نرمال‌سازی وقتی خیلی اهمیت پیدا می‌کند که ویژگی‌های مختلف، در مقیاس‌ها و بازه‌های متفاوت قرار دارند (برای مثال، حقوق و سن). برای رسیدن به معیارِ فاصله‌ی دقیق، نیاز داریم تا ویژگی‌ها را به بازه‌ی یکسان تغییر دهیم.

در حالی‌که اکثرِ داده‌کاوها، از کلمه‌ی نرمال‌سازی (normalization) برای اشاره به این تغییرِ بازه استفاده می‌کنند، بعضی از افراد هستند که بین نرمال‌سازی (normalization) و استانداردسازی (standardization) تفاوت قائل می‌شوند، برای این افراد، نرمال‌سازی، برابر است با تغییرِ بازه‌ی مقادیرِ مختلف به صورتی است که بعد از این تغییر، مقادیر در بازه‌ی ۰ تا ۱ قرار بگیرند. در حالی که استانداردسازی، در واقع، تغییر بازه و مقیاسِ یک ویژگی است به گونه‌ای که به طور متوسط (میانگین یا میانه‌ی آن) برابر ۰ شود و بقیه‌ی مقادیر، حاصلِ انحراف از این میانگین باشند (مانند انحرافِ استاندارد یا انحرافِ استانداردِ مطلق). پس برای این دسته از داده‌کاوها، امتیازِ استاندارد یا امتیازِ استانداردِ اصلاح‌شده، مثال‌هایی از استانداردسازی است.

به یاد بیاورید که یکی از راه‌های نرمال‌سازیِ یک ویژگی به صورتی‌که در بازه‌ی ۰ تا ۱ قرار بگیرد، محاسبه‌ی کمینه (min) و بیشینه (max) برای مقادیرِ آن ویژگی بود. مقدار نرمال‌شده برای هر یک از مقادیرِ آن ویژگی به صورت زیر محاسبه می‌شد:

$$\frac{value - min}{\max - min}$$

بیایید دقتِ یک طبقه‌بند را برای هنگامی که از این نرمال‌سازی استفاده می‌کند در مقایسه با هنگامی که از امتیازِ استانداردِ اصلاح‌شده استفاده می‌کند، پیدا کنیم.



کد بزنید:

آیا می‌توانید، طبقه‌بندی که نوشتیم را طوری تغییر دهید که از نرمال‌سازیِ نوشته شده در صفحه‌ی قبل استفاده کند؟

می‌توانید بعد از تغییرِ کد، این داده‌ها را بر روی سه مجموعه‌ی داده‌ای که داشتیم، آزمایش کنید:

	الگوریتم طبقه‌بندی		
مجموعه‌های داده	بدون استفاده از نرمال‌سازی	با استفاده از فرمول صفحه‌ی قبل	با استفاده از فرمول امتیاز استاندارد اصلاح‌شده
Athletes	80.00%	?	80.00%
Iris	100.00%	?	93.33%
MPG	32.00%	?	56.00%

کد بزنید – نتایجِ من

این نتایجی است که من گرفته‌ام:

	الگوریتم طبقه‌بندی		
مجموعه‌های داده	بدون استفاده از نرمال‌سازی	با استفاده از فرمول صفحه‌ی قبل	با استفاده از فرمول امتیاز استاندارد اصلاح‌شده
Athletes	80.00%	60.00%	80.00%
Iris	100.00%	83.33%	93.33%
MPG	32.00%	36.00%	56.00%

خب! اگر بخواهیم نتایج را برای این نرمال‌سازی (min-max) با روشِ امتیازِ استانداردِ اصلاح‌شده مقایسه کنیم، باید بگوییم که نتایج ناامیدکننده بوده است.

جالب می‌شود اگر که با مجموعه داده‌های مختلف سر و کله بزنید و روش‌های مختلفی را بر روی آن‌ها آزمایش کنید. من مجموعه‌ی داده‌های گل‌های زنبق (Iris) و مایل به ازای یک گالن (MPG) را از سایتِ UCI که مخزنی برای داده‌های مربوط به یادگیریِ ماشین است، دریافت کردم (archive.ics.uci.edu/ml). به شما هم پیشنهاد می‌کنم که به این سایت بروید و یکی دو تا از مجموعه‌داده‌های این وب‌سایت را دانلود کنید. سپس آن‌ها را به فرمتِ دلخواه تبدیل کرده و ببینید که طبقه‌بندِ ما چه نتایجی بر روی این داده‌ها خواهد گرفت.

فصل ۵. کمی بیشتر در مورد طبقه‌بندی

توسعه‌های دیگری از طبقه‌بندی

الگوریتم‌های ارزیابی (Evaluating Algorithm) و الگوریتم KNN

بیایید به مثالِ ورزشکارانِ زن در فصلِ قبل برگردیم.
در آن مثال، ما یک طبقه‌بندی ساختیم که قد و وزنِ هر ورزشکار را به عنوان ورودی می‌گرفت، و آن ورودی را به ورزش‌های مختلف مانند ژیمیناستیک، دو و میدانی یا بسکتبال، طبقه‌بندی می‌کرد.
حالا، فردی مانندِ Marissa Coleman، که ۶ فوت و ۱ اینچ قد، و ۱۶۰ پوند وزن دارد را در نظر بگیرید. طبقه‌بندِ ما به درستی توانست پیش‌بینی کند که این خانم، بازیکنِ بسکتبال است:

```
>>> cl = Classifier('athletesTrainingSet.txt')
>>> cl.classify([73, 160])
'Basketball'
```

و همچنین این طبقه‌بند پیش‌بینی کرد که شخصی با قدِ ۴ فوت و ۹ اینچ، و وزنِ ۹۰ پوند، احتمالاً یک ژیمیناستیک کار است:

```
>>> cl.classify([59, 90])
'Gymnastics'
```

هنگامی که ما یک طبقه‌بند را ایجاد می‌کنیم، شاید مایل باشیم به برخی از سوالاتِ پیرامون آن پاسخ دهیم:

اما چگونه می‌توانیم به این سوالات پاسخ دهیم؟

مجموعه‌ی آموزشی (Training Set) و مجموعه‌ی آزمون (Test Set)

در انتهای فصلِ قبلی، ما با سه مجموعه‌ی داده‌ی مختلف کار کردیم: مجموعه‌ی داده‌ی ورزشکاران زن، مجموعه‌ی داده‌ی گل‌های زنبق (Iris) و مجموعه‌ی داده‌ی مایل به ازای هر گالن برای اتومبیل‌ها. هر کدام از این مجموعه داده‌ها را به **دو زیر قسمت** تقسیم کردیم. از یکی از زیر قسمت‌ها برای ساختِ طبقه‌بند (classifier) استفاده کردیم. به این مجموعه‌ی

داده که برای آموزشِ طبقه‌بند مورد استفاده قرار گرفت، **مجموعه‌ی آموزشی (training set)** گفته می‌شود. قسمت دوم اما، برای ارزیابیِ طبقه‌بند مورد استفاده قرار می‌گیرد. به این مجموعه‌ی داده، **مجموعه‌ی آزمون (test set)** گفته می‌شود. این دو عبارت (مجموعه‌ی آموزشی و مجموعه‌ی آزمون) دو عبارتِ آشنا و مرسوم در دنیای داده‌کاوی هستند.

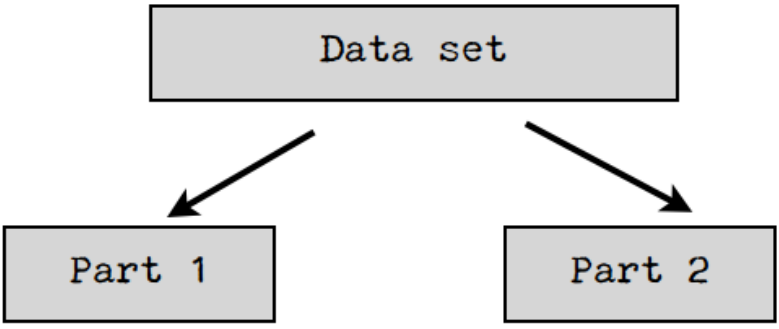
در دنیای داده‌کاوی، انسان‌ها هیچ‌گاه الگوریتم‌های خود را با داده‌هایی که از آن‌ها برای یاد گرفتن و آموزش استفاده کرده‌اند، امتحان نمی‌کنند. (در واقع داده‌هایی که از آن‌ها یاد می‌گیرید بایستی با داده‌هایی که توسط آن، الگوریتم را ارزیابی می‌کنید، تفاوت داشته باشد)

بیایید ببینیم که چرا ما از مجموعه‌ی آموزش، برای ارزیابی استفاده نمی‌کنیم. برای این‌کار همان الگوریتمِ نزدیک‌ترین همسایه را در نظر بگیرید. اگر خانمِ Marissa Coleman که یک بازیکنِ بسکتبال است، در مجموعه‌ی آموزشیِ ما باشد، او با ۶ فوت و ۱ اینچ قد، و ۱۶۰ پوند وزن، نزدیک‌ترین شخص به خودش خواهد بود. پس هر گاه بخواهیم الگوریتمِ نزدیک‌ترین همسایه را آزمایش کنیم و دقت آن را بسنجیم، اگر داده‌های آزمونِ ما که برای آزمایش کردنِ دقتِ الگوریتم استفاده می‌شود، زیر مجموعه‌ای از داده‌های آموزشی باشد، همواره دقتی نزدیک به ۱۰۰درصد خواهیم داشت. به بیانِ عمومی‌تر، در ارزیابیِ هر الگوریتمِ داده‌کاوی، اگر داده‌های مجموعه‌ی آزمون، زیر مجموعه‌ای از داده‌های مجموعه‌ی آموزشی باشد، نتایج، خوشبینانه و معمولاً بیش از حد خوشبینانه خواهد بود. پس این‌کار به نظر ایده‌ی خوبی نمی‌رسد.

حالا بیایید به ایده‌ای که در فصلِ قبل از آن استفاده کردیم، برگردیم. ما داده‌هایمان را به دو قسمت تقسیم کردیم. قسمتِ بزرگ‌تر را برای آموزش و قسمتِ کوچک‌تر را برای آزمون و ارزیابی استفاده کردیم. البته که این کار هم مشکلاتِ خودش را دارد. می‌توانیم در چگونگیِ تقسیمِ داده‌هایمان بسیار بدشانس باشیم. برای مثال، تمامیِ بازیکنانِ بسکتبال، در مجموعه‌ی آزمون، کوتاه قامت باشند. (مانند خانم Debbie Black که ۵ فوت و ۳ اینچ قد و ۱۲۴ پوند وزن دارد) و طبیعتاً به عنوانِ وزرشکارهای دو ماراتن طبقه‌بندی می‌شوند. و یا تمامیِ ورزشکارانِ دو و میدانی در مجموعه‌ی آزمون، کوتاه قامت و سبک‌وزن باشند، مانند خانم Tatyana Petrova که ۵ فوت و ۳ اینچ قد و ۱۰۸ پوند وزن دارد. در این صورت

بازهم این ورزشکاران به اشتباه ژیمیناستیک‌کار طبقه‌بندی می‌شوند. در مجموعه‌های آزمونی مانندِ این، **دقت (accuracy)** کم خواهد بود. و یا شاید، در انتخابِ مجموعه‌ی آزمون، خیلی خوش‌شانس باشیم. برای مثال، هر شخصی در مجموعه‌ی آزمون از لحاظ قد و وزن کاملاً مطابق و متناسب با ورزشِ مورد نظر خود باشد و در این صورت دقتِ ما نزدیک به ۱۰۰درصد خواهد شد که اشتباه است. در هر کدام از حالت‌ها (خوش‌شانسی یا بدشانسی)، اگر دقت را بر مبنای یک مجموعه آزمون قرار دهیم، نمی‌توان انتظار داشت که دقتِ واقعی را در زمانِ اجرایِ طبقه‌بند با داده‌های واقعی به ما نشان دهد.

راه حل برای این مشکل، می‌تواند این باشد که این فرآیند را چندین بار تکرار کنیم و میانگینِ دقت را برای تمامیِ نتایج محاسبه کرده و معیار عمل قرار دهیم. برای مثال، می‌توانیم داده‌ها را به صورت مساوی (نصف) تقسیم کنیم. اجازه دهید این دو قسمت را، قسمتِ ۱ (Part 1) و قسمتِ ۲ (Part 2) در نظر بگیریم.



می‌توانیم داده‌های موجود در قسمتِ ۱ را برای آموزشِ طبقه‌بند مورد استفاده قرار دهیم و داده‌های موجود در قسمتِ ۲ را برای آزمون و ارزیابی استفاده کنیم. و دوباره فرآیند را تکرار کنیم، ولی این‌بار با قسمتِ ۲، عملیات آموزش را انجام داده و با قسمتِ ۱ طبقه‌بند را بیازماییم. در آخر، میانگینِ دقت را محاسبه کرده و به عنوانِ دقتِ نهایی برگردانیم. یک مشکل با این طرز تفکر، این است که فقط از $1/2$ داده‌ها برای آموزشِ طبقه‌بند در هر دور استفاده کرده‌ایم. خوشبختانه این موضوع قابل حل است و راه‌حلِ آن این است که تعداد قسمت‌ها را افزایش دهیم. برای مثال، می‌توانیم سه قسمت داشته باشیم و در هر دور، دو

سوم ($^2/_3$) از داده‌ها را برای آموزش و یک سوم ($^1/_3$) را برای آزمون استفاده می‌کنیم. چیزی شبیه به جدول زیر می‌شود:

iteration 1	train with parts 1 and 2	test with part 3
iteration 2	train with parts 1 and 3	test with part 2
iteration 3	train with parts 2 and 3	test with part 1

Average the results.

در داده‌کاوی، معمول‌ترین روش این است که قسمت‌ها را به صورتِ ۱۰ تایی تقسیم کنیم و این روش به روشِ **اعتبار سنجیِ متقابلِ ۱۰-تکه‌ای (10-Fold Cross Validation)** معروف است.

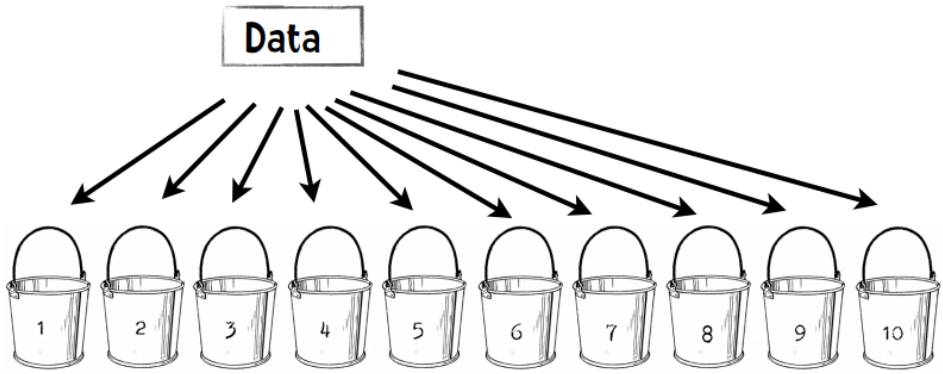
اعتبار سنجیِ متقابلِ ۱۰-تکه‌ای (10-Fold Cross Validation)

با استفاده از این روش، یک مجموعه‌ی داده داریم که آن را به صورت تصادفی به ۱۰ قسمت تقسیم می‌کنیم. از ۹ قسمتِ آن برای آموزش، و از ۱ قسمتِ آن برای آزمون استفاده می‌کنیم. این فرآیند را ۱۰ بار تکرار کرده و هر بار قسمت‌های مختلف را (یک قسمت از ۱۰ قسمت را) برای آزمون انتخاب می‌کنیم.

بیایید به یک مثال بپردازیم. فرض کنید من می‌خواهم یک طبقه‌بند درست کنم که فقط بتواند جواب‌های بلی (yes) یا خیر (no) را به سوالِ «آیا این شخص یک بازیکنِ حرفه‌ایِ بسکتبال هست؟» بدهد. داده‌های من بر اساسِ ۵۰۰ بازیکنِ بسکتبال و ۵۰۰ بازیکنِ غیر بسکتبال هستند.

مثالِ اعتبار سنجیِ متقابلِ ۱۰-تکه‌ای (10-Fold Cross Validation)

قدمِ ۱: داده‌ها را به ۱۰ قسمتِ مساوی تقسیم می‌کنیم:



یعنی در هر سطل (قسمت) ۵۰ بازیکنِ بسکتبال و ۵۰ بازیکنِ غیر بسکتبال قرار می‌دهیم. در واقع هر کدام از سطل‌ها، حاوی اطلاعاتِ ۱۰۰ شخص هستند. (به هر قسمت اصطلاحاً یک سطل – bucket می‌گوییم)

قدمِ ۲: قدم‌های زیر را ۱۰ بار تکرار می‌کنیم:

۲-۱ در هر دور، یکی از سطل‌ها را کنار می‌گذاریم. مثلاً برای دورِ اول، سطلِ شماره‌ی ۱ را کنار می‌گذاریم، دورِ دوم سطلِ شماره‌ی ۲ و به همین ترتیب تا آخر.

۲-۲ با استفاده از داده‌های موجود در بقیه‌ی سطل‌ها (که کنار گذاشته نشده‌اند)، طبقه‌بند را آموزش می‌دهیم (مثلاً در دورِ اول، طبقه‌بند را بر اساسِ داده‌های موجود در سطل‌های ۲ تا ۱۰ آموزش می‌دهیم)

۲-۳ حالا طبقه‌بندی که ساخته‌ایم را با داده‌های آن سطلی که کنار گذاشته‌ایم مورد آزمایش قرار داده و نتایج را برای آن دور ذخیره می‌کنیم. برای مثال نتایج برای یک دور، می‌تواند به شکل زیر باشد:

۳۵ نفر از بازیکنانِ بسکتبال، به درستی طبقه‌بندی شده‌اند

۲۹ نفر از بازیکنان غیر بسکتبال نیز به درستی طبقه‌بندی شده‌اند

قدمِ ۳: نتایجِ حاصل از قدمِ دو را تجمیع می‌کنیم

معمولاً نتایجِ نهایی را در جدولی مانندِ جدولِ زیر قرار می‌دهیم:

	به عنوان بازیکن بسکتبال طبقه‌بندی شده‌اند	به عنوان بازیکن بسکتبال طبقه‌بندی نشده‌اند
بازیکن بسکتبال (واقعی)	372	128
بازیکن غیر بسکتبال (واقعی)	220	280

مشاهده می‌کنید در مثالِ جدول بالا، از ۵۰۰ بازیکنِ بسکتبال، ۳۷۲ نفر از آن‌ها به صورتِ صحیح طبقه‌بندی شده‌اند. یک کاری که می‌توانیم انجام دهیم این است که این‌ها را با هم جمع کرده و بگوییم از ۱۰۰۰ نفری که مورد ارزیابی قرار دادیم، ۶۵۲ (۳۷۲ + ۲۸۰) نفر از آن‌ها را به درستی طبقه‌بندی کرده‌ایم. پس در این‌جا دقت ۶۵٫۲٪ است. معیاری که ما از اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای (10-Fold Cross Validation) به دست آوریم، نسبت به اعتبار سنجیِ ۲ یا ۳ تکه‌ای، ارزش بیشتری دارد. این ارزش، به این دلیل اتفاق می‌افتد که در هر بار آموزشِ طبقه‌بند، ما از ۹۰ درصدِ داده‌ها استفاده می‌کنیم (در حالی‌که برای مثال در اعتبار سنجیِ متقابلِ ۲ تکه‌ای، در هر بار آموزش، از ۵۰ درصدِ داده‌ها استفاده می‌کردیم).

طبق گفته‌ی این خانم در شکل بالا، برای مثال، اگر ما ۱۰۰۰ نمونه (مثلاً ۱۰۰۰ ورزشکار) داریم، طبقه‌بندمان را بر اساسِ ۹۹۹ نفر آموزش داده و بر اساس ۱ نفرِ باقی‌مانده آزمایش می‌کنیم و این کار را در ۱۰۰۰ دور (۱۰۰۰ مرتبه) تکرار خواهیم کرد. در واقع با استفاده از حداکثر تعداد نمونه‌های موجود برای آموزشِ طبقه‌بند، دقت آن می‌تواند بسیار بیشتر شود.

روش کنار گذاشتنِ یکی (Leave-One-Out)

در ادبیاتِ یادگیری‌ماشین، اعتبار سنجیِ متقابل n-تایی (n-fold cross validation) که در آن **n**، تعداد نمونه‌های موجود در مجموعه‌ی داده‌ی ماست، روشِ «کنار گذاشتنِ یکی (leave-one-out)» نیز نامیده می‌شود. در حال حاضر یکی از ویژگی‌های این روش را گفته‌ایم – این‌که در هر دور، حداکثر مقادیرِ موجود برای آموزشِ طبقه‌بند را استفاده می‌کنیم. یکی از ویژگی‌های مثبتِ دیگر، این است که این روش یک روشِ قطعی (deterministic) است.

منظورمان از قطعی (deterministic) چیست؟

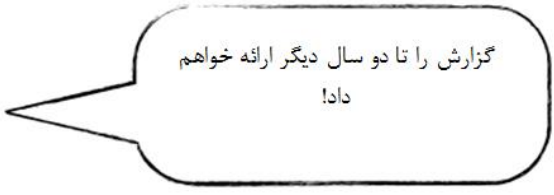
فرض کنید Lucy، 80 ساعتِ فشرده را در هفته به ساخت و کدنویسیِ یک الگوریتمِ طبقه‌بندی پرداخته باشد. الان جمعه است و Lucy دیگر خسته شده‌است. او از دو دوستِ دیگر خود (Emily و Li) درخواست می‌کند تا الگوریتمِ او را مورد ارزیابی قرار دهند. او به آن دو، الگوریتمِ ساخته شده‌ی خود را همراه مجموعه‌ی داده‌ی یکسان می‌دهد و از آن‌ها می‌خواهد این این الگوریتم را با استفاده از اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای ارزیابی کنند. دوشنبه، Lucy از آن‌ها می‌خواهد تا نتایج را به او بدهند:

خب! به نظر می‌رسد آن‌ها نتایجِ یکسانی نگرفته‌اند. آیا یکی از آن دو اشتباهی مرتکب شده‌است؟ نه الزماً. در روشِ اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای، ما داده‌ها را به صورت تصادفی در سطل‌های مختلف قرار می‌دهیم. به خاطرِ وجودِ این عنصرِ تصادفی، احتمالاً Emily و Li داده‌ها را به قسمت‌های شبیه به هم تقسیم نکرده‌اند. در واقع، خیلی احتمال کمی دارد که آن‌ها دقیقاً مانندِ یکدیگر عملیاتِ تقسیم‌کردنِ داده‌ها را انجام داده باشند و دقیقاً داده‌های مشابه را در قسمت‌ها یا همان سطل‌های مشابه، قرار داده باشند. پس در هنگامِ آموزشِ الگوریتمِ طبقه‌بندی، آن‌ها از داده‌های مختلفی استفاده کرده‌اند. همین‌کار نیز در هنگامِ

آزمودنِ الگوریتمِ طبقه‌بندی رخ داده است و آن‌ها با داده‌های مختلفی، الگوریتمِ ساخته‌شده را ارزیابی کرده‌اند. پس بسیار منطقی است که نتایجِ مختلفی گرفته باشند و این نتیجه ربطی به این ندارد که دو نفرِ جدا، عملیاتِ ارزیابی را انجام داده‌اند. اگر خودِ Lucy هم یک بار دیگر عملیات اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای را انجام می‌داد، دو نتیجه‌ی مشابه را هم دریافت می‌کرد. دلیلِ این اختلاف این است که یک قسمت از آن به صورت تصادفی انجام می‌شود. برای همین است که اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای یک روشِ غیرِ قطعی (non-deterministic) نامیده می‌شود، یعنی اگر دو بار آزمون را انجام دهیم، هیچ تضمینی وجود ندارد که نتایجِ یکسانی را دریافت کنیم. **برعکس، روشِ «کنار گذاشتنِ یکی (leave-one-out)» یک روشِ قطعی است**. هر باری که ما این روش را بر روی یک الگوریتمِ طبقه‌بندیِ خاص با مجموعه‌ی داده‌ی یکسان اجرا کنیم، نتیجه‌ی یکسانی خواهیم گرفت. که این برایمان خوب است!

معایب روشِ کنار گذاشتنِ یکی (Leave-One-Out)

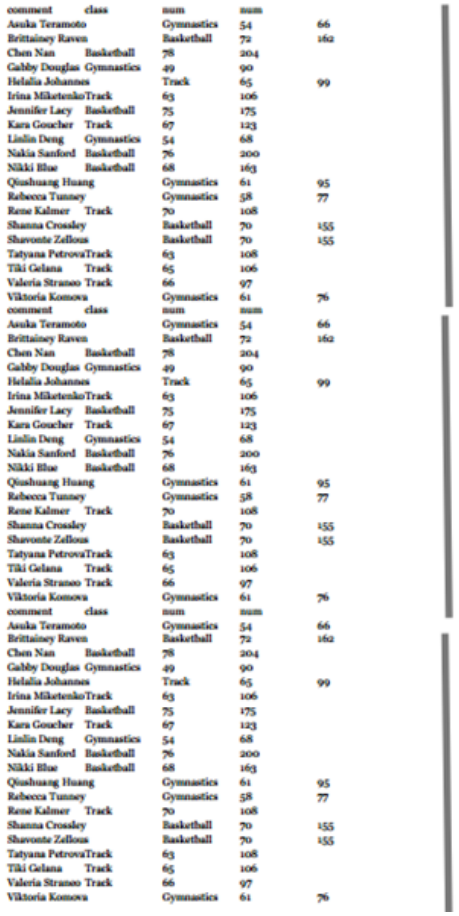
عیب اصلی در روشِ کنار گذاشتنِ یکی (leave-one-out)، هزینه‌ی محاسباتی آن است. فرض کنید یک مجموعه‌ی داده با ۱۰۰۰ نمونه داشته باشیم و برای آموزشِ الگوریتمِ طبقه‌بندی با این داده‌ها، به یک دقیقه زمان احتیاج باشد. برای اعتبارسنجیِ متقابلِ ۱۰ تکه‌ای، طبیعتاً به ده دقیقه زمان برای آموزش احتیاج داریم. ولی در روشِ leave-one-out، این زمان به ۱۶ ساعت می‌رسد. اگر مجموعه‌ی داده‌ی ما شامل یک میلیون نمونه باشد، مجموعِ زمانِ یادگیری در روشِ leave-one-out تقریباً ۲ سال می‌شود!



مشکلِ دیگر این روش، مربوط به لایه‌بندی (stratification) می‌شود.

لایه‌بندی (Stratification)

بیایید به مثالِ فصل قبل برگردیم – ساخت یک طبقه‌بند که بتواند پیش‌بینی کند یک ورزشکار، چه ورزشی را انجام می‌دهد (بسکتبال، ژیمیناستیک یا دو و میدانی). در هنگام آموزشِ الگوریتمِ طبقه‌بندی، می‌خواهیم داده‌های آموزشی نماینده‌ای از داده‌ها بوده و از هر یک از سه طبقه‌ی موجود (بسکتبال، ژیمیناستیک و دو و میدانی) در این داده‌ها موجود باشند. فرض کنید ما داده‌ها را به مجموعه‌ی آموزشی به صورت کاملاً تصادفی اختصاص دهیم. این امکان وجود دارد که هیچ بازیکن بسکتبالی در این مجموعه‌ی تصادفیِ آموزشی، موجود نباشد و به همین خاطر الگوریتمِ ساخته‌شده، برای تشخیص بازیکنان بسکتبال، خیلی خوب عمل نکند. یا مثلاً فرض کنید یک مجموعه‌ی داده از ۱۰۰ ورزشکار داشته باشیم. اول به وب‌سایتِ مسابقات WNBA (بسکتبال NBA زنان) می‌رویم و اطلاعاتِ ۳۳ ورزشکار را استخراج می‌کنیم. سپس به وب‌سایت ویکی‌پدیا رفته و تعداد ۳۳ ورزشکار ژیمیناستیک که در مسابقات المپیک ۲۰۱۲ شرکت کرده‌اند را یادداشت می‌کنیم. در آخر دوباره به وب‌سایت ویکی‌پدیا سر می‌زنیم و تعداد ۳۴ نفر از ورزشکارانِ زنِ دو و میدانی که در المپیک‌های مختلف شرکت کرده‌اند را استخراج می‌کنیم. پس در واقع مجموعه‌ی داده‌ی ما چیزی شبیه به شکل زیر می‌شود:



comment	class	num	num	
Asuka Teramoto		Gymnastics	54	66
Brittainey Raven		Basketball	72	162
Chen Nan	Basketball	78	204	
Gabby Douglas	Gymnastics	49	90	
Helalia Johannes		Track	65	99
Irina Miketenko	Track	63	106	
Jennifer Lacy	Basketball	75	175	
Kara Goucher	Track	67	123	
Linlin Deng	Gymnastics	54	68	
Nakia Sanford	Basketball	76	200	
Nikki Blue	Basketball	68	163	
Qiushuang Huang		Gymnastics	61	95
Rebecca Tunney		Gymnastics	58	77
Rene Kalmer	Track	70	108	
Shanna Crossley		Basketball	70	155
Shavonte Zellous		Basketball	70	155
Tatyana Petrova	Track	63	108	
Tiki Gelana	Track	65	106	
Valeria Strunso	Track	66	97	
Viktoria Komova		Gymnastics	61	76
comment	class	num	num	
Asuka Teramoto		Gymnastics	54	66
Brittainey Raven		Basketball	72	162
Chen Nan	Basketball	78	204	
Gabby Douglas	Gymnastics	49	90	
Helalia Johannes		Track	65	99
Irina Miketenko	Track	63	106	
Jennifer Lacy	Basketball	75	175	
Kara Goucher	Track	67	123	
Linlin Deng	Gymnastics	54	68	
Nakia Sanford	Basketball	76	200	
Nikki Blue	Basketball	68	163	
Qiushuang Huang		Gymnastics	61	95
Rebecca Tunney		Gymnastics	58	77
Rene Kalmer	Track	70	108	
Shanna Crossley		Basketball	70	155
Shavonte Zellous		Basketball	70	155
Tatyana Petrova	Track	63	108	
Tiki Gelana	Track	65	106	
Valeria Strunso	Track	66	97	
Viktoria Komova		Gymnastics	61	76
comment	class	num	num	
Asuka Teramoto		Gymnastics	54	66
Brittainey Raven		Basketball	72	162
Chen Nan	Basketball	78	204	
Gabby Douglas	Gymnastics	49	90	
Helalia Johannes		Track	65	99
Irina Miketenko	Track	63	106	
Jennifer Lacy	Basketball	75	175	
Kara Goucher	Track	67	123	
Linlin Deng	Gymnastics	54	68	
Nakia Sanford	Basketball	76	200	
Nikki Blue	Basketball	68	163	
Qiushuang Huang		Gymnastics	61	95
Rebecca Tunney		Gymnastics	58	77
Rene Kalmer	Track	70	108	
Shanna Crossley		Basketball	70	155
Shavonte Zellous		Basketball	70	155
Tatyana Petrova	Track	63	108	
Tiki Gelana	Track	65	106	
Valeria Strunso	Track	66	97	
Viktoria Komova		Gymnastics	61	76

بیایید از روشِ اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای (10-Fold Cross Validation) استفاده کنیم. از اولِ لیست شروع کرده و هر ۱۰ نفر را در یک قسمت (سطل) مجزا قرار می‌دهیم. در موردِ مثالِ بالا، ما ۱۰ بازیکنِ بسکتبال در سطل‌های یک و دو داریم. سطلِ سوم شامل بازیکنانِ بسکتبال و ژیمیناستیک می‌شود. سطل‌های چهارم و پنجم تنها به بازیکنان ژیمیناستیک تعلق می‌گیرند و به همین ترتیب تا سطلِ دهم. احتمالاً متوجه شده‌اید که هیچ کدام از سطل‌های ما، نماینده‌ی مجموعه‌ی داده‌ی نیستند. این باعث چولگی در نتایج خواهد شد. روش خوب و مناسب برای اختصاصِ داده‌ها به سطل‌ها این است که اطمینان حاصل کنیم که طبقه‌ها (بسکتبال، ژیمیناستیک و دو و میدانی) به صورت متناسب، با نسبتی که در مجموعه‌ی داده‌ی اصلی موجود هستند، در هر کدام از سطل‌ها نیز وجود

داشته باشند. پس چون در مثالِ ما یک سومِ داده‌های مجموعه‌ی اصلی، بازیکنان بسکتبال هستند، در هر کدام از قسمت‌ها (سطل‌ها) نیز بایستی یک سوم، بازیکنِ بسکتبال داشته باشیم. به همین نسبت در هر کدام از سطل‌ها، یک سومِ دیگر بایستی ژیمیناستیک‌کار و یک سوم، ورزشکارانِ دو و میدانی باشند. به این کار در اصطلاح، **لایه‌بندی (stratification)** گفته می‌شود و کار خوبی هم هست. ولی مشکلِ روشِ leave-one-out این است که هیچ کدام از مجموعه‌های آزمون، لایه‌بندی شده نیستند و در واقع non-stratified هستند، چون فقط یک نمونه در هر دور تکرار، برای داده‌ی آزمون در نظر گرفته می‌شود. در مجموع، با این‌که روشِ leave-one-out برای مجموعه داده‌های کوچک، می‌تواند مفید باشد، روشِ اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای (10-fold cross validation) به دلایلِ ذکر شده، معروف‌ترین روش در میان روش‌های گفته شده در دنیای داده‌کاوی و یادگیری‌ماشین است.

ماتریس اغتشاش (Confusion Matrices)

تا به این‌جای کار، ما الگوریتمِ طبقه‌بندیِ خودمان را بر اساس معیارِ دقت (accuracy) ارزیابی می‌کردیم. این معیار به صورت زیر محاسبه می‌شود:

تعداد مواردی که به درستی طبقه‌بندی شده است (در مجموعه‌ی آزمون)
تقسیم بر
تعدادِ کلِ مواردی که در مجموعه‌ی آزمون قرار داشته است

اما گاهی اوقات، ما نیاز داریم تا اطلاعاتِ دقیق‌تری از کیفیتِ الگوریتم‌های طبقه‌بندی به دست آوریم. یکی از مفاهیمی که به صورت دقیق و با جزئیات این کار را انجام می‌دهد، **ماتریس اغتشاش (confusion matrix)** است. در این ماتریس، سطرها بیان‌گرِ طبقه‌های واقعی هستند، و ستون‌ها به طبقه‌ای که الگوریتمِ طبقه‌بندی پیش‌بینی کرده است، اشاره می‌کنند.

اسمِ «ماتریسِ اغتشاش (confusion matrix)» از آن‌جایی می‌آید که با دیدن این ماتریس، می‌فهمیم که الگوریتممان در کجاها «گیج (confused)» شده است. اجازه بدهید به مثالِ خودمان، درباره زنانِ ورزشکار برگردیم. فرض کنید ما مجموعه‌ی داده‌ای داریم که ویژگی‌های زنانِ ورزشکار در رشته‌های مختلف را در خود جای داده است. ۱۰۰ زن ژیمیناستیک‌کار، ۱۰۰ زن که در مسابقات بسکتبال WNBA بازی می‌کنند و ۱۰۰ زن که در دو و میدانی (دوی ماراتُن) شرکت کرده‌اند. الگوریتمِ طبقه‌بندی را با روشِ اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای (10-fold cross validation) ارزیابی می‌کنیم. در این روش، هر کدام از نمونه‌ها، یعنی هر کدام از زنانِ ورزشکار را دقیقاً یک بار برای آزمون استفاده می‌کنیم. نتایجِ این آزمون، در ماتریسِ اغتشاشِ زیر نمایش داده شده‌است (ژیمیناستیک‌کار = gymnast، بازیکن بسکتبال = basketball player و ورزشکار دو و میدانی (دوی ماراتن) = marathoner):

	gymnast	basketball player	marathoner
gymnast	83	0	17
basketball player	0	92	8
marathoner	9	16	75

برای تفسیر جدول بالا، همان‌طور که گفتیم، سطرها بیان‌گرِ طبقه‌های واقعی بوده و ستون‌ها نشان‌دهنده‌ی طبقه‌هایی هستند که الگوریتمِ ما طبقه‌بندی کرده است. پس برای مثال ۸۳ مورد از ژیمیناستیک‌کارها به درستی طبقه‌بندی شده‌اند و ۱۷ مورد از آن‌ها به اشتباه، ورزشکارِ دوی ماراتن، توسط الگوریتم، طبقه‌بندی شده‌اند. ۹۲ نفر از بازیکنانِ بسکتبال به درستی توسط الگوریتم، طبقه‌بندی شده‌اند و ۸ نفر از آن‌ها، به اشتباه به عنوان بازیکنان دوی ماراتن شناخته شده‌اند. در آخر، ۷۵ نفر از بازیکنانِ دوی ماراتن به درستی طبقه‌بندی شده‌اند و الگوریتم، ۹ نفر از آن‌ها را، به اشتباه برچسبِ ژیمیناستیک‌کار و ۱۶ نفر را، به طبقه‌ی بسکتبالیست نسبت داده است.

قطرِ اصلی در ماتریسِ اغتشاش، بیان‌گر نمونه‌هایی است که به درستی طبقه‌بندی شده‌اند:

	gymnast	basketball player	marathoner
gymnast	**83**	0	17
basketball player	0	**92**	8
marathoner	9	16	**85**

در این مثال، دقت (accuracy) برای الگوریتمِ ما به صورت زیر محاسبه می‌شود:

$$\frac{83+92+75}{300}=\frac{250}{300}=83.33\%$$

برای این‌که بفهمیم الگوریتممان چه نوع خطاهایی دارد، می‌توانیم به سادگی، ماتریسِ اغتشاش را بررسی کنیم. در مثالِ بالا، به نظر می‌رسد که الگوریتمِ طبقه‌بندیِ ما، توانسته تمایزِ خوبی بینِ ورزشکاران ژیمیناستیک و بسکتبالیست‌ها قائل شود. ولی بعضی اوقات بازیکنان بسکتبال و ژیمیناستیک‌کاران به عنوان ورزشکارانِ دوی ماراتن توسط الگوریتم شناخته می‌شوند و برخی دیگر از اوقات، ورزشکارانِ دوی ماراتن به عنوان ژیمیناستیک‌کاران یا ورزشکارانِ بسکتبالیست، برچسب می‌خورند.

درکِ ماتریسِ اغتشاش، خیلی هم سخت نیس!

یک مثال برنامه‌نویسی

بیایید به مجموعه‌ی داده‌ای که در فصلِ پیش به آن پرداختیم، برگردیم. مجموعه‌ی داده‌ی «مایل به ازای هر گالن برای اتومبیل‌ها (Auto Miles Per Gallon)» از دانشگاهِ کارنی ملون (Carnegie Mellon University). فرمتِ داده‌ها چیزی شبیه به جدول زیر بود:

mpg	cylinders	c.i.	HP	weight	secs. 0-60	make/model
30	4	68	49	1867	19.5	fiat 128
45	4	90	48	2085	21.7	vw rabbit (diesel)
20	8	307	130	3504	12	chevrolet chevelle malibu

من می‌خواهم مصرف اتومبیل‌ها را به صورتِ مایل به ازای هر گالن، پیش‌بینی کنم. این پیش‌بینی بر اساس ویژگی‌های مختلف مانند «تعداد سیلندرها (cylinders)»، «ظرفیت پیستون (بر اساس اینچِ مکعب (c.i.))»، اسبِ بخار (HP)، وزن (weight) و شتاب (یعنی زمانِ بین ۰ تا ۶۰ (secs. 0-60)) است. من تمامیِ این ۳۹۲ نمونه را در فایلی با نام mpgData.txt گذاشتم و کدِ پایتونِ زیر را برای آن نوشتم. این کدِ پایتون داده‌ها را به ۱۰ قسمتِ مختلف (سطل‌ها) با روشِ لایه‌بندی (stratified) تقسیم می‌کند. (داده‌ها و کدها در وب‌سایت guidetodataminig.com و chistio.ir موجود است)

```
import random
def buckets(filename, bucketName, separator, classColumn
):
    """the original data is in the file named filename
    bucketName is the prefix for all the bucket names
    separator is the character that divides the columns
    (for ex., a tab or comma and classColumn is the colu
mn
    that indicates the class"""
    # put the data in 10 buckets
    numberOfBuckets = 10
    data = {}
    # first read in the data and divide by category
    with open(filename) as f:
        lines = f.readlines()
    for line in lines:
```

```
        if separator != '\t':
            line = line.replace(separator, '\t')
        # first get the category
        category = line.split()[classColumn]
        data.setdefault(category, [])
        data[category].append(line)
    # initialize the buckets
    buckets = []
    for i in range(numberOfBuckets):
        buckets.append([])
    # now for each category put the data into the bucket
s
    for k in data.keys():
        #randomize order of instances for each class
        random.shuffle(data[k])
        bNum = 0
        # divide into buckets
        for item in data[k]:
            buckets[bNum].append(item)
            bNum = (bNum + 1) % numberOfBuckets
    # write to file
    for bNum in range(numberOfBuckets):
        f = open("%s-%02i" % (bucketName, bNum + 1), 'w'
)
        for item in buckets[bNum]:
            f.write(item)
        f.close()
buckets("mpgData.txt", 'mpgData','\t',0)
```

با اجرای این کد، ۱۰ فایل با نام‌های mpgData01، mpgData02 و به همین ترتیب تا آخر، ساخته می‌شود.

کد بزنید

آیا می‌توانید کدِ نزدیک‌ترین همسایه را از فصلِ قبل طوری تغییر دهید که تابعِ آزمون (test)، عملیاتِ اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای را بر روی ۱۰ فایلِ داده‌ای که ساخته‌ایم تولید کند؟ (داده‌ها را می‌توانید از سایت guidetodatamining.com و chistio.ir دریافت کنید)

برنامه‌ای که می‌نویسید، بایستی در خروجی، یک ماتریسِ اغتشاش مانندِ زیر تولید کند:

ac-tual MPG \ predicted MPG	10	15	20	25	30	35	40	45
10	3	10	0	0	0	0	0	0
15	3	68	14	1	0	0	0	0
20	0	14	66	9	5	1	1	0
25	0	1	14	35	21	6	1	1
30	0	1	3	17	21	14	5	2
35	0	0	2	8	9	14	4	1
40	0	0	1	0	5	5	0	0
45	0	0	0	2	1	1	0	2

53.316% accurate
total of 392 instances

کد بزنید – راه‌حل

یکی از راه‌حل‌ها برای مسئله‌ی بالا می‌تواند این باشد:

تابع اولیه (initializer) را تغییر داده تا داده‌ها را از ۹ قسمت (۹ سطل) بخواند

یک تابعِ جدید برای آزمودن (test) از داده‌های یک قسمت بسازیم

یک تابعِ جدید بسازیم که عملیات اعتبارسنجیِ متقابل ۱۰-تکه‌ای را محاسبه کند

بیایید به صورت عملی به این قضیه نگاهی بیندازیم

تابع اولیه __init__

امضا (signature) برای تابعِ __init__ به این صورت است:

```
def __init__(self, bucketPrefix, testBucketNumber, dataFormat):
```

اسمِ فایل‌ها (filenames) برای هر قسمت چیزی مانند mpgData-01، mpgData-02 و... خواهد بود. در این مثالِ ما، bucketPrefix در واقع «mpgData» است به معنای پیشوند.

منظور از bucket همان قسمت‌ها یا سطل‌ها هستند که وظیفه‌ی جداسازی ۱۰ تکه را برعهده دارند. testBucketNumber همان سطلی است که داده‌های آزمون را در خود نگه می‌دارد. مثلاً اگر مقدارِ testBucketNumber برابر ۳ بود، الگوریتمِ طبقه‌بندی، بر اساسِ داده‌های قسمت‌های ۱، ۲، ۴، ۵، ۶، ۷، ۸، ۹ و ۱۰ عملیاتِ یادگیری را انجام می‌داد و بر اساسِ داده‌های قسمتِ ۳، خود را ارزیابی می‌کرد. dataFormat رشته‌ای است که مشخص می‌کند چگونه ستون‌ها در داده‌ها بایستی تفسیر شوند. برای مثال:

”class num num num num num comment“

مشخص می‌کند که اولین ستون (از سمتِ چپ) به معنای طبقه (class) برای هر نمونه است. ۵ ستونِ بعدی، بیان‌گرِ ویژگی‌های عددی برای نمونه‌ها هستند و ستونِ آخر به عنوان توضیحات در نظر گرفته می‌شود.

تابع __init__، به صورت کامل، مانند کدهای زیر است:

```
import copy
class Classifier:
def __init__(self, bucketPrefix, testBucketNumber, dataF
ormat):
    """ a classifier will be built from files with the b
ucketPrefix
    excluding the file with textBucketNumber. dataFormat
 is a
    string that describes how to interpret each line of
the data
    files. For example, for the mpg data the format is:
    "class num num num num num comment"
    """
    self.medianAndDeviation = []
    # reading the data in from the file
    self.format = dataFormat.strip().split('\t')
    self.data = []
    # for each of the buckets numbered 1 through 10:
    for i in range(1, 11):
        # if it is not the bucket we should ignore, read
 the data
        if i != testBucketNumber:
            filename = "%s-%02i" % (bucketPrefix, i)
            f = open(filename)
            lines = f.readlines()
            f.close()
```

```
            for line in lines:
                fields = line.strip().split('\t')
                ignore = []
                vector = []
                for i in range(len(fields)):
                    if self.format[i] == 'num':
                        vector.append(float(fields[i]))
                    elif self.format[i] == 'comment':
                        ignore.append(fields[i])
                    elif self.format[i] == 'class':
                        classification = fields[i]
                self.data.append((classification, vector
, ignore))
    self.rawData = copy.deepcopy(self.data)
    # get length of instance vector
    self.vlen = len(self.data[0][1])
    # now normalize the data
    for i in range(self.vlen):
        self.normalizeColumn(i)
```

تابع آزمون قسمت‌ها (testBucket)

بعد از آن که تابعِ __init__ را نوشتیم، تابعی را می‌نویسیم که بتواند داده‌های موجود در یک قسمت (سطل) را مورد آزمون و ارزیابی قرار دهد:

```
def testBucket(self, bucketPrefix, bucketNumber):
    """Evaluate the classifier with data from the file
    bucketPrefix-bucketNumber"""
    filename = "%s-%02i" % (bucketPrefix, bucketNumber)
    f = open(filename)
    lines = f.readlines()
    totals = {}
    f.close()
    for line in lines:
        data = line.strip().split('\t')
        vector = []
        classInColumn = -1
        for i in range(len(self.format)):
            if self.format[i] == 'num':
                vector.append(float(data[i]))
            elif self.format[i] == 'class':
                classInColumn = i
        theRealClass = data[classInColumn]
        classifiedAs = self.classify(vector)
```

```
        totals.setdefault(theRealClass, {})
        totals[theRealClass].setdefault(classifiedAs, 0)
        totals[theRealClass][classifiedAs] += 1
    return totals
```

این تابع یک bucketPrefix و یک bucketNumber به عنوان ورودی دریافت می‌کند. مثلاً اگر مقدارِ پیشوند (bucketPrefix) برابر «mpgData» و عددِ قسمت (bucketNumber) برابر ۳ بود، داده‌ی آزمون از فایلِ mgpData-03 خوانده می‌شود. تابعِ testBucket یک دیکشنری با فرمتِ زیر برمی‌گرداند:

```
{'35':   {'35': 1, '20': 1, '30': 1},
 '40':   {'30': 1},
 '30':   {'35': 3, '30': 1, '45': 1, '25': 1},
 '15':   {'20': 3, '15': 4, '10': 1},
 '10':   {'15': 1},
 '20':   {'15': 2, '20': 4, '30': 2, '25': 1},
 '25':   {'30': 5, '25': 3}}
```

کلیدِ این دیکشنری، بیان‌گرِ طبقه‌ی واقعی برای آن نمونه است. برای مثال، خطِ اول بیان‌گر نتایج برای نمونه‌هایی است که مقدار درستِ طبقه‌بندی برای آن‌ها برابر ۳۵ mpg (مایل به ازای هر گالن) بوده است. مقادیرِ هر کدام از این کلیدها یک دیکشنریِ دیگر است که نشان می‌دهد الگوریتمِ طبقه‌بندی، چگونه این نمونه‌ها را برچسب زده است. برای مثال، خطِ

```
'15':   {'20': 3, '15': 4, '10': 1},
```

نشان دهنده‌ی آزمونی است که در آن ۳ نمونه، مورد بررسی قرار گرفته‌اند و دو تای آن‌ها به اشتباه طبقه‌بندی شده‌اند. نمونه‌ی شماره‌ی ۳، که به عددِ ۲۰ mpg به اشتباه طبقه‌بندی شده است، نمونه‌ی شماره‌ی ۴ که به درستی به عددِ ۱۵ mpg طبقه‌بندی شده است و نمونه‌ی شماره‌ی ۱ که به اشتباه به عددِ ۱۰ mpg طبقه‌بندی شده است.

فرآیند اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای (10-Fold Cross Validation)

در آخر، بایستی تابعی بنویسیم که بتواند فرآیند اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای را انجام دهد. در واقع این تابع، ۱۰ طبقه‌بند می‌سازد. هر کدام از این طبقه‌بندها با ۹ قسمتِ آموزشی، آموزش می‌بینند و بر اساسِ ۱ قسمتِ باقی‌مانده، ارزیابی می‌شوند.

```
def tenfold(bucketPrefix, dataFormat):
    results = {}
    for i in range(1, 11):
        c = Classifier(bucketPrefix, i, dataFormat)
        t = c.testBucket(bucketPrefix, i)
        for (key, value) in t.items():
            results.setdefault(key, {})
            for (ckey, cvalue) in value.items():
                results[key].setdefault(ckey, 0)
                results[key][ckey] += cvalue

    # now print results
    categories = list(results.keys())
    categories.sort()
    print( "\n Classified as: ")
    header = " "
    subheader = " +"
    for category in categories:
        header += category + " "
        subheader += "----+"
    print (header)
    print (subheader)
    total = 0.0
    correct = 0.0
    for category in categories:
        row = category + " |"
        for c2 in categories:
            if c2 in results[category]:
                count = results[category][c2]
            else:
                count = 0
            row += " %2i |" % count
            total += count
            if c2 == category:
                correct += count
        print(row)
    print(subheader)
```

```
    print("\n%5.3f percent correct" %((correct * 100) /
total))
    print("total of %i instances" % total)

tenfold("mpgData", "class num num num num num comment")
```

با اجرای این برنامه، نتایجِ زیر حاصل می‌شود:

```
          Classified as:
            10   15   20   25   30   35   40   45
          +----+----+----+----+----+----+----+----+
10        |  5 |  8 |  0 |  0 |  0 |  0 |  0 |  0 |
15        |  8 | 63 | 14 |  1 |  0 |  0 |  0 |  0 |
20        |  0 | 14 | 67 |  8 |  5 |  1 |  1 |  0 |
25        |  0 |  1 | 13 | 35 | 22 |  6 |  1 |  1 |
30        |  0 |  1 |  3 | 17 | 21 | 14 |  5 |  2 |
35        |  0 |  0 |  2 |  7 | 10 | 13 |  5 |  1 |
40        |  0 |  0 |  1 |  0 |  5 |  5 |  0 |  0 |
45        |  0 |  0 |  0 |  2 |  1 |  1 |  0 |  2 |
          +----+----+----+----+----+----+----+----+

52.551 percent correct
total of 392 instances
```

آمار کاپا (Kappa Statistic)

ابتدای این فصل، چند سوال در مورد الگوریتم‌های طبقه‌بندی پرسیدیم که احتمالاً دوست داشته باشیم به آن‌ها پاسخ دهیم. مثلاً این سوال که این طبقه‌بند چقدر خوب است؟ همچنین ما معیار و روشِ ارزیابیِ خود را اصلاح کردیم و به اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای و ماتریسِ اغتشاش روی آوردیم. در مثالِ چند صفحه قبل، مشخص کردیم که الگوریتم طبقه‌بندی، برای پیش‌بینیِ «مایل به ازای هر گالن» برای یک اتومبیل، ۵۳٫۳۱۶٪ دقت داشت. ولی آیا ۵۳٫۳۱۶٪ به معنیِ این است که الگوریتمِ طبقه‌بندیِ ما خوب است یا خیر؟ برای پاسخ به این سوال، می‌خواهیم به نوعی آمار بپردازیم به نام آمارِ کاپا (Kappa statistic).

آمارِ کاپا، طبقه‌بند را با یک طبقه‌بندِ دیگر مقایسه می‌کند به طوری که طبقه‌بندِ دوم کاملاً بر اساسِ شانس ساخته شده باشد. برای این‌که نشان دهیم آمارِ کاپا چگونه کار می‌کند، با یک مثالِ ساده‌تر از «مایل به ازای هر گالن – mpg» شروع می‌کنم و برای این‌کار دوباره به مجموعه‌ی داده‌ی زنانِ ورزشکار می‌پردازم. شکلِ زیر، نتایجِ یک الگوریتمِ طبقه‌بندی برای ورزشکاران زن است:

	gymnast	basketball player	marathoner	TOTALS
gymnast	35	5	20	60
basketball player	0	88	12	100
marathoner	5	7	28	40
TOTALS	40	100	60	200

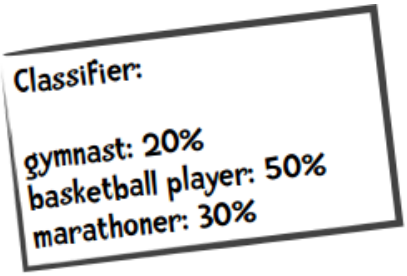
در شکلِ بالا، که در واقع **ماتریس اغتشاش (confusion matrix)** برای یک طبقه‌بندِ خاص است، مجموع را هم در ستون TOTALS نمایش دادم. برای مشخص کردنِ دقت (accuracy) ما جمعِ اعداد برای قطرِ اصلی را محاسبه می‌کنیم (۳۵ + ۸۸ + ۲۸ = ۱۵۱) و حاصلِ این عدد را بر تعدادِ تمامیِ نمونه‌ها یعنی ۲۰۰ عدد تقسیم می‌کنیم. پاسخ به صورت زیر می‌شود:

```
151 / 200 = .755
```

حالا می‌خواهم یک ماتریس اغتشاشِ دیگر را تولید کنم. این ماتریسِ اغتشاش بیان‌گر نتایجِ یک طبقه‌بندِ مبتنی بر شانس (random) است. یعنی طبقه‌بندی که پیش‌بینی‌هایش به جای یادگیری، کاملاً تصادفی است. در ابتدا، یک کپی از ماتریسِ بالا را ایجاد می‌کنیم و فقط مجموع (TOTALS) را برای آن‌ها نگه می‌داریم:

	gymnast	basketball player	marathoner	TOTALS
gymnast				60
basketball player				100
marathoner				40
TOTALS	40	100	60	200

با مشاهده‌ی سطرِ پایینی، مشاهده می‌کنیم که در ۵۰ درصدِ موارد (۱۰۰ مورد از ۲۰۰ مورد)، طبقه‌بندِ ما، بازیکنی را به عنوانِ بازیکنِ بسکتبال انتخاب کرده‌است. در ۲۰ درصدِ موارد (۴۰ مورد از ۲۰۰ مورد) ژیمیناستیک‌کار و در ۳۰ درصدِ موارد دوی ماراتن را برای ورزشکاران انتخاب کرده است.



حالا ما از این درصدها برای پر کردنِ بقیه‌ی جدول استفاده خواهیم کرد. الان تعداد ۶۰ ژیمیناستیک‌کار به صورت واقعی حضور دارند. الگوریتمِ طبق‌بندیِ ما که به صورت تصادفی کار می‌کند، ۲۰ درصد از این افراد را، ژیمیناستیک‌کار طبقه‌بندی می‌کند. پس ۲۰ درصدِ ۶۰، می‌شود ۱۲ و ما عددِ ۱۲ را در جدول قرار می‌دهیم. بعد از آن، ۵۰ درصد از این افراد را به عنوانِ بازیکنِ بسکتبال (۵۰ درصدِ ۶۰ که می‌شود ۳۰)، و ۳۰ درصدِ آن‌ها را ورزشکارانِ دوی ماراتن (۳۰ درصدِ ۶۰ که می‌شود ۱۸) طبقه‌بندی می‌کند.

	gymnast	basketball player	marathoner	TOTALS
gymnast	12	30	18	60
basketball player				100
marathoner				40
TOTALS	40	100	60	200

و به همین صورت ادامه می‌دهیم. تعدادِ ۱۰۰ بازیکن بسکتبال حضور دارند. طبقه‌بندِ تصادفی، ۲۰ درصد از آن‌ها را به عنوان ژیمیناستیک‌کار، ۵۰ درصد را به عنوان بازیکنِ بسکتبال و ۳۰ درصد را به عنوانِ ورزشکار دوی مارتن طبقه‌بندی می‌کند و به همین صورت تا آخر تمامیِ خانه‌های ماتریس را پر می‌کنیم:

	gymnast	basketball player	marathoner	TOTALS
gymnast	12	30	18	60
basketball player	20	50	30	100
marathoner	8	20	12	40
TOTALS	40	100	60	200

برای مشخص کردنِ دقتِ این روش که به صورت تصادفی عمل کرده‌است، بایستی مانندِ قبل، تمامی اعداد در ستونِ اصلی را جمع کرده و تقسیم بر کلِ نمونه‌ها کنیم.

$$P(r) = \frac{12+50+12}{200} = \frac{74}{200} = .37$$

حالا، آمارِ کاپا، به ما می‌گوید الگوریتم طبقه‌بندِ ما، چقدر بهتر از یک الگوریتمِ طبقه‌بندِ تصادفی کار می‌کند. فرمول کاپا به صورت زیر است:

$$k = \frac{P(c) - P(r)}{1 - P(r)}$$

که در آن P(c) دقتِ الگوریتمِ طبقه‌بندِ واقعی (طبقه‌بندِ طراحی شده توسطِ ما) است و P(r) دقتِ الگوریتمِ طبقه‌بندِ تصادفی است. در این مثال، دقتِ الگوریتمِ طبقه‌بندِ واقعی، برای ۰٫۷۵۵ شد و دقتِ طبقه‌بندِ تصادفی برابرِ ۰٫۳۷ شد. پس:

$$k = \frac{0.755 - 0.37}{1 - 0.37} = 0.61$$

حالا چگونه عددِ ۰٫۶۱ را ارزیابی کنیم؟ آیا این به آن معناست که الگوریتمی که طراحی کرده‌ایم ضعیف است؟ یا به معنی این است که الگوریتمِ ما قوی و یا حتی عالی است؟ در زیر، نموداری را مشاهده می‌کنید که به ما در تفسیر عددِ کارپا کمک می‌کند:

0 < : یعنی طبقه‌بندِ ما خوب نیست، و دقتِ آن کمتر از طبقه‌بندِ تصادفی شده است

0.01-0.20: خیلی کم خوب

0.21-0.40: عملکرد منصفانه

0.41-0.60: عملکرد معقول

0.61-0.80: به صورت قابل ملاحظه‌ای خوب

0.81-1.00: عملکرد نزدیک به عالی

منبع:

Landis, JR, Koch, GG. 1977. The measurement of observer agreement for categorical data. Biometrics 33:159-74

مدادتان را تیز کنید

فرض کنید ما یک طبقه‌بندِ نادان داریم که رشته‌ی دانشجویانِ جاریِ دانشگاه را بر اساس ۱۰ فیلمی که دیده‌اند، بررسی می‌کنند. ما مجموعه‌ی داده‌ای شامل ۶۰۰ دانشجو داریم. این دانشجویان در رشته‌های علوم کامپیوتر (cs)، علوم آموزشی (ed)، زبان انگلیسی (eng) و روانشناسی (psych) مشغول به تحصیل هستند. ماتریسِ اغتشاش برای این دانشجویان در زیر نمایش داده شده است. آیا شما می‌توانید آمارِ کاپا (Kappa statistic) را برای این ماتریس محاسبه کرده و آن را تفسیر کنید؟

	predicted major				
	cs	ed	eng	psych	Total
cs	50	8	15	7	
ed	0	75	12	33	
eng	5	12	123	30	
psych	5	25	30	170	

مدادتان را تیز کنید – راه‌حل

الگوریتمِ طبقه‌بندِ ما برا اساس این ماتریس اغتشاش، چه کیفیتی دارد؟ آیا شما می‌توانید آمارِ کاپا (Kappa Statistic) را برای این ماتریس محاسبه کرده و آن را تفسیر کنید؟

در ابتدا بایستی جمع تمامیِ ستون‌ها را داشته باشیم:

	cs	ed	eng	psych	TOTAL
SUM	60	120	180	240	600
%	10%	20%	30%	40%	100%

در قدمِ بعدی، بایستی ماتریسِ اغتشاش را برای طبقه‌بندِ تصادفی بسازیم:

predicted major

	cs	ed	eng	psych	Total
cs	8	16	24	32	80
ed	12	24	36	48	120
eng	17	34	51	68	170
psych	23	46	69	92	230
Total	60	120	180	240	600

دقتِ این طبقه‌بندِ تصادفی، برابرِ ۰,۲۹۲ است که مانند زیر محاسبه می‌شود:

(8 + 24 + 51 + 92) / 600 = (175 / 600) = 0.292

دقتِ طبقه‌بندِ ما برابر ۰,۶۹۷ شده است

دقتِ طبقه‌بندِ تصادفی هم برابر ۰,۲۹۲ شده است

آمارِ کاپا به صورت زیر برای این طبقه‌بند محاسبه می‌شود:

$$k = \frac{P(c) - P(r)}{1 - P(r)}$$

$$k = \frac{0.697 - 0.292}{1 - 0.292} = 0.572$$

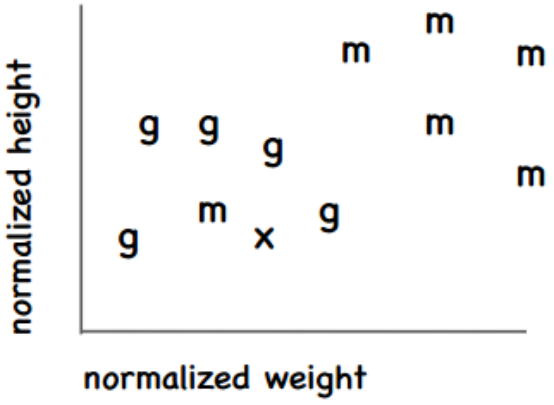
و تفسیرِ آن به این معناست که الگوریتمِ ما به صورت معقولی خوب عمل کرده است.

بهبودِ الگوریتمِ نزدیک‌ترین همسایه (Nearest Neighbor)

یک مثالِ بدیهی از یک طبقه‌بند، طبقه‌بندِ ساده (rote classifier) است – یعنی طبقه‌بندی که بدون فکر و تحلیلِ خاصی تصمیم‌گیری می‌کند. این طبقه‌بند تمامیِ مجموعه‌ی آموزشی را در حافظه‌ی خود ذخیره می‌کند و یک نمونه‌ی جدید را فقط وقتی طبقه‌بندی می‌کند که این نمونه دقیقاً مانندِ نمونه‌های موجود در حافظه (یعنی همان نمونه‌های آموزشی) باشد. اگر ما طبقه‌بندها را فقط بر اساس داده‌های موجود در مجموعه‌ی آموزشی، ارزیابی کنیم، الگوریتمِ طبقه‌بندِ ساده (rote classifier) همیشه دقتِ ۱۰۰ درصدی را کسب می‌کند. در دنیای واقعی، طبقه‌بندِ ساده، انتخابِ خوبی نیست، چون همواره نمونه‌هایی وجود دارند که می‌خواهیم آن‌ها را طبقه‌بندی کنیم در حالی‌که در مجموعه‌ی آموزشی، نمونه‌های دقیقاً مشابهِ آن‌ها، وجود ندارد. الگوریتمِ نزدیک‌ترین همسایه (nearest neighbor) را می‌توان به عنوان توسعه‌ای از الگوریتمِ طبقه‌بندِ ساده در نظر گرفت. در این الگوریتم (نزدیک‌ترین همسایه)، به جای این‌که به دنبال نمونه‌هایی دقیقاً مشابه برای طبقه‌بندیِ نمونه‌های جدید بگردیم، به دنبال نمونه‌هایی در مجموعه‌ی آموزشی می‌گردیم که به نمونه‌ی جدید، نزدیک‌تر هستند. Pang Ning Tan، Mecheal Steinback و Vipin Kumar در کتاب خود با نامِ «مقدمه‌ای بر داده‌کاوی» به این صورت این روش را توضیح داده‌اند: اگر یک چیز، مانند

اردک راه برود، مانند اردک صدا در بیاورد، و از نظر ظاهری هم شبیه به اردک باشد، پس احتمالاً این چیز، یک اردک است.

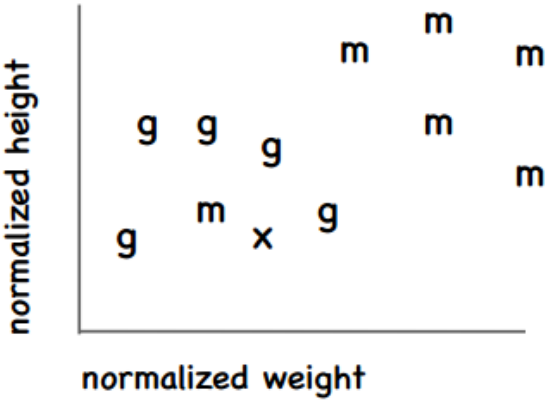
یکی از مشکلاتِ الگوریتمِ نزدیک‌ترین همسایه هنگامی اتفاق می‌افتد که ما داده‌های پَرت (outliers) داریم. اجازه بدهیم با مثالی، این موضوع را روشن کنم. بیایید به مثالِ ورزشکارانِ زن برگردیم. این‌بار فقط می‌خواهیم به ورزش‌های ژیمیناستیک و دوی ماراتن نگاهی بیندازیم. فرض کنید ما یک ورزشکارِ زنِ قد کوتاه و سبک‌وزن داریم که در رشته‌ی دوی ماراتن شرکت کرده است. اگر بخواهیم داده‌ها را بر روی نمودار رسم کنیم، چیزی مانندِ شکل زیر می‌شود (m به معنای ورزشکارِ دوی ماراتن و g به معنای ورزشکار ژیمیناستیک است):



محور عمودی، قد نرمال‌سازی شده (normalized height) را نمایش می‌دهد و محور افقی، وزن نرمال‌سازی شده (normalized weight). همان‌طور که مشاهده می‌کنیم، آن ورزشکارِ دوی ماراتن که قدِ کوتاهی داشت و سبک‌وزن هم بود، در میانِ ورزشکارانِ ژیمیناستیک خیلی تنها قرار گرفت. حالا فرض کنید x یک نمونه‌ی جدید است که می‌خواهیم طبقه‌ی آن را توسط الگوریتمِ طبقه‌بندی، مشخص کنیم. نزدیک‌ترین همسایه به x، همان داده‌ی پَرتِ m است، پس x به طبقه‌ی «دوی مارتن - m» تعلق می‌گیرد. ولی با یک نگاهِ چشمی به نمودارِ بالا، می‌توانیم بفهمیم که x بیشتر به یک ورزشکارِ ژیمیناستیک نزدیک است، چون در بینِ gها احاطه شده است.

K نزدیک‌ترین همسایه (KNN)

یکی از راه‌های بهبودِ روشِ نزدیک‌ترین همسایه این است که به جای نگاه کردن به یکی از همسایه‌ها که نزدیک‌ترینِ آن‌ها هست، به چند همسایه‌ی نزدیک از یک نمونه، نگاه کنیم – به این روش k نزدیک‌ترین همسایه یا KNN گفته می‌شود. در روشِ KNN هر کدام از همسایه‌ها یک رای می‌دهند و الگوریتم نمونه‌ها را بر اساسِ رای‌های همسایه‌های خود، طبقه‌بندی می‌کند. در واقع طبقه‌ای که بیشترین رای را در میانِ همسایه‌ها آورده باشد، طبقه‌ی پیش‌بینی شده برای نمونه‌ی جدید می‌شود. برای مثال، فرض کنید ما از الگوریتمِ سه نزدیک‌ترین همسایه (k=3) استفاده می‌کنیم. در موردِ مثالِ بالا، دو رای برای ژیمیناستیک‌کاران و یک رای برای ورزشکار دوی ماراتن داشتیم، پس الگوریتم، x را یک ورزشکار ژیمیناستیک معرفی می‌کند.



با این تفاسیر هنگامی که می‌خواهیم یک طبقه‌ی گسسته (discrete class) برای مثال دوی ماراتن، ژیمیناستیک یا بسکتبال را پیش‌بینی کنیم، می‌توانیم از این روشِ رای‌گیری استفاده کنیم. **طبقه یا کلاسی که بیشترین رای را داشته باشد، به عنوانِ طبقه‌ی پیش‌بینی شده برای نمونه‌ی مورد پیش‌بینی، انتخاب می‌شود.** اگر تعدادِ رای‌ها مساوی بود، طبقه یا کلاس، به صورت تصادفی از بین آن طبقه‌هایی که رایشان مساوی شده است، انتخاب می‌شود. ولی هنگامی که می‌خواهیم یک مقدارِ عددی (و نه گسسته) را پیش‌بینی کنیم، مثلاً اگر بخواهیم پیش‌بینی کنیم که یک شخص چه امتیازی به یک گروهِ موسیقی می‌دهد، می‌توانیم تاثیر رای‌ها را بین همسایه‌ها تقسیم کنیم تا به یک مقدارِ **مبتنی بر وزن–مسافت (distance-weighted)** برسیم. یعنی رایِ همسایه‌های نزدیک‌تر، تاثیرِ بیشتری داشته باشد. اجازه بدهید کمی این بحث را بیشتر باز کنم. فرض کنید می‌خواهیم پیش‌بینی کنیم که Ben چقدر گروهِ موسیقیِ Funky Meters را دوست دارد. سه همسایه‌ی نزدیک برای Ben نیز، Sally، Tara، و Jade هستند. در زیر فاصله‌ی این افراد از Ben و همچنین امتیازی که آن‌ها به گروهِ موسیقیِ Funky Meters داده‌اند را مشاهده می‌کنید:

User (کاربر)	Distance (مسافت)	Rating (امتیاز)
Sally	5	4
Tara	10	5
Jade	15	5

همان‌طور که مشخص است، Sally نزدیک‌ترین شخص به Ben است و به گروهِ Funky Meters امتیازِ ۴ داده‌است. چون می‌خواهم به شخصی که به Ben نزدیک‌تر است وزنِ بیشتر بدهم، اولین قدم اینست که معیارِ فاصله را طوری تغییر دهم که بیشترین عدد، نزدیک‌ترین شخص به Ben باشد یعنی بیشترین عدد، وزن بیشتری می‌گیرد. برای این‌کار می‌توانیم از معکوسِ فاصله استفاده کنیم (برای این‌کار کافیست عددِ ۱ را بر آن فاصله تقسیم کنیم). مثلاً برای Sally که فاصله‌اش با Ben برابر ۵ است، به صورت زیر، معکوسِ فاصله، محاسبه می‌شود:

$$\frac{1}{5} = 0.2$$

(معکوس مسافت)

User (کاربر)	Inverse Distance	Rating (امتیاز)
Sally	0.2	4
Tara	0.1	5
Jade	0.067	5

حالا باید هر کدام از این فاصله‌های معکوس شده را بر جمعِ آن‌ها تقسیم کنم (تا نرمال شوند). جمعِ تمامیِ معکوس‌ها برابر ۰,۳۶۷ می‌شود.

$$0.2 + 0.1 + 0.067 = 0.367$$

User (کاربر)	Influence (تاثیر)	Rating (امتیاز)
Sally	0.545	4
Tara	0.272	5
Jade	0.183	5

دو چیز را باید مد نظر داشته باشیم. اول اینکه جمعِ تمامیِ مقادیرِ تاثیرات (influence) برابرِ ۱ شود. ثانیاً در اعدادِ فاصله‌ی اصلی، Sally به میزان ۲ برابر بیشتر از Tara به Ben نزدیک بود و این نزدیک بودن در تاثیرِ آن‌ها نیز حفظ شده است. یعنی Sally دو برابر بیشتر از Tara بر Ben تاثیر می‌گذارد (چون دو برابر نزدیک‌تر بود). در آخر ما تاثیرِ هر فرد را در امتیازِ او ضرب کرده و مجموعِ آن‌ها را به عنوان نتیجه‌ی نهایی محاسبه می‌کنیم:

امتیازی که Ben به گروهِ موسیقیِ Funcky Meters می‌دهد:

$$= 0.545 \cdot 4 + 0.272 \cdot 5 + 0.183 \cdot 5$$
$$= 2.18 + 1.36 + 0.915 = 4.455$$

مدادتان را تیز کنید:

من می‌خواهم بدانم که Sofia چقدر گروهِ موسیقیِ Hiromi را دوست خواهد داشت. با توجه به داده‌های زیر و الگوریتمِ k نزدیک‌ترین همسایه با k=3، چه مقداری را پیش‌بینی می‌کنید؟

person (شخص)	distance from Sofia (فاصله تا Sofia)	rating for Hiromi (امتیاز به Hiromi)
Gabriela	4	3
Ethan	8	3
Jayden	10	5

مدادتان را تیز کنید – راه حل

اولین کاری که باید انجام دهیم اینست که معکوسِ هر کدام از فاصله‌ها را (در ستون **Inverse Distance**) محاسبه کنیم:

Person	Inverse Distance	Rating
Gabriela	1/4 = 0.25	3
Ethan	1/8 = 0.125	3
Jayden	1/10 = 0.1	5

مجموعِ معکوسِ فاصله‌ها برابر ۰,۴۷۵ می‌شود. قدمِ بعدی این است که تاثیرِ هر شخص را با تقسیم معکوسِ فاصله بر جمعِ این معکوس‌ها (در ستون **Influence**) محاسبه کنیم:

Person	Influence	Rating
Gabriela	0.526	3
Ethan	0.263	3
Jayden	0.211	5

در آخر، تمامیِ تاثیرات را در امتیازهایی که هر کدام از اشخاص داده‌اند، ضرب می‌کنیم و همه‌ی آن‌ها را با یکدیگر جمع می‌کنیم:

$$= (0.526 \times \cdot\ 3) + (0.263 \times 3) + (0.211 \times 5)$$
$$= 1.578 + 0.789 + 1.055 = 3.422$$

یک مجموعه‌ی داده‌ی جدید و یک چالش

وقت این شده که به یک مجموعه‌ی داده‌ی جدید نگاهی بیندازیم. مجموعه‌ی داده‌ی دیبابت هندی‌های پیما -Pima (یک قبیله‌ی هندی) که توسط مرکزِ ملیِ دیبایت، گوارش و کلیه‌ی ایالات متحده جمع‌آوری است.

به طرز شگفت‌آوری، بیش از ۳۰ درصدِ قبلیه‌ی Pima دیابت دارند. برعکس، نرخِ دیابت در ایالات متحده برابر ۸,۳ درصد و در چین برابر ۴,۲ درصد است.

هر نمونه در این مجموعه‌ی داده، بیان‌گرِ اطلاعاتِ یک زنِ عضوِ قبلیه‌ی Pima بالای سن ۲۱ سال است و هر کدام از آن‌ها به یکی از طبقه‌ها تعلق دارند: **فردی که ۵ سال است دیابت دارد**، یا **فردی که این طور نیست**. تعدادِ ۸ ویژگی نیز موجود است:

تعداد دفعاتی که این شخص حامله شده است
غلظت گلوکز پلاسما ۲ ساعته در تست تحمل قند خون
فشارِ خون دیاستولیک (mm Hg)

ضخامتِ پوست ماهیچه‌ی سه سر (mm)
انسولین سرم ۲ ساعته (mu U/ml)
شاخص توده‌ی بدنی (وزن به کیلو گرم تقسیم بر قد به متر به توانِ ۲)
تابع شجره‌نامه‌ی دیابت
سن (به سال)

قسمتی از داده‌ها را می‌توانید در زیر مشاهده کنید (ستونِ آخر طبقه یا کلاس را مشخص می‌کند، **۰ به معنای نداشتنِ دیابت** و **۱ به معنای داشتن دیابت** است)

2	99	52	15	94	24.6	0.637	21	0
3	83	58	31	18	34.3	0.336	25	0
5	139	80	35	160	31.6	0.361	25	1
3	170	64	37	225	34.5	0.356	30	1

پس برای مثال، اولین زن (در شکلِ بالا)، دو بچه دارد، دارای غلظتِ گلوکز پلاسما برابر ۹۹ است، فشار خونِ دیاستولیک برابر ۵۲ دارد و به همین ترتیب تا آخر.

کد بزنید – قسمت اول

دو فایل در وب‌سایتِ ما موجود است. pimaSmall.zip که یک فایل فشرده شامل ۱۰۰ نمونه از داده‌ها بوده و به ۱۰ فایل تقسیم شده است (۱۰ سطلِ جدا برای عمل اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای). و همچنین فایلِ pima.zip که فایلِ کامل، شامل ۳۹۳ نمونه است. هنگامی که از مجموعه‌ی داده‌ی pimaSmall همراه با الگوریتمِ طبقه‌بندِ نزدیک‌ترین

همسایه (که در فصل قبلی ساختیم) همراه با اعتبارسنجیِ متقابلِ ۱۰-تکه‌ای استفاده می‌کنم، به نتایج زیر می‌رسم:

```
          Classified as:
            0   1
          +----+----+
0         | 45 | 14 |
1         | 27 | 14 |
          +----+----+
```

(تابع **heapq.nsmallest(n, list)** لیستی از **n** کوچک‌ترین آیتم‌ها را برمی‌گرداند)

حالا این تکلیفِ شماست:
کدِ الگوریتمِ طبقه‌بندی را از وب‌سایت دانلود کرده و الگوریتم **KNN** را پیاده‌سازی کنید. اجازه دهید ما تابع اولیه (**initializer**) را تغییر دهیم تا این تابع بتواند آرگومانِ **k** را هم بپذیرد:

```
def __init__(self, bucketPrefix, testBucketNumber, dataFormat, k):
```

تابع چیزی شبیه به شکل زیر می‌شود:

```
def knn(self, itemVector):
```

این تابع بهتر است از **self.k** استفاده کند (یادتان باشد که مقدار **k** را در تابعِ **init** مقداردهی کنید) و در نهایت بایستی طبقه‌ی مربوطه (۰ و ۱ برای دیابتی نبودن و دیابتی بودن) را برگرداند. البته باید تابعِ **tenfold** را نیز تغییر داده تا بتوانید **k** را به **initializer** ارجاع دهید.

کد بزنید – راه‌حل

تابعِ __init__ را به صورت زیر تغییر می‌دهم:

```
def __init__(self, bucketPrefix, testBucketNumber, dataFormat, k):
        self.k = k
        ...
```

و حالا تابعِ knn:

```
def knn(self, itemVector):
    """returns the predicted class of itemVector using k
    Nearest Neighbors"""
    # changed from min to heapq.nsmallest to get the
    # k closest neighbors
    neighbors = heapq.nsmallest(self.k,
            [(self.manhattan(itemVector, item[1]), item)
            for item in self.data])
    # each neighbor gets a vote
    results = {}
    for neighbor in neighbors:
        theClass = neighbor[1][0]
        results.setdefault(theClass, 0)
        results[theClass] += 1
    resultList = sorted([(i[1], i[0]) for i in results.i
tems()],
            reverse=True)
    #get all the classes that have the maximum votes
    maxVotes = resultList[0][0]
    possibleAnswers = [i[1] for i in resultList if i[0]
== maxVotes]
    # randomly select one of the classes that received t
he max votes
    answer = random.choice(possibleAnswers)
    return( answer)
```

تابعِ tenfold را هم کمی تغییر می‌دهیم:

```
def tenfold(bucketPrefix, dataFormat, k):
    results = {}
    for i in range(1, 11):
        c = Classifier(bucketPrefix, i, dataFormat, k)
```

. . .

می‌توانید کدِ کامل را از سایتِ guidetodatamining.com و یا chistio.ir دانلود کنید. یادتان باشد که این فقط یکی از راه‌های پیاده‌سازیِ این تابع است و لزوماً بهترین راه نیست.

کد بزنید – قسمتِ دوم

کدام تاثیرِ بیشتری دارد؟ این‌که داده‌های بیشتری داشته باشیم (با توجه به نتایج به دست آمده از pimaSmall و pima) یا این‌که الگوریتمِ بهتری داشته باشیم (با توجه به k=1 تا k=3)؟

کد بزنید- قسمت دوم – راه حل

در شکلِ زیر دقتِ به دست آمده، برای مقادیر مختلف را مشاهده می‌کنید:

	pimaSmall	pima
k=1	59.00%	71.247%
k=3	61.00%	72.519%

پس به نظر می‌رسد که سه برابر کردنِ داده‌ها، بسیار بیشتر دقت را افزایش می‌دهد تا این‌که الگوریتم را بهینه‌تر کنیم.

مدادتان را تیز کنید

خب! ۷۲٫۵۱۹٪ به نظر خوب می‌رسد. ولی آیا واقعاً این طور است؟ آمارِ کاپا را برای این نتایج حساب کنید:

	no diabetes	diabetes
no diabetes	219	44
diabetes	64	66

Performance:

- ☐ slightly good
- ☐ fair
- ☐ moderate
- ☐ substantially good
- ☐ near perfect

مدادتان را تیز کنید – راه حل

	no diabetes	diabetes	TOTAL
no diabetes	219	44	263
diabetes	64	66	130
TOTAL	283	110	393
ratio	0.7201	0.2799	

(سطر ratio نسبت را نشان می‌دهد)

طبقه‌بندِ تصادفی (random (r) classifier):

	no diabetes	diabetes
no diabetes	189.39	73.61
diabetes	93.61	36.39

دقت (accuracy):

$$p(r) = \frac{189.39 - 36.61}{393} = 0.5745$$

$$K = \frac{P(c) - P(r)}{1 = P(r)} = \frac{0.72519 - 0.5745}{1 - 0.5745} = 0.35415$$

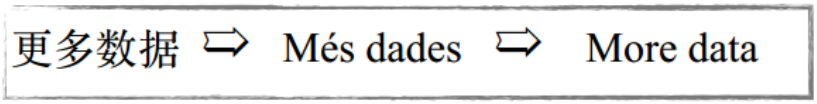
پس متوجه می‌شویم که این طبقه‌بند، عملکرد نسبتاً خوب (و نه خیلی خوبی) داشته است.

داده‌های بیشتر، الگوریتم‌های بهتر و یک اتوبوسِ خراب

چند سال پیش، در کنفرانسی در مکزیکوسیتی بودم. این کنفرانس تقریباً یک کنفرانس غیر معمول بود چون یک روزِ آن به ارائه‌ها سپری می‌شد و روز دیگر بایستی به تور می‌رفتیم (دشتِ سلطان، خرابه‌های Inca و...). روزی که در تور بودیم بایستی مسافتِ زیادی را با اتوبوس می‌رفتیم و اتوبوس هم زیاد حالِ خوشی نداشت. در نتیجه، عده‌ای از ما که PhD داشتیم کنار جاده ایستادیم و با یکدیگر صحبت کردیم تا اتوبوس تعمیر شود. این تبادلاتِ کنارِ جاده‌ای، نقطه‌ی عطفی در این کنفرانس برای من بود. یکی از افرادی که با او صحبت کردم شخصی به نامِ Eric Brill بود. او به خاطر توسعه‌ی نرم‌افزاری به اسمِ Brill tagger (برچسب زننده‌ی Brill) شهرت داشت که عملیاتِ برچسب‌زنیِ اجزای زبان (Part of Speech) را انجام می‌داد. شبیه به کاری که ما در این چند فصل انجام دادیم، برچسب‌زننده‌ی Brill، داده‌ها را طبقه‌بندی می‌کرد – در این مورد، او کلمات را بر اساس

اجزای زبان (اسم، فعل، و...) برچسب می‌زد و طبقه‌بندی می‌کرد. الگوریتمِ ارائه شده توسطِ او خیلی بهتر از الگوریتم‌های قبلی در همان حوزه کارکرد داشت (و به همین خاطر Brill در حوزه‌ی پردازش زبانِ طبیعی (NLP) به شهرت رسیده بود). در کنارِ جاده، با Eric Brill درباره‌ی بهبودِ عملکردِ الگوریتم‌ها صحبت کردم. از نقطه نظرِ او، با اضافه کردنِ بیشترِ داده‌ها در مجموعه‌ی آموزشی، شما به نتایجِ بهتری دست پیدا می‌کنید تا این‌که بخواهید الگوریتمتان را بهبود دهید. در واقع، او احساس می‌کرد که اگر همان الگوریتمِ برچسب‌زنیِ اجزای زبان (Part of Speech) را نگه دارد و فقط تعدادِ داده‌های آموزشی را افزایش دهد، بهبودِ بیشتری نسبت به الگویتمِ مشهورش رخ خواهد داد. البته که می‌گفت، شما نمی‌توانید فقط با جمع کردنِ داده‌های بیشتر PhD بگیرید ولی با بهبودِ یک الگوریتم می‌توانید!

یک مثالِ دیگر. در مسابقاتِ مختلفِ ترجمه‌ی ماشینی (که کامپیوتر عملیاتِ ترجمه را انجام می‌دهد)، گوگل معمولاً در صدر می‌ایستد. قبول دارم گوگل، آدم‌های با استعداد و باهوشی برای توسعه‌ی الگوریتم‌هایش دارد، اما قسمت زیادی از این سیطره‌ی گوگل، به خاطرِ مجموعه‌ی داده‌ی بسیار بزرگی است که از سراسر وب جمع‌آوری کرده است.



منظورم این نیست که بهترین الگوریتم را برای کارتان انتخاب نکنید. همان‌طور که دیدیم، انتخابِ الگوریتمِ مناسب باعثِ بهبودِ زیادی خواهد شد. ولی در عین حال، اگر بخواهید یک مسئله را حل کنید (به جای انتشارِ یک مقاله‌ی علمی) ارزشش را ندارد که وقتتان را خیلی صرفِ تحقیق و انتخابِ الگوریتم مناسب کنید. **بهتر است وقتتان را صرف به دست آوردنِ داده‌های بیشتر کنید.**

با این‌که در موردِ اهمیتِ به دست آوردنِ داده‌ها صحبت کردم، راهم را با معرفی الگوریتم‌های جدید در فصولِ آینده ادامه خواهم داد!

> معمولاً از الگوریتمِ KNN برای طبقه‌بندیِ مسائلِ زیر استفاده می‌شود:
> پیشنهادِ محصولِ جدید در سایتی مانند آمازون
> ارزیابیِ ریسک اعتباریِ یک مشتری

طبقه‌بندی پوشش زمین با استفاده از تحلیل تصاویر
تشخیص چهره
تشخیص جنسیتِ افراد در تصویر
پیشنهادِ صفحاتِ وب
پیشنهادِ بسته‌های تفریحی و مسافرتی

فصل ٦. بِیز ساده

احتمالات و بیز ساده (Naïve Bayes)

اجازه بدهید برای بارِ چندم، به مثالِ ورزشکارانِ زن برگردیم. فرض کنید از شما سوال کنم که Brittney Griner چه ورزشی را انجام می‌دهد (ژیمیناستیک، دوی ماراتن یا بسکتبال) و به شما این اطلاعات را بدهم که او ۶ فوت و ۸ اینچ قد، و ۲۰۷ پوند وزن دارد. تصورِ من این است که شما بسکتبال را انتخاب می‌کنید و اگر از شما بپرسم که چقدر از انتخابِ خود اطمینان دارید؟ احتمالاً بگویید «تقریباً مطمئنم».

حالا از شما می‌پرسم که Heather Zurich (که در تصویر مشخص است) چه ورزشی را انجام می‌دهد؟ او ۶ فوت و ۱ اینچ قد و ۱۷۶ پوند وزن دارد. الان دقیقاً نمی‌دانم که شما چه پاسخی خواهید داد. ممکن است بگویید بازیکن بسکتبال است و اگر بپرسم که چقدر از این انتخاب مطمئن هستید، اطمینانِ کمتری نسبت به انتخابِ قبلی یعنیBrittney Griner خواهید داشت. این شخص ممکن است یک ورزشکارِ دوی ماراتن با قدی نسبتاً بلند برای این رشته باشد.

نهایتاً از شما می‌پرسم که Yumiko Hara در کدام رشته‌ی ورزشی، بازی می‌کند؟ او ۵ فوت و ۴ اینچ قد و ۹۵ پوند وزن دارد. فرض کنید شما ورزشِ ژیمیناستیک را انتخاب می‌کنید و باز هم من از شما سوال

می‌کنم که چقدر از انتخابِ خود اطمینان دارید. ممکن است شما چیزی معادلِ «خیلی مطمئن نیستم» را به من پاسخ دهید. چند ورزشکارِ دوی ماراتن نیز قد و وزنی شبیه به این داشته‌اند.

با استفاده از خانواده‌ی الگوریتم‌های نزدیک‌ترین همسایه (nearest neighbor) نمی‌توان به سادگی، اطمینانِ طبقه‌بندی را محاسبه کرد. در این حالی است که با روش‌های طبقه‌بندیِ مبتنی بر احتمالات – مانند روش‌های **بیزین (Bayesian)** – نه تنها می‌توانیم طبقه‌بندی را انجام دهیم، بلکه می‌توانیم طبقه‌بندی‌های احتمالی (probabilistic classifications) را نیز در اختیار داشته باشیم. مثلاً بگوییم این ورزشکار، به احتمالِ ۸۰ درصد بازیکنِ بسکتبال است. یا این‌که بگوییم این بیمار به احتمال ۴۰ درصد در پنج سالِ آینده به بیماریِ دیابت مبتلا خواهد شد. یا مثلاً، احتمال بارش باران در شهرِ لاس کروسز (Las Cruces) در بیست و چهار ساعتِ آینده، برابر ۱۰ درصد است.

روش‌های مبتنی بر نزدیک‌ترین همسایه به روش‌های **تنبل (lazy learners)** معروف هستند. دلیل تنبل بودنِ این روش‌ها این است که وقتی یک مجموعه داده به آن‌ها می‌دهیم، به صورت ساده، آن‌ها فقط این مجموعه‌ی داده را ذخیره می‌کنند (یعنی فقط خودِ مجموعه‌ی داده را بدون ساخت مدلِ خاصی، به یاد می‌سپارند). و هر موقع که بخواهیم یک طبقه‌بندیِ جدید انجام دهیم، این الگوریتم‌های تنبل بایستی تمامیِ مجموعه‌ی داده را بررسی کنند. مثلاً اگر ۱۰۰ هزار قطعه‌ی موسیقی در مجموعه داده‌های آموزشی داشته باشیم، این روش‌ها بایستی تمامیِ این ۱۰۰ هزار قطعه را برای هر بار طبقه‌بندی، جستجو کنند.

اما در مقابل، روش‌های بیزین (Bayesian)، به **روش‌های مشتاق (eager learners)** معروف هستند. هنگامی که یک مجموعه‌ی

داده را به یک یادگیرنده‌ی مشتاق می‌دهیم، این روش‌ها، فوراً داده‌ها را تحلیل کرده و یک مُدل از داده‌ها می‌سازند. حال اگر این روش‌ها بخواهند یک نمونه را طبقه‌بندی کنند، از این **مدلِ از پیش ساخته شده** استفاده کرده و عملیات طبقه‌بندی را انجام می‌دهند. برای همین این روش‌ها، با سرعتِ بیشتری می‌توانند نمونه‌های جدید را طبقه‌بندی کنند.
توانایی طبقه بندیِ احتمالی، و همچنین یادگیریِ مشتاق، دو مزیت برای روش‌های بیزین، شمرده می‌شوند.

احتمالات

فرضِ من این است که شما یک دانشِ کمی در موردِ احتمالات دارید. یک سکه می‌اندازیم (شیر یا خط). احتمالِ این‌که شیر بیاید چقدر است؟ یک تاس (۱ تا ۶) می‌اندازیم، احتمالِ این‌که عددِ ۱ بیاید چقدر است؟ منظورم از احتمال، این جور چیزهاست. یک فردِ ۱۹ ساله را انتخاب می‌کنم و از شما می‌خواهم به من بگویید که با احتمالِ چند درصد، این فرد، مونث است. شما خواهید گفت ۵۰ درصد. این‌ها، مثال‌هایی از احتمالاتِ پیشین (prior probability) بودند که با P(h) مشخص می‌شوند – یعنی **احتمالِ فرضیه‌ی h**.
پس مثلاً برای یک سکه:
احتمال شیر آمدن = P(heads) = 0.5
برای یک تاسِ ۶ تایی، احتمال این‌که عددِ ۱ بیاید:
P(1) = 1/6
اگر یک جمعیت از افرادِ ۱۹ ساله، به صورت مساوی مرد یا زن داشته باشیم. احتمالِ این‌که نفرِ انتخاب شده مونث باشد:
P(female) = 0.5

حالا فرض کنید بگویم از میان جمعیتی از زنان و مردان، یکی را انتخاب می‌کنم. این شخص دانشجوی معماری در مدرسه‌ی Frank Lloyd Wright در شهرِ آریزونا است. شما با یک جستجوی گوگل، متوجه می‌شوید که در این مدرسه، ۸۶ درصدِ دانشجویان مونث و ۱۴ درصدِ دانشجویان مذکر هستند. پس می‌گویید احتمالِ این‌که این شخصِ انتخاب شده، مونث باشد، ۸۶ درصد است (نه ۵۰ درصد).

این چیزیست که ما به آن P(h | D) می‌گوییم– یعنی احتمالِ فرضیه‌ی h با مشاهده‌ی داده‌ی D.

برای مثال:

P(female | attends Frank Lloyd Wright School) = 0.86

(یعنی احتمال این‌که یک شخص مونث (female) باشد، با فرضِ اینکه بدانیم این شخص دانشجوی مدرسه‌ی Frank Lloyd Wright است)

فرمولِ کلی به صورت زیر است:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

یک مثال

در زیر لیستی از افراد به همراه نوع لپ‌تاپ (laptop) و تلفنِ همراهشان (phone) را آورده‌ام:

name	laptop	phone
Kate	PC	Android
Tom	PC	Android
Harry	PC	Android
Annika	Mac	iPhone
Naomi	Mac	Android
Joe	Mac	iPhone
Chakotay	Mac	iPhone
Neelix	Mac	Android
Kes	PC	iPhone
B'Elanna	Mac	iPhone

احتمال این که یک شخص، که به صورتِ تصادفی انتخاب شده است، از iPhone استفاده کند چقدر است؟

تعدادِ ۵ کاربر دارای iPhone در جدول هستند و تعداد کلِ کاربران نیز ۱۰ نفر است. پس:

$$P(iPhone) = \frac{5}{10} = 0.5$$

حال احتمالِ این‌که یک شخص، که به صورتِ تصادفی انتخاب شده است، از iPhone استفاده کند، با این شرط که بدانیم لپ‌تاپِ او Mac است، چیست؟

$$P(iPhone \mid mac) = \frac{P(mac \cap iPhone)}{P(mac)}$$

اولاً تعدادِ ۴ نفر وجود دارند که هم Mac دارند و هم iPhone. پس:

$$P(mac \cap iPhone) = \frac{4}{10} = 0.4$$

و احتمالِ این‌که یک شخص که به صورتِ تصادفی انتخاب شده باشد، لپ‌تاپ Mac داشته باشد به صورت زیر محاسبه می‌شود:

$$P(mac) = \frac{6}{10} = 0.6$$

پس احتمالِ این‌که آن شخصِ انتخاب شده، دارای iPhone باشد در حالی که بدانیم لپ‌تاپِ او Mac است، به صورت زیر محاسبه می‌شود:

$$P(iPhone \mid mac) = \frac{0.4}{0.6} = 0.667$$

البته این تعریفِ رسمیِ احتمالِ پسین (posterior probability) است. بعضی اوقات که بخواهیم این فرمول را پیاده‌سازی کنیم، فقط از تعداد (به صورتِ خام) استفاده می‌کنیم:

$$P(iPhone|mac) = \frac{\text{تعداد افرادی که از mac و iPhone استفاده می‌کنند}}{\text{تعدادی افرادی که از mac استفاده می‌کنند}}$$

$$P(iPhone \mid mac) = \frac{4}{6} = 0.667$$

مدادتان را تیز کنید
احتمالِ این‌که شخصی دارای Mac باشد، به شرطی که بدانیم او iPhone دارد چیست؟
یعنی P(mac|iPhone)؟

نکته:
اگر فکر می‌کنید به تمرین های بیشتری درباره‌ی احتمالاتِ مقدماتی نیاز دارید، لینک های موجود در سایتِ guidetotdatamining.com و chistio.ir را مشاهده کنید.

مدادتان را تیز کنید – راه حل
احتمالِ این‌که شخصی دارای Mac باشد، به شرطی که بدانیم او iPhone دارد چیست؟

$$P(mac \mid iPhone) = \frac{P(iPhone \cap mac)}{P(iPhone)}$$

$$= \frac{0.4}{0.5} = 0.8$$

یادگیری چند عبارت

p(h)، یعنی احتمالِ این‌که فرضیه‌ی h درست باشد، **احتمالِ پیشین (prior probability) برای h** نامیده می‌شود. پیشین به معنای این است که قبل از هر گونه شواهدی باشد. برای مثال، احتمالِ این‌که یک شخص، لپ‌تاپِ Mac داشته باشد ۰٫۶ است (شواهد ممکن است نشان دهند که این شخص همچنین یک iPhone هم داشته است)
p(h | d) را **احتمالِ پَسین (posterior probability) برای h** می‌نامند. یعنی بعد از مشاهده‌ی داده‌ی d، احتمالِ رخ دادنِ h چقدر است؟ برای مثال، بعد از این که متوجه شدیم شخصی دارای iPhone است، چقدر احتمال دارد که همین شخص لپ‌تاپِ Mac هم داشته باشد؟ همچنین به این احتمال، **احتمالِ شرطی (conditional probability)** نیز گفته می‌شود.

حالا اگر بخواهیم یک طبقه‌بندِ بیزین (Bayesian classifier) بسازیم، دو احتمالِ دیگر نیز نیاز داریم، P(D) و P(D | h). مثالِ زیر را برای فهمِ بهتر، نگاه کنید.

کارتِ خرید ماکروسافت

آیا می‌دانستید که ماکروسافت کارت‌های خریدِ خرده‌فروشی را به صورت هوشمند تولید می‌کند؟ بله، در واقع، ماکروسافت در قرارداد با شرکتی به نامِ چااوتیک مون (Chaotic Moon) برای توسعه‌ی این‌کارت‌ها اقدام کرده است. شعار این شرکت این است: ما از شما باهوش‌تریم. ما از شما خلاق‌تریم. البته شما می‌توانید فکر کنید این شرکت خیلی مغرور و پررو یا هر چیز دیگری است. این شرکت برای انجام این کار یک کارت خرید را با یک تبلت دارای سیستم‌عاملِ ویندوز ۸، یک کینکت، یک بلندگوی بلوتوثی برای این‌که کارت بتواند با شما صحبت کند، و یک پلتفرمِ رباتیِ موبایلی برای همراهیِ این کارت در سراسرِ فروشگاه با شما همراه کرده است.

شما با کارتِ اعتباری خود واردِ مغازه می‌شوید. کارت شما را می‌شناسد. تمامیِ خریدهای قبلیِ شما را هم می‌داند (همانندِ خرید هر کس دیگری در این فروشگاه).

تصور کنید که نرم‌افزارِ پشت این کارت، می‌خواهد تصمیم بگیرد که آیا تبلیغی از یک نوع «چای سبزی سِنشای ژاپنی» را به شما نشان دهد یا خیر. این نرم‌افزار فقط وقتی این تبلیغات را به شما نشان می‌دهد که احتمال دهد شما این چای را خواهید خرید.
نرم‌افزار پشت کارت، یک مجموعه‌ی داده‌ی کوچک را از تمامیِ خریداران جمع‌آوری کرده است. این مجموعه‌ی داده، در زیر قابل مشاهده است. (ستون Customer ID شماره‌ی مشتری را نشان می‌دهد، Zipcode کد پستی است، ستونِ bought organic product ستونی است که نشان می‌دهد آیا شخص محصولات ارگانیک خریده است یا خیر و ستون bought Sencha green tea نشان می‌دهد که آیا شخص چای سبز سنچا خریده است یا خیر)

Customer ID	Zipcode	bought organic produce?	bought Sencha green tea?
1	88005	Yes	Yes
2	88001	No	No
3	88001	Yes	Yes
4	88005	No	No
5	88003	Yes	No
6	88005	No	Yes
7	88005	No	No
8	88001	No	No
9	88005	Yes	Yes
10	88003	Yes	Yes

P(D)، احتمالِ این است که یک داده‌ی آموزشی مشاهده شده باشد. برای مثال، با نگاه به داده‌های جدولِ بالا، مشاهده می‌کنیم که احتمال این که کدپستی (zip code) برابرِ ۸۸۰۰۵ باشد، $5/10$ یا همان ۰٫۵ است.

P(88005) = 0.5

P(D | h) احتمالِ این است که داده‌ای رخ دهد با شرط آن‌که فرضیه‌ای داشته باشیم. برای مثال، احتمالِ این‌که کد پستی (zip code) برابر ۸۸۰۰۵ بوده به شرط آن‌که این شخص چای سبزِ سنچا را خرید باشد به صورت زیر است:

P(88005|Sencha Tea)

برای حل این مسئله، ما به مواردی نگاه می‌کنیم که شخصی چای سبز سنشا را خریده باشد. تعدادِ این افراد ۵ مورد است. ۳ نفر از آن‌ها، کدِ پستیِ ۸۸۰۰۵ را دارند. پس:

$$\mathbf{P(88005 \mid SenchaTea) = \frac{3}{6} = 0.6}$$

مدادتان را تیز کنید
احتمالِ این‌که کد پستی برابر ۸۸۰۰۵ باشد با این شرایط که بدانیم آن شخص چای سبزِ سنشا را نخریده است را به دست آورید؟

مدادتان را تیز کنید – راه حل
احتمالِ این‌که کد پستی برابر ۸۸۰۰۵ باشد با این شرایط که بدانیم آن شخص چای سبزِ سنشا را نخریده است را به دست آورید؟
۵ مورد وجود دارد که چای سبزِ سنچا را نخریده‌اند. ۲ مورد از آن‌ها، در محدوده‌ی کد پستیِ ۸۸۰۰۵ زندگی می‌کنند. پس:

$$\boldsymbol{P(88005 \mid \neg\, SenchaTea) = \frac{2}{5} = 0.4}$$

نشانه‌ی «¬» به معنای «منفی» است. مثلاً «$\neg\, SenchaTea$» به معنای نخریدنِ چای سبز سنچا است.

مدادتان را تیز کنید

این مثال، کلیدی برای فهمِ ادامه‌ی این فصل است، پس اجازه دهید کمی بیشتر با آن سر و کله بزنیم.

۱. احتمالِ این‌که شخصی در کد پستیِ ۸۸۰۰۱ باشد چقدر است (بدون دانستنِ چیزِ دیگری)؟

۲. احتمالِ این‌که شخص در کد پستیِ ۸۸۰۰۱ باشد چقدر است با این شرط که بدانیم این شخص چایِ سبزِ سنچا خریده است؟

۳. احتمالِ این‌که شخصی در کد پستیِ ۸۸۰۰۱ باشد چقدر است با این شرط که بدانیم او چای سبزِ سنچا را نخریده است؟

مدادتان را تیز کنید – راه حل

این مثال، کلیدی برای فهمِ ادامه‌ی این فصل است، پس اجازه دهید کمی بیشتر با آن سر و کله بزنیم.

۱. احتمالِ این‌که شخصی در کد پستیِ ۸۸۰۰۱ باشد چقدر است (بدون دانستنِ چیزِ دیگری)؟

تعدادِ ۱۰ نمونه در مجموعه‌ی داده‌هایمان داریم و فقط ۳ نفر از آن‌ها در کدپستیِ ۸۸۰۰۱ هستند. پس **P(88001)** برابر ۰٫۳ است.

۲. احتمالِ این‌که شخصی در کد پستیِ ۸۸۰۰۱ باشد چقدر است با این شرط که بدانیم این شخص چایِ سبزِ سنچا خریده است؟

تعداد ۵ نمونه هستند که چای سبزِ سنچا را خریده‌اند و فقط یکی از آن‌ها در محدوده‌ی کد پستیِ ۸۸۰۰۱ قرار دارد. پس:

$$P(88001 \mid SenchaTea) = \frac{1}{5} = 0.2$$

۳. احتمالِ این‌که شخصی در کد پستیِ ۸۸۰۰۱ باشد چقدر است با این شرط که بدانیم او چای سبزِ سنچا را نخریده‌است؟
تعداد ۵ نمونه هستند که چای سبزِ سنچا را نخریده‌اند و ۲ نفر از آن‌ها در محدوده‌ی کد پستیِ ۸۸۰۰۱ هستند. پس:

$$P(88001 \mid \neg SenchaTea) = \frac{2}{5} = 0.4$$

تئوریِ بیز (Bayes Theorem)

تئوریِ بیز، رابطه‌ی بین P(h)، P(h | D)، P(D) و P(D | h) را بیان می‌کند:

$$P(h \mid D) = \frac{P(D \mid h)\, P(h)}{P(D)}$$

این تئوری، هسته‌ی اصلیِ تمامیِ روش‌های بیزین است. معمولاً در داده‌کاوی از این تئوری برای تصمیم‌گیری بین فرضیه‌های جایگزین استفاده می‌کنیم. مثلاً با دادنِ شواهد خاص، بتوانیم بفهمیم که آیا یک فرد، ژیمیناستیک‌کار، ورزشکار دوی ماراتن یا بازیکن بسکتبال هست یا خیر؟ و یا با دادن شواهدِ خاصی، بتوانیم بفهمیم که آیا یک شخص چای سبزِ سنچا را خواهد خرید یا خیر؟ برای تصمیم‌گیری در مورد جایگزین‌ها ما احتمالِ هر کدام از فرضیه‌ها را محاسبه می‌کنیم. برای مثال:
می‌خواهیم تبلیغی برای چای سبزِ سنچا در سبدِ خریدِ هوشمندِ خود داشته باشیم. این تبلیغ زمانی نمایش داده خواهد شد که احتمال بدهیم این شخص، چای را خواهد خرید. می‌دانیم که این شخص در کد پستیِ ۸۸۰۰۵ زندگی می‌کند.
دو فرضیه‌ی رقیب و متضاد هم وجود دارد:

این شخص، چای سبزِ سنچا را خواهد خرید
در این شرایط ما P(buySenchaTea | 88005) را محاسبه می‌کنیم

این شخص چای سبزِ سنچا را نخواهد خرید
در این شرایط ما P(¬buySenchaTea | 88005) محاسبه می‌کنیم

و آن فرضی را که احتمالِ بالاتری دارد انتخاب می‌کنیم
پس اگر:

$$P(\text{buySenchaTea} \mid 88005) = 0.6$$
و
$$P(\neg\text{buySenchaTea} \mid 88005) = 0.4$$

باشد، احتمالِ بیشتر این است که این شخص، چای سبزِ سنچا را خواهد خرید، پس ما تبلیغ را به او نمایش می‌دهیم.

فرض کنید ما برای یک فروشگاه الکترونیکی فعالیت می‌کنیم و این فروشگاه سه نوع ایمیل تبلیغاتی دارد. اولین نوع تبلیغات، یک لپ‌تاپ را تبلیغ می‌کند، دومی یک کامپیوتر دسکتاپ را و آخری یک تبلت را تبلیغ می‌کند. بر اساسِ چیزی که ما از هر مشتری می‌دانیم، یکی از این ایمیل‌ها را به او ارسال می‌کنیم به صورتی‌که بیشترین احتمال را برای خرید داشته باشد. برای مثال، ممکن است من بدانم که یک مشتری در محدوده‌ی کد پستیِ ۸۸۰۰۵ زندگی می‌کند، یک دختر دانشگاهی دارد که او هم در همان خانه زندگی می‌کند و به کلاسِ یوگا هم می‌رود. به نظر شما کدام یک از تبلیغات را برای او بفرستم؟ لپ‌تاپ، دسکتاپ یا تبلت را؟

فرض کنید D تمام چیزی است که از یک شخص می‌دانم:

- **او در محدوده‌ی کد پستیِ ۸۸۰۰۵ زندگی می‌کند**
- **یک دختر دانشگاهی دارد**
- **به کلاس یوگا می‌رود**

فرضیه‌ی من این است که کدام یک از تبلیغات برای او مناسب‌تر هستند: لپ‌تاپ، دسکتاپ یا تبلت. پس من بایستی احتمالات زیر را حساب کنم:

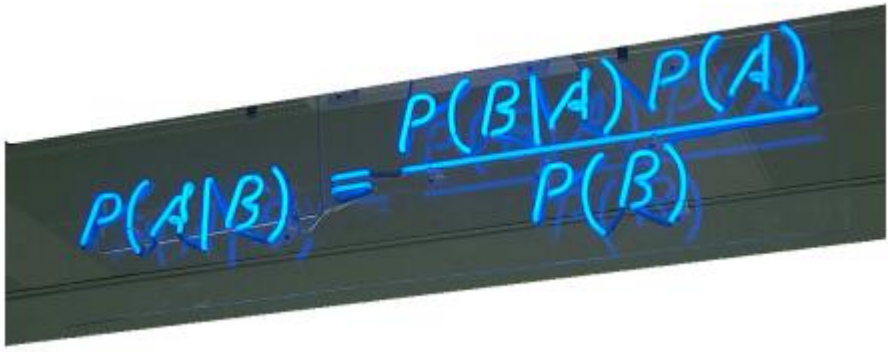
$$P(laptop \mid D) = \frac{P(D \mid laptop)P(laptop)}{P(D)}$$

$$P(desktop \mid D) = \frac{P(D \mid desktop)P(desktop)}{P(D)}$$

$$P(tablet \mid D) = \frac{P(D \mid tablet)P(tablet)}{P(D)}$$

و فرضیه‌ای که بیشترین احتمال را دارد را انتخاب کنم.

اگر بخواهم انتزاعی‌تر صحبت کنم، در طبقه‌بندی، ما چندین فرضیه داریم: $h_1, h_2, \ldots h_n$. این فرضیه‌ها همان دسته‌ها و طبقه‌های مختلف در مسئله‌ی ما هستند. برای مثال، در مسئله‌ی تشخیصِ ورزشکاران این که یک شخص، بازیکنِ بسکتبال، دوی مارتن یا ژیمیناستیک‌کار است و یا در مسئله‌ی پیش‌بینیِ دیابت این‌که آیا یک شخص دیابت خواهد گرفت یا خیر.



$$P(h_1 \mid D) = \frac{P(D \mid h_1)P(h_1)}{P(D)} \qquad P(h_2 \mid D) = \frac{P(D \mid h_2)P(h_2)}{P(D)}$$

,

$$\ldots \qquad P(h_n \mid D) = \frac{P(D \mid h_n)P(h_n)}{P(D)}$$

هنگامی که ما همه‌ی این احتمالات را حساب کنیم، آن فرضیه را انتخاب می‌کنیم که بیشترین احتمال را دارد. به این کار در اصطلاح **فرضیه‌ی بیشینه‌ی پسین (the maximum a posteriori hypothesis)** یا h_{MAP} می‌گویند.

می‌توانیم جمله‌ی بالا را که در واقع محاسبه‌ی فرضیه‌ی بیشینه‌ی پسین است به صورت زیر ترجمه به زبان ریاضیات ترجمه کنیم:

$$h_{MAP} = argmax_{h \in H} P(h|D)$$

در این فرمول H، مجموعه‌ی فرضیات است. پس h∈H به این معنی است: «برای هر فرضیه در مجموعه‌ی فرضیات». در کل فرمول چیزی شبیه این را می‌گوید که: «برای هر فرضیه در مجموعه‌ی فرضیات، P(h|D) را محاسبه کن و فرضیه‌ای که بیشترین احتمالِ وقوع دارد را انتخاب کن». با استفاده از تئوریِ بیزین می‌توانیم فرمول را به صورت زیر تبدیل کنیم:

$$h_{MAP} = argmax_{h \in H} \frac{P(D|h)\,P(h)}{P(D)}$$

پس برای هر فرضیه، ما می‌خواهیم فرمول زیر را محاسبه کنیم:

$$\frac{P(D|h)\,P(h)}{P(D)}$$

احتمالاً متوجه شده‌اید که برای تمامیِ این محاسبات، مخرجِ کسر یکتا و برابر P(D) است. بنابراین، این محاسبات، مستقل از فرضیات هستند. اگر یک فرضیه‌ی خاص دارای بیشترین احتمالِ وقوع باشد، با فرمول بالا، این فرضیه هنوز هم بیشترین احتمال وقوع را دارد البته اگر ما تمام فرضیات را بر P(D) تقسیم نکرده باشیم. اگر هدف ما پیدا کردن فرضیه‌ای با بیشترین احتمالِ وقوع باشد، می‌توانیم محاسباتمان را به صورت زیر ساده کنیم:

$$h_{MAP} = argmax_{h \in H} P(D|h)\,P(h)$$

برای این‌که ببینیم این فرمول چگونه کار می‌کند، از مثالی که Tom M. Mitchell در کتاب یادگیری ماشینِ خودش آورده است استفاده می‌کنیم. Tom M. Mitchell رئیس دپارتمان یادگیری ماشین دانشگاه کارنی مِلُن (Carnegie Mellon) است. او یک محقق بزرگ و یک

مرد خوش‌برخورد و خوب است. به سراغ مثال کتاب برویم. یک حوزه‌ی پزشکی را فرض کنید که در آن می‌خواهیم ببینیم آیا یک بیمار، یک سرطانِ خاصی را دارد یا خیر. یک آزمایش خونِ ساده وجود دارد که از طریق آن می‌توانیم بفهمیم آیا شخص دارای این بیماری هست یا نه. آزمایش یک حالت دودویی دارد – یعنی نتایج را به صورت مثبت (POS) یا منفی (NEG) برمی‌گرداند. هنگامی که سرطان وجود داشته باشد، آزمایش نتیجه‌ی مثبت (POS) را در ۹۸ درصدِ مواقع درست برخواهد گرداند. این آزمایش نتایج منفی (NEG) را نیز در ۹۷ درصد مواقع درست نشان خواهد داد. یعنی در مواقعی که بیماری وجود نداشته باشد، با دقت ۹۷ این آزمایش می‌تواند عدم وجود بیماری را نشان دهد.

فرضیه‌های ما:

. بیمار، یک سرطان خاص را دارد

. بیمار، آن سرطان خاص را ندارد

مدادتان را تیز کنید

اجازه بدهید چیزی که نوشته بودم را به نماد احتمالی ترجمه کنیم. لطفاً جملات زیر را با نمادهای متناسب، مرتبط کرده و احتمالات را بنویسید. اگر جمله‌ای باقی نمانده بود، خودتان بنویسید.

ما می‌دانیم که ۰٫۸ درصد از افراد آمریکا، این نوع از سرطان را دارند.

هنگامی که این بیماری در شخصی وجود داشته باشد، آزمایش در ۹۸ درصد مواقع نتیجه‌ی مثبت (POS) را به درستی تشخیص می‌دهد.

این آزمایش، در ۹۷ درصد موارد نیز نتیجه‌ی آزمایشِ منفی (NEG) را درست تشخیص می‌دهد (هنگامی که بیماری وجود ندارد)

$P(POS|cancer)$ = _____

$P(POS|\neg cancer)$ = _____

$P(cancer)$ = _____

$P(\neg cancer)$ = _____

$P(NEG|cancer)$ = _____

$P(NEG|\neg cancer)$ = _____

مدادتان را تیز کنید – راه حل

ما می‌دانیم که ۰٫۸ درصد از افراد آمریکا، این نوع از سرطان را دارند. پس:
P(cancer) = 0.008

۹۹٫۲ درصد افراد این نوع بیماری را ندارند. پس:
P(¬cancer) = 0.992

هنگامی که بیماری در شخصی وجود داشته باشد، آزمایش در ۹۸ درصد مواقع نتیجه‌ی مثبت (POS) را به درستی تشخیص می‌دهد. پس:
P(POS | cancer) = 0.98

هنگامی که بیماری وجود داشته باشد، آزمایش به اشتباه در ۲ درصد مواقع، پاسخ را منفی (NEG) اعلام می‌کند. پس:
P(NEG | cancer) = 0.02

این آزمایش، در ۹۷ درصد موارد نیز نتیجه‌ی آزمایشِ منفی (NEG) را درست تشخیص می‌دهد (هنگامی که بیماری وجود ندارد). پس:
P(NEG | ¬cancer) = 0.97

این آزمایش در ۳ درصد موارد، هنگامی که بیماری وجود ندارد، آن را به اشتباه مثبت (POS) اعلام می‌کند. پس:

$$P(POS \mid \neg\ cancer) = 0.03$$

مدادتان را تیز کنید

فرض کنید Ann به مطب دکتر بیاید. یک آزمایش خون برای سرطان از او گرفته می‌شود و پاسخ مثبت (POS) است.

اتفاق خوشایندی برای Ann نیست. با این همه، آزمایش دقت ۹۸ درصدی دارد.

و با استفاده از تئوری بیزین می‌توانیم مشخص کنیم که آیا Ann به سرطان مبتلا هست یا خیر.

$$P(cancer) = 0.008$$
$$P(\neg cancer) = 0.992$$
$$P(POS \mid cancer) = 0.98$$
$$P(POS \mid \neg\ cancer) = 0.03$$
$$P(NEG \mid cancer) = 0.02$$
$$P(NEG \mid \neg cancer) = 0.97$$

مدادتان را تیز کنید – راه حل

فرض کنید Ann به مطب دکتر بیاید. یک آزمایش خون برای سرطان از او گرفته می‌شود و پاسخ مثبت (POS) است.

اتفاق خوشایندی برای Ann نیست. با این همه، آزمایش دقت ۹۸ درصدی دارد.

با استفاده از تئوری بیزین می‌توانیم مشخص کنیم که آیا Ann به سرطان مبتلا هست یا خیر.

$$P(cancer) = 0.008$$
$$P(\neg cancer) = 0.992$$
$$P(POS \mid cancer) = 0.98$$
$$P(POS \mid \neg\ cancer) = 0.03$$
$$P(NEG \mid cancer) = 0.02$$
$$P(NEG \mid \neg cancer) = 0.97$$

ما به دنبال پیدا کردن احتمال بشینه‌ی پسین هستم:

$$P(POS \mid cancer)\ P(cancer) = 0.98\ (0.008) = 0.0078$$
$$P(POS \mid \neg cancer)\ P(\neg cancer) = 0.03\ (0.992) = 0.0298$$

حالا ما $\mathbf{h_{MAP}}$ را انتخاب می‌کنیم و بیمار را در دسته‌ی «بدون بیماری سرطان» طبقه‌بندی می‌کنیم.

اگر بخواهیم احتمال دقیق را بدانیم، می‌توانیم نرمال‌شده‌ی این مقادیر را داشته باشیم. این کار را طوری انجام می‌دهیم که جمع آن‌ها برابر ۱ شود:

$$P(cancer \mid POS) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

پس Ann به احتمال ۲۱ درصد مبتلا به سرطان است.

من این مقدار ۸۵ درصد را زیاد نگفتم. مقالات زیر را نگاهی بیندازید:

>> Casscells, W., Schoenberger, A., and Grayboys, T. (1978): "Interpretation by physicians of clinical laboratory results." N Engl J Med. 299:999-1001

>> Gigerenzer, Gerd and Hoffrage, Ulrich (1995): "How to improve Bayesian reasoning without instruction: Frequency formats." Psychological Review. 102: 684-704.

>> Eddy, David M. (1982): "Probabilistic reasoning in clinical medicine: Problems and opportunities." In D. Kahneman, P. Slovic, and A. Tversky, eds, Judgement under uncertainty: Heuristics and biases. Cambridge University Press, Cambridge, UK

در ادامه توضیح می‌دهم که چرا نتایج با تصور شما مغایرت داشت. بیشتر افراد می‌بینند که در گزارش‌های آماری گفته است ۹۸ درصد از افرادی که این نوع سرطان را دارند، نتیجه‌ی آزمایششان مثبت است و با این فرض، نتیجه می‌گیرند که ۹۸ درصد از افرادی که نتیجه‌ی آزمایش آن‌ها مثبت شود، یک همچین سرطانی را دارند. اشتباه است که به این صورت فکر کنیم که این سرطان فقط ۰٫۸ درصد از افراد جمعیت را تحت تاثیر قرار می‌دهد. اجازه بدهید بگوییم که آزمایش را بر روی تمام یک میلیون جمعیتِ شهر انجام می‌دهیم. این بدان معنی است که ۸۰۰۰ نفر در شهر این سرطان را دارند. و طبیعتاً ۹۹۲۰۰۰ نفر خیر. ابتدا، تصور کنید که آزمایش را بر روی ۸۰۰۰ نفری که سرطان دارند انجام دهیم. می‌دانیم که ۹۸ درصد از مواقع هنگامی که آزمایش را بر روی افرادِ دارای سرطان انجام می‌دهیم، این آزمایش نتیجه‌ی درست را (که مثبت است) برمی‌گرداند. پس ۷۸۴۰ نتیجه‌ی آزمایش مثبت را به درستی دریافت کرده‌اند و ۱۶۰ نفر از آن‌ها، به اشتباه، نتیجه‌ی منفی را دریافت کرده‌اند. حالا اجازه دهید سراغ ۹۹۲۰۰۰ نفری که سرطان نداشته‌اند برویم. هنگامی که آزمایش را بر روی آن‌ها انجام می‌دهیم، در ۹۷ درصد موارد نتیجه‌ی درست (که همان نتیجه‌ی منفی است) را می‌گیریم. پس ۹۶۲۲۴۰ نفر از آن‌ها را به درستی نتیجه‌ی منفی داده و ۳۰۰۰ نفر را به اشتباه، نتیجه‌ی آزمایششان را مثبت ارزیابی می‌کنیم. این نتایج در جدول زیر خلاصه شده است:

	نتیجه‌ی آزمایش مثبت	نتیجه‌ی آزمایش منفی
افراد دارای سرطان	7,840	160
افرادی که سرطان ندارند	30,000	962,240

حالا، تصور کنید که Ann، نتیجه‌ی مثبت از آزمایش بگیرد و داده‌ها در قسمتِ «نتیجه‌ی آزمایش مثبت» را نیز به او بدهیم. ۳۰۰۰۰ نفر از افرادی که نتیجه‌ی آزمایش آن‌ها مثبت اعلام شده است، سرطانی نداشته‌اند و فقط ۷۸۴۰ نفر از آن‌ها، سرطان داشته‌اند. پس محتمل است که Ann سرطان نداشته باشد.

هنوز متوجه نشده‌اید؟
اشکالی ندارد. خیلی از افراد متوجه نمی‌شوند.

با تمرین‌های بیشتری، به درک بهتری از این قضیه می‌رسید.

چرا به تئوری بیزین احتیاج داریم؟

تئوری بیز به صورت زیر نوشته می‌شود:

$$P(h\,|D) = \frac{P(D\,|h)\,P(h)}{P(D)}$$

اجازه بدهید به مثال سبد خرید هوشمند ماکروسافت بازگردیم. در آن مثال، اطلاعاتی که در جدول زیر آمده است را از مشتریان به دست آوردیم (ستون Customer ID شماره‌ی مشتری را نشان می‌دهد، Zipcode کد پستی است، ستونِ bought organic product ستونی است که نشان می‌دهد آیا شخص محصولات ارگانیک خریده است یا خیر و ستون bought Sencha green tea نشان می‌دهد که آیا شخص چای سبز سنچا را خریده است یا خیر):

Customer ID	Zipcode	bought organic produce?	bought Sencha green tea?
1	88005	Yes	Yes
2	88001	No	No
3	88001	Yes	Yes
4	88005	No	No
5	88003	Yes	No
6	88005	No	Yes
7	88005	No	No
8	88001	No	No
9	88005	Yes	Yes
10	88003	Yes	Yes

فرض کنید می‌دانیم که یک مشتری در محدوده‌ی کدپستی ۸۸۰۰۵ زندگی می‌کند. در این‌جا دو فرضیه‌ی متضاد داریم (دو فرضیه‌ی رقیب). یکی اینکه «این مشتری چای سبز سنچا را خواهد خرید» و دیگری اینکه «این مشتری چای سبز سنچا را نمی‌خرد». پس فرمول‌ها به صورت زیر خواهند بود:

$$P(h_1 \mid D) = P(buySenchaTea \mid 88005)$$

و

$$P(h_2 \mid D) = P(\neg buySenchaTea \mid 88005)$$

احتمالاً تعجب کنید که چرا باید فرمول زیر را برای به دست آوردنِ این مقدار حساب کرد:

$$\frac{P(88005 \mid buySenchaTea)\, P(buySenchaTea)}{P(88005)}$$

این در حالی است که به راحتی می‌توانیم $\frac{P(88005\,|buySenchaTea)\,P(buySenchaTea)}{P(88005)}$ را به صورت مستقیم از جدول بالا محاسبه کنیم. بله، در این مثالِ ساده حرف شما درست است، ولی در بسیاری از مثال‌های دنیای واقعی، محاسبه‌ی $P(h\,|D)$ بسیار مشکل خواهد بود.

مثال قبلی که در مورد پزشکی و سرطان بود را تصور کنید. می‌خواستیم با توجه به جوابِ مثبت آزمایش، ببینیم که آیا آن شخص مبتلا به بیماریِ سرطان هست یا خیر:

$$P(cancer\,|POS)\ \approx P(POS\,|cancer)\,P(cancer)$$

$$P(\neg cancer\,|POS)\ \approx P(POS\,|\neg cancer)\,P(\neg cancer)$$

محاسبه‌ی آیتم‌های سمت راست ساده است. ما می‌توانیم $P(POS\,|cancer)$ را تخمین بزنیم. این تخمین می‌تواند با دادنِ جوابِ آزمایش به یک نمونه از افراد که سرطان دارند محاسبه شود. همچنین $P(POS\,|\neg cancer)$ را نیز می‌توان با دادنِ جواب آزمایش به نمونه‌ای از افراد که سرطان ندارند محاسبه کرد. $P(cancer)$ هم یک آماری است که می‌توان از سایت‌های دولت استخراج کرد و $P(\neg cancer)$ هم که معادل $1-P(cancer)$ است.

ولی در طرف مقابل، محاسبه‌ی $P(cancer\,|POS)$ به صورت مستقیم، می‌تواند چالشِ زیادی داشته باشد. این فرمول از ما می‌خواهد که احتمالی را به دست بیاوریم که جواب آزمایش را به یک میانگین تصادفی از افراد در کل جمعیت بدهیم و جواب تست هم مثبت (POS) باشد و نتیجتاً آن شخص دارای سرطان باشد. برای این کار ما نیاز به یک نمونه داریم که بتواند بیان‌گر جامعه باشد ولی از آن جایی که ۰٫۸ درصد از جامعه سرطان دارد، به این معنی است که نمونه‌ای از ۱۰۰۰ نفر، فقط ۸ بیمارِ دارای سرطان دارد– که این خیلی کم است. پس احتیاج به یک نمونه‌ی خیلی بزرگ داریم. و اینجاست که تئوری بیز یک استراتژی برای محاسبه‌ی $P(h\,|D)$ برای ما فراهم می‌کند. یعنی این استراتژی هنگامی کارا است که محاسبه‌ی این فرمول به صورت مستقیم کار سختی باشد.

بیزین ساده

در اکثر اوقات، ما به جای داشتن یک قسمتِ کوچک از داده‌ها، تعداد بیشتری نمونه‌ی داده داریم. در مثالِ «چای سبز سنچا» ما دو نوع علامت یا همان ویژگی داشتیم: یکی «کد پستی» بود و دیگری اینکه «آیا شخص غذای ارگانیک خریده است یا خیر». برای محاسبه‌ی احتمال فرضیه، در حالی که تمامیِ علائم و شواهد به ما داده شده باشد، به سادگی هر یک از احتمالات را در هم ضرب می‌کنیم. برای مثال:

Customer ID	Zipcode	bought organic produce?	bought Sencha green tea?
1	88005	Yes	Yes
2	88001	No	No
3	88001	Yes	Yes
4	88005	No	No
5	88003	Yes	No
6	88005	No	Yes
7	88005	No	No
8	88001	No	No
9	88005	Yes	Yes
10	88003	Yes	Yes

Code:

tea = فردی که چای سنچا خریده است

¬ tea = فردی که چای سنچا نخریده است

P(88005|tea) =
احتمال اینکه شخصی در محدوده‌ی کد پستیِ ۸۸۰۰۵ زندگی کرده با این شرط که بدانیم او چای سبز سنچا خرید است.

و...

ما می‌خواهیم بدانیم، شخصی که در محدوده‌ی کدپستی ۸۸۰۰۵ زندگی می‌کند و محصولات ارگانیک را نیز خریداری کرده است، چای سبز سنچا را خواهد خرید یا خیر:
فرمول به صورت P(tea | 88005 & organic) نوشته می‌شود و برای محاسبه‌ی آن احتمالات را با هم ضرب می‌کنیم:

$$P(tea \mid 88005 \ \& \ organic) = P(88005 \mid tea)\, P(organic \mid tea)\, P(tea)$$
$$= 0.6(0.8)(0.5) = 0.24$$

$$P(\neg tea \mid 88005 \ \& \ organic)$$
$$= P(88005 \mid \neg\, tea)\, P(organic \mid \neg tea)\, P(\neg tea)$$
$$= 0.4(0.25)(0.5) = 0.05$$

پس شخصی که در محدوده‌ی کدپستیِ ۸۸۰۰۵ زندگی کرده و غذاهای ارگانیک خریداری کند، به احتمال بیشتری چای سبز سنچا را خواهد خرید تا این‌که این محصول را خریداری نکند. پس به این خریدار در سبدِ هوشمندِ خریدش، چای سبز سنچا را نمایش می‌دهیم تا در صورت نیاز خریداری کند.

حالا می‌خواهم جمله‌ای از Stephen Baker را بیاورم که توضیحی است بر تکنولوژی سبد خرید:

«... در این‌جا می‌خواهیم با هم ببینیم که خرید از این سبدهای هوشمند چگونه می‌تواند کمک‌کننده باشد. شما سبد خرید خود را پر کرده و کارتِ مشتری خود را رو می‌کنید. صفحه‌ی خوش‌آمدگویی باز می‌شود البته همراه با لیست خرید. این لیست بر اساس الگوهای خریدهای قبلیِ شماست. شیر، تخم‌مرغ، کدو و هر چیز دیگری می‌تواند در این سبد خرید قرار بگیرد. سیستم‌های هوشمند ممکن است سریع‌ترین راه برای خرید هر آیتم را جلوی پای شما قرار دهند. یا شاید به شما اجازه دهند که لیست خرید را تغییر دهید و به سیستم بگویید که برای مثال دیگر گل کلم یا بادام زمینی شور را پیشنهاد ندهد. این یک کار ساده است. ولی با توجه به مطالعات شرکت اکسنچر (Accenture) که شرکتی در زمینه‌ی مشاوره استراتژی است، خریداران به طور میانگین ۱۱ درصد از اقلامی که می‌خواستند خرید کنند را در هنگامِ خرید فراموش می‌کنند. اگر فروشگاه‌ها به طور موثر به ما یادآوری کنند که چه چیزهایی احتیاج داریم، باعث می‌شود که کمتر نصف شب مجبور شویم به فروشگاه برویم و خرید کنیم و البته فروش آن‌ها هم بیشتر می‌شود.»

Baker. 2008. P49

کتاب شمارش (Numerati)

من از این کتاب که توسط استفان بیکر (Stephen Baker) نوشته شده است چندین بار نقل و قول کرده‌ام. به شما هم توصیه می‌کنم که این کتاب را بخوانید. کتاب کاغذی فقط ۱۰ دلار است و به دردِ خواندن قبل از خواب هم می‌خورد.

مثالِ محصولات i100 و i500

فرض کنید می‌خواهیم به شرکتِ iHealth که در زمینه‌ی فروشِ وسیله‌های پوشیدنی برای مراقبت و نظارت بر تمرین (مثل دستبندهای هوشمند) فعالیت می‌کند، کمک کنیم. این شرکت رقیب وسیله‌هایی مانند Nike Fuel و Fitbit Flex (دو محصول مشابه دیگر) است. شرکت iHealth دو مدل از این محصول را می‌فروشد: مدل i100 و مدل i500 که تفاوت آن‌ها در عملکرد هر یک است.

ویژگی‌های محصول iHealth100:
نظارت بر ضربان قلب، جی‌پی‌اس (برای محاسبه‌ی پیمودن مسیر و...)، وای‌فای برای اتصال خودکار به سایت iHealth جهت فرستادن داده‌ها.

ویژگی‌های محصول iHealth500:
تمامی ویژگی‌های iHealth100 + نبض اکسیمتری (اکسیژن در خون) + اتصال 3G خودکار به وب‌سایت iHealth

این شرکت، محصولات را به صورت آنلاین می‌فروشد و ما را استخدام کرده است تا سیستمِ توصیه‌گری را برای مشتریانشان راه‌اندازی کنیم. برای جمع‌آوری داده‌ها در ساخت سیستم توصیه‌گر، هنگامی که شخصی یک دستگاه را خرید، از او می‌خواهیم که پرسشنامه‌ای را پر کند. هر کدام از پرسش‌های این پرسشنامه، به یک ویژگی مرتبط است. اول، از او می‌پرسیم که دلیل اصلی خود را برای شروع تمرین‌های ورزشی بگوید و این کار را بایستی با انتخاب از گزینه‌های زیر انجام دهد: **سلامتی (health)**، **ظاهر (appearance)** و یا **هر دو (both)**. سپس ، از آن‌ها می‌پرسیم که در حال حاضر در چه سطحی تمرین می‌کنند: **نشسته (sedentary)**، **متعادل (moderate)**، یا **فعال (active)**. از آن‌ها سوال می‌کنیم که تا چه

اندازه مشتاق و علاقه‌مند هستند: **به صورت نسبی (moderate)** یا **بسیار شدید (aggressive)**. و در آخر از آن ها این سوال را می‌پرسیم که آیا با دستگاه‌های مبتنی بر تکنولوژی و استفاده از آن‌ها **راحت هستند** یا **خیر**. نتایج به صورت زیر درآمده است (ستون Main Interest دلیل اصلی استفاده را نشان می‌دهد، ستون Current Exercise Level سطح فعلی خریدار است، ستون How Motivated نشان‌دهنده‌ی میزان اشتیاق مشتری بوده و ستون Comfortabe with tech. Devices هم این معنا را می‌رساند که آیا کاربر در استفاده وسائل تکنولوژی راحت است یا خیر. ستون # Model هم نشان‌دهنده‌ی مدلی است که مشتری، خریده است):

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Model #
both	sedentary	moderate	yes	i100
both	sedentary	moderate	no	i100
health	sedentary	moderate	yes	i500
appearance	active	moderate	yes	i500
appearance	moderate	aggressive	yes	i500
appearance	moderate	aggressive	no	i100
health	moderate	aggressive	no	i500
both	active	moderate	yes	i100
both	moderate	aggressive	yes	i500
appearance	active	aggressive	yes	i500
both	active	aggressive	no	i500
health	active	moderate	no	i500
health	sedentary	aggressive	yes	i500
appearance	active	moderate	no	i100
health	sedentary	moderate	no	i100

مدادتان را تیز کنید

با استفاده از روش بیز ساده (Naïve Bayes)، کدام مدل را به شخصی که:
اولویت اولش سلامتی است. یعنی Main Interest = health
در سطح متعادل تمرین می‌کند. یعنی Current Exercise Level = moderate
به صورت نسبی مشتاق به تمرین کردن است. یعنی How Motivated = moderate
و با استفاده از دستگاه‌های مبتنی بر تکنولوژی راحت است. یعنی Comfortable with tech. Devices = yes
پیشنهاد می‌کنید؟
اگر دنبال کمک هستید قسمت زیر را بخوانید

مداتان را تیز کنید – سرنخ
خب. در نهایت ما بایستی فرمول زیر را محاسبه کنیم:

$$P(i100 \mid health, moderateExercise, moderateMotivation, techComfortable)$$

و همچنین:

$$P(i500 \mid health, moderateExercise, moderateMotivation, techComfortable)$$

و بعد از محاسبه‌ی این دو، مدلی با بیشترین احتمال را انتخاب کنیم.
اجازه بدهید مشخص کنم که برای محاسبه‌ی فرمول اولی چه چیز نیاز داریم:

$$P(i100 \mid health, moderateExercise, moderateMotivation, techComfortable) = P(health|i100)\, P(moderateExercise|i100)\, P(moderateMotivated|i100)\, P(techComfortable|i100) P(i100)$$

پس این چیزی است که در ابتدا بایستی محاسبه کنیم:

P(health|i100) = 1/6
P(moderateExercise|i100) =
P(moderateMotivated|i100) =
P(techComfortable|i100) =
P(i100) = 6 / 15

این سر نخی بود که به شما دادم. حالا امیدوارم که بتوانید مثال را حل کنید

مدادتان را تیز کنید – راه‌حل

اول بایستی فرمول زیر را محاسبه کنیم:

P(i100 | health, moderateExercise, moderateMotivation, techComfortable)

که معادل ضرب تمامی جملات زیر است:

P(health|i100) P(moderateExercise|i100) P(moderateMotivated|i100) P(techComfortable|i100) P(i100)

P(health|i100) = 1/6
P(moderateExercise|i100) = 1/6
P(moderateMotivated|i100) = 5/6
P(techComfortable|i100) = 2/6
P(i100) = 6 / 15

پس:

P(i100| evidence) = .167 * .167 * .833 * .333 * .4 = .00309

و حالا برای **i500** محاسبات را انجام می‌دهیم:

P(i500 | health, moderateExercise, moderateMotivation, techComfortable)

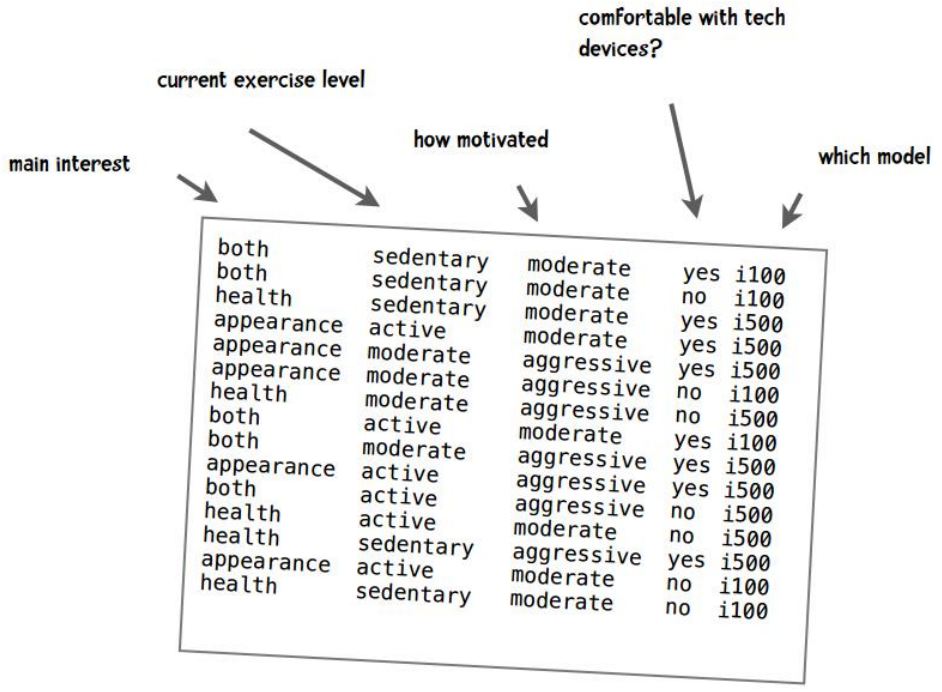
P(health|i500) = 4/9
P(moderateExercise|i500) = 3/9
P(moderateMotivated|i500) = 3/9
P(techComfortable|i500) = 6/9
P(i500) = 9 / 15

P(i500| evidence) = .444 * .333 * .333 * .667 * .6 = .01975

و نتیجه هم که مشخص است.

حالا این کارها را در پایتون انجام می‌دهیم

خیلی خوب! حالا که فهمیدیم، طبقه‌بند بیزین ساده چگونه کار می‌کند، اجازه دهید ببینیم که چگونه می‌توانیم آن را در پایتون پیاده‌سازی کنیم. فرمت فایل‌های داده همانند فرمت فصل قبل است. در فصل قبل فایل متنی موجود بود که در هر خط مقادیر با فاصله‌ی tab از هم جدا شده بودند. حالا برای مثالِ iHealth، فایل داده چیزی مانند شکل زیر است:



ما از مثالی استفاده می‌کنیم که کمی بیشتر از این داده‌ها را دارد و همچنین من می‌خواهم از اعتبارسنجیِ متقابل ۱۰-تکه‌ای (10-fold cross validation) که در فصل قبلی از آن بهره بردیم، این‌جا هم استفاده کنم. اگر خاطرتان باشد، در این روش داده‌ها به ۱۰ سطل (قسمت – bucket) تقسیم می‌شدند. ما عملیات یادگیری را بر روی ۹ قسمت انجام داده و طبقه‌بند را بر روی بقیه (در این‌جا ۱ قسمتِ باقی‌مانده) آزمایش می‌کردیم. و این کار را ۱۰ مرتبه انجام می‌دادیم و هر بار یک قسمت خاصی را برای آزمایش کنار می‌گذاشتیم. مثال iHealth، با فقط ۱۵ نمونه ساخته شده است، پس می‌توانیم عملیات بیز ساده (Naïve

Bayes) را به صورت دستی بر روی آن انجام دهیم. احتمالاً خیلی عاقلانه نیست که این ۱۵ نمونه را به ۱۰ قسمت تقسیم کنیم. یک راه حل، که خیلی هم راه حل خوبی نیست، این است که همان ۱۰ قسمت را داشته باشیم ولی تمام ۱۵ نمونه در یک قسمت (یک سطل) قرار بگیرند و بقیه‌ی قسمت‌ها خالی باشند.

کد الگوریتم طبقه‌بندیِ بیز ساده (Naïve Bayes) از دو قسمت تشکیل شده است. یک قسمت برای آموزش و قسمت دیگر برای طبقه‌بندی.

قسمت آموزش

خروجی قسمت آموزش بایستی داده‌های زیر باشد:

یک مجموعه‌ی احتمالات پیشین (prior probabilities) – برای مثال P(i100) = 0.4

یک مجموعه‌ی احتمالات شرطی (conditional probabilities) – برای مثال P(health|i100) = 0.167

می‌خواهم احتمالات پیشین را به صورت دیکشنری در پایتون (جدول درهم‌سازی – hash table) نمایش دهم:

```
self.prior = {'i500': 0.6, 'i100': 0.4}
```

احتمالاتِ شرطی کمی پیچیده‌تر هستند. روش من – که احتمالاً روش‌های بهتری هم موجود باشد – این است که مجموعه‌ای از احتمالات شرطی را به هر کلاس مرتبط کنم. چیزی مانند کدِ زیر:

```
{'i500': {1: {'appearance': 0.3333333333333, 'health': 0.4444444444444,
              'both': 0.2222222222222},
          2: {'sedentary': 0.2222222222222, 'moderate': 0.333333333333,
              'active': 0.4444444444444444},
          3: {'moderate': 0.333333333333, 'aggressive': 0.66666666666},
          4: {'no': 0.3333333333333333, 'yes': 0.6666666666666666}},

 'i100': {1: {'appearance': 0.333333333333, 'health': 0.1666666666666,
              'both': 0.5},
          2: {'sedentary': 0.5, 'moderate': 0.16666666666666,
              'active': 0.3333333333333},
          3: {'moderate': 0.83333333334, 'aggressive': 0.166666666666},
          4: {'no': 0.6666666666666, 'yes': 0.3333333333333}}}
```

اعداد ۱، ۲، ۳، ۴ بیان‌گر شماره‌های ستون‌ها است. پس اولین خط کد بالا به این صورت تفسیر می‌شود: «احتمال این‌که مقدار اولین ستون appearance باشد، به این شرط که بدانیم دستگاه، مدل i500 بوده، برابر ۰,۳۳۳ درصد است».
اولین قدم در محاسبه‌ی این احتمالات، این است که به سادگی آیتم‌ها را شمارش کنیم. در زیر چند خطِ اول فایلِ ورودی به صورت زیر نمایش داده شده است:

```
both        sedentary   moderate    yes i100
both        sedentary   moderate    no  i100
health      sedentary   moderate    yes i500
appearance  active      moderate    yes i500
```

دوباره می‌خواهم از دیکشنری‌ها استفاده کنم. classes، تعداد کلاس‌هایی است که مشاهده شده‌اند. پس بعد از اجرای خط اول، کلاس‌ها (classes) مانند زیر خواهد بود:

```
{'i100': 1}
```

بعد از خط دوم:

```
{'i100': 2}
```

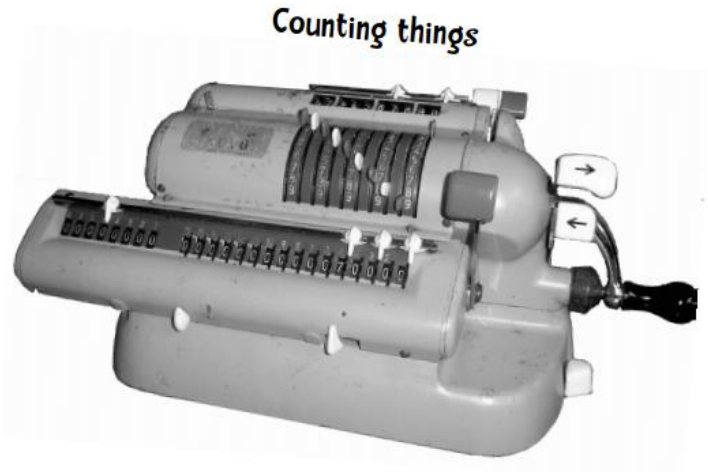
و بعد از خط سوم:

```
{'i500': 1, 'i100': 2}
```

بعد از پردازش همه‌ی داده‌ها، مقدار classes به صورت زیر در می‌آید:

```
{'i500': 9, 'i100': 6}
```

برای به دست آوردنِ احتمالِ پیشین (prior probabilities)، به سادگی این اعداد را تقسیم بر تعداد تمامی نمونه‌ها می‌کنم



.

برای مشخص کردن **احتمالات شرطی (conditional probabilities)** تعداد پیشامدهای مقادیر ویژگی‌ها، در ستون های مختلف را در یک دیکشنری به اسم count ذخیره کرده و برای هر کلاس، تعداد را جدا ذخیره می‌کنم.
پس برای پردازش رشته‌ی «both» در خط اول، مقدار counts به صورت زیر خواهد بود:

```
{'i100': {1: {'both': 1}}
```

و در نهایت با پردازش داده‌ها، مقدار counts مانند کد زیر خواهد بود:

```
{'i100': {1: {'appearance':2, 'health': 1, 'both': 3},
          2: {'active': 2, 'moderate': 1, 'sedentary': 3},
          3: {'moderate': 5, 'aggressive': 1},
          4: {'yes': 2, 'no': 4}},
 'i500': {1: {'health': 4, 'appearance': 3, 'both': 2},
          2: {'active': 4, 'moderate': 3, 'sedentary': 2},
          3: {'moderate': 3, 'aggressive': 6},
          4: {'yes': 6, 'no': 3}}}
```

پس، در اولین ستونِ i100، دو بار ویژگی «ظاهر (appearance)» و یک بار ویژگی «سلامتی (health)» مشاهده شده است. مقدارِ «both (یعنی هر دو)» برای آن‌ها نیز برابر ۳ خواهد بود. برای به دست آوردن احتمالات شرطی، ما آن مقادیر را بر تعداد تمامی نمونه‌های آن کلاس، تقسیم می‌کنیم. برای مثال، تعداد ۶ نمونه از i100 وجود دارد که ۲تای آن‌ها، مقدار «ظاهر (apearance)» را برای ستون اولشان دارند. پس:

P('appearance'|i100) = 2/6 = .333

با استفاده از این پیش‌زمینه، کد پایتونِ زیر برای آموزش این طبقه‌بند قرار داده شده است (یادتان هست که می‌توانید این کدها را از سایت guidetodatamining.com و chistio.ir دانلود کنید):

```
class Classifier:
    def __init__(self, bucketPrefix, testBucketNumber, data
Format):

        """ a classifier will be built from files with the
bucketPrefix
        excluding the file with textBucketNumber. dataForma
t is a string that
        describes how to interpret each line of the data fi
les. For example,
        for the iHealth data the format is:
        "attr  attr    attr    attr    class"
        """

        total = 0
        classes = {}
```

```
        counts = {}


        # reading the data in from the file

        self.format = dataFormat.strip().split('\t')
        self.prior = {}
        self.conditional = {}
        # for each of the buckets numbered 1 through 10:
        for i in range(1, 11):
            # if it is not the bucket we should ignore, rea
d in the data
            if i != testBucketNumber:
                filename = "%s-%02i" % (bucketPrefix, i)
                f = open(filename)
                lines = f.readlines()
                f.close()
                for line in lines:
                    fields = line.strip().split('\t')
                    ignore = []
                    vector = []
                    for i in range(len(fields)):
                        if self.format[i] == 'num':
                            vector.append(float(fields[i]))
                        elif self.format[i] == 'attr':
                            vector.append(fields[i])

                        elif self.format[i] == 'comment':
                            ignore.append(fields[i])
                        elif self.format[i] == 'class':
                            category = fields[i]
                    # now process this instance
                    total += 1
                    classes.setdefault(category, 0)
                    counts.setdefault(category, {})
                    classes[category] += 1
                    # now process each attribute of the ins
tance
                    col = 0
                    for columnValue in vector:
                        col += 1
                        counts[category].setdefault(col, {}
)
                        counts[category][col].setdefault(co
lumnValue, 0)
                        counts[category][col][columnValue]
+= 1
```

این کد فقط برای آموزش بود. همان‌طور که می‌بینید هیچ محاسبات پیچیده‌ی ریاضی نداشت. فقط چند تا شمارش ساده بود.

عملیات طبقه‌بندی

خب، ما طبقه‌بند را آموزش دادیم. حالا می‌خواهیم چند نمونه‌ی مختلف را طبقه‌بندی کنیم. برای مثال، چه مدلی از محصول را بایستی به شخصی که در درجه‌ی اول به سلامت (health) علاقه‌مند است، به صورت نسبی (moderated) فعال است، نسبتاً (moderated) مشتاق است و در نهایت با استفاده از تکنولوژی هم راحت (yes) است، پیشنهاد دهیم:

c.classify(['health', 'moderate', 'moderate', 'yes'])

برای محاسبه‌ی کد بالا، نیاز داریم تا فرمول زیر را محاسبه کنیم:

$$h_{MAP} = \text{argmax}_{h \in H} P(D \mid h)P(h)$$

قبلاً این کار را به صورت دستی انجام دادیم، و برای این کار احتمالِ وقوعِ هر فرضیه را با توجه به اطلاعاتی که داشتیم محاسبه می‌کردیم و حالا به سادگی این فرمول را به کد تبدیل می‌کنیم:

```
def classify(self, itemVector):
    """Return class we think item Vector is in"""
    results = []
    for (category, prior) in self.prior.items():
        prob = prior
        col = 1
        for attrValue in itemVector:
            if not attrValue in self.conditional[categor
y][col]:
                # we did not find any instances of this
attribute value
                # occurring with this category so prob =
 0
                prob = 0
            else:
```

```
                prob = prob * self.conditional[category]
[col][attrValue]
                col += 1
        results.append((prob, category))
    # return the category with the highest probability
    return(max(results)[1])
```

و هنگامی که کد را امتحان می‌کنیم، همان نتیجه‌ای را میگیریم که قبلاً با محاسبه‌ی دستی گرفته بودیم:

```
>>c = Classifier("iHealth/i", 10, "attr\tattr\tattr\tattr\tclass")
>>print(c.classify(['health', 'moderate', 'moderate', 'yes'])
i500
```

جمهوری‌خواهان در مقابل دموکرات‌ها

اجازه بدهید به یک مجموعه‌ی داده‌ی جدید، نگاهی داشته باشیم. مجموعه‌ی داده‌ی رای‌های کنگره (Congressional Voting Records)، که از سایت http://archive.ics.uci.edu/ml/index.html قابل دانلود است. این مجموعه‌ی داده در فرمتی مطابق با برنامه‌ی نوشته شده توسط ما استفاده می‌شود. برنامه‌ی ما در سایت guidetodatamining.com و chistio.ir قرار دارد. این داده‌ها متشکل از رای‌های نمایندگان کنگره‌ی ایالات متحده است. **ستون‌ها شامل ۱۶ ویژگی می‌شوند.**

اطلاعات ویژگی‌ها:

نام کلاس: ۲(دموکرات، جمهوری‌خواه)

کودکان معلول:۲ (آری، خیر)
به اشتراک گذاری هزینه‌ی پروژه‌ی آب: ۲(آری، خیر)
تصویب قطعنامه بودجه:۲ (آری، خیر)
ثابت نگه داشتن نرخ پزشکان:۲ (آری، خیر)
کمک‌های السالوادور:۲ (آری، خیر)
گروه‌های مذهبی در مدارس:۲ (آری، خیر)
جلوگیری از آزمایش ضد ماهواره‌ای:۲ (آری، خیر)
کمک به گروه کانترا در نیکاراگوئه:۲ (آری، خیر)
موشک mx: ۲ (آری، خیر)
ضد مهاجرت:۲ (آری، خیر)
تقلیل شرکت سینفیول (یک شرکت وابسته به دولت آمریکا): ۲ (آری، خیر)
هزینه برای تحصیلات:۲ (آری، خیر)
حق قانونی شکایت:۲ (آری، خیر)
جرم و جنایت:۲ (آری، خیر)
صادرات بدون مالیات:۲ (آری، خیر)
فعالیت‌های صادراتی به آفریقای جنوبی:۲ (آری، خیر)

فایل، از مقادیری که با tab از هم جدا شده‌اند، تشکیل شده و چیزی شبیه به شکل زیر است:

democrat	y	n	y	n	n	n	y	y	y	n	n	n	n	n	y	y
democrat	y	y	y	n	n	n	y	y	y	y	n	n	n	n	y	y
democrat	y	y	y	n	n	n	y	y	n	n	n	n	n	y	n	y
republican	y	y	y	n	n	y	y	y	y	y	n	n	n	n	n	y

الگوریتم طبقه‌بندی بیز ساده (Naïve Bayes) برای این مثال، به خوبی کار می‌کند .فرمت رشته‌ها به ما می‌گوید که اولین ستون همان **طبقه یا برچسب** برای هر مورد است و بقیه‌ی ستون‌ها، بایستی به عنوان **ویژگی‌ها (ابعاد)** تفسیر شوند:

```
format = """class\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tattr\tattr
\tattr\tattr\tattr\tattr\tattr\tattr\tattr"""

tenfold("house-votes/hv", format)

            Classified as:
               democrat   republican
               +-------+-------+
   democrat    |   111 |    13 |
               |-------+-------|
 republican    |     9 |    99 |
               +-------+-------+

90.517 percent correct
total of 232 instances
```

عالی شد!

برای اینکه یکی از مشکلاتِ این روش را درک کنید چند صندلیِ نمایندگی مختلف در ایالات متحده را به عنوان نمونه در نظر بگیرید. از ۴۳۵ نمونه‌ی مختلف که رای داده‌اند، یک نمونه‌ی ۲۰۰ تایی از استخراج کرده‌ام – ۱۰۰ نفر دموکرات و ۱۰۰ نفر جمهوری‌خواه. جدول زیر نشان می‌دهد که چند درصد از آن‌ها به ۴ لایحه‌ی مختلف، «بلی» گفته‌اند. (هر

کدام از ستون‌ها یک لایحه است. برای مثال CISPA قانون امنیت و اشتراک هوشمند سایبری است)

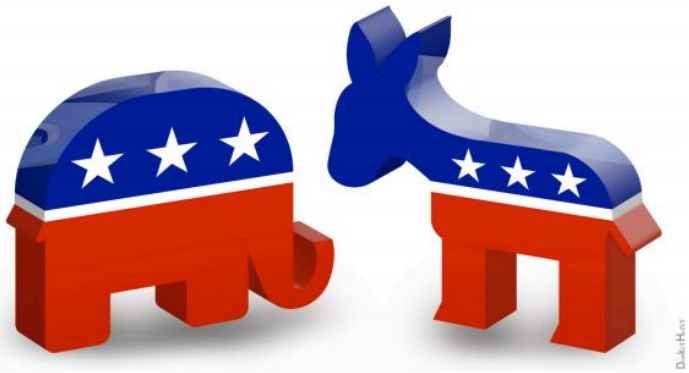
	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
جمهوری خواه	0.99	0.01	0.99	0.5
دموکرات	0.01	0.99	0.01	1.0

درصد رای به «بله»

این جدول نشان می‌دهد ۹۹ درصد جمهوری‌خواهان در نمونه، به CISPA (قانون امنیت و اشتراک هوشمند سایبری) جواب «بله» داده‌اند. فقط ۱ درصد جمهوری‌خواهان به قانون حریم خصوصیِ خواننده (Reader Privacy Act) رای بله دادند، در حالی که ۹۹ درصد آن‌ها به مالیات فروش اینترنت (Internet Sales Tax) رای داده و ۵۰ درصد آن‌ها به لایحه‌ی شنود اینترنتی (Internet Snooping Bill) رای بله داده‌اند (البته من این مقادیر را از خودم ساختم، یعنی واقعی نیستند). ما یک نماینده‌ی ایالات متحده را انتخاب می‌کنیم که در نمونه‌ی ما نباشد. مثلاً نماینده‌ی X که با Rep. X در جدول نشان داده می‌شود، ومی‌خواهیم ببینیم آیا او یک جمهوری‌خواه است یا یک دموکرات؟ در زیر، نحوه‌ی رای دادنِ این نماینده را آورده‌ام:

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
جمهوری خواه	0.99	0.01	0.99	0.5
دموکرات	0.01	0.99	0.01	1.0
Rep. X	N	Y	N	N

به نظرتان، این شخص، دموکرات است یا جمهوری‌خواه؟



به نظر من که دموکرات است. اجازه بدهید به صورت قدم به قدم مثال را با استفاده از بیز ساده (Naïve Bayes) پیش ببریم. احتمالاتِ پیشین برای P(Democrat) (دموکرات‌ها) و P(Republican) (جمهوری‌خواه‌ها) هر دو برابر ۰٫۵ است، چون دقیقاً تعداد ۱۰۰ نفر دموکرات و ۱۰۰ نفر جمهوری‌خواه در مجموعه‌ی داده‌ی ما موجود بودند. ما می‌دانیم که نماینده‌ی X یعنی همان که می‌خواهیم حزبِ او را مشخص کنیم، به CISPA (قانون امنیت و اشتراک هوشمند سایبری) رایِ «خیر - N» داده است. همچنین می‌دانیم که:

$$P(\text{Republican}|C=\text{no}) = 0.01 \text{ and } P(\text{Democrat}|C=\text{no}) = 0.99$$

که در این فرمول، منظورمان از C، همان CISPA (قانون امنیت و اشتراک هوشمند سایبری) است و با همان شواهد کم، مقدار احتمال P(h|D) به صورت زیر محاسبه می‌شود (منظور از Republication جمهوری خواه و منظور از Democrat دموکرات است):

h=	p(h)	P(C=no|h)				P(h|D)
Republican	0.5	0.01				0.005
Democrat	0.5	0.99				0.495

با توجه به اینکه نماینده‌ی X، به قانون محافظت از خوانندگان (Reader Privacy Act) رای «بلی» داده و رای «خیر» که به مالیات فروش اینترنت (Internet Sales Tax) داده است، جدول زیر به دست می‌آید:

h=	p(h)	P(C=no|h)	P(R=yes|h)	P(T=no|h)		P(h|D)
Republican	0.5	0.01	0.01	0.01		0.0000005
Democrat	0.5	0.99	0.99	0.99		0.485

اگر این احتمالات را نرمال‌سازی (normalize) کنیم، به عدد زیر می‌رسیم:

$$\mathbf{P(Democrat \mid D)} = \frac{\mathbf{0.485}}{\mathbf{0.485 + 0.0000005}} = \frac{\mathbf{0.485}}{\mathbf{0.4850005}} = \mathbf{0.99999}$$

پس تا به این‌جا با احتمال ۹۹,۹۹ درصد یقین پیدا کردیم که نمونه‌ی X، یک نماینده‌ی دموکرات است.

در نهایت، ما رای «خیر» برای لایحه‌ی شنود اینترنت (Internet Snooping Bill) را، برای این نماینده مورد محاسبه قرار می‌دهیم:

h=	p(h)	P(C=no|h)	P(R=yes|h)	P(T=no|h)	P(S=no|h)	P(h|D)
Republican	0.5	0.01	0.01	0.01	0.50	2.5E-07
Democrat	0.5	0.99	0.99	0.99	0.00	0

اوه. ما از یقین ۹۹,۹۹ درصدی به احتمال ۰ درصدی برای دموکرات بودنِ این شخص رسیدیم. این به آن دلیل است که تعداد ۰ عدد (یعنی هیچ) دموکرات به قبض شنود رایِ «خیر» داده بودند. بر اساس این احتمالات، به این نتیجه می‌رسیم که شخصِ X، یک جمهوری خواه است. که این بر خلاف شهود اولیه‌یمان بود.

تخمین احتمالات

احتمالات در الگوریتم بیز ساده (Naïve Bayes) در واقع تخمینی از احتمالات واقعی هستند. احتمالات واقعی (true probabilities) آن‌هایی هستند که از تمام جمعیت داده‌ها

به دست می‌آیند. برای مثال، اگر می‌توانستیم آزمایش سرطان را به تک‌تکِ افراد در جامعه بدهیم، می‌توانستیم، برای مثال، با دریافت احتمال واقعی و دقیقِ آزمایش، نتیجه‌ی منفی را با توجه به آگاه بودن از این‌که شخصی سرطان ندارد، مشخص کنیم. با این حال، انجام آزمایش برای هر تک‌تکِ افراد در جامعه تقریباً غیر ممکن است. ما می‌توانیم این احتمال را با انتخاب یک نمونه‌ی تصادفی که نماینده‌ای از جمعیت مورد نظر باشد، تخمین بزنیم. برای مثال ۱۰۰۰ نفر را انتخاب کرده، آزمایش سرطان را بر روی آن‌ها انجام می‌دهیم و احتمالات را محاسبه می‌کنیم. در اکثر موارد، این کار تخمین خوبی از احتمالات واقعی را به ما می‌دهد، ولی هنگامی که احتمالات واقعی خیلی کوچک باشند، این تخمین‌ها، رو به ضعف می‌روند. در این‌جا یک مثال را با هم مرور می‌کنیم. فرض کنید که احتمال واقعیِ این‌که یک دموکرات به لایحه‌ی شنود اینترنت (Internet Snooping Bill) رایِ «خیر» بدهد، برابر ۰٫۰۳ درصد است. P(S=no|Democrat) = 0.03.

ورزش سبکِ ذهنی

فرض کنید، ما همین احتمال بالا را، با انتخاب ۱۰ نفر دموکرات و ۱۰ نفر جمهوری‌خواه، تخمین بزنیم. کدام یک از اعداد زیر، بیشترین احتمال را برای فرد دموکراتی که رای «خیر» به لایحه‌ی شنود اینترنتی داده باشد، دارد؟

□0 □1 □2 □3

ورزش سبکِ ذهنی – راه حل

فرض کنید، ما همین احتمال بالا را، با انتخاب ۱۰ نفر دموکرات و ۱۰ نفر جمهوری‌خواه، تخمین بزنیم. کدام یک از اعداد زیر، بیشترین احتمال را برای فرد دموکراتی که رای «خیر» به قبض شنود اینترنتی داده باشد، دارد؟

0

پس بر اساس نمونه، P(S=no|Democrat) = 0.

همان‌طور که در مثال بالا دیدیم، هنگامی که احتمال برابر صفر است، این احتمالِ صفر بر سراسر الگوریتم بیز ساده (Naïve Bayes) تاثیر می‌گذارد – مهم نیست که بقیه‌ی مقادیر چه هستند. مشکلِ دیگر این است که، احتمالاتی که بر اساس یک نمونه ساخته می‌شوند، به سمتِ **دست کم گرفته شدنِ (underestimate)** احتمالِ واقعی انحراف پیدا می‌کنند. یعنی تمایل دارند که دست کم گرفته شوند.

حل مشکل

اگر سعی کنیم که چیزی شبیه به P(S=no|Democrat) را حساب کنیم، محاسباتمان مانند زیر می‌شود:

$$P(S = \text{no}|\text{Democrat}) = \frac{\text{the number that both are Democrats and voted no on the snooping bill.}}{\text{total number of Democrats}}$$

صورت کسر: تعدادی که هر دوی آن‌ها دموکرات بوده و به لایحه‌ی شنود اینترنتی رایِ خیر داده باشند

مخرج کسر: تعداد کل دموکرات‌ها

برای توضیح ساده‌تر اجازه بدهید، با استفاده از متغیرهای کوتاه‌تر، این فرمول را توضیح دهم:

$$P(x \mid y) = \frac{n_c}{n}$$

که در آن n تعداد نمونه‌های موجود در کلاسِ y در مجموعه‌ی آموزشی هستند و n_c تعداد تمامی نمونه‌های موجود از کلاسِ y است که مقدار x را دارند.
مشکلی که داریم هنگامی است که $\mathbf{n_c}$ **برابر صفر** است. ما این مشکل را با تغییر فرمول به صورت زیر حل می‌کنیم:

$$P(x \mid y) = \frac{n_c + mp}{n + m}$$

در این فرمول، m یک عدد ثابت است که **«اندازه‌ی نمونه‌ی مساوی (equivalent sample size)»** نامیده می‌شود. روشِ مشخص کردنِ m متفاوت است. اما برای الان، من m را برابر تعداد مقادیر متفاوتی در نظر می‌گیرم که این یک ویژگی می‌تواند داشته باشد. برای مثال، ۲ مقدارِ متفاوت برای نحوه‌ی رای دادنِ یک شخص به لایحه‌ی شنود اینترنت وجود دارد. مقادیر «بلی» و «خیر». پس مقدار m را برابر ۲ قرار می‌دهم. p هم که مقدار تخمین پیشین احتمال است. در اکثر موارد، احتمال یکپارچه را در نظر می‌گیریم. برای مثال، چقدر احتمال دارد که شخصی به «لایحه‌ی شنود» رایِ «خیر» بدهد، با توجه به این‌که هیچ چیزی در مورد این شخص ندانیم؟ ½. پس p در این مثال برابر ½ است.

اجازه بدهید به سراغ همان مثال قبلی برویم تا ببنیم این فرمول به چه شکل کار می‌کند. ابتدا، جدول زیر را نگاه کنید که رای‌ها را نمایش می‌دهد:

رای‌های جمهوری‌خواه‌ها (Republican)

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Yes	99	1	99	50
No	1	99	1	50

رای‌های دموکرات‌ها (Democrats)

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Yes	1	99	1	100
No	99	1	99	0

شخصی که می‌خواهیم او را طبقه‌بندی کنیم، به CISPA رایِ «خیر» داده‌است. ابتدا ما احتمالِ این‌که این شخص جمهوری‌خواه باشد (با این شرط که بدانیم که او رایِ «خیر» را به CISPA داده‌است) محاسبه می‌کنیم. فرمول جدیدمان به این صورت بود:

$$P(x \mid y) = \frac{n_c + mp}{n + m}$$

در این فرمول n، برابر تعداد جمهوری‌خواه‌ها (یعنی برابر ۱۰۰) است و n_c تعداد جمهوری‌خواه‌هایی است که رایِ «خیر» را به CISPA داده‌اند (یعنی تعداد ۱ نفر). m هم که تعداد مقادیر مختلف برای رای است (یعنی همان «بلی» و «خیر») که برابر ۲ می‌شود. پس با جایگذاری این اعداد در فرمولِ بالا، نتیجه به صورت زیر در می‌آید:

$$P(\text{cispa} = \text{no} \mid \text{republican}) = \frac{1 + 2(.5)}{100 + 2} = \frac{2}{102} = 0.01961$$

حالا همین فرآیند را برای شخصی استفاده می‌کنیم که به CISPA رایِ «خیر» داده باشد با این شرط که بدانیم او دموکرات است:

$$P(\text{cispa} = \text{no} \mid \text{republican}) = \frac{99 + 2(.5)}{100 + 2} = \frac{100}{102} = 0.9804$$

با این شواهد، احتمالات مربوط به P(h|D) مانند جدول زیر می‌شود:

h=	p(h)	P(C=no|h)				P(h|D)
Republican	0.5	0.01961				0.0098
Democrat	0.5	0.9804				0.4902

حال می‌توانیم همین کار را برای رای‌های «بلی» که X به لایحه‌ی Reader Privacy Act و همچنین رای‌های «خیر» که X به Internet Sales Tax داده‌است محاسبه کنیم.

مدادتان را تیز کنید

این مسئله را حل کنید. ببینید که آیا این شخص یک دموکرات است یا یک جمهوری‌خواه.

یادتان باشد که این شخص، به CISPA رایِ «خیر» داده است، به لایحه‌ی Reader Privacy Act رایِ «بلی» داده است و به دو لایحه‌ی Internet Sales Tax و Internet Snooping Bill هم رای «خیر» داده است.

مدادتان را تیز کنید – راه حل

این مسئله را حل کنید. ببینید که آیا این شخص یک دموکرات است یا یک جمهوری‌خواه.

یادتان باشد که این شخص، به CISPA رایِ «خیر» داده است، به لایحه‌ی Reader Privacy Act رایِ «بلی» داده است و به دو لایحه‌ی Internet Sales Tax و Internet Snooping Bill هم رای «خیر» داده است.

برای محاسبه‌ی دو ستون دیگر در این مسئله، دقیقاً مانندِ همان فرمول و محاسباتی که برای CISPA کردیم، عمل می‌کنیم. احتمال این‌که این شخص به لایحه‌ی Internet Snooping Bills رای «خیر» داده باشد با این شرط که بدانیم او یک جمهوری‌خواه است به صورت زیر محاسبه می‌شود:

$$\mathrm{P}(\text{cispa} = \text{no} \mid \text{republican}) = \frac{50 + 2(.5)}{100 + 2} = \frac{51}{102} = 0.5$$

و همین احتمال با شرط دموکرات بودنِ شخص:

$$\mathrm{P}(\text{cispa} = \text{no} \mid \text{democrat}) = \frac{0 + 2(.5)}{100 + 2} = \frac{1}{102} = 0.0098$$

با ضرب این احتمالات با یکدیگر، به جدول زیر می‌رسیم:

h=	p(h)	P(C=no|h)	P(R=yes|h)	P(I=no|h)	P(S=no|h)	P(h|D)
Republican	0.5	0.01961	0.01961	0.01961	0.5	0.000002
Democrat	0.5	0.9804	0.9804	0.9804	0.0098	0.004617

پس بر خلاف روش قبلی، ما این شخص را یک دموکرات طبقه‌بندی می‌کنیم. که البته این با شهود ما هم جور در می‌آید.

رفع یک ابهام

در همین مثالِ قبلی، مقدار m برای تمامی محاسبات برابر ۲ بود. با این حال، لزومی ندارد که مقدار m برای تمامیِ ویژگی‌ها (ابعاد) یک مقدار ثابت باشد. مثال نظارت بر سلامتی که قبلاً در مورد آن صحبت کردیم را به یاد بیاورید. ویژگی‌هایی که آن‌جا داشتیم به صورت زیر بود:

پرسشنامه:

دلیل اصلی علاقه‌مندیِ شما برای خرید این محصول چیست؟

- سلامتی (health)
- **ظاهر (appearance)**
- هر دو (both)

(برای این ویژگی، مقدار m برابر ۳ است. چون این ویژگی می‌تواند یکی از سه مقدار (سلامتی، ظاهر یا هر دو) را داشته باشد. اگر احتمالات را یکسان (uniform) در نظر بگیریم، پس در واقع p = 1/3 می‌شود)

سطح تمرینی شما در حال حاضر چقدر است؟

- **غیر متحرک یا نشسته (sedentary)**
- به صورت نسبی (moderate)
- فعال (active)

(در این ویژگی هم، m برابر ۳ و p = 1/3 است)

آیا شما با محصولات فناوری راحت هستید؟

- **بلی (yes)**

- خیر (No)

(برای این ویژگی، m برابر ۲ است، زیرا این ویژگی می‌تواند دو مقدار «بلی» یا «خیر» را بپذیرد و p = 1/2 است زیرا احتمال این‌که یکی از این دو انتخاب شود ½ است)

بیایید فرض کنیم که تعداد افرادی که مورد بررسی قرار گرفتند و محصول i500 را خریداری کرده‌اند برابر ۱۰۰ نفر باشد (یعنی n برابر ۱۰۰ است). تعداد افرادی که i500 دارند و سطح آن‌ها (در سوال دوم) برابر «غیر متحرک یا نشسته - sedentary» است، برابر ۰ عدد است (یعنی n_c برابر صفر). پس احتمال این‌که شخصی در وضعیت «غیر متحرک یا نشسته - sedentary» باشد (با توجه به ویژگی دوم یعنی همان سوال دوم) با این شرط که بدانیم او i500 دارد توسط فرمول زیر می‌شود:

$$\mathrm{P}(sedentary|\mathrm{i500}) = \frac{0 + 3(.333)}{100 + 3} = \frac{1}{103} = 0.0097$$

اعداد

احتمالا متوجه شدید که من داده‌ها را از حالت عددی که در تمام روش‌های نزدیک‌ترین همسایه (nearest neighbor) از آن‌ها استفاده می‌کردم، به حالت طبقه‌ای (categorical) در الگوریتم بیز ساده (Naïve Bayes) تغییر دادم. منظورمان از **«داده‌های طبقه‌ای (categorical data)»** این است که این داده‌ها در دسته‌های گسسته و جدا قرار می‌گیرند. برای مثال، ما مردم را بر اساس رایی که داده‌اند تقسیم می‌کنیم. مثلاً افرادی که به یک لایحه رایِ «بله» داده باشند به یک دسته تعلق می‌گیرند و افرادی که رایِ «خیر» داده باشند، به دسته‌ای دیگر. یا ممکن است موسیقی‌دان‌ها را با سازی که می‌نوازند تقسیم‌بندی کنیم. در نتیجه، تمامیِ ساکسیفون‌نوازها به یک قسمت تعلق پیدا می‌کنند و تمامیِ درام‌نوازها به قسمتی دیگر، و همین‌طور تمامیِ پیانیست‌ها نیز به یک قسمت دیگر و... . این دسته‌ها و قسمت‌ها یک مقیاس مشخص ندارد. برای مثال، ساکسیفون‌نوازها نسبت به درام‌نوازها، به پیانونوازها نزدیک‌تر نیستند. این در حالی است که داده‌های عددی، در یک مقیاس مشخص به کار گرفته می‌شوند. برای مثال، دستمزدِ سالانه‌ی ۱۰۵۰۰۰ دلاری به نسبتِ دستمزدِ ۴۰۰۰۰ دلاری، به دستمزدِ ۱۱۰۰۰۰ دلاری نزدیک‌تر است.

در روش‌های بیزین (Bayesian) ما چیزهای مختلف را می‌شماریم – چه تعدادی از افراد غیر متحرک یا نشسته (در مثال خرید دستگاه نظارت بر سلامتی) هستند – و البته در ابتدا مشخص نیست که چگونه می‌توانیم چیزهایی را که در یک مقیاس هستند (مثلا عددی هستند) بشماریم. برای مثال، چگونه می‌توانیم میانگینِ نمرات را بشماریم؟ برای پاسخ به این سوال دو روش وجود دارد:

روش ۱: ساخت طبقه‌ها و دسته‌بندی

یک روش این است که طبقه‌ها و دسته‌بندی‌های مختلفی را با استفاده از روش‌های گسسته‌سازیِ ویژگی‌ها انجام دهیم. این کار را معمولاً در وب‌سایت‌ها یا فرم‌های پرسش‌نامه دیده‌اید. برای مثال:

سن:

. کوچک‌تر از ۱۸
. ۱۸ تا ۲۲
. ۲۳ تا ۳۰
. ۳۰ تا ۴۰
. بزرگ‌تر از ۴۰

درآمد سالیانه:

. بیشتر از ۲۰۰۰۰۰ دلار
. بین ۱۵۰۰۰۰ تا ۲۰۰۰۰۰
. بین ۱۰۰۰۰۰ تا ۱۵۰۰۰۰
. بین ۶۰۰۰۰ تا ۱۰۰۰۰۰
. بین ۴۰۰۰۰ تا ۶۰۰۰۰

هنگامی که ما این اطلاعات را (که مقادیر پیوسته هستند) به مقادیر گسسته تبدیل کنیم، می‌توانیم از روش بیز ساده (Naïve Bayes) مانند قبل استفاده کنیم.

روش ۲: توزیع‌های گوسی (Gussian)

عبارت‌های «توزیع گوسی (Gussian Distribution)» و «تابع چگالی احتمال (Probability Density Function)» به نظر جالب می‌آیند، اما این عبارات بیشتر از یک عبارتِ عادی و ساده، جذاب هستند. اجازه بدهید ببینیم معنی آن‌ها چیست و چگونه می‌توان از آن‌ها در بیز ساده (Naïve Bayes) استفاده کرد. مثالی که را تا این‌جا (در مورد نظارت بر سلامت با محصولات i100 و i500) مرور می‌کردیم را در نظر بگیرید. یک ویژگیِ دیگر به اسم درآمد (Income) نیز به این داده‌ها اضافه شده است:

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Income (in $1,000)	Model #
both	sedentary	moderate	yes	60	i100
both	sedentary	moderate	no	75	i100
health	sedentary	moderate	yes	90	i500
appearance	active	moderate	yes	125	i500
appearance	moderate	aggressive	yes	100	i500
appearance	moderate	aggressive	no	90	i100
health	moderate	aggressive	no	150	i500
both	active	moderate	yes	85	i100
both	moderate	aggressive	yes	100	i500
appearance	active	aggressive	yes	120	i500
both	active	aggressive	no	95	i500
health	active	moderate	no	90	i500
health	sedentary	aggressive	yes	85	i500
appearance	active	moderate	no	70	i100
health	sedentary	moderate	no	45	i100

یک خریدارِ عادی، که محصول i500 ما را خریداری کرده است در نظر بگیرید. این محصول، محصولی فوق‌العاده و درجه یک است. اگر از شما بخواهم این شخص را توصیف کنید، امکان دارد میانگین درآمد را برای خریداران این محصول به من بگویید:

$$\text{mean} = \frac{90 + 125 + 100 + 150 + 100 + 120 + 95 + 90 + 85}{9}$$
$$= \frac{955}{9} = 106.111$$

و شاید با مطالعه‌ی فصل چهارم از این کتاب، انحراف استاندارد را هم محاسبه کنید:

$$\text{sd} = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{\text{card}(x)}}$$

حتماً یادتان می‌آید که انحراف استاندارد (sd) بازه‌ی پراکندگی را نشان می‌داد. اگر تمامی مقادیر در محدوده‌ی نزدیک به میانگین قرار داشته باشند، انحراف استاندارد هم کم است. ولی اگر مقادیر پراکنده باشند، انحراف استاندارد نیز زیاد است.

مدادتان را تیز کنید

انحراف استاندارد درآمد برای افرادی که i500 خریده‌اند، چقدر است؟ (این مقادیر در جدول زیر آمده است):

Income (in $1,000)
90
125
100
150
100
120
95
90
85

مدادتان را تیز کنید – راه‌حل

انحراف استاندارد درآمد برای افرادی که i500 خریده‌اند، چقدر است؟ (این مقادیر در جدول زیر آمده است):

Income (in $1,000)	(x-106.111)	(x-106.111)²
90	-16.111	259.564
125	18.889	356.794
100	-6.111	37.344
150	43.889	1926.244
100	-6.111	37.344
120	13.889	192.904
95	-11.111	123.454
90	-16.111	259.564
85	-21.111	445.674
	∑ =	3638.889

$$sd = \sqrt{\frac{3638.889}{9}}$$

$$= \sqrt{404.321} = 20.108$$

انحراف استانداردِ جمعیت و انحراف استاندارد نمونه

فرمولی که در مثال بالا برای انحراف استاندارد استفاده کردیم، **انحراف استاندارد جمعیت (population standard deviation)** نام دارد. این نام را برای این فرمول گذاشته‌اند، زیرا هنگامی از این فرمول استفاده می‌کنیم که **داده‌هایمان شامل تمامی جمعیتی می‌شود که می‌خواهیم در مورد آن‌ها مطالعه و تحلیل کنیم**. برای مثال، ممکن است بخواهیم یک آزمایش را بر روی ۵۰۰ دانش‌آموز انجام دهیم و سپس میانگین و انحراف استاندارد را برای

نتیجه‌ی آن آزمایش به دست آوریم. در این مورد، ما از انحراف استاندارد جمعیت استفاده می‌کنیم. با این حال، معمولاً داده‌های ما کامل نیستند، یعنی داده‌های مربوط به تمامیِ جمعیت را نداریم. برای مثال، فرض کنید که من می‌خواهم تاثیر خشکسالی را بر موش‌های نَر در نیومکزیکوی شمالی اندازه‌گیری کنم و به عنوان بخشی از تحقیق، میانگین و انحراف استاندارد وزنِ این موش‌ها را اندازه بگیرم. در این مورد، طبیعتاً تک‌تکِ موش‌های نیومکزیکوی شمالی را وزن‌کشی نمی‌کنم. به جای این کار، یک نمونه‌ی کوچکتری از موش‌ها را جمع‌آوری کرده و وزنِ آن‌ها را اندازه می‌گیرم.

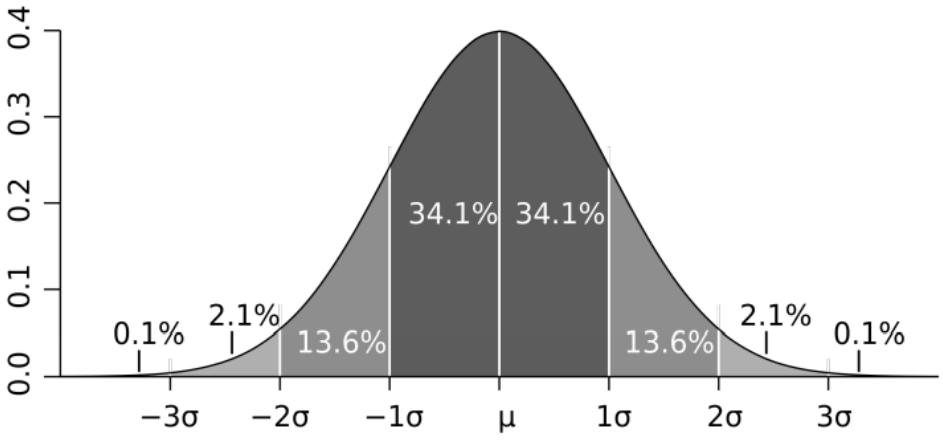
در این مثال، با این‌که می‌توانم از فرمول بالا، برای انحراف استاندراد استفاده کنم، ولی فرمول دیگری وجود دارد که می‌تواند بهتر و دقیق‌تر انحراف استاندارد را برای تمامیِ جمعیت، تخمین بزند. این فرمول به **انحراف استانداردِ نمونه (sample standard deviation)** معروف است و با تغییر کمی نسبت به فرمول قبل، به این فرمول می‌رسیم:

$$\mathrm{sd} = \sqrt{\frac{\sum_i (\mathrm{x}_i - \bar{\mathrm{x}})^2}{\mathrm{card}(\mathrm{x}) - 1}}$$

پس انحراف استاندارد برای ویژگیِ «درآمد (Income)» با توجه به این فرمول به صورت زیر محاسبه می‌شود:

$$\text{sd} = \sqrt{\frac{3638.889}{9-1}} = \sqrt{454.861} = 21.327$$

در ادامه‌ی این فصل، ما از انحراف استانداردِ نمونه استفاده خواهیم کرد.
احتمالاً چیزهایی درباره‌ی **توزیع نرمال (normal distribution)**، نمودار زنگوله‌ای یا توزیع گوسی شنیده‌اید. توزیع گوسی در واقع همان توزیع نرمال است. تابعی که این توزیع را معرفی می‌کند، **تابع گوسی (gussian) یا تابع زنگوله (bell)** است. در اکثر مواقع، محاسبه‌گران (همان داده‌کاوها) فرضشان این است که ویژگی‌ها از یک توزیع گوسی پیروی می‌کنند. و منظورشان این است که حدود ۶۸ درصد از نمونه‌ها در یک توزیع گوسی، در فاصله‌ی یک انحراف استاندارد، نسبت به میانگین، قرار دارند و ۹۵ درصد نمونه‌ها، در فاصله‌ی ۲ برابریِ انحراف استاندارد نسبت به میانگین قرار می‌گیرند:



(در شکلِ بالا، μ میانگین است و σ انحراف استاندارد)

در مثال ما، میانگین درآمد ۱۰۶,۱۱۱ و انحراف استاندارد نمونه برابر ۲۱,۳۲۷ بود. پس ۹۵ درصد از افرادی که i500 خریده بودند، در محدوده‌ی ۴۲۶۶۰ دلار تا ۱۴۹۷۷۰ دلار درآمد

دارند (با توجه به توزیع گوسی). اگر از شما سوال کنم P(100k| i500) چقدر می‌شود – یعنی احتمال این‌که شخصی که i500 خریده است، درآمد ۱۰۰۰۰۰ دلاری داشته باشد – احتمالاً فکر می‌کنید که این درآمد برای این شخص محتمل است. حالا اگر سوال کنم که احتمال P(20k| i500) – یعنی احتمال این‌که شخصی که i500 خریده است، درآمد ۲۰۰۰۰ دلاری داشته باشد – چقدر است، احتمالاً فکر می‌کنید که این درآمد برای این شخص، کمی غیر معقول است.

برای این‌که این قضیه را فرموله کنیم، به سراغ میانگین و انحراف استاندارد می‌رویم تا بتوانیم با کمک آن‌ها، این احتمال را به صورت زیر محاسبه کنیم:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

شاید اگر کمی این فرمول را بزرگ‌تر کنیم، ساده‌تر به نظر برسد!

هر وقت که یک فرمول به ظاهر پیچیده را در این کتاب می‌نویسم، حس می‌کنم باید سریعاً بگویم «نترسید». البته ممکن است که هیچ‌کدام از شما خوانندگان نترسیده باشید و من تنها کسی باشم که دچار ترس شده است.

با این‌حال اجازه بدهید که این را بگویم. داده‌کاوی دارای فرمول‌ها و واژگان حرفه‌ای است. قبل از این‌که به حوزه‌ی داده‌کاوی وارد بشوید، ممکن است فکر کنید که این چیزها به نظر سخت می‌آیند. ولی بعد از خواندنِ آن‌ها، حتی برای مدت کوتاهی، متوجه می‌شوید که این فرمول‌ها چیز خاصی نیستند. در واقع فقط باید قدم به قدم با فرول جلو برویم و روی آن کار کنیم.

حال اجازه دهید سریعاً به واکاویِ این فرمول بپردازیم، تا بفهمیم که چقدر این فرمول ساده است. فرض کنید می‌خواهیم P(100k|i500) را حساب کنیم. یعنی احتمال این‌که شخصی

۱۰۰۰۰۰ دلار درآمد داشته باشد به شرطی که بدانیم این شخص i500 خریده است. چند صفحه قبل، ما میانگین درآمد برای افرادی را که i500 خریده‌اند را محاسبه کردیم. همچنین انحراف استاندارد نمونه را هم حساب کردیم. این مقادیر در زیر قابل مشاهده هستند. اگر بخواهیم به صورت محاسباتی صحبت کنیم، ما میانگین را به صورت حروف یونانی μ نشان داده‌ایم. همچنین انحراف استاندارد را با σ نشان می‌دهیم:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

$\mu_{ij} = 106.111$
$\sigma_{ij} = 21.327$
$x_i = 100$

حالا اجازه دهید این مقادیر را به فرمول تبدیل کنیم:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}(21.327)} e^{\frac{-(100-106.111)^2}{2(21.327)^2}}$$

حالا کمی محاسباتش را انجام دهیم:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{6.283}(21.327)} e^{\frac{-(37.344)}{909.68}}$$

و کمی بیشتر:

$$P(x_i \mid y_j) = \frac{1}{53.458} e^{-0.0411}$$

حرف e، یک ثابت ریاضی است که بر مبنای لگاریتم طبیعی کار می‌کند. مقدار آن به طور تقریبی برابر ۲,۷۱۸ است. پس:

$$P(x_i \mid y_j) = \frac{1}{53.458}(2.718)^{-0.0411} = (0.0187)(0.960) = 0.0180$$

پس احتمال این‌که درآمد شخصی که i500 خریده است، ۱۰۰۰۰۰ دلار باشد، برابر ۰٫۰۱۸۰ است.

مدادتان را تیز کنید

در جدول زیر، من قدرت اسب بخار (HP) را برای اتومبیل‌هایی که ۳۵ مایل به ازای هر گالن بنزین مصرف می‌کنند را آورده‌ام. می‌خواهم بدانم، احتمال این‌که یک Datsun 210 دارای ۱۳۲ اسب بخار باشد با این شرط که بدانیم این اتومبیل ۳۵ مایل را با یک گالن بنزین می‌پیماید، چقدر است.

car	HP
Datsun 210	65
Ford Fiesta	66
VW Jetta	74
Nissan Stanza	88
Ford Escort	65
Triumph tr7 coupe	88
Plymouth Horizon	70
Suburu DL	67

μ_{ij} = _____

σ_{ij} = _____

x_i = _____

مدادتان را تیز کنید – راه حل – بخش ۱

در جدول زیر، من قدرت اسب بخار را برای اتومبیل‌هایی که ۳۵ مایل به ازای هر گالن بنزین مصرف می‌کنند را آورده‌ام. تمایل دارم که بدانم، احتمال این‌که یک **Datsun 210** دارای ۱۳۲ اسب بخار باشد با این شرط که بدانیم این اتومبیل ۳۵ مایل را با یک گالن بنزین می‌پیماید، چقدر است.

car	HP
Datsun 210	65
Ford Fiesta	66
VW Jetta	74
Nissan Stanza	88
Ford Escort	65
Triumph tr7 coupe	88
Plymouth Horizon	70
Suburu DL	67

$\mu_{ij} = 72{,}875$

$\sigma_{ij} = 9.804$

$x_i = 132$

$$\sigma = \sqrt{\frac{(65-\mu)^2+(66-\mu)^2+(74-\mu)^2+(88-\mu)^2+(65-\mu)^2+(88-\mu)^2+(70-\mu)^2+(67-\mu)^2}{7}}$$

$$\sigma = \sqrt{\frac{672.875}{7}} = \sqrt{96.126} = 9.804$$

مدادتان را تیز کنید – راه‌حل – بخش ۲

در جدول زیر، من قدرت اسب بخار را برای اتومبیل‌هایی که ۳۵ مایل به ازای هر گالن بنزین مصرف می‌کنند را آورده‌ام. تمایل دارم که بدانم، احتمال این‌که یک **Datsun 210**

دارای ۱۳۲ اسب بخار باشد با این شرط که بدانیم این اتومبیل ۳۵ مایل را با یک گالن بنزین می‌پیماید، چقدر است.

$\mu_{ij} = 72{,}875$

$\sigma_{ij} = 9.804$

$x_i = 132$

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} e^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

$$P(132hp \mid 35mpg) = \frac{1}{\sqrt{2\pi}(9.804)} e^{\frac{-(132-72.875)^2}{2(9.804)^2}}$$

$$= \frac{1}{\sqrt{6.283}(9.804)} e^{\frac{-3495.766}{192.237}} = \frac{1}{24.575} e^{-18.185}$$

$$= 0.0407(0.00000001266)$$

$$= 0.0000000005152$$

خب! به نظر خیلی غیر محتمل می‌رسد که **Datsun 280z**، با این شرط که بدانیم ۳۵ مایل به ازای هر گالن مصرف می‌کند، ۱۳۲ اسب بخار قدرت داشته باشد. (ولی این‌طور شد!).

یک سری نکات برای پیاده‌سازی

در فاز آموزش برای الگوریتم بیزین ساده (Naïve Bayes)، ما میانگین و انحراف استانداردِ نمونه را برای ویژگی عددی محاسبه خواهیم کرد. کمی جلوتر، خواهیم دید که چگونه می‌توانیم این‌ها را پیاده‌سازی کنیم.
در فاز طبقه‌بندی، فرمول بالا، می‌تواند توسط چند خط کد پایتون، پیاده‌سازی شود:

```
import math
def pdf(mean, ssd, x):
    """Probability Density Function computing P(x|y)
    input is the mean, sample standard deviation for all
 the items in y,
    and x."""
    ePart = math.pow(math.e, -(x-mean)**2/(2*ssd**2))
    return (1.0 / (math.sqrt(2*math.pi)*ssd)) * ePart
```

بیایید این تابع را با همان مثال بالا امتحان کنیم:

```
>>>pdf(106.111, 21.327, 100)
0.017953602706962717

>>>pdf(72.875, 9.804, 132)
5.152283971078022e-10
```

خب! حالا کمی استراحت نیاز داریم.

پیاده‌سازی پایتون

فاز یادگیری

روش بیزین ساده، بر اساس احتمالات پیشین و احتمالات شرطی استوار است. اجازه بدهید به مثال دموکرات/جمهوری‌خواه برگردیم. احتمال پیشین (prior probability) احتمالی است که قبل از مشاهده‌ی هر رخدادی، نگه داشته می‌شوند. برای مثال، اگر من بدانم که تعداد ۲۳۳ جمهوری‌خواه و ۲۰۰ دموکرات وجود دارند، پس احتمال پیشین، برای این‌که یک فرد دلخواه در مجلس نمایندگان آمریکا جمهوری‌خواه (republican) باشد به صورت زیر محاسبه می‌شود:

$$\mathrm{P(republican)} = \frac{233}{433} = 0.54$$

که این نشان‌دهنده‌ی P(h) است. احتمال شرطی (conditional probability) یا همان P(h|D) احتمال این است که h را به دست آوریم با این شرط که D را بدانیم. برای مثال، P(democrat|bill1Vote=yes). در بیز ساده (Naïve Bayes) ما احتمال را عوض می‌کنیم و P(D|h) را محاسبه می‌کنیم. برای مثال، P(bill1Vote=yes|democrat).

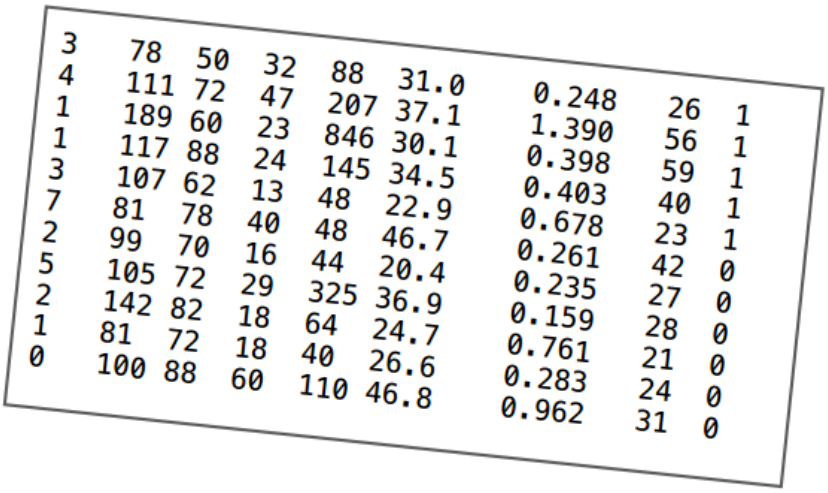
در این کدِ پایتون، ما این احتمالات شرطی را در یک دیکشنری (dictionary در پایتون) به صورت زیر نگه‌داری می‌کنیم:

```
{'democrat': {'bill 1': {'yes': 0.333, 'no': 0.667},
              'bill 2': {'yes': 0.778, 'moderate': 0.222}}

 'republican': {'bill 1': {'yes': 0.811, 'no': 0.189},
                'bill 2': {'yes': 0.250, 'no': 0.750}}}
```

پس احتمال این‌که شخصی به لایحه‌ی شماره‌ی ۱ (bill 1) رای «بلی» داده باشد با این شرط که بدانیم او یک دموکرات است، P(bill 1=yes|democrat) برار ۰,۳۳۳ می‌شود.
ما این ساختار داده‌ای را برای ویژگی‌هایی استفاده می‌کنیم که مقادیرشان گسسته است (برای مثال «بلی» و «خیر»، یا «مرد» و «زن»). با این حال، هنگامی که ویژگی‌ها، مقادیر عددی هستند، ما از **تابع چگالی احتمال (probability density function)** استفاده می‌کنیم و برای این کار بایستی میاگین و انحراف استانداردِ نمونه را برای آن ویژگیِ خاص، نگه داریم. برای این مقادیرِ عددی، از ساختار زیر استفاده خواهم کرد:

```
mean = {'democrat': {'age': 57,  'years served': 12}
        'republican': {'age': 53, 'years served': 7}}

ssd = {'democrat': {'age': 7,  'years served': 3}
       'republican': {'age': 5, 'years served': 5}}
```

مانند گذشته، هر کدام از نمونه‌ها با یک خط در فایل داده نشان داده می‌شود. ویژگی‌های هر کدام از نمونه‌ها با tab از هم جدا می‌شوند. برای مثال، چند خط اول فایل داده، برای مجموعه‌ی داده‌ی دیابت پیما در هند (Pima Indians Diabetes) به صورت زیر است:



ستون‌ها به ترتیب بیان‌گر، تعداد دفعات باردار شدن، غلظت گلوکز پلاسما، فشار خون، ضخامت ماهیچه‌ی سه سر، سطح انسولین سرم، شاخص توده‌ی بدنی، شجره‌نامه دیابت، سن است. و در ستون آخر هم ۱ نشان‌دهنده‌ی داشتن دیابت و ۰ نشان‌دهنده‌ی نداشتن دیابت است.

همچنین مانند گذشته، می‌خواهیم نشان دهیم که چگونه برنامه، هر کدام از ستون‌ها را به وسیله‌ی فرمت آن‌ها تفسیر می‌کند. این برنامه از عبارات زیر استفاده می‌کند:

عبارت attr، ستون‌هایی هستند که دارای مقدار غیر عددی هستند و از روش بیز (Bayes) که قبلاً در این فصل نشان داده شد، استفاده می‌کنند.

عبارت num، ستون‌هایی را مشخص می‌کند که بایستی به عنوان ویژگیِ عددی تفسیر شوند. این ستون‌ها از تابع چگالی احتمال (probability density function) بهره می‌برند. یعنی برای این ستون‌ها بایستی مقدار میانگین و انحراف استاندارد را در هنگام آموزش محاسبه کنیم.

عبارت class نمایان‌گر ستونی است که طبقه‌ی یک نمونه را مشخص می‌کند (در واقع چیزی که می‌خواهیم یاد بگیریم)

در مجموعه‌ی داده‌ی دیابت پیما در هند (Pima Indian Diabetes)، فرمت به صورت زیر خواهد بود:

```
"num    num    num    num    num    num    num    num    class"
```

برای محاسبه‌ی میانگین و انحراف استانداردِ نمونه، به یک ساختار داده‌ایِ موقتی احتیاج داریم تا از آن در فاز آموزش استفاده کنیم. پدوباره، اجازه دهید به یک قسمتی از مجموعه‌ی داده‌ی دیابت پیما (Pima)، نگاهی بیندازیم:

```
3   78  50  32  88  31.0  0.248  26  1
4   111 72  47  207 37.1  1.390  56  1
1   189 60  23  846 30.1  0.398  59  1
2   142 82  18  64  24.7  0.761  21  0
1   81  72  18  40  26.6  0.283  24  0
0   100 88  60  110 46.8  0.962  31  0
```

ستون آخر، نشان‌گر طبقه‌ی هر نمونه است. پس سه نمونه‌ی اول (یعنی سه نفر اول) دیابت دارند و سه نمونه‌ی آخر در شکل بالا دیابت ندارند. تمامیِ ستون‌های دیگر، ویژگی‌های عددی هستند. حالا می‌خواهیم میانگین و انحراف استاندارد برای هر کدام از این دو طبقه محاسبه کنیم. برای محاسبه‌ی میانگین در هر طبقه و هر ویژگی، نیاز دارم تا حواسم را به همه‌ی آن‌ها جمع کنم. در کدِ فعلی ما، در حال حاضر تعداد همه‌ی نمونه‌ها را ذخیره کرده‌ایم. من این را با استفاده از یک دیکشنری، پیاده‌سازی کرده‌ام:

```
totals   {'1': {1: 8, 2: 378, 3: 182, 4: 102, 5: 1141,
              6: 98.2, 7: 2.036, 8: 141},

         {'0': {1: 3, 2: 323, 3: 242, 4: 96, 5: 214,
              6: 98.1, 7: 2.006, 8: 76}
```

پس برای طبقه‌ی ۱ (class 1)، جمع ستونِ ۱ برابر ۸ است (۳ + ۴ + ۱)، جمع ستونِ ۲ برابر ۳۷۸ و همین‌طور تا آخر است. برای طبقه‌ی ۰ (class 0)، جمع ستون ۱ برابر ۳ (۲ + ۱ + ۰)، جمع ستون ۲ برابر ۳۲۳ است و همین‌طور تا آخر.

برای انحراف استاندارد، بایستی داده‌های اصلی و اولیه را نیز نگه داریم، و برای آن، یک دیکشنری با فرمت زیر می‌سازیم:

```
numericValues
        {'1': 1: [3, 4, 1], 2: [78, 111, 189], ...},
        {'0': {1: [2, 1, 0], 2: [142, 81, 100]}
```

من این کدها را اضافه کردم تا این ساختار داده‌ای موقت را در تابع __init__() در کلاس Classifier مانند کد زیر بسازم:

```
import math
class Classifier:
    def __init__(self, bucketPrefix, testBucketNumber, d
ataFormat):
        """ a classifier will be built from files with t
he bucketPrefix
        excluding the file with textBucketNumber. dataFo
rmat is a string that
        describes how to interpret each line of the data
 files. For example,
        for the iHealth data the format is:
        "attr attr attr attr class"
        """
        total = 0
        classes = {}
        # counts used for attributes that are not numeri
c
        counts = {}
        # totals used for attributes that are numereric
        # we will use these to compute the mean and samp
le standard deviation
        # for each attribute - class pair.
        totals = {}
        numericValues = {}
        # reading the data in from the file
        self.format = dataFormat.strip().split('\t')
        #
        self.prior = {}
        self.conditional = {}
        # for each of the buckets numbered 1 through 10:
```

```
        for i in range(1, 11):
            # if it is not the bucket we should ignore,
read in the data
            if i != testBucketNumber:
                filename = "%s-%02i" % (bucketPrefix, i)
                f = open(filename)
                lines = f.readlines()
                f.close()
                for line in lines:
                    fields = line.strip().split('\t')
                    ignore = []
                    vector = []
                    nums = []
                    for i in range(len(fields)):
                        if self.format[i] == 'num':
                            nums.append(float(fields[i])
)
                        elif self.format[i] == 'attr':
                            vector.append(fields[i])
                        elif self.format[i] == 'comment'
:
                            ignore.append(fields[i])
                        elif self.format[i] == 'class':
                            category = fields[i]
                    # now process this instance
                    total += 1
                    classes.setdefault(category, 0)
                    counts.setdefault(category, {})
                    totals.setdefault(category, {})
                    numericValues.setdefault(category, {
})
                    classes[category] += 1
                    # now process each non-numeric attri
bute of the instance
                    col = 0
                    for columnValue in vector:
                        col += 1
                        counts[category].setdefault(col,
 {})
                        counts[category][col].setdefault
(columnValue, 0)
                        counts[category][col][columnValu
e] += 1
                    # process numeric attributes
                    col = 0
                    for columnValue in nums:
```

```
                        col += 1
                        totals[category].setdefault(col,
 0)
                        #totals[category][col].setdefaul
t(columnValue, 0)
                        totals[category][col] += columnV
alue
                        numericValues[category].setdefau
lt(col, [])
                        numericValues[category][col].app
end(columnValue)
        #
        # ok done counting. now compute probabilities
        # first prior probabilities p(h)
        #
        for (category, count) in classes.items():
            self.prior[category] = count / total
        #
        # now compute conditional probabilities p(h|D)
        #
        for (category, columns) in counts.items():
            self.conditional.setdefault(category, {})
            for (col, valueCounts) in columns.items():
                self.conditional[category].setdefault(co
l, {})
                for (attrValue, count) in valueCounts.it
ems():
                    self.conditional[category][col][attr
Value] = (
                        count / classes[category])
        self.tmp = counts
        #
        # now compute mean and sample standard deviation
        #
```

کد بزنید

آیا می‌توانید کدی بزنید که میانگین و انحراف استاندارد را محاسبه کند؟ فایل naiveBayesDensityFunctionTraining.py را از سایت guidetodatamining.com یا chistio.ir دانلود کنید.

برنامه‌ی شما بایستی ساختارهای داده‌های «انحراف استاندارد نمونه» و «میانگین» را مانند زیر تولید کند:

```
c = Classifier("pimaSmall/pimaSmall",  1,
               "num num    num    num    num    num    num    num    class")
>> c.ssd
{'0': {1: 2.54694671925252, 2: 23.454755259159146,  ...},
 '1': {1: 4.21137914295475, 2: 29.52281872377408,}}
>>> c.means
{'0': {1: 2.8867924528301887, 2: 111.90566037735849,  ...},
 '1': {1: 5.25, 2: 146.05555555555554, ...}}
```

کد بزنید – راه حل

این راه حل من است:

```
#
# now compute mean and sample standard deviation
#
self.means = {}
self.ssd = {}
self.totals = totals
for (category, columns) in totals.items():
    self.means.setdefault(category, {})
    for (col, cTotal) in columns.items():
        self.means[category][col] = cTotal / classes[c
ategory]
# standard deviation
for (category, columns) in numericValues.items():
    self.ssd.setdefault(category, {})
    for (col, values) in columns.items():
        SumOfSquareDifferences = 0
        theMean = self.means[category][col]
        for value in values:
            SumOfSquareDifferences += (value - theMean
)**2
        columns[col] = 0
        self.ssd[category][col] = math.sqrt(SumOfSquar
eDifferences
            / (classes[category] - 1))
```

فایلی که این راه‌حل را دارد به اسم naiveBayesDensityFunctionTrainingSolution.py در وب‌سایت ما قابل دانلود است.

کد بزنید ۲

آیا می‌توانید تابع طبقه‌بندی را طوری بازنگری کنید که از تابع «چگالیِ احتمالی (probability density function)» برای مقادیر عددی استفاده کند؟ فایلی که باید تغییرات در آن انجام شود naiveBayesDensityFunctionTemplate.py است. در زیر تابع طبقه‌بندیِ اصلی نمایش داده شده است:

```
def classify(self, itemVector, numVector):
    """Return class we think item Vector is in"""
    results = []
    sqrt2pi = math.sqrt(2 * math.pi)
    for (category, prior) in self.prior.items():
        prob = prior
        col = 1
        for attrValue in itemVector:
            if not attrValue in self.conditional[categ
ory][col]:
                # we did not find any instances of thi
s attribute value
                # occurring with this category so prob
 = 0
                prob = 0
            else:
                prob = prob * self.conditional[categor
y][col][attrValue]
            col += 1
    # return the category with the highest probability
    #print(results)
    return(max(results)[1])
```

کد بزنید ۲ – راه حل

آیا می‌توانید تابع طبقه‌بندی را بازنگری کنید به طوری که از تابع «چگالی احتمالی (probability density function)» برای مقادیر عددی استفاده کند؟ فایلی که باید تغییرات در آن انجام شود naiveBayesDensityFunctionTemplate.py است.

راه حل:

```
def classify(self, itemVector, numVector):
    """Return class we think item Vector is in"""
    results = []
    sqrt2pi = math.sqrt(2 * math.pi)
    for (category, prior) in self.prior.items():
        prob = prior
        col = 1
        for attrValue in itemVector:
            if not attrValue in self.conditional[categ
ory][col]:
                # we did not find any instances of thi
s attribute value
                # occurring with this category so prob
 = 0
                prob = 0
            else:
                prob = prob * self.conditional[categor
y][col][attrValue]
            col += 1
        col = 1
        for x in numVector:
```

```
            mean = self.means[category][col]
            ssd = self.ssd[category][col]
            ePart = math.pow(math.e, -(x - mean)**2/(2
*ssd**2))
            prob = prob * ((1.0 / (sqrt2pi*ssd)) * ePa
rt)
            col += 1
        results.append((prob, category))
    # return the category with the highest probability
    #print(results)
    return(max(results)[1])
```

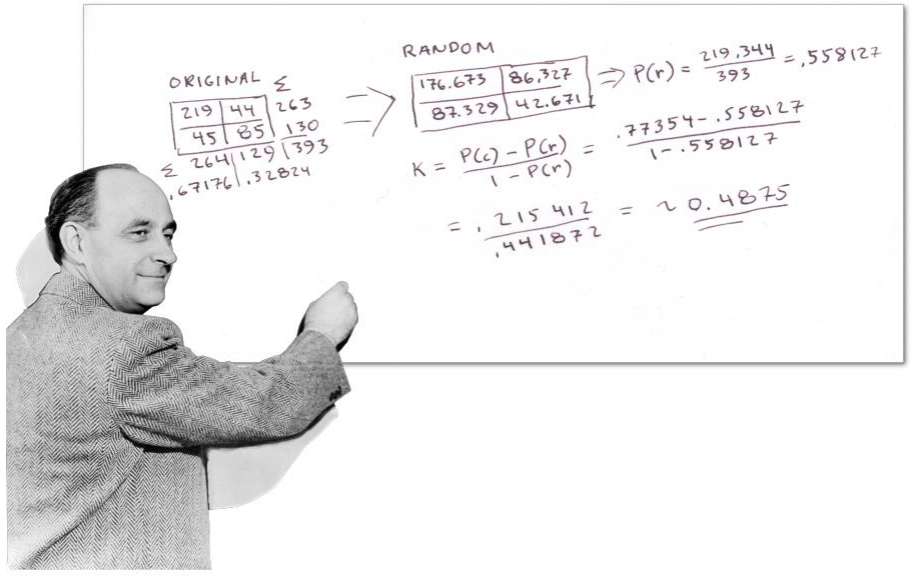
آیا این راه حل بهتر از روش« نزدیک‌ترین همسایه (Nearest Neighbor)» است؟

در فصل ۵، ما الگوریتم k نزدیک‌ترین همسایه (KNN) را بر روی مجموعه داده‌های دیابت پیما (Pima) و یک زیر مجموعه‌ای از آن که pimaSmall نام داشت، آزمایش کردیم. اگر خاطرتان باشد، نتایج به صورت زیر بود:

	pimaSmall	pima
k=1	59.00%	71.247%
k=3	61.00%	72.519%

نتایج برای الگوریتم «بیز ساده (Naïve Bayes)» به صورت زیر است:

	pimaSmall	pima
Bayes	72.000%	77.354%



امتیاز کاپا برا بیز ساده برابر ۰٫۴۸۷۵ شد، که به صورت نسبی عدد خوبی است.

پس برای این مثال، بیز ساده (Naïve Bayes) بهتر از k نزدیک‌ترین همسایه (KNN) شده است.

مزایای روش بیز

- پیاده‌سازی ساده (فقط نیاز به شمارش چیزهای مختلف دارد)
- نیاز به داده‌های کمتر برای آموزش نسبت به روش‌های دیگر دارد
- روش خوبی است برای وقتی که هم کیفیت و دقت بالا و هم سرعت اجرای مناسب را احتیاج دارید

مزایای KNN

- پیاده‌سازی ساده
- فرضِ این‌که داده‌ها دارای یک ساختار مشخص هستند را ندارد – یک ویژگی مهم!
- مقدار زیادی حافظه برای ذخیره‌ی داده‌های آموزشی احتیاج دارد

عیب اصلی روش بیز

این روش، تعاملات بین ویژگی‌ها را یاد نمی‌گیرد. برای مثال، نمی‌تواند این نکته را یاد بگیرد که من غذاهای پنیری را دوست دارم و همچنین غذاهای برنجی را هم دوست دارم. ولی غذاهایی که هم پنیر و هم برنج داشته باشد را دوست ندارم.

روش KNN

روش KNN یک انتخاب معقول و منطقی برای زمانی هست که مجموعه‌ی آموزشی بزرگ باشد. روش KNN یک روش کاملاً همه‌کاره (مثل چاقوی سوییسی) است و می‌تواند در رشته‌ها و مثال‌های متعددی کاربرد داشته باشد. برای مثال سیستم‌های پیشنهادگر، پروتئومیکس (دانش بررسی مجموعه‌ای از پروتئین‌ها در یک ارگان)، تعاملات بین پروتئین‌ها، و طبقه‌بندی تصاویر از نمونه‌هایی هستند که KNN در آن‌ها کاربرد دارد.

چیزی که ما را برای ضرب احتمالات در یکدیگر توانمند می‌کند این حقیقت است این احتمالات از یکدیگر **مستقل** هستند. برای مثال، یک بازی را تصور کنید که در آن یک سکه را پرتاب می‌کنیم و یک تاس می‌اندازیم. این دو رویداد از یکدیگر مستقل هستند به این معنا که مقداری که در تاس می‌افتد به رو یا پشت آمدنِ سکه ارتباطی ندارد و در واقع به آن وابسته نیست. و همان‌طورکه گفتم، اگر رخدادها مستقل (غیر وابسته) باشند می‌توانیم **احتمال توام (joint probability)** را با ضرب احتمالات هر کدام از رخدادها در یکدیگر به دست بیاوریم (منظور از احتمال توام یا همان joint probability، احتمال اتفاق افتادن هر دو رخداد با هم است). پس احتمال این‌که سکه به رو بیاید و تاس هم عدد شش بشود به صورت زیر محاسبه می‌گردد:

$$\mathrm{P}(\text{heads} \wedge 6) = \mathrm{P}(\text{heads}) \times \mathrm{P}(6) = 0.5 \times \frac{1}{6} = 0.08333$$

فرض کنید من یک سری کارت بازی دارم که دارای ۵۲ ورق است. ۲۶ ورق سیاه و ۲۶ ورق قرمز. ۱۲ تا از این ورق‌ها دارای عکس هستند و بقیه‌ی ۴۰ تای دیگر، عکس‌دار نیستند. حالا من فقط ورق‌های قرمز را انتخاب می‌کنم (۲۶ عدد) ولی از میان ورق‌های سیاه، آن‌هایی که عکس‌دار هستند را هم انتخاب می‌کنم (۶ عدد) که جمعاً برابر ۳۲ کارت می‌شود. احتمال انتخاب یک ورق عکس‌دار (facecard) چقدر است؟

$$\mathrm{P}(\text{ facecard}) = \frac{12}{32} = 0.375$$

احتمال انتخاب یک کارت قرمز؟

$$\mathrm{P}(\text{red}) = \frac{6}{32} = 0.1875$$

احتمال انتخاب یک کارت، که هم قرمز باشد و هم عکس‌دار چقدر است؟ در این‌جا ما احتمالات را در هم ضرب نمی‌کنیم. در واقع ما عملیات زیر را انجام نمی‌دهیم:

$$\mathrm{P}(\text{red} \wedge \text{facecard}) = \mathrm{P}(\text{red}) \times \mathrm{P}(\text{ facecard}) = 0.375 \times 0.185 = 0.0703$$

در این‌جا، حس ششمِ ما به ما می‌گوید که احتمال انتخاب یک کارت قرمز برابر ۰,۱۸۷۵ است. ولی اگر ما یک کارت قرمز را برداریم، به احتمال ۱۰۰ درصد این کارت یک کارت عکس‌دار است. پس به نظر می‌رسد که احتمال انتخاب یک کارت که هم قرمز باشد و هم عکس‌دار، برابر ۰,۱۸۷۵ است.

یا می‌توانیم یک راه دیگر را برویم. احتمال این‌که یک کارت عکس‌دار انتخاب شود، برابر ۰,۳۷۵ است. به صورتی که کارت‌ها، پشت سر هم قرار داده شده‌اند، نصف کارت‌های عکس‌دار، قرمز هستند. پس احتمال انتخاب یک کارت که هم قرمز باشد و هم عکس‌دار برابر ۰,۳۷۵ ضربدر ۰,۵ یعنی برابر ۰,۱۸۷۵ است.

در این‌جا ما نمی‌توانیم احتمالات را در یکدیگر ضرب کنیم به خاطر این‌که ویژگی‌ها مستقل نیستند – اگر یک کارت قرمز را برداریم، احتمال یک کارت عکس‌دار تغییر پیدا می‌کند – و بالعکس.

در بسیاری از موارد (اگر نه همه) در مسائلِ واقعیِ داده‌کاوی، ویژگی‌ها از یکدیگر مستقل نیستند و به هم وابسته هستند.

مجموعه‌ی داده‌ی ورزشی را تصور کنید. در این مجموعه، ما دو ویژگی داشتیم (قد و وزن). قد و وزن از یکدیگر مستقل نبودند. هر چقدر شما قد بلندتر باشید، احتمال این‌که سنگین وزن‌تر هم باشید، بیشتر است.

فرض کنید من ویژگی‌های کدپستی، درآمد و سن را دارم. این ویژگی‌ها مستقل از هم نیستند. یک سری کدپستی‌های خاص هستند که خانه‌های بزرگی دارند و بعضی دیگر که بر اساس پارک‌های آزمایشی ساخته شده‌اند. مثلاً کدپستی منطقه‌ی Palo Alto، ممکن است توسط افراد ۲۰ ساله تصرف شده باشد یا منطقه‌ی Arizona ممکن است خانه‌ی بسیاری از بازنشستگان باشد.

درباره‌ی ویژگی‌های موسیقی نیز کمی فکر کنید – چیزهایی مانند تعداد صداهای گیتار ناهمگون (از ۰ تا ۵) یا تعداد صداهای ویولن کلاسیک. در این‌جا هم تعداد زیادی از ویژگی‌ها، مستقل نیستند. برای مثال اگر من مقدار زیادی صدای گیتار ناهمگون داشته باشم، احتمال وجود صدای ویولن کلاسیک کم و کم‌تر می‌شود.

فرض کنید من مجموعه‌ی داده‌ای از نتایج آزمایش خون داشته باشم. تعداد زیادی از این مقادیر، مستقل از هم نیستند. برای مثال، تعدادی از آزمایش‌های تیروئید مانند T4 آزاد و TSH موجود هستند که ارتباط معکوسی بین مقادیر این دو آزمایش وجود دارد.

خودتان هم در موارد مختلف که با آن برخورد کرده‌اید، تفکر کنید. برای مثال، ویژگی‌های اتومبیل‌ها را در ذهن خود متصور شوید. آیا آن‌ها مستقل هستند؟ ویژگی‌های فیلم‌های مختلف چطور؟ یا ویژگی‌های خریدهای سایت آمازون؟

پس، برای این‌که روش بیز (Bayes) خوب عمل کند، ما نیاز داریم تا از ویژگی‌هایی استفاده کنیم که مستقل از هم باشند، ولی بسیاری از مسائل دنیای واقعی این رفتارِ مستقل را ندارند. کاری که ما می‌کنیم این است که فرض می‌کنیم این ویژگی‌ها مستقل هستند! ما از عصای جادوییِ قایم کردن چیزهای مختلف زیر فرش استفاده می‌کنیم و این مشکل را نادیده می‌گیریم. ما این روش را **بیز ساده (Naïve Bayes)** می‌نامیم زیرا که به صورت

ساده‌ای **فرضِ استقلالِ ویژگی‌ها** را (حتی اگر واقعاً مستقل نباشند) در مسئله، در نظر می‌گیریم. به نظر می‌رسد که روش بیز ساده، بسیار بسیار خوب عمل می‌کند، حتی با این فرض ساده!

کد بزنید

آیا می‌توانید الگوریتم بیز ساده را بر روی مجموعه‌ی داده‌های دیگرمان اجرا کنید؟ برای مثال، الگوریتم KNN بر روی مجموعه‌ی داده‌ی مصرف سوخت اتومبیل‌ها به ازای یک گالن (MPG)، دقت ۵۳ درصدی می‌داد. آیا روش بیز، راه حل بهتری برای این مجموعه‌ی داده است؟

```
tenfold("mpgData/mpgData", "class attr  num  num  num  num  comment")
```

?????

فصل ۷. پردازش متون بدون ساختار

طبقه‌بندی متون بدون ساختار

در فصل‌های قبلی، به سیستم‌های پیشنهاددهنده نگاهی انداختیم که در آن‌ها مردم به صورت صریح چیزهای مختلف را با سیستم‌های ستاره‌ای امتیاز دهی می‌کردند. برای مثال ۵ ستاره برای گروه موسیقیِ Phoenix، شصت بالا به نشانه‌ی موافقت (thumbs-up)/شصت پایین به نشانه‌ی مخالفت (thumbs-down)، و یا مقیاس‌های مختلفِ عددی از نمونه‌های این امتیازدهی بودند. همچنین به چیزهایی که به صورت ضمنی برای ما تولیدِ داده می‌کردند نیز نگاه کردیم. مثالی از این روش، رفتار کاربران است – برای مثال آیا این کاربران یک آیتمِ خاص را خریده‌اند یا آیا بر روی یک لینکِ خاص، کلیک کرده‌اند؟ همچنین به سیستم‌های طبقه‌بندی نگاهی انداختیم. این سیستم‌ها از ویژگی‌هایی مانند قد، وزن و یا حتی این‌که چگونه شخصی به یک لایحه در مجلس، رای داده است، استفاده می‌کردند. در تمامی این نمونه‌ها، اطلاعات در مجموعه‌ی داده به سادگی می‌توانست توسط یک جدول نمایش داده شود.

age	glucose level	blood pressure	diabetes?
26	78	50	1
56	111	72	1
23	81	78	0

mpg	cylinders	HP	sec. 0-60
30	4	68	19.5
45	4	48	21.7
20	8	130	12

این نوع از داده‌ها به **«داده‌های ساختاریافته (structured data)»** معروف هستند – یعنی داده‌ای که در آن‌ها نمونه‌ها یعنی همان سطرها در جدولی مانند جدول بالا، به وسیله‌ی تعدادی ویژگی مشخص می‌شوند .برای مثال، یک سطر در جدول ممکن است به یک خودرو اشاره کند و این کار را به وسیله‌ی مجموعه‌ای از ویژگی‌ها مانند: مسافت به ازای یک گالن (mpg)، تعداد سیلندر (cylinders) و... انجام دهد. داده‌های بدون ساختار یا غیرِساختاریافته، مانند پیام‌های ایمیلی، پیام‌های وب‌سایت توییتر، نوشته‌های وبلاگ‌ها یا مقالات روزنامه‌ها هستند. این نوع چیزها را (حداقل در یک نگاه اجمالی) نمی‌توان به صورت تمیز و مرتب در یک جدول جای داد.

برای مثال، فرض کنید ما علاقه‌مند هستیم که مشخص کنیم آیا یک سری فیلم، خوب هستند یا خیر و می‌خواهیم پیام‌های توییتر در مورد این فیلم‌ها را تحلیل کنیم:

ما انگلیسی زبان‌ها، می‌توانیم از توییت‌های بالا ببینیم که مثلاً Andy Gavin از فیلم Gravity خوشش آمده است. چون او در یک توییت گفته است که «آدم را به هیجان وا می‌دارد (puts the thrill back in thriller)» و یا گفته است که «بازی خوبی داشت (good acting)». ما می‌دانیم که Debra Murphy به نظر خیلی از فیلم خوشش نیامده است چون او در توییتی نوشته «پولاتون رو ذخیره کنید ($$$ save your)» و اگر کسی نوشته باشد «خیلی خیلی بد می‌خواهم فیلم Gravity را ببینم، همه باید این فیلم را ببینیم (I wanna see Gravity sooo bad, we should all go see it!!!)» این شخص احتمالاً فیلم را دوست داشته است اگر چه که از کلمه‌ی «بد (bad)» استفاده کرده است.

فرض کنید من در مغازه‌ی اغذیه‌فروشی باشم و چیزی به اسم ماست یونانی چبانی (Chobani) را ببینم. به نظر جذاب می‌آید ولی آیا واقعاً خوب است؟ موبایلم را برمی‌دارم و در گوگل جستجو می‌کنم و چیزی شبیه به این را در یک وبلاگ با عنوان «زن، تنها با نان زندگی نمی‌کند (Woman Does Not Live on Bread Alone)» خواندم که محتوایش چیزی مانند زیر بود:

آیا تا به حال ماست یونانی خورده‌اید؟ اگر نه، کلیدهایتان را بردارید (و البته اگر در نیویورک زندگی می‌کنید، یک کت هم بردارید) و به بقالیِ محل خود بروید. حتی وقتی این ماست‌ها بدون‌چربی و ساده باشند، باز هم خیلی ضخیم و خامه‌ای بوده، و من بعد از خوردن آن‌ها

احساس گناه می‌کنم. هدف اصلی ماست هم همین است. طعم ساده‌ی آن ترش و عالی است. آن‌هایی که می‌توانند این ماست را تهیه کنند، نوع عسلیِ آن را بخرند. شکری در کار نیست، ولی مقداری عسل برای ایجاد طعم شیرین به آن اضافه شده است (همچنین می‌توانید خودتان به آن عسلِ محلی اضافه کنید – عسل محلی برای آلرژی‌ها خوب است!). اگر چه که من نباید از لحاظ فنی عسل مصرف کنم، ولی اگر یک روز بد را پشت سر گذاشته، و نامیدانه فقط به شیرینی احتیاج داشته باشم، یک قاشق چای‌خوری عسل به ماستم اضافه می‌کنم و کاملاً ارزشش را دارد. طعم‌های میوه‌ای از شرکت چوبانی (Chobani)، تمامشان در خود مقداری شکر دارند، این در حالی است که میوه در این ماست‌ها کاملاً غیرضروری هستند. اگر بقالی شما، مارکِ چوبانی (Chobani) را ندارد، مارک Fage، نیر شناخته شده است و به همان مقدار خوشمزه است.

حالا برای ماست یونانی، شما مقدار ۵۰ سنت تا ۱ دلار بیشتر پرداخت می‌کنید و مقدار حدوداً ۲۰ کالری در هر بار مصرف بیشتر وارد بدنتان می‌شود. ولی ارزشش را دارد، حداقل برای من، تا حس ناامیدی و ناراحتی بعد از یک اسنک عصرانه را نداشته باشم.

http://womandoesnotliveonbreadalone.blogspot.com/2009/03/sugar-free-yogurt-reviews.html

آیا این متن، برای شرکت چوبانی (Chobani) یک متن مثبت بود یا منفی؟ فقط بر اساس جمله‌ی دوم که گفته بود: «اگر نه، خواندن را متوقف کنید و کلیدهایتان را جمع کنید... و خودتان را به بقالی محلتان برسانید»، به نظر می‌رسد که این متن یک متن مثبت است. این متن توضیح می‌دهد که طعم و مزه بسیار خوشمزه و عالی است و در واقع این ماست را یک ماست خوشمزه توصیف کرده است. به نظر می‌رسد که باید این ماست را خریده و امتحانش کنم. الان برمی‌گردم...

یک سیستم خودکار برای تشخیص مثبت یا منفی بودن متون

آقای John، به نظر یک توییت مثبت در مورد فیلم Gravity آمده است!

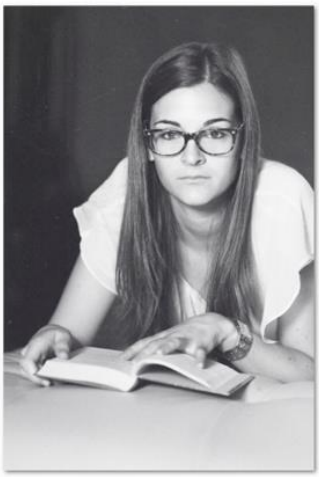
فرض کنید یک سیستم خودکار که می‌تواند متنی را خوانده و تصمیم بگیرد که آیا این متن که در مورد یک محصولِ خاص نوشته شده، نظری مثبت است یا منفی. حال بیایید ببینیم اصلاً چرا به یک همچین سیستمی نیاز داریم؟ فرض کنید یک شرکت، فروشنده‌ی محصولی است که بر سلامت نظارت دارد و احتمالاً می‌خواهد بداند که مردم در مورد محصولاتش چه نظری دارند؟ آیا چیزی که مردم می‌گویند، غالباً مثبت است یا منفی؟ این شرکت یک کمپین تبلیغاتی برای یک محصول جدید راه‌اندازی کرده است. آیا مردم از محصول راضی هستند (مثلاً جملاتی مثل: «من حتماً به یک همچین چیزی احتیاج دارم!» را می‌گویند؟) و یا ناراضی هستند (مثلاً جملاتی مثل: «به نظر مزخرف میاد» را می‌گویند). یا مثلاً شرکت اپل یک کنفراس دارد که در مورد مشکلات آی‌فون صبحت می‌کند. آیا جمع‌بندی مثبت بوده است؟ یا برای مثالی دیگر یکی از نامزدهای مجلس سنای آمریکا، یک سخنرانی سیاسی برگزار کرده است و می‌خواهیم بدانیم آیا بلاگرهای سیاسی، این سخنرانی را مثبت ارزیابی کردند یا خیر؟ پس با این اوصاف یک سیستم خودکار به نظر مفید و کارا می‌رسد.

فرض کنید من بخواهم یک سیستم توسعه دهم و این سیستم بتواند تشخیص دهد که آیا یک شخص، محصول غذایی را دوست داشته است یا خیر. ممکن است با این ایده شروع کنیم که: بیاییم یک مجموعه‌ای از کلمات که بیان‌گر دوست داشتنِ یک محصول توسط یک کاربر هستند را جمع‌آوری کنیم و یک مجموعه‌ی دیگر از کلمات که بیان‌گر دوست نداشتن یک محصول توسط کاربر هستند را نیز جمع‌آوری کنیم.

کلماتی که مربوط به «دوست داشتن – like» یک محصول هستند:
خوشمزه (delicious)
خوش طعم (tasty)
خوب (good)
عشق (love)
ملایم (smooth)

کلماتی که مربوط به «دوست نداشتن – dislike» یک محصول هستند:
افتضاح (awful)
پرهیز (bad)
بد (bad)
متنفر (hate)
ریگ‌دار (gritty)

اگر بخواهیم تلاش کنیم تا ببینیم آیا یک شخص برای مثال ماست Chobani را دوست دارد یا خیر، می‌توانیم به صورت ساده، فقط کلماتی که در مجموعه‌ی کلمات مثبت هستند را در متنی که کاربر گذاشته، شمارش کنیم و بعد از آن کلماتی که در مجموعه کلمات منفی هم هستند را نیز بشماریم. بر اساس این که کدام مجموعه کلمات در این متنِ کاربر، بیشتر استفاده شده باشد، این متن به مثبت یا منفی طبقه‌بندی می‌شود. این کار را همچنین می‌توانیم برای عملیات طبقه‌بندیِ دیگر نیز انجام دهیم. برای مثال، آیا یک گروه pro-choice (موافق حق سقط جنین در آمریکا) است یا pro-life (مخالف حق سقط جنین)؟ این طبقه‌بندی را نیز می‌توانیم بر اساس کلماتی که این گروه استفاده می‌کند، بفهمیم. اگر آن‌ها از کلماتی مانند «کودک نابالغ (unborn child)» استفاده کنند، این بدان معنا است که این گروه pro-life است. ولی اگر از کلمه‌ی «جنین - fetus» استفاده کنند، با احتمال بالاتری، pro-choice هستند. خیلی عجیب نیست که بتوانیم از وجودِ کلمات در دسته‌بندی‌های مختلف، برای طبقه‌بندیِ متن استفاده کنیم.



به جای این‌که فقط از تعداد خامِ کلمات برای طبقه‌بندی متن استفاده کنیم، بیایید از «بیز ساده» کمک بگیریم

$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

بیایید فرمول را واکاوی کنیم

می‌خواهم تمامیِ فرضیه‌ها را بررسی کرده و فرضیه‌ای با بیشترین احتمال را انتخاب کنم

احتمال آن فرضیه‌ها

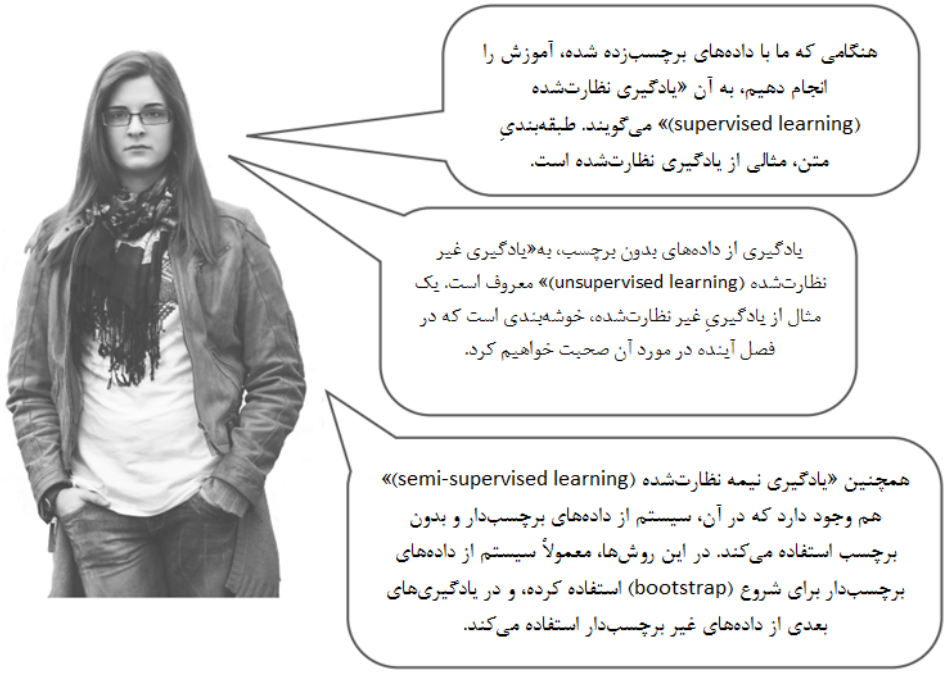
$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

برای هر کدام از فرضیه h، در میان تمامی فرضیه‌ها H...

احتمال این‌که یک داده‌ی خاص D را با داشتنِ فرضیه‌ی h مشاهده کنیم (برای مثال احتمال مشاهده‌ی یک کلمه‌ی خاص در یک متن، با فرض این‌که آن متن را داشته باشیم)

ما از روش بیز ساده (Naïve Bayes) که در فصل قبل در مورد آن صحبت کردیم، استفاده می‌کنیم. با استفاده از مجموعه‌ی داده‌ی آموزشی شروع کرده و از آن‌جایی که حالا می‌خواهیم از متون بدون ساختار استفاده کنیم، به این مجموعه، **مجموعه‌ی اسناد آموزشی (corpus training)** گفته می‌شود. هر کدام از نمونه‌های وارد شده، یک **سند (document)** نامیده می‌شود، حتی اگر کلاً ۱۴۰ کاراکتر (مانند یک پست توییتر) شود. هر کدام از سندها، با طبقه‌ی مربوط به خودش برچسب‌زنی (label) می‌شوند. پس برای مثال، ما ممکن است مجموعه‌ای از پست‌های توییتر داشته باشیم، که یک سری فیلم‌ها را امتیازدهی کرده باشند. هر کدام از پست‌ها، به نوعی برچسب‌زده می‌شوند. برای برچسب‌ها می‌توانیم از گزینه‌های «مطلوب (favorable)» یا «نامطلوب (unfavorable)» استفاده کنیم. و حالا می‌خواهیم الگوریتم طبقه‌بندیمان را بر اساس این مجموعه‌ی داده‌ی متنی و برچسب‌های آن‌ها، آموزش دهیم. در این حالت، P(h)، در فرمول بالا، احتمال این برچسب‌ها است. اگر ۱۰۰۰ سندِ متنی در مجموعه‌ی آموزشی داشته باشیم و ۵۰۰ سند از آن‌ها، برچسب «مطلوب» و ۵۰۰ سند، برچسب «نامطلوب» داشته باشند، پس:

$$\mathrm{P}(\text{favorable}) = 0.5, \mathrm{P}(\text{unfavorable}) = 0.5$$



اوکی، برگردیم به:

$$h_{MAP} = \text{argmax}_{h \in H} P(D \mid h) P(h)$$

حالا بیایید قسمت P(D|h) را هم مرور کنیم – احتمال مشاهده‌ی یک رخداد خاص داده‌ای مانند D با این شرط که فرضیه‌ی h را داشته باشیم. در این‌جا داده‌ی D که می‌خواهیم از آن استفاده کنیم، کلمات در متن هستند. یک روش، این‌است که اولین جمله از سند را بررسی کنیم. برای مثال، جمله‌ی «Puts the Thrill back in Thriller» را در نظر بگیرید. بیایید ببینیم که احتمال این‌که یک جمله‌ای مثبت بوده و با کلمه‌ی «puts» شروع شده باشد، چقدر است؟ حالا احتمال این‌که یک جمله‌ی مثبت، دارای کلمه‌ی دومِ «the» باشد (مثل همین جمله که کلمه‌ی دومش «the» است) چقدر است؟ و همچنین احتمال این‌که یک جمله‌ی مثبت، دارای سومین کلمه‌ی «Thrill» باشد چقدر است و همین‌طور الی آخر برای تمامی کلماتِ موجود در این جمله. و همچنین احتمال هر کدام از کلمات را برای

جملات منفی نیز محاسبه می‌کنیم تا ببینیم احتمال مثبت بودن آن جمله بیشتر است یا منفی بودن آن جمله.

گوگل تخمین زده است که تعداد حدود ۱ میلیون کلمه در زبان انگلیسی وجود دارد. اگر پیغام‌های توییتر تقریباً ۱۴ کلمه داشته باشند، بایستی احتمالات زیر را محاسبه کنیم:

1,000,000 x 1,000,000 x 1,000,000 x , 1,000,000 x 1,000,000 x 1,000,000 x ,1,000,000 x 1,000,000 x 1,000,000 x 1,000,000 x 1,000,000 x 1,000,000 x 1,000,000 x 1,000,000

مشاهده می‌کنید که تعداد خیلی زیادی از احتمالات را بایستی محاسبه کرد! باید یک روش بهتری هم وجود داشته باشد!

خب. تعداد بسیار زیادی از احتمالات باید محاسبه شوند و همین موضوع باعث می‌شود که این روش غیرکارا شود. و خوشبختانه، یک راه بهتر وجود دارد. می‌خواهیم مقداری چیزهای مختلف را ساده کنیم و این کار را با استفاده از کلمات به صورت غیر ترتیبی (یعنی به صورت کوله‌ای) انجام می‌دهیم. یعنی به جای این‌که مثلاً بپرسیم «**احتمال این‌که سومین کلمه در جمله‌ی مثبت، «thrill» باشد، چقدر است**؟»، می‌پرسیم «**احتمال این‌که کلمه‌ی «thrill» در یک جمله‌ی مثبت باشد، چقدر است**؟» در ادامه، این احتمالات را با هم محاسبه خواهیم کرد.

فاز آموزش

ابتدا، کلمات را مشخص می‌کنیم – کلمات یکتا را. از تمامی اسناد موجود (یعنی از تمامی اسناد که مثبت یا منفی باشند کلمات را استخراج می‌کنیم). پس برای مثال، اگر کلمه‌ی «the» هزار بار در تمامی اسناد تکرار شده باشد، فقط یکبار در لیست کلمات آن را می‌آوریم. خب! حالا فرض کنید:

|Vocabulary|

نشان‌گرِ تعدادِ کلمات باشد. بعد از آن، برای هر کدام از کلمات که با w_k مشخص می‌شوند، در مجموعه‌ی کلمات، می‌خواهیم احتمال وقوع یک کلمه را با این شرط که بدانیم یک فرض خاص اتفاق می‌افتد، محاسبه کنیم. یعنی $P(w_k|h_i)$
می‌خواهیم این فرمول را به صورت زیر محاسبه کنیم. برای هر فرضیه‌ای (در این مورد، «مثبت» یا «منفی»):

۱. اسنادی که با آن فرضیه‌ی خاص برچسب خورده‌اند در یک فایل ترکیب می‌کنیم
۲. تعداد تکرار کلمات موجود در این فایل را می‌شماریم. این بار، مثلاً اگر کلمه‌ی «the»، ۵۰۰ مرتبه تکرار شده باشد، این تعداد را برابر ۵۰۰ قرار می‌دهیم. اجازه دهید این تعداد را n بنامیم.
۳. برای هر کلمه، در مجموعه‌ی کلمات که با w_k نشان می‌دادیم، تعداد تکرار آن کلمه را در متن می‌شماریم. آن را با n_k نمایش می‌دهیم.
۴. برای هر کلمه در مجموعه، یعنی همان w_k، با فرمول زیر محاسبه می‌کنیم:

$$P(w_k|h_i) = \frac{n_k + 1}{n + Vocabulary}$$

فاز طبقه‌بندی بیز ساده (Naïve Bayes)

هنگامی که فاز آموزش را انجام دادیم، می‌توانیم سندها را طبقه‌بندی کنیم. این کار را با استفاده از فرمولی که معرفی کردیم، انجام می‌دهیم:

$$h_{MAP} = \operatorname{argmax}_{h \in H} P(D \mid h)P(h)$$

فرض کنید داده‌های مجموعه‌ی آموزشیِ ما شامل ۵۰۰ توییت با محتوای مثبت و ۵۰۰ توییت با محتوای منفی در مورد یک محصول باشند. پس:

$$P(\text{like}) = 0.5, P(\text{dislike}) = 0.5$$

بعد از آموزش، احتمالات به صورت زیر در آمده است (ستون اول یعنی word نشان‌دهنده‌ی کلمه‌ی مورد نظر ماست):

word	P(word|like)	P(word|dislike)
am	0.007	0.009
by	0.012	0.012
good	0.002	0.0005
gravity	0.00001	0.00001
great	0.003	0.0007
hype	0.0007	0.002
I	0.01	0.01
over	0.005	0.0047
stunned	0.0009	0.002
the	0.047	0.0465

حالا فرض کنید می‌خواهیم توییتِ زیر را طبقه‌بندی کنیم (که آیا مثبت است یا منفی):

I am stunned by the hype over gravity

حالا ما می‌خواهیم احتمال زیر را حساب کنیم:

$$P(like) \times P(I \mid like) \times P(am \mid like) \times P(stunned \mid like) \times \ldots$$

و همچنین احتمال زیر را:

$$P(dislike) \times P(I \mid dislike) \times P(am \mid dislike) \times P(stunned \mid dislike) \times \ldots$$

و در نهایت فرضیه‌ای را قبول کنیم که دارای بیشترین احتمال است (مثبت = like و منفی = dislike)

word	P(word|like)	P(word|dislike)
	P(like) = 0.5	P(dislike) =0.05
I	0.01	0.01
am	0.007	0.009
stunned	0.0009	0.002
by	0.012	0.012
the	0.047	0.0465
hype	0.0007	0.002
over	0.005	0.0047
gravity	0.00001	0.00001
∏	6.22E-22	4.72E-21

(ستون آخر، همان ضرب احتمالات است)

پس احتمالات به صورت زیر محاسبه می‌شود:

like= 0.000000000000000000000622
dislike= 0.000000000000000000004720

یک یادآوری کوچک

عبارت e در ریاضیات، نشان می‌داد که چقدر باید بعد از اعشار جلو برویم. اگر عدد مثبت بود، یعنی باید اعشار را به سمت راست برد و اگر عدد منفی بود، یعنی بایستی اعشار را به سمت چپ هدایت کرد. پس:

$$1.23e-1 = 0.123$$
$$1.23e-2 = 0.0123$$
$$1.23e-3 = 0.00123$$

در بالا، احتمال منفی بودن (dislike) بیشتر از مثبت بودن (like) است. پس در این‌جا، این توییت را به دسته‌ی منفی‌ها طبقه‌بندی می‌کنیم. به این معنی که این توییت در نقد و دوست‌نداشتنِ یک محصول خاص بوده است.

همان‌طور که متوجه شدید مشکلی در اعداد اعشار کوچک وجود دارد و راه‌حل آن به این صورت است. فرض کنید ما ۱۰۰ کلمه در سند داریم و به طور میانگین احتمال هر کلمه ۰,۰۰۱ است (کلماتی مانند tell، reported، average، morninig، و and حدوداً احتمال

۰٫۰۰۱ دارند). اگر این احتمالات را در زبان پایتون با هم ضرب کنیم، نتیجه عدد صفر خواهد شد.

```
>>> 0.0001**100
0.0
```

ولی اگر لگاریتمِ احتمالات با هم جمع کنیم، به یک عددی غیر از صفر می‌رسیم:

```
>>> import math
>>> p = 0
>>> for i in range(100):
      p += math.log(0.0001)

>>> p
-921.034037197617
```

ممکن است فراموش کرده باشید... $b^n = x$

لگاریتم (یا همان log) برای یک عدد (مثلاً عدد x در بالا) همان توان (عدد n در بالا) است که می‌تواند عدد پایه (b) را به عدد x برساند. برای مثال، فرض کنید عدد پایه (b) برابر ۱۰ باشد:

$$\log_{10}(1000) = 3$$

چون ۱۰ به توان ۳ برابر ۱۰۰۰ می‌شود.

در پایتون عدد پایه برای تابع لگاریتم، عدد e است. فعلاً به عدد e کاری نداریم. چیزی که الان برای ما جذاب است دو مورد زیر است:

۱. لگاریتم‌ها، مقیاس اعداد را فشرده می‌کنند (یعنی با استفاده لگاریتم می‌توانیم اعدادِ کوچکتر را در پایتون نمایش دهیم)

برای مثال:

$$.0000001 \times .000005 = .000000000005$$

لگاریتمِ اعداد بالا به صورت زیر می‌شود:

$$-16.11809 + -9.90348 = -26.02157$$

۲. به جای این‌که احتمالات را با یکدیگر ضرب کنیم، لگاریتمِ احتمالات را با هم جمع می‌کنیم.

مجموعه‌ی داده‌ی متون خبری (Newsgroup Corpus)

ابتدا نحوه‌ی کارکرد این الگوریتم را بررسی می‌کنیم. این کار را با استفاده از یک مجموعه‌ی داده‌ی مرجع که از نوشته‌های خبریِ usenet استخراج شده است، انجام می‌دهیم. داده‌ها، شامل مجموعه‌ای از نوشته‌ها شامل ۲۰ گروه خبری (دسته‌ها یا همان طبقه‌ها) هستند: (برای مثال گروه «کامپیوتر.گرافیک» که با comp.graphics نشان داده می‌شود)

comp.graphics - misc.forsale - soc.religion.christian - alt.atheism - comp.os.ms-windows-misc - rec.autos - talk.politics.guns - sci.space - comp.sys.ibm.pc.hardware - rec.motorcycles - talk.politics.mideast - sci.crypt - comp.sys.mac.hardware - rec.sport.baseball - talk.politics.misc - sci.electronics - comp.windows.x - rec.sport.hockey - talk.religion.misc - sci.med

ما به دنبال این هستیم که یک نوع طبقه‌بند را توسعه دهیم که مشخص کند هر کدام از نوشته‌ها در کدام گروه خبری (از گروه‌های بالا) اختصاص دارد. برای مثال می‌خواهیم نوشته‌ی زیر را طبقه‌بندی کنیم:

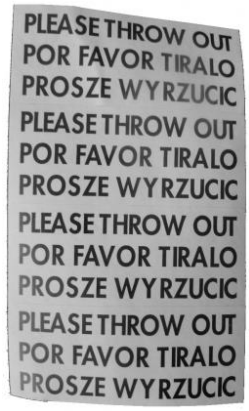
From: essbaum@rchland.vnet.ibm.com
(Alexander Essbaum)
Subject: Re: Mail order response time
Disclaimer: This posting represents the poster's views, not necessarily those of IBM Nntp-Posting-Host: relva.rchland.ibm.com

Organization: IBM Rochester
Lines: 18
> I have ordered many times from Competition
> accesories and ussually get 2-3 day delivery.
ordered 2 fork seals and 2 guide bushings from CA for my FZR. two weeks later get 2 fork seals and 1 guide bushing. call CA and ask for remaining *guide* bushing and order 2 *slide* bushings (explain on the phone which bushings are which; the guy seemed to understand). two weeks later get 2 guide bushings.
sigh
how much you wanna bet that once i get ALL the parts and take the fork apart that some parts won't fit?

که در واقع متن بالا به گروه **rec.motorcycles** تعلق دارد.

توجه کنید که در پُستِ بالا، بعضی از کلمات دارای اشتباهات املایی هستند (برای مثال کلمات accesories یا ussually) که این خود می‌تواند چالش‌هایی را برای الگوریتمِ طبقه‌بندی ایجاد کند.

این مجموعه‌ی داده در http://qwone.com/~jason/20Newsgroups/ قابل دانلود است (ما از مجموعه‌ی داده‌ی 20news=bydate استفاده کردیم). همچنین این داده‌ها در وب‌سایت کتاب به نشانی guidetodatamining.com و chistio.ir قابل مشاهده و دانلود است. این مجموعه‌ی داده شامل ۱۸۸۴۶ سند است و به دو قسمت آموزشی (که ۶۰ درصد داده‌ها را تشکیل می‌دهد) و آزمون (شامل ۴۰ درصد داده‌ها) تقسیم شده است. قسمت‌های آموزشی و آزمون در دایرکتوری‌های جداگانه‌ای قرار دارند. در هر کدام از دایرکتوری‌ها یک سری زیردایرکتوری‌هایی قرار دارند که هر کدام یک گروه خبری را پوشش می‌دهند. در هر کدام از این زیردایرکتوری‌ها، اسناد مختلف موجودند که در واقع نوشته‌های آن گروه خبری هستند.



همه چیزی را به بیرون پرت کنید!

قبل از این‌که شروع به کد زدن بکنیم، کمی بیشتر در مورد این مثال فکر کنید.

(خانم‌ها و آقایان. در صحنه‌ی اصلی ... فقط بر اساس کلماتِ داخل متن، تلاشمان را می‌کنیم که بگوییم هر کدام از نوشته‌ها مربوط به کدام دسته‌ی خبری می‌شوند)

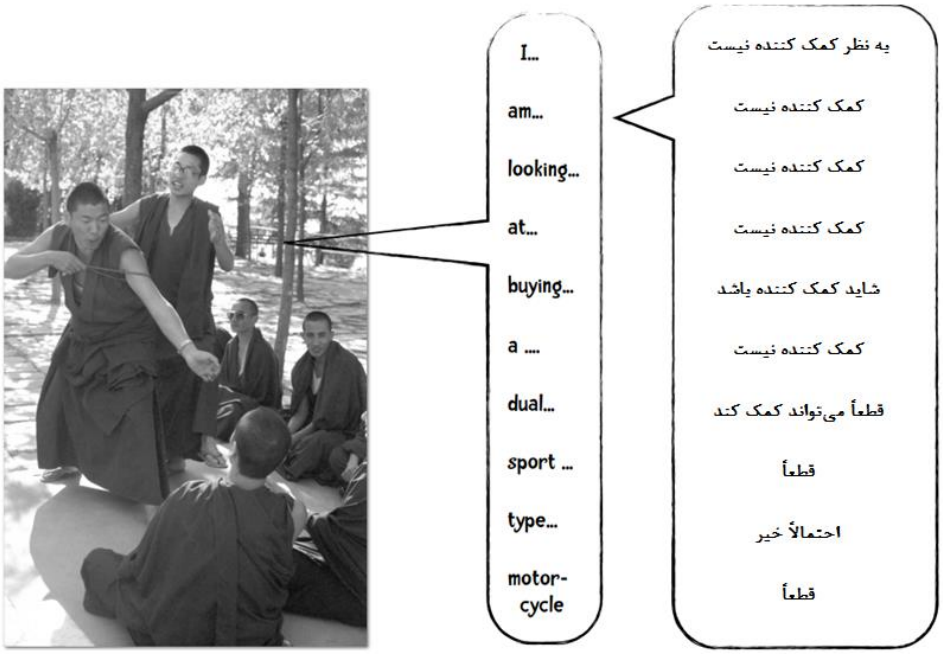
برای مثال، ما می‌خواهیم سیستمی داشته باشیم که بتواند پستی مانند پست زیر را به دسته‌ی rec.motocycle برچسب‌زنی یا همان طبقه‌بندی کند.

I am looking at buying a Dual Sport type motorcycle. This is my first cycle as well. I am interested in any experiences people have with the following motorcycles, good or bad.

Honda XR250L
Suzuki DR350S
Suzuki DR250ES
Yamaha XT350

Most XXX vs. YYY articles I have seen in magazines pit the Honda XR650L against another cycle, and the 650 always comes out shining. Is it safe to assume that the 250 would be of equal quality?

بیایید ببینیم که کدام کلمات ممکن است برای طبقه‌بندی کمک کند:



اگر ما ۲۰۰ کلمه‌ی پرتکرار انگلیسی را از میان متن حذف کنیم، سند ما به صورت زیر می‌شود:

I am looking at buying a Dual Sport type motorcycle. This is my first cycle as well. I am interested in any experiences people have with the following motorcycles, good or bad.

Honda XR250L
Suzuki DR350S
Suzuki DR250ES
Yamaha XT350

Most XXX vs. YYY articles I have seen in magazines pit the Honda XR650L against another cycle, and the 650 always comes out shining. Is it safe to assume that the 250 would be of equal quality ?

حذف این کلمات، باعث کاهش ۵۰ درصدی حجم متن می‌شود. به علاوه، به نظر می‌رسد که حذفِ این کلمات، تاثیری بر عملکردِ طبقه‌بندِ ما ندارد. در واقع، داده‌کاوها این کلمات را،

کلماتِ بدون محتوا یا **کلمات پرت (fluff words)** می‌نامند. H. P. Luhn در مقاله‌ی اصلیِ خود با نام «ساخت خودکار انتزاع‌های زبانی (The automatic creation of literature abstracts)» گفته است که این کلمات اینقدر معمول هستند که تاثیرِ مطلوبی نداشته باشند و باعث تشکیل داده‌های نویزدار در سیستم می‌شوند. در واقع حذف این کلمات، باعث افزایش بهینگی و کارایی می‌شود. کلماتی که حذف می‌کنیم به **«کلمات توقف (stop words)»** نیز شناخته می‌شوند. ما لیستی از این‌ها داریم، که به آن‌ها «لیست کلمات توقف (stop words list)» می‌گوییم و این کلمات را از میان متون در مرحله‌ی پیش‌پردازش (preprocess) حذف می‌کنیم. در واقع ما این کلمات را حذف می‌کنیم چون اولاً این‌کار باعث کاهش مقدار پردازش ما می‌شود و ثانیاً این‌کار تاثیر بدی بر روی کارایی سیستم نمی‌گذارد – همان‌طور که در بالا اشاره شد، حذف آن‌ها حتی می‌تواند باعث افزایش کارایی نیز شود.

کلمات معمول (Common Words) در مقابل کلمات توقف (Stop Words)
اگر چه این جمله که کلماتی مانند the یا a به فرآیند طبقه‌بندی کمکی نمی‌کنند، کلمات دیگری مانند work یا write یا school ممکن است کمک کننده باشند که البته بستگی به الگوریتم طبقه‌بندی دارد. هنگامی که یک مجموعه از کلمات توقف (stop words) می‌سازیم، معمولاً کلمات معمول (common words) که بتوانند کمک کننده باشند را نادیده می‌گیریم (یعنی در لیست کلمات توقف نمی‌آوریم). می‌توانید برای درک کردن تفاوت بین این دو مجموعه‌ی داده، در اینترنت جستجو کنید و مجموعه کلمات توقف را با کلمات پرتکرار و معمول مقایسه کنید (stop words list یا frequent word list)

خطرات حذف کلمات توقف

هِی تو جوانِ خام. نباید این کلمات معمول را دور بریزی!

در حالی که حذف کلمات توقف، در بعضی از موارد می‌تواند کارا باشد، بهتر است آن‌ها را بدون فکر حذف نکنید. برای مثال، به نظر می‌رسد که فقط با استفاده از کلماتِ پرتکرار (و بدون استفاده از بقیه‌ی کلمات) که در واقع برعکسِ کاری است که در بالاتر انجام دادیم، می‌توانیم اطلاعات کافی را برای به دست آوردنِ اسنادی که با زبان عربی نوشته شده‌اند، به دست بیاوریم. اگر به این موضوع علاقه دارید، مقاله‌ی زیر را جستجو کنید (این مقاله‌ای است که من با یکی از همکلاسی‌هایم در دانشگاه ایالتی نیومکزیو نوشته‌ام):

Linguistic Dumpster Diving: Geographical Classification of Arabic Text

همچنین این مقاله در سایت http://zacharski.org در دسترس است.

یا مثلاً در مورد چت‌های آنلاین، شکارچی‌های جنسی (یعنی کسانی که می‌خواهند مردم را فریب دهند) از کلماتی مانند I، me و you بسیار بیشتر از سایر افراد استفاده می‌کنند. اگر هدف ما در یک پروژه، مشخص کردنِ شکارچیان جنسی در چت‌ها باشد، حذف کلمات پرتکرار می‌تواند به کاراییِ الگوریتم طبقه‌بندی، صدمه وارد کند.

پس، کورکورانه کلمات توقف را حذف نکنید. ابتدا فکر کنید.

کد بزنید – استایل پایتونی

اجازه بدهید ابتدا قسمت آموزش بیز ساده (Naïve Bayes) را کد بزنیم. یادتان باشد که داده‌های آموزشی برای مجموعه‌ی متن‌های newsgroup به صورت زیر بود:

```
20news-bydate-train
      alt.atheism
            text file 1 for alt.atheism
            text file 2
            …
            text file n
      comp.graphics
            text file 1 for comp.graphics
            ...
```

پس من یک دایرکتوری یا همان فولدر دارم (که در این مثال به اسم 20news-bydate-train است). در این دایرکتوری یک سری زیردایرکتوری وجود دارد که طبقه‌های مختلف را نمایش می‌دهند (برای مثال alt.athesim یا comp.graphics و...). اسم این زیر

دایرکتوری‌ها همان اسم طبقه‌ها هستند. داده‌های آزمون نیز، به همین شکل سازماندهی شده‌اند. پس برای تطبیق این ساختار، کد پایتون نیاز به دانستنِ دایرکتوریِ آموزشی دارد (برای مثال، /Users/raz/Downloads/20news-bydate/20news-bydate-train/).

توضیحِ کلیِ کد، برای قسمت آموزشی به صورت زیر است:

کلاس BayesText

۱. تابع init:

اول: کلمات را از مجموعه کلمات توقف (stop words) می‌خواند

دوم: دایرکتوریِ آموزشی را برای به دست آوردن نام زیردایرکتوری‌ها می‌خواند (علاوه بر این‌که این‌ها نام زیردایرکتوری‌ها هستند، نام گروه‌ها یا همان طبقه‌ها نیز هستند)

سوم: برای هر کدام از آن زیردایرکتوری‌ها که همان طبقه‌های ما هستند، تابع train را صدا می‌زند، که در واقع تعداد تکرارِ هر کدام از کلمات را در تمامی فایل‌های آن زیردایرکتوری محاسبه می‌کند

چهارم: احتمالات را توسط فرمول زیر محاسبه می‌کند:

$$P(w_k \mid h_i) = \frac{n_k + 1}{n + |\text{Vocabulary}|}$$

```
from __future__ import print_function
import os, codecs, math

class BayesText:

    def __init__(self, trainingdir, stopwordlist):
        """This class implements a naive Bayes approach
to text
        classification
        trainingdir is the training data. Each subdirect
ory of
        trainingdir is titled with the name of the class
ification
        category -- those subdirectories in turn contain
 the text
        files for that category.
```

```
        The stopwordlist is a list of words (one per lin
e) will be
        removed before any counting takes place.
        """
        self.vocabulary = {}
        self.prob = {}
        self.totals = {}
        self.stopwords = {}
        f = open(stopwordlist)
        for line in f:
            self.stopwords[line.strip()] = 1
        f.close()
        categories = os.listdir(trainingdir)
        #filter out files that are not directories
        self.categories = [filename for filename in cate
gories
                           if os.path.isdir(trainingdir
+ filename)]
        print("Counting ...")
        for category in self.categories:
            print('    ' + category)
            (self.prob[category],
             self.totals[category]) = self.train(trainin
gdir, category)
        # I am going to eliminate any word in the vocabu
lary
        # that doesn't occur at least 3 times
        toDelete = []
        for word in self.vocabulary:
            if self.vocabulary[word] < 3:
                # mark word for deletion
                # can't delete now because you can't del
ete
                # from a list you are currently iteratin
g over
                toDelete.append(word)
        # now delete
        for word in toDelete:
            del self.vocabulary[word]
        # now compute probabilities
        vocabLength = len(self.vocabulary)
        print("Computing probabilities:")
        for category in self.categories:
            print('    ' + category)
            denominator = self.totals[category] + vocabL
ength
```

```
            for word in self.vocabulary:
                if word in self.prob[category]:
                    count = self.prob[category][word]
                else:
                    count = 1
                self.prob[category][word] = (float(count
 + 1)
                                             / denominat
or)
        print ("DONE TRAINING\n\n")


    def train(self, trainingdir, category):
        """counts word occurrences for a particular cate
gory"""
        currentdir = trainingdir + category
        files = os.listdir(currentdir)
        counts = {}
        total = 0
        for file in files:
            #print(currentdir + '/' + file)
            f = codecs.open(currentdir + '/' + file, 'r'
, 'iso8859-1')
            for line in f:
                tokens = line.split()
                for token in tokens:
                    # get rid of punctuation and lowerca
se token
                    token = token.strip('\'".,?:-')
                    token = token.lower()
                    if token != '' and not token in self
.stopwords:
                        self.vocabulary.setdefault(token
, 0)
                        self.vocabulary[token] += 1
                        counts.setdefault(token, 0)
                        counts[token] += 1
                        total += 1
            f.close()
        return(counts, total)
```

نتایج فاز آموزش در یک دیکشنری (جدول درهم‌ساز - hash table) به اسم prob ذخیره می‌شود. کلیدها برای این دیکشنری انواع طبقه‌بندی‌هاست (مثلاً comp.graphics،

rec.motocycles، soc.religion.christian و...). مقادیر هم همان دیکشنری‌ها هستند. کلیدهای این زیردیکشنری‌ها کلمات بوده و مقادیرِ آن‌ها، احتمالات آن کلمات است (در واقع به صورت کلید/مقدار یا همان key/value پیاده‌سازی شده است). در زیر یک مثال از این مورد را آورده‌ایم:

```
bT = BayesText(trainingDir, stoplistfile)
>>>bT.prob["rec.motorcycles"]["god"]
0.00013035445075435553
>>>bT.prob["soc.religion.christian"]["god"]
0.004258192391884386
>>>bT.prob["rec.motorcycles"]["the"]
0.028422937849264914
>>>bT.prob["soc.religion.christian"]["the"]
0.039953678998362795
```

پس برای مثال، احتمال وجود کلمه‌ی god (به معنی خدا) در گروه rec.motorcycles (که مربوط به موتورسیکلت است)، برابر ۰,۰۰۰۱۳ است. (یعنی در این گروه تقریباً در هر ۱۰۰۰۰ کلمه، یک کلمه‌ی god داریم). ولی احتمال وجود کلمه‌ی god در گروه soc.religion.christian (که مربوط به مذهب مسحیت است) برابر ۰,۰۰۴۲۴ است (یعنی در این گروه تقریباً در هر ۲۵۰ کلمه، یک کلمه‌ی god داریم).

فاز آموزش همچنین طبقه ها را هم برمی‌گرداند، که در آن، همان طور که احتمالاً حدس می‌زنید، لیستی از تمامی گروه‌ها موجود است:

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', ...]
```

این فاز آموزش بود. بیایید حالا یک سند را طبقه‌بندی کنیم

کد بزنید

آیا می‌تواند تابع طبقه‌بند را به گونه‌ای بنویسید که بتواند طبقه‌ی یک سند را پیش‌بینی کند؟ برای مثال:

```
>>> bT.classify("20news-bydate-test/rec.motorcycles/104673")
'rec.motorcycles'
>>> bT.classify("20news-bydate-test/sci.med/59246")
'sci.med'
>>> bT.classify("20news-bydate-test/soc.religion.christian/21424")
'soc.religion.christian'
```

همان طور که می‌بینید، تابع طبقه بند (classify) یک نام فایل را می‌گیرد و یک رشته برمی‌گرداند که این رشته همان طبقه‌ی مربط به آن فایل (سند) است.

یک فایل پایتونی در وب‌سایت موجود است که می‌توانید به صورت قالب از آن استفاده کنید. این فایل با نام bayesText-ClassifyTemplate.py در سایت ما موجود است.

```
class BayesText:

  def __init__(self, trainingdir, stopwordlist):
    self.vocabulary = {}
    self.prob = {}
    self.totals = {}
    self.stopwords = {}
    f = open(stopwordlist)
    for line in f:
        self.stopwords[line.strip()] = 1
    f.close()
    categories = os.listdir(trainingdir)
    #filter out files that are not directories
    self.categories = [filename for filename in categories
                       if os.path.isdir(trainingdir +
filename)]
    print("Counting ...")
    for category in self.categories:
        print('    ' + category)
        (self.prob[category],
         self.totals[category]) = self.train(trainingdir,
category)
    # I am going to eliminate any word in the vocabulary
```

کد بزنید – راه حل

```
def classify(self, filename):
        results = {}
        for category in self.categories:
            results[category] = 0
        f = codecs.open(filename, 'r', 'iso8859-1')
        for line in f:
            tokens = line.split()
            for token in tokens:
                #print(token)
                token = token.strip('\'".,?:-').lower(
)
                if token in self.vocabulary:
                    for category in self.categories:
                        if self.prob[category][token]
== 0:
                            print("%s %s" % (category,
 token))
                        results[category] += math.log(
                            self.prob[category][token]
)
        f.close()
        results = list(results.items())
        results.sort(key=lambda tuple: tuple[1], rever
se = True)
        # for debugging I can change this to give me t
he entire list
```

```
        return results[0][0]
```

در آخر، بیایید کدی بزنیم که بتوانیم توسط آن، هر کدام از اسناد را در دایرکتوریِ آزمون به گروهِ مربوط به خود طبقه‌بندی کند و دقتِ این تابع را بر اساس درصد برگرداند.

```
def testCategory(self, directory, category):
    files = os.listdir(directory)
    total = 0
    correct = 0
    for file in files:
        total += 1
        result = self.classify(directory + file)
        if result == category:
            correct += 1
    return (correct, total)

def test(self, testdir):
    """Test all files in the test directory--that direct
ory is
    organized into subdirectories--each subdir is a clas
sification
    category"""
    categories = os.listdir(testdir)
    #filter out files that are not directories
    categories = [filename for filename in categories if
                  os.path.isdir(testdir + filename)]
    correct = 0
    total = 0
    for category in categories:
        print(".", end="")
        (catCorrect, catTotal) = self.testCategory(
            testdir + category + '/', category)
        correct += catCorrect
        total += catTotal
    print("\n\nAccuracy is  %f%%  (%i test instances)" %
          ((float(correct) / total) * 100, total))
```

هنگامی که این کد را همراه با فایل کلمات توقف (که خالی هستند) اجرا می‌کنم، نتیجـه‌ی زیر را می‌گیرم:

```
DONE TRAINING
Running Test ...
.....................
Accuracy is  77.774827%  (7532 test instances)
```

کد بزنید

آیا می‌توانید همین طبقه‌بند را برای چند لیست از کلمات توقف اجرا کنید؟ آیا کارایی افزایش پیدا می‌کند؟ کدام یک دقیق‌تر هستند؟ (برای پیدا کردن یک یا چند لیست از کلمات توقف، می‌توانید در اینترنت جستجو کنید) (مثلا برای پیدا کردن کلمات توقف در

انگلیسی عبارت English stop words list را جستجو کنید، و برای فارسی هم perisan stop words list)

ستون stop list size، سایز مجموعه‌ی کلمات توقف هستند و ستون accuracy، دقت را نشان می‌دهد:

stop list size	accuracy
0	77.774827
list 1	
list 2	

کد بزنید – راه حل

من یک لیستی از ۲۵ کلمه‌ی توقف انگلیسی را در http://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html و یک لیستِ ۱۷۴ کلمه‌ای را در http://www.ranks.nl/resources/stopwords.html پیدا کرده‌ام.

البته در وب‌سایت خودمان هم برای دانلود موجود است.

این نتایجی است که من به دست آورده‌ام:

stop list size	accuracy
0	77.774827%
25 word list	78.757302%
174 word list	79.938927%

پس در این مورد، به نظر می‌رسد که لیستِ ۱۷۴تایی، کارایی را حدوداً ۲ درصد نسبت به حالتی که از کلمات توقف استفاده نمی‌کردیم، بهبود داده است. آیا با نتایجی که شما به دست آوردید، مطابقت داشت؟

بیز ساده (Naïve Bayes) و تحلیل احساسات (Sentiment Analysis)

هدف اصلی در تحلیل احساس، مشخص کردن گرایش نویسنده یا نظر او است.

یکی از شکل‌های رایج در تحلیل احساسات، تعیین **قطبیت (polarity)** برای یک نوشته است، مثلاً یک نظر (مثبت یا منفی) یا چیزی شبیه به این. برای حل این مسئله، می‌توانیم از الگوریتمِ طبقه‌بندی بیزساده (Naïve Bayes) استفاده کنیم. برای این‌کار می‌توانیم از مجموعه‌ی داده‌ی **قطبیت نظرات فیلم** که ابتدا در سال ۲۰۰۴ توسط Pang and Lee ارائه شد، استفاده کنیم. مقاله‌ی کامل آن‌ها در زیر آمده است:

Pang, Bo and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. Proceedings of ACL.

مجموعه‌ی داده‌ی آن‌ها، شامل ۱۰۰۰ نظر مثبت و ۱۰۰۰ نظر منفی بود. دو مورد از آن‌ها را در زیر مشاهده می‌کنید: (سمت چپ نظر منفی و سمت راست نظر مثبتی است)

the second serial-killer thriller of the month is just awful . oh , it starts deceptively okay , with a handful of intriguing characters and some solid location work . ...

when i first heard that romeo & juliet had been " updated " i shuddered . i thought that yet another of shakespeare's classics had been destroyed .
fortunately , i was wrong . baz luhrman has directed an " in your face " , and visually

مجموعه‌ی داده را می‌توانید از http://www.cs.cornell.edu/people/pabo/movie-review-data/ دانلود کنید. من داده‌ها را در ۱۰ قسمت (bucket) یعنی همان ۱۰ تکه (fold) به صورت زیر در دایرکتوری‌های مختلف جمع‌آوری کرده‌ام (neg نشان‌دهنده‌ی منفی بودن و pos نشان‌دهنده‌ی مثبت بودنِ نظرات موجود در آن دایرکتوری است):

```
review_polarity_buckets
    txt_sentoken
        neg
            0
                files in fold 0
            1
                files in fold 1
            ...
            9
                files in fold 9
        pos
            0
                files in fold 0
            ...
```

این ساختار در وب‌سایت ما قابل دانلود است (guidetodatamining.com و chistio.ir)

کد بزنید

آیا می‌توانید روش بیز ساده (naïve bayes) را طوری تغییر دهید که اعتبارسنجی متقابل ۱۰-تکه‌ای (10-fold cross validation) را بر روی این مجموعه‌ی داده انجام دهد؟ مثلاً خروجی چیزی شبیه به شکل زیر شود:

```
          Classified as:
             neg   pos
           +-----+-----+
neg        |   1 |   2 |
pos        |   3 |   4 |
           +-----+-----+
12.345 percent correct
total of 2000 instances
```

همچنین ضریب کاپا (kappa coefficient) را هم محاسبه کنید.

رفع مسئولیت واضح

قطعاً شما فقط با خواندن این کتاب، یک داده‌کاوِ خبره نخواهید شد، همان‌طور که خواندن یک کتابِ پیانو، شما را یک پیانیستِ حرفه‌ای نخواهد کرد. حتماً و لزوماً نیاز به تمرین دارید!

در تصویر زیر، زنی در حال نواختن برام را مشاهده می‌کنید!

و متقابلاً در تصویر زیر، زنی را مشاهده می‌کنید که دارد بیز ساده را تمرین می‌کند (تمرین باعث می‌شود تا قلب با علاقه‌ی بیشتری رشد کند)

کد بزنید – راه حل

نتیجه‌ای که من گرفتم به این صورت بوده است:

```
          Classified as:
             neg   pos
           +-----+-----+
 neg       |   1 |   2 |
 pos       |   3 |   4 |
           +-----+-----+
12.345 percent correct
total of 2000 instances
```

همچنین ضریب کاپا را هم حساب کرده‌ام:

$$\kappa = \frac{P(c) - P(r)}{1 - P(r)} = \frac{.8115 - 0.5}{1 - 0.5} = \frac{.3115}{.5} = 0.623$$

پس ما نتیجه‌ی خوبی برای این الگوریتم بر روی این داده‌ها گرفته‌ایم.

کد نیز در ادامه آمده است:

و باز هم یادآوری که

کدها را می‌توانید از سایت guidetodataminig.com یا chistio.ir دانلود کنید.

```
from __future__ import print_function
import os, codecs, math

class BayesText:

    def __init__(self, trainingdir, stopwordlist, ignore
Bucket):
        """This class implements a naive Bayes approach
to text
        classification
        trainingdir is the training data. Each subdirect
ory of
        trainingdir is titled with the name of the class
ification
        category -- those subdirectories in turn contain
 the text
        files for that category.
        The stopwordlist is a list of words (one per lin
e) will be
        removed before any counting takes place.
        """
        self.vocabulary = {}
        self.prob = {}
        self.totals = {}
        self.stopwords = {}
        f = open(stopwordlist)
```

```
        for line in f:
            self.stopwords[line.strip()] = 1
        f.close()
        categories = os.listdir(trainingdir)
        #filter out files that are not directories
        self.categories = [filename for filename in cate
gories
                           if os.path.isdir(trainingdir
+ filename)]
        print("Counting ...")
        for category in self.categories:
            #print('    ' + category)
            (self.prob[category],
             self.totals[category]) = self.train(trainin
gdir, category,
                                                 ignoreB
ucket)
        # I am going to eliminate any word in the vocabu
lary
        # that doesn't occur at least 3 times
        toDelete = []
        for word in self.vocabulary:
            if self.vocabulary[word] < 3:
                # mark word for deletion
                # can't delete now because you can't del
ete
                # from a list you are currently iteratin
g over
                toDelete.append(word)
        # now delete
        for word in toDelete:
            del self.vocabulary[word]
        # now compute probabilities
        vocabLength = len(self.vocabulary)
        #print("Computing probabilities:")
        for category in self.categories:
            #print('    ' + category)
            denominator = self.totals[category] + vocabL
ength
            for word in self.vocabulary:
                if word in self.prob[category]:
                    count = self.prob[category][word]
                else:
                    count = 1
                self.prob[category][word] = (float(count
 + 1)
```

```
                                                / denominat
or)
        #print ("DONE TRAINING\n\n")


    def train(self, trainingdir, category, bucketNumberT
oIgnore):
        """counts word occurrences for a particular cate
gory"""
        ignore = "%i" % bucketNumberToIgnore
        currentdir = trainingdir + category
        directories = os.listdir(currentdir)
        counts = {}
        total = 0
        for directory in directories:
            if directory != ignore:
                currentBucket = trainingdir + category +
 "/" + directory
                files = os.listdir(currentBucket)
                #print("   " + currentBucket)
                for file in files:
                    f = codecs.open(currentBucket + '/'
+ file, 'r', 'iso8859-1')
                    for line in f:
                        tokens = line.split()
                        for token in tokens:
                            # get rid of punctuation and
 lowercase token
                            token = token.strip('\'".,?:
-')
                            token = token.lower()
                            if token != '' and not token
 in self.stopwords:
                                self.vocabulary.setdefau
lt(token, 0)
                                self.vocabulary[token] +
= 1
                                counts.setdefault(token,
 0)
                                counts[token] += 1
                                total += 1
                    f.close()
        return(counts, total)


    def classify(self, filename):
```

```
        results = {}
        for category in self.categories:
            results[category] = 0
        f = codecs.open(filename, 'r', 'iso8859-1')
        for line in f:
            tokens = line.split()
            for token in tokens:
                #print(token)
                token = token.strip('\'".,?:-').lower()
                if token in self.vocabulary:
                    for category in self.categories:
                        if self.prob[category][token] ==
 0:
                            print("%s %s" % (category, t
oken))
                        results[category] += math.log(
                            self.prob[category][token])
        f.close()
        results = list(results.items())
        results.sort(key=lambda tuple: tuple[1], reverse
 = True)
        # for debugging I can change this to give me the
 entire list
        return results[0][0]

    def testCategory(self, direc, category, bucketNumber
):
        results = {}
        directory = direc + ("%i/" % bucketNumber)
        #print("Testing " + directory)
        files = os.listdir(directory)
        total = 0
        correct = 0
        for file in files:
            total += 1
            result = self.classify(directory + file)
            results.setdefault(result, 0)
            results[result] += 1
            #if result == category:
            #                correct += 1
        return results

    def test(self, testdir, bucketNumber):
        """Test all files in the test directory--that di
rectory is
```

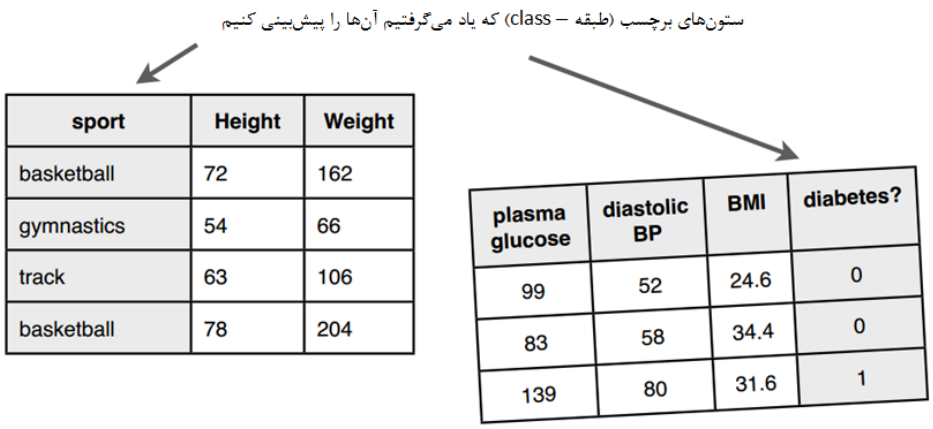
```
        organized into subdirectories--each subdir is a
classification
        category"""
        results = {}
        categories = os.listdir(testdir)
        #filter out files that are not directories
        categories = [filename for filename in categorie
s if
                      os.path.isdir(testdir + filename)]
        correct = 0
        total = 0
        for category in categories:
            #print(".", end="")
            results[category] = self.testCategory(
                testdir + category + '/', category, buck
etNumber)
        return results

def tenfold(dataPrefix, stoplist):
    results = {}
    for i in range(0,10):
        bT = BayesText(dataPrefix, stoplist, i)
        r = bT.test(theDir, i)
        for (key, value) in r.items():
            results.setdefault(key, {})
            for (ckey, cvalue) in value.items():
                results[key].setdefault(ckey, 0)
                results[key][ckey] += cvalue
                categories = list(results.keys())
    categories.sort()
    print(   "\n       Classified as: ")
    header =    "          "
    subheader = "        +"
    for category in categories:
        header += "% 2s   " % category
        subheader += "-----+"
    print (header)
    print (subheader)
    total = 0.0
    correct = 0.0
    for category in categories:
        row = " %s    |" % category
        for c2 in categories:
            if c2 in results[category]:
                count = results[category][c2]
            else:
```

```
                count = 0
            row += " %3i |" % count
            total += count
            if c2 == category:
                correct += count
        print(row)
    print(subheader)
    print("\n%5.3f percent correct" %((correct * 100) /
total))
    print("total of %i instances" % total)

# change these to match your directory structure
prefixPath = "/Users/raz/Dropbox/guide/data/review_polar
ity_buckets/"
theDir = prefixPath + "/txt_sentoken/"
stoplistfile = prefixPath + "stopwords25.txt"
tenfold(theDir, stoplistfile)
```

فصل ۸. خوشه بندی

کشف گروه‌ها و خوشه‌ها (clusters)

در فصول گذشته، ما سیستم‌های طبقه‌بندی (classification) را توسعه دادیم. در این سیستم‌ها، یک طبقه‌بند را بر روی یک مجموعه‌ی داده **دارای برچسب** آموزش می‌دادیم.



sport	Height	Weight
basketball	72	162
gymnastics	54	66
track	63	106
basketball	78	204

plasma glucose	diastolic BP	BMI	diabetes?
99	52	24.6	0
83	58	34.4	0
139	80	31.6	1

بعد از آموزشِ طبقه‌بند، می‌توانیم از آن برای برچسب‌زنیِ نمونه‌ها و مثال‌های جدید استفاده کنیم.

برای مثال، این شخص یک بازیکن بسکتبال است. آن یکی ژیمیناستیک‌کار است. آن شخص با احتمال کمی در ۳ سال آینده دیابت می‌گیرد و همین‌طور تا آخر. به بیانی دیگر،

الگوریتمِ طبقه‌بندی، **یک برچسب را از مجموعه‌ای از برچسب‌ها که در فاز آموزش دریافت کرده است، انتخاب می‌کند** – در واقع این الگوریتم، برچسب‌های ممکن را می‌داند.

این وظیفه به عهده‌ی الگوریتم‌هایی است که به **خوشه‌بندی (clustering)** معروف هستند. سیستم، نمونه‌ها را به خوشه‌ها یا گروه‌هایی تقسیم می‌کند و این کار را بر اساس یک معیارِ شباهت انجام می‌دهد. دو نوع الگوریتم اصلیِ خوشه‌بندی وجود دارد.

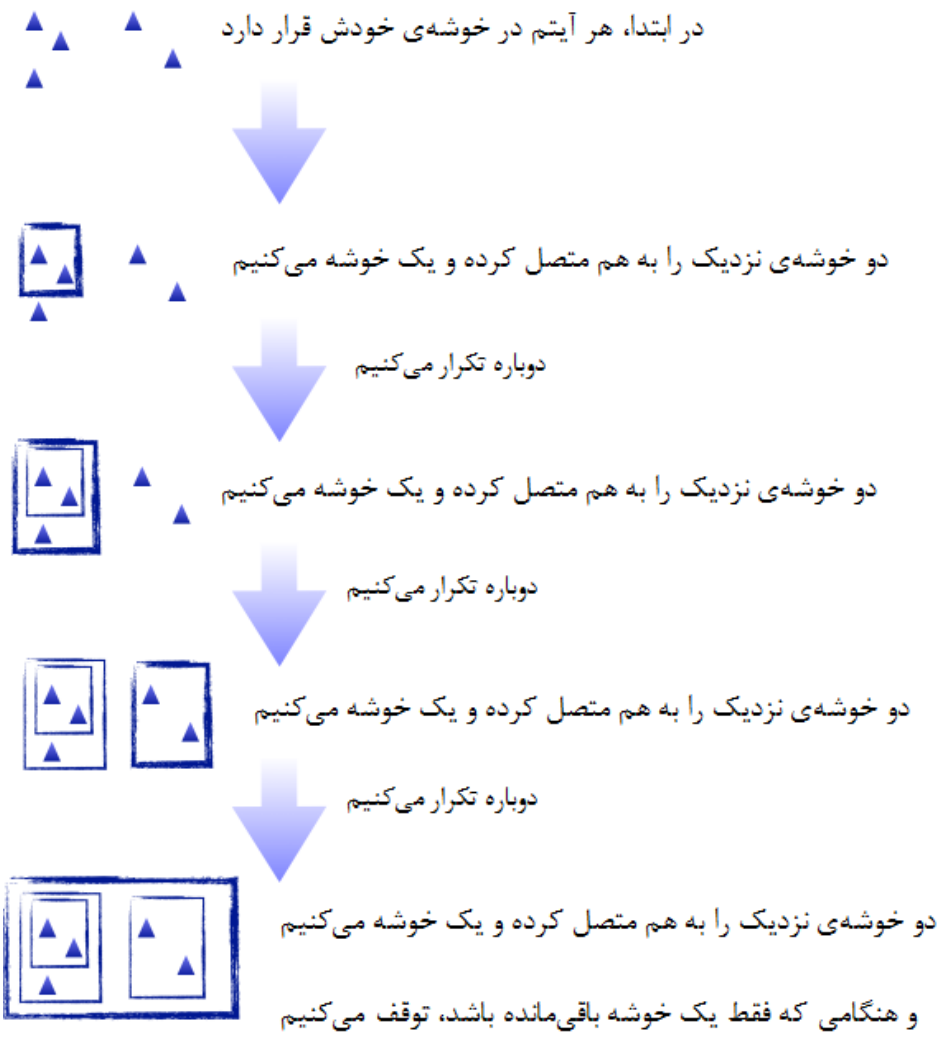
خوشه‌بندی k-means

در این نوع خوشه‌بندی، ما به الگوریتم می‌گوییم که چند عدد خوشه می‌خواهیم به دست بیاوریم. برای مثال، به الگوریتم می‌گوییم، لطفاً این ۱۰۰۰ نفر را به ۵ گروه تقسیم کن. یا لطفاً این صفحات وب را به ۱۵ گروه قسمت‌بندی کن. این گروه‌بندی‌ها به وسیله‌ی روشی

به نام **خوشه‌بندی k-means** انجام می‌شوند و کمی جلوتر در همین فصل درباره‌ی این‌ها صحبت خواهیم کرد.

خوشه‌بندی سلسله‌مراتبی (Hierarchical)

در این روش، ما مشخص نمی‌کنیم که می‌خواهیم به چه تعداد خوشه یا گروه برسیم. به جای آن، الگوریتم، به این صورت شروع می‌کند که هر کدام از نمونه‌ها را یک خوشه در نظر می‌گیرد. در هر بار تکرارِ الگوریتم، **دو خوشه‌ای را که بیشتر از همه شبیه به هم هستند، یکی می‌کند**. این کار را تا موقعی تکرار می‌کند که فقط یک خوشه باقی بماند. به این روش **خوشه‌بندی سلسله‌مراتبی** می‌گویند که کاملاً اسم بامسمایی است. با اجرای این الگوریتم، در نهایت به یک خوشه می‌رسیم که شامل دو زیر خوشه (sub-cluster) هست. هر کدام از این دو زیر خوشه به نوبه‌ی خود، به ۲ زیر زیر خوشه تقسیم می‌شود و همین‌طور تا آخر.



دوباره با هم مرور کنیم. در هر بار تکرار، دو خوشه‌ی نزدیک به هم را یکی می‌کنیم. برای مشخص کردنِ «نزدیک‌ترین خوشه‌ها»، بایستی از یک **فرمولِ فاصله** استفاده کنیم. ولی ما روش‌هایی داریم که توسط آن می‌توانیم فاصله‌ی بین دو خوشه را نسبت به هم مقایسه کنیم. هر کدام از این روش‌های مقایسه، منتج به روش‌های خوشه‌بندیِ مختلف می‌شود. سه خوشه‌ی A، B و C را فرض کنید که هر کدام دو عضو (دو نمونه) دارند. کدام دو جفت

خوشه را در مرحله‌ی بعد بایستی با یکدیگر متصل کرده و یکی کنیم؟ خوشه‌ی A با B، یا خوشه‌ی C با B؟

خوشه‌بندی Single-linkage

در روش single-linkage ما فاصله‌ی میان دو خوشه را به صورت زیر تفسیر می‌کنیم:
نزدیک‌ترین فاصله بین هر عضوی از یکی از خوشه‌ها به هر عضو دیگری از خوشه‌ی دیگر.
با این تعریف، فاصله‌ی بین خوشه‌ی A و خوشه‌ی B (در تصویر بالا) برابر فاصله‌ی بین A1 و B1 می‌شود، زیرا این فاصله نزدیک‌تر از هر جف فاصله‌ی دیگر بین اعضای این دو خوشه است. یعنی این فاصله نسبت به فاصله‌های A1 و B2، فاصله‌ی A2 و B1 و فاصله‌ی A2 و B2 کمتر است. با استفاده از خوشه‌بندی single-linkage، خوشه‌ی A به خوشه‌ی B نزدیک‌تر است تا خوشه‌ی C. پس در یک مرحله، ما خوشه‌های A و B را با هم ترکیب کرده و به یک خوشه تبدیل می‌کنیم.

خوشه‌بندی Complete-linkage

در روش complete-linkage ما فاصله‌ی میان دو خوشه را به صورت زیر تفسیر می‌کنیم:
دورترین فاصله بین هر عضوی از یکی از خوشه‌ها به هر عضو دیگری از خوشه‌ی دیگر.
با این تعریف، فاصله‌ی بین خوشه‌ی A و خوشه‌ی B، فاصله‌ی بین A2 و B2 است. در واقع در خوشه‌بندیِ complete-linkage، در شکل بالا، خوشه‌ی C به خوشه‌ی B نزدیک‌تر است تا خوشه‌ی A به خوشه‌ی B. پس در یک مرحله، خوشه‌ی B و C را با هم ترکیب می‌کنیم.

خوشه‌بندی Average-linkage

در روش average-linkage، ما فاصله‌ی میان دو خوشه را به صورت زیر تفسیر می‌کنیم: میانگین فاصله‌ی بین هر عضوی از یکی از خوشه‌ها، به هر عضو از خوشه‌ی دیگر.
در شکل بالا به نظر می‌رسد به صورت میانگین، فاصله‌ی خوشه‌های C و B کمتر از A و B است. پس در یک مرحله، خوشه‌های C و B را با هم ترکیب می‌کنیم.

فکر خوبیه! بیایید نژاد سگ‌ها را بر اساس قد و وزنشان خوشه‌بندی کنیم!

نژاد	قد	وزن
breed	height (inches)	weight (pounds)
Border Collie	20	45
Boston Terrier	16	20
Brittany Spaniel	18	35
Bullmastiff	27	120
Chihuahua	8	8
German Shepherd	25	78
Golden Retriever	23	70
Great Dane	32	160
Portuguese Water Dog	21	50
Standard Poodle	19	65
Yorkshire Terrier	6	7

فکر کنم یک چیزی را فراموش کردیم! به نظرت قبل از محاسبه‌ی فاصله، نباید کاری بکنیم؟

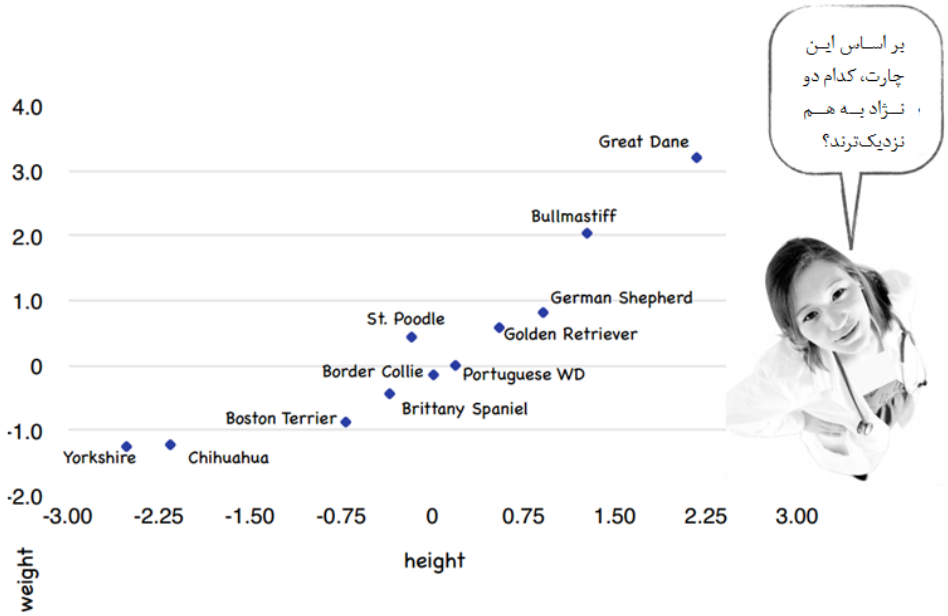
امتیازات استاندارد اصلاح‌شده
Modified Standard Scores

breed	height	weight
Border Collie	0	-0.1455
Boston Terrier	-0.7213	-0.873
Brittany Spaniel	-0.3607	-0.4365
Bullmastiff	1.2623	2.03704
Chihuahua	-2.1639	-1.2222
German Shepherd	0.9016	0.81481
Golden Retriever	0.541	0.58201
Great Dane	2.16393	3.20106
Portuguese Water Dog	0.1803	0
Standard Poodle	-0.1803	0.43651
Yorkshire Terrier	-2.525	-1.25132

بعد از آن، می‌خواهیم فاصله‌ی اقلیدسی را بین نژادهای مختلف محاسبه کنیم

فاصله‌های اقلیدسی (چند عدد از فاصله‌هایی که به هم نزدیک‌ترند با رنگ قرمز مشخص شده‌اند) – سطرها و ستون‌ها نشان‌دهده‌ی نژادهای مختلف سگ‌ها هستند که ستون‌ها به صورت مخفف نمایش داده شده‌اند، مثلاً BT به معنای Boston Terrier است:

	BT	BS	B	C	GS	GR	GD	PWD	SP	YT
Border Collie	1.024	0.463	2.521	2.417	1.317	0.907	3.985	0.232	0.609	2.756
Boston Terrier		0.566	3.522	1.484	2.342	1.926	4.992	1.255	1.417	1.843
Brittany Spaniel			2.959	1.967	1.777	1.360	4.428	0.695	0.891	2.312
Bullmastiff				4.729	1.274	1.624	1.472	2.307	2.155	5.015
Chihuahua					3.681	3.251	6.188	2.644	2.586	0.362
German Shphrd						0.429	2.700	1.088	1.146	4.001
Golden Retriever							3.081	0.685	0.736	3,572
Great Dane								3.766	3.625	6.466
Portuguese WD									0.566	2.980
Standard Poodle										2.889



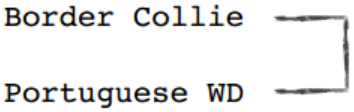
(در این شکل، هر کدام از نژاد سگ‌ها در دو بُعد با توجه به مقادیرِ نرمال‌شده، قرار می‌گیرند. محور افقی نماینده‌ی «قد» و محور عمودی نماینده‌ی «وزن» است)

اگر گفته باشید Border Collie و Portuguese Water Dog به هم نزدیک‌تر هستند، درست گفته‌اید!

الگوریتم

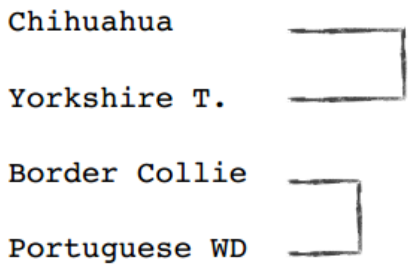
مرحله‌ی ۱.

در ابتدا، هر کدام از نژادها در خوشه‌ی خودشان هستند. ما دو خوشه را که به هم نزدیک‌تر هستند، پیدا کرده و آن‌ها را به یک خوشه تبدیل می‌کنیم. در جدولِ صفحه‌ی قبل، مشاهده می‌کنیم دو خوشه‌ای که به هم نزدیک‌ترند Border Collie و Portuguese Water Dog (با فاصله‌ی ۰,۲۳۲) هستند، پس آن‌ها را با هم ترکیب می‌کنیم.



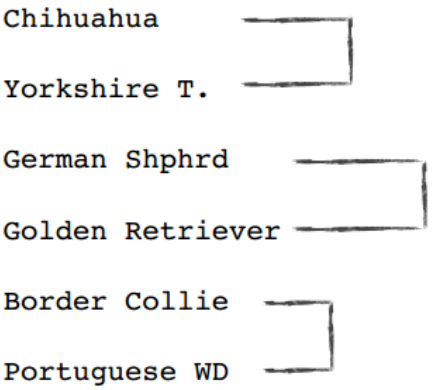
مرحله‌ی ۲.

دو مرتبه، دو خوشه‌ای که به هم نزدیک‌تر باشند را مشخص کرده و با هم ترکیب می‌کنیم. در جدول بالا، مشاهده می‌کنیم که این دو خوشه Chihuahua و Yorkshire Terrier (با فاصله‌ی ۰,۳۶۲) هستند. پس این دو را با هم ترکیب می‌کنیم.



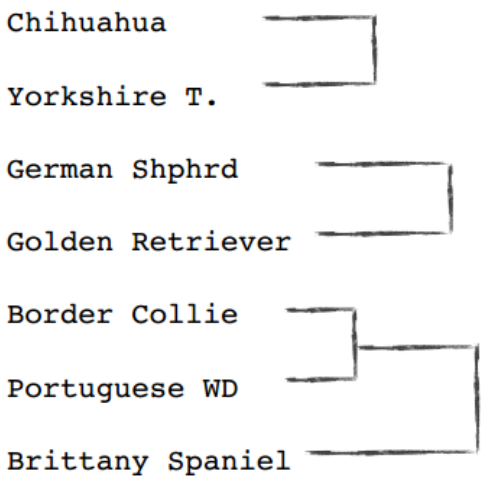
مرحله‌ی ۳.

فرآیند را دوباره تکرار می‌کنیم. این بار نوبت ترکیبِ German Shepherd و Golden Retriever است.

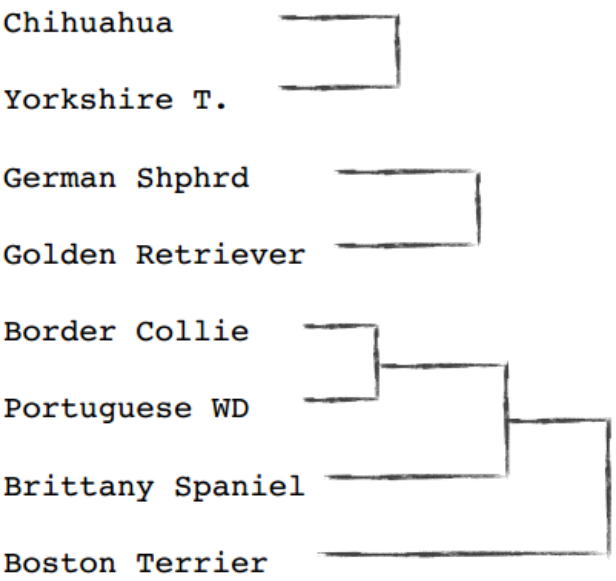


مرحله‌ی ۴.

دوباره همان فرآیند را تکرار می‌کنیم. از جدول مشاهده می‌کنیم که نزدیک‌ترین جفت به یکدیگر Border Collie و Brittany Spaniel هستند. ولی Border Collie در حال حاضر در یک خوشه با Portuguese Water Dog قرار دارد (در مرحله‌ی ۱ آن دو را در یک خوشه قرار دادیم). پس در این مرحله این خوشه را با Brittany Spaniel ترکیب می‌کنیم.



و ادامه می‌دهیم:



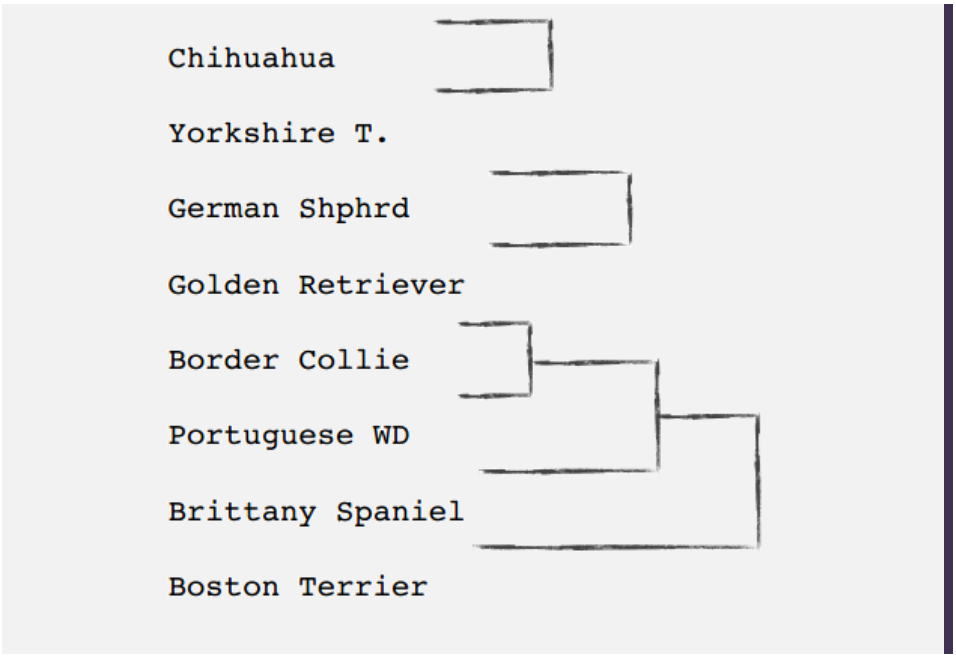
این نوع دیاگرام‌ها، به دِندروگرام (dendrogram) معروف هستند. در واقع یک دیاگرامِ درختی است که خوشه‌ها را نشان می‌دهد.

مدادتان را تیز کنید

خوشه‌بندی را برای داده‌های نژادهای سگ‌ها به اتمام برسانید!

برای حل این مسئله، یک لیست مرتب شده‌ای از فاصله بین نژاد سگ‌ها در لینک زیر موجود است:

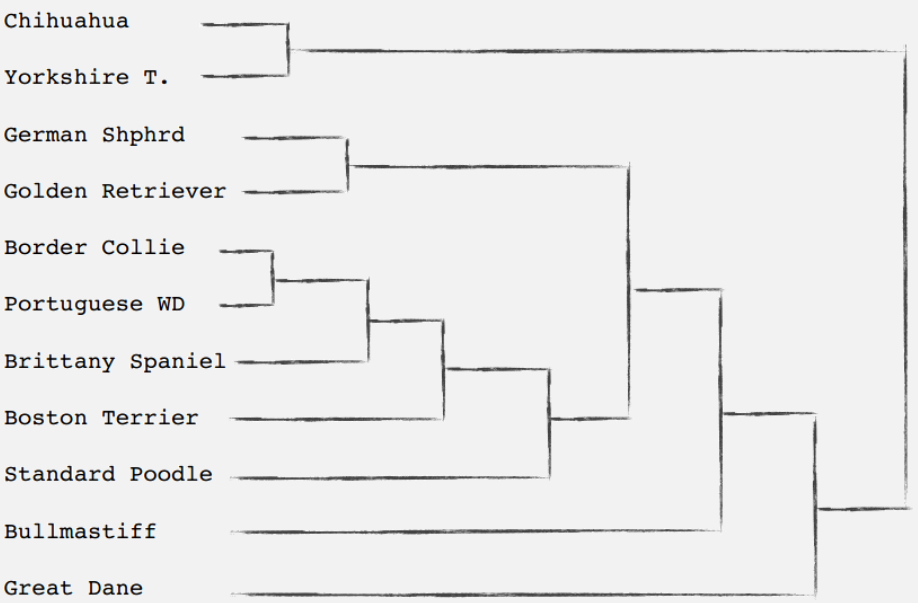
https://raw.githubusercontent.com/zacharski/pg2dm-python/0684ec677a1a1baaecb47bc0f8f21ec121e83339/data/ch8/dogDistanceSorted.txt



مدادتان را تیز کنید – راه حل

برای حل این مسئله، یک لیست مرتب شده‌ای از فاصله بین نژاد سگ‌ها در لینـک زیـر موجود است:

https://raw.githubusercontent.com/zacharski/pg2dm-python/0684ec677a1a1baaecb47bc0f8f21ec121e83339/data/ch8/dogDistanceSorted.txt

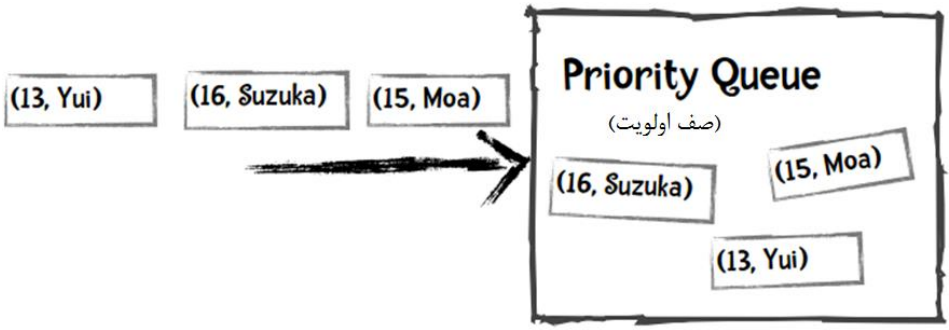


کُد پایتون برای یک الگوریتم خوشه‌بندی سلسله‌مراتبی

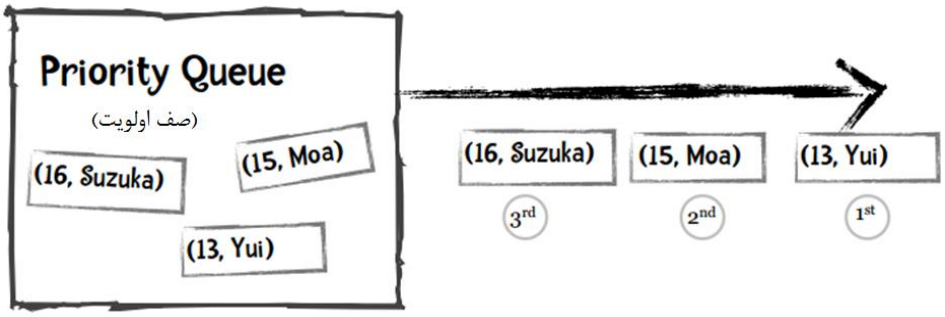
.

در صف اولویت، هر کدام از آیتم‌هایی که به صف وارد می‌شوند، یک صفت به نام «**اولویت**» دارند و آیتم‌ها به ترتیبی که ویژگیِ اولویت مشخص می‌کند، از صف خوانده می‌شوند.

آیتم‌هایی که اولویت بیشتری داشته باشند، زودتر خارج می‌شوند. در مثال ما، مثلاً شخصی که جوان‌تر باشد، زودتر خارج می‌شود. چون اولویت را سن افراد گذاشته‌ایم.
در شکل زیر داده‌ها را به همان ترتیب قبل وارد صف می‌کنیم!



ولی این‌بار اولین آیتمی که از صف خارج می‌شود، Yui خواهد بود به خاطر این‌که او جوان‌ترین فرد است، پس اولویت بیشتری دارد.

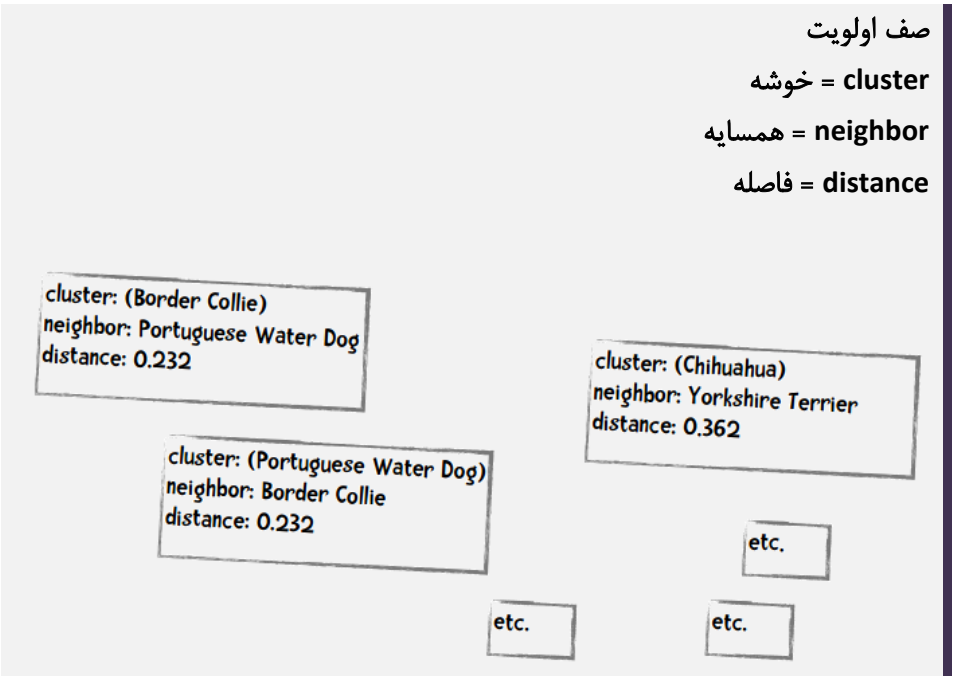


حالا ببینیم که این صف به چه صورتی در پایتون کار می‌کند:

```
>>> from queue import PriorityQueue          # load the PriorityQueue library
>>> singersQueue = PriorityQueue()           # create a PriorityQueue called
                                             # singersQueue
>>> singersQueue.put((16, 'Suzuka Nakamoto')) # put a few items in the queue
>>> singersQueue.put((15, 'Moa Kikuchi'))
>>> singersQueue.put((14, 'Yui Mizuno'))
```

```
>>> singersQueue.put((17, 'Ayaka Sasaki'))
>>> singersQueue.get()                          # The first item retrieved
(14, 'Yui Mizuno')                              # will be the youngest, Yui.
>>> singersQueue.get()
(15, 'Moa Kikuchi')
>>> singersQueue.get()
(16, 'Suzuka Nakamoto')
>>> singersQueue.get()
(17, 'Ayaka Sasaki')
```

برای این مثال که می‌خواهیم خوشه‌بندی سلسله مراتبی را انجام دهیم، خوشه‌ها را در یک صف اولویت قرار می‌دهیم. در این‌جا اولویت، کوتاه‌ترین فاصله به نزدیک‌ترین همسایه از یک خوشه خواهد بود. برای مثال، برای مسئله‌ی نژادِ سگ‌ها، ما نژاد Border Collie را طوری در صف قرار می‌دهیم که نزدیک‌ترین همسایه به آن یعنی Portuguese Water Dog در فاصله‌ی ۰٫۲۳۲ قرار بگیرد. ورودی‌های دیگرِ مشابه را هم، به همین شکل، برای بقیه‌ی نژادها در صف قرار می‌دهیم:



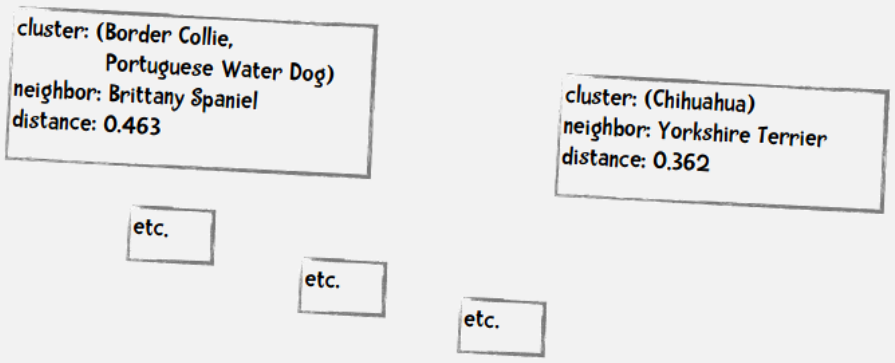
ما دو ورودی با نزدیک‌ترین فاصله را انتخاب می‌کنیم، تا مطمئن شویم که زوج‌های متناسبی را انتخاب کرده‌ایم. در این مثال، ما دو نژاد Border Collie و Portuguese Water Dog را انتخاب می‌کنیم و این دو خوشه را به یک خوشه ترکیب می‌کنیم. در این مورد، یک خوشه با نام Border Collie – Poruguese Water Dog خواهیم ساخت و این خوشه را در صف قرار می‌دهیم:

صف اولویت

cluster = خوشه

neighbor = همسایه

distance = فاصله



و این‌کار را تا وقتی انجام می‌دهیم که فقط یک خوشه باقی بماند. ورودی‌هایی که در صف قرار می‌دهیم، بایستی کمی پیچیده‌تر از این باشند. پس بیایید کمی دقیق‌تر به این مثال نگاه کنیم.

خواندن داده‌ها از فایل

داده‌ها در فایل CSV (مقادیر جداشده شده با کاما) هستند که ستون اول، نام آن نمونه‌ی خاص (نژاد سگ) و بقیه‌ی ستون‌ها، ویژگی‌های (attributes) دیگر هستند. اولین خط از این فایل هم، سرآیند (header) است که ویژگی‌ها را توضیح می‌دهد:

```
breed,height (inches),weight (pounds)
Border Collie,20,45
Boston Terrier,16,20
Brittany Spaniel,18,35
Bullmastiff,27,120
Chihuahua,8,8
German Shepherd,25,78
Golden Retriever,23,70
Great Dane,32,160
Portuguese Water Dog,21,50
Standard Poodle,19,65
Yorkshire Terrier,6,7
```

داده‌های این فایل در لیستی به اسم data، در برنامه ذخیره می‌شوند. این متغیر data، اطلاعات را به صورت ستونی ذخیره می‌کند. پس، data[0] لیستی است که نام نژادها را در خود دارد (برای مثال data[0][0] رشته‌ی «Border Collie» است و یا data[0][1] رشته‌ی «Boston Terrier» هست). همچنین data[1] لیستی است که شامل قدها (heights) می‌شود و data[2] برای وزن‌هاست (weights). تمامیِ داده‌ها، به جز آن‌هایی که در ستون اول باشند، به float (اعشار) تبدیل می‌شوند. برای مثال، data[1][0] یک مقدار عددیِ اعشاری برابر ۲۰٫۰ است و یا data[2][0] یک مقدار عددیِ اعشاری برابر ۴۵٫۰ است. هنگامی که داده‌ها خوانده شدند، نرمال‌سازی انجام می‌شود. در توضیح الگوریتم، از عبارت index (به معنای شاخص) جهتِ اشاره به شماره‌ی سطر برای هر کدام از نمونه‌ها استفاده خواهم کرد. برای مثال، Border Collie در شاخصِ (index) شماره‌ی **صفر** قرار دارد و Boston Terrier در شاخص (index) شماره‌ی **یک** و یا Yorkshine Terrier در شاخص (index) شماره‌ی ۱۰.

ایجاد صف اولویت اولیه

برای شروع الگوریتم، هر کدام از نژادها را در صف قرار می‌دهیم. فرض کنید ورودیِ Border Collie را می‌خواهیم در صف قرار دهیم. ابتدا، فاصله‌ی Border Collie را با تمامیِ نژادهای دیگر به دست آورده و این اطلاعات را در یک دیکشنریِ پایتونی ذخیره می‌کنیم.

```
{1: ((0, 1), 1.0244),      the distance between the Border Collie (index 0) and the Boston Terrier
                           (index 1), is 1.0244
 2: ((0, 2), 0.463),       the distance between the Border Collie the Brittany Spaniel is 0.463
 ...
 10: ((0, 10), 2.756)}     the Border Collie -- Yorkshire Terrier distance is 2.756
```

همچنین حواسمان را به نزدیک‌ترین همسایه‌ی Border Bollie جمع می‌کنیم و می‌دانیم که فاصله‌ی نزدیک‌ترین همسایه‌ی آن به صورت زیر به دست می‌آید:

```
closest distance: 0.232
nearest pair: (0, 8)
```

(نزدیک همسایه به Border Collie یعنی index 0، نژاد Portuguese Water Dog یعنی index 8 است و بالعکس)

مشکل مسافت‌های یکسان و این‌که با این داده‌ها چه کار کنیم؟

ممکن است متوجه شده باشید که در جدولِ بالا، فاصله‌ی بین Portuguese Water Dog و Standard Poodle و فاصله‌ی بین Boston Terrier و Brittany Spaniel با هم مساوی و برابر ۰,۵۶۶ هستند. اگر ما آیتم‌ها را از صف اولویت بر اساس فاصله واکشی کنیم، احتمال دارد Standard Poodle و Boston Terrier را انتخاب کرده و آن‌ها را در یک خوشه قرار دهیم، که در واقع به یک اشتباه ختم می‌شود. برای جلوگیری از این خطا، ما از یک چندتایی (tuple) استفاده خواهیم کرد که شامل یک شاخص (بر اساس داده‌های مجموعه) از دو نژاد باشد به نحوی که نمایان‌گر فاصله‌ی بین آن‌ها باشد. برای مثال Portuguese Water Dog شاخصِ شماره‌ی ۸ را در مجموعه دارد (index 8) و Standard Poodle

شاخصِ شماره‌ی ۹ را، پس این چند تایی به صورت (8, 9) خواهد شد و این چندتایی به لیستِ نزدیک‌ترین همسایه داده می‌شود. پس نزدیک‌ترین همسایه برای Poodle به صورت زیر خواهد شد:

```
['Portuguese Water Dog', 0.566, (8,9)]
```

و نزدیک‌ترین همسایه برای Portuguese Water Dog نیز به این صورت است:

```
['Standard Poodle', 0.566, (8,9)]
```

با استفاده از این چندتایی (tuple)، هنگامی که ما آیتم‌ها را از صف واکشی می‌کنیم، می‌توانیم ببینم که آیا آن‌ها زوج‌های درستی هستند یا خیر.

یک نکته‌ی دیگر در مورد فاصله‌های یکسان

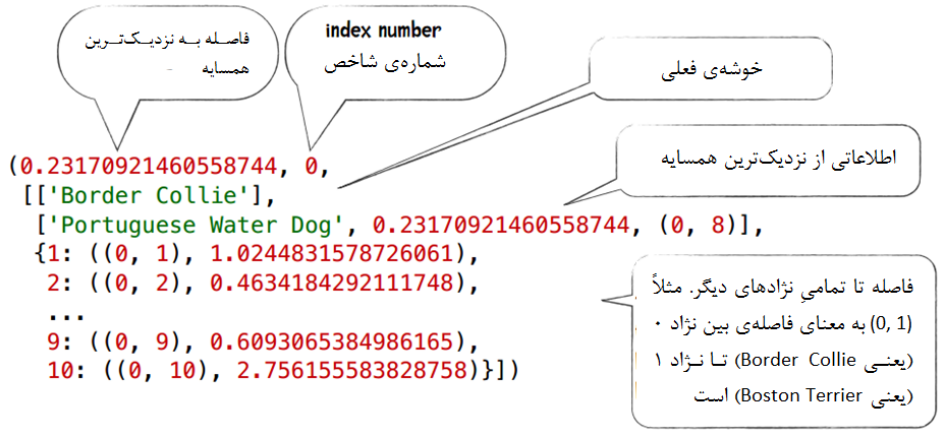
هنگامی که صف اولویت را در پایتون معرفی کردم (کدهای بالاتر)، چندتایی‌هایی شامل سن و نام بازیگران ژاپنی را هم واردِ صف کردم. این ورودی‌ها، بر اساس ویژگیِ سن (age) از صف واکشی می‌شدند. حالا فرض کنید دو ورودی، دارای سن‌های یکسان می‌بودند، آن وقت چه اتفاقی می‌افتاد؟

فرض کنید:

```
>>> singersQueue.put((15,'Suzuka Nakamoto'))
>>> singersQueue.put((15,'Moa Kikuchi'))
>>> singersQueue.put((15, 'Yui Mizuno'))
>>> singersQueue.put((15, 'Avaka Sasaki'))
>>> singersQueue.put((12, 'Megumi Okada'))
>>> singersQueue.get()
(12, 'Megumi Okada')
>>> singersQueue.get()
(15, 'Avaka Sasaki')
>>> singersQueue.get()
(15, 'Moa Kikuchi')
>>> singersQueue.get()
(15, 'Suzuka Nakamoto')
>>> singersQueue.get()
(15, 'Yui Mizuno')
>>>
```

همان‌طور که می‌بینید، اگر اولین آیتم در چندتاییِ مورد نظر ما، مثل هم باشند، پایتون به سراغ آیتم بعدی می‌رفت. مثلاً هنگامی که تمامی آیتم‌ها ۱۵ سال داشته باشند، ورودی‌ها بر اساس آیتم بعدی، یعنی «نام شخص» واکشی می‌شوند. و چون این نام‌ها همگی رشته هستند، به ترتیبِ حروف الفبا پشت سر هم چیده می‌شوند. پس مثلاً، نامِ Avaka Sasaki زودتر از Moa Kikuchi واکشی می‌شد و نامِ Moa هم طبیعتاً زودتر از Suzuka واکشی می‌شود.

در مثال ما که همان **خوشه‌بندیِ سلسله‌مراتبی** بود، از فاصله‌ی بین نژاد سگ‌ها، برای تعیین اولویت استفاده کردیم و برای حل مشکل، از عدد شاخص (index) استفاده می‌کنیم. اولین آیتمی که در صف قرار می‌دهیم، شماره‌ی صفر دارد، آیتم بعدی، شاخص شماره‌ی ۱ و همین‌طور تا آخر. ورودی‌هایی که به صف اضافه می‌کنیم، به شکل زیر هستند:



برای هر کدام از این نژادها، صفِ اولویت را مقداردهی اولیه می‌کنیم.

آنقدر این کارها را انجام می‌دهیم تا هنگامی که فقط یک خوشه داشته باشیم. یعنی همه‌ی خوشه‌ها تجمیع شوند:

ما دو آیتم از صف می‌گیریم، آن‌ها را به یک خوشه تبدیل می‌کنیم و آن خوشه‌ی ترکیب شده را در صف درج می‌کنیم. در مثالِ نژادهای سگ، برای مثال، دو ورودیِ Border Collie و Portuguese Water Dog را واکشی کرده و آیتم زیر را تولید می‌کنیم:

```
['Border Collie', 'Portuguese Water Dog']
```

بعد از آن، فاصله‌ی این خوشه‌ی جدید را با تمام نژادهای سگ‌های دیگر (که هر کدام یک خوشه هستند) محاسبه می‌کنیم (البته به جز خودشان). ما این کار را با ادغام دیکشنریِ فاصله‌ها، از دو خوشه‌ی اولیه به صورت زیر انجام می‌دهیم. (دیکشنریِ فاصله‌ها را برای اولین آیتمی که از صف می‌گیریم، به صورت distanceDict1 نام‌گذاری کنیم و دومی را distanceDict2 و دیکشنری‌ای که جدید ایجاد شده را هم newDistanceDict می‌نامیم)

```
Initialize newDistanceDict to an empty dictionary
for each key, value pair in distanceDict1:
  if there is an entry in distanceDict2 with that key:
     if the distance for that entry in distanceDict1 is
        shorter than that in distanceDict2:
           place the distanceDict1 entry in newDistanceDict
        else:
           place the distanceDict1 entry in newDistanceDict
```

(لطفاً از روی کد پرش نکنید، سعی کنید یک‌بار کد را بخوانید و درک کنید. لازم نیست خیلی به خودتان فشار بیاورید. فقط سعی کنید این کار را انجام دهید)

key	value in the Border Collie Distance List مقدار در لیست فاصله‌ی Border Collie	value in the Portuguese Water Dog Distance List مقدار در لیست فاصله‌ی Portuguese Water Dog	value in the Distance List for the new cluster مقدار در لیست فاصله برای خوشه‌ی جدید
0	–	((0, 8), 0.2317092146055)	–
1	((0, 1), 1.02448315787260)	((1, 8), 1.25503395239308)	((0, 1), 1.02448315787260)
2	((0, 2), 0.46341842921117)	((2, 8), 0.69512764381676)	(0, 2), 0.46341842921117)
3	((0, 3), 2.52128307411504)	((3, 8), 2.3065500082408)	((3, 8), 2.3065500082408)
4	((0, 4), 2.41700998092941)	((4, 8), 2.643745991701)	((0, 4), 2.41700998092941)
5	((0, 5), 1.31725590972761)	((5, 8), 1.088215707936)	((5, 8), 1.088215707936)
6	((0, 6), 0.90660838225252)	((6, 8), 0.684696194462)	((6, 8), 0.684696194462)
7	((0, 7), 3.98523295438990)	((7, 8), 3.765829069545)	((7, 8), 3.765829069545)
8	((0, 8), 0.23170921460558)	–	–
9	((0, 9), 0.60930653849861)	((8, 9), 0.566225873458)	((8, 9), 0.566225873458)
10	((0, 10), 2.7561555838287)	((8, 10), 2.980333906137)	((0, 10), 2.7561555838287)

ورودیِ کاملی که قرار است در صف وارد شود به صورت زیر خواهد بود (این ورودی از ترکیب نژادِ Border Collie و Portuguese Water Dog ساخته شده است):

```
(0.4634184292111748, 11, [('Border Collie', 'Portuguese Water Dog'),
 [2, 0.4634184292111748, (0, 2)],
 {1: ((0, 1), 1.0244831578726061), 2: ((0, 2), 0.4634184292111748),
  3: ((3, 8), 2.306550008240866), 4: ((0, 4), 2.4170099809294157),
  5: ((5, 8), 1.0882157079364436), 6: ((6, 8), 0.6846961944627522),
  7: ((7, 8), 3.7658290695451373), 9: ((8, 9), 0.5662258734585477),
  10: ((0, 10), 2.756155583828758)}])
```

کد بزنید

آیا الگوریتم بالا را می‌توانید در پایتون پیاده‌سازی کنید؟

برای کمک به شما در مورد این سوال، فایل پایتونی در وب‌سایت به اسم hierarchicalClustererTemplate.py موجود است که نقطه‌ی شروع را به شما نشان می‌دهد. برای این کار بایستی:

۱. تابع init را بنویسید

برای هر کدام از ورودی در داده‌ها:

۱. فاصله‌ی اقلیدسی را بین آن ورودی و تمامی ورودی‌های دیگر محاسبه کرده و یک دیکشنریِ پایتونی مانند چیزی که در بالا توضیح داده شد، ایجاد کنید

۲. نزدیک‌ترین همسایه را پیدا کنید

۳. اطلاعات این ورودی را در صف وارد کنید

۲. تابع خوشه‌بندی را بنویسید. این تابع باید به صورت متناوب:

۱. دو ورودیِ اول را از صف دریافت کند

۲. آن‌ها را با هم ترکیب کند و یک خوشه‌ی جدید بسازد

۳. خوشه‌ی جدید را در صف قرار دهد

تا وقتی که فقط یک خوشه در صف باقی بماند

کد بزنید – راه حل

یادتان باشد که:

این راه‌حل، راه‌حل من است و لزوماً بهترین راه‌حل نیست. شاید شما بتوانید بهتر از آن را بنویسید!

```
from queue import PriorityQueue
import math


"""
Example code for hierarchical clustering
"""

def getMedian(alist):
    """get median value of list alist"""
    tmp = list(alist)
    tmp.sort()
    alen = len(tmp)
    if (alen % 2) == 1:
        return tmp[alen // 2]
    else:
        return (tmp[alen // 2] + tmp[(alen // 2) - 1]) /
 2
```

```
def normalizeColumn(column):
    """Normalize column using Modified Standard Score"""
    median = getMedian(column)
    asd = sum([abs(x - median) for x in column]) / len(c
olumn)
    result = [(x - median) / asd for x in column]
    return result

class hClusterer:
    """ this clusterer assumes that the first column of
the data is a label
    not used in the clustering. The other columns contai
n numeric data"""

    def __init__(self, filename):
        file = open(filename)
        self.data = {}
        self.counter = 0
        self.queue = PriorityQueue()
        lines = file.readlines()
        file.close()
        header = lines[0].split(',')
        self.cols = len(header)
        self.data = [[] for i in range(len(header))]
        for line in lines[1:]:
            cells = line.split(',')
            toggle = 0
            for cell in range(self.cols):
                if toggle == 0:
                   self.data[cell].append(cells[cell])
                   toggle = 1
                else:
                    self.data[cell].append(float(cells[c
ell]))
        # now normalize number columns (that is, skip th
e first column)
        for i in range(1, self.cols):
                self.data[i] = normalizeColumn(self.data
[i])

        ###
        ###  I have read in the data and normalized the
        ###  columns. Now for each element i in the data
, I am going to
```

```
        ###     1. compute the Euclidean Distance from e
lement i to all the
        ###        other elements.  This data will be pl
aced in neighbors,
        ###        which is a Python dictionary. Let's s
ay i = 1, and I am
        ###        computing the distance to the neighbo
r j and let's say j
        ###        is 2. The neighbors dictionary for i
will look like
        ###        {2: ((1,2), 1.23),  3: ((1, 3), 2.3).
.. }
        ###
        ###     2. find the closest neighbor
        ###
        ###     3. place the element on a priority queue
, called simply queue,
        ###        based on the distance to the nearest
neighbor (and a counter
        ###        used to break ties.



        # now push distances on queue
        rows = len(self.data[0])

        for i in range(rows):
            minDistance = 99999
            nearestNeighbor = 0
            neighbors = {}
            for j in range(rows):
                if i != j:
                    dist = self.distance(i, j)
                    if i < j:
                        pair = (i,j)
                    else:
                        pair = (j,i)
                    neighbors[j] = (pair, dist)
                    if dist < minDistance:
                        minDistance = dist
                        nearestNeighbor = j
                        nearestNum = j
            # create nearest Pair
            if i < nearestNeighbor:
                nearestPair = (i, nearestNeighbor)
            else:
```

```
                nearestPair = (nearestNeighbor, i)

            # put instance on priority queue
            self.queue.put((minDistance, self.counter,
                            [[self.data[0][i]], nearestP
air, neighbors]))
            self.counter += 1


    def distance(self, i, j):
        sumSquares = 0
        for k in range(1, self.cols):
            sumSquares += (self.data[k][i] - self.data[k
][j])**2
        return math.sqrt(sumSquares)


    def cluster(self):
         done = False
         while not done:
             topOne = self.queue.get()
             nearestPair = topOne[2][1]
             if not self.queue.empty():
                 nextOne = self.queue.get()
                 nearPair = nextOne[2][1]
                 tmp = []
                 ##
                 ##  I have just popped two elements off
 the queue,
                 ##  topOne and nextOne. I need to check
 whether nextOne
                 ##  is topOne's nearest neighbor and vi
ce versa.
                 ##  If not, I will pop another element
off the queue
                 ##  until I find topOne's nearest neigh
bor. That is what
                 ##  this while loop does.
                 ##

                 while nearPair != nearestPair:
                     tmp.append((nextOne[0], self.counte
r, nextOne[2]))
                     self.counter += 1
                     nextOne = self.queue.get()
                     nearPair = nextOne[2][1]
```

```
                    ##
                    ## this for loop pushes the elements I
popped off in the
                    ## above while loop.
                    ##
                    for item in tmp:
                        self.queue.put(item)

                    if len(topOne[2][0]) == 1:
                       item1 = topOne[2][0][0]
                    else:
                        item1 = topOne[2][0]
                    if len(nextOne[2][0]) == 1:
                       item2 = nextOne[2][0][0]
                    else:
                        item2 = nextOne[2][0]
                    ##  curCluster is, perhaps obviously, t
he new cluster
                    ##  which combines cluster item1 with c
luster item2.
                    curCluster = (item1, item2)

                    ## Now I am doing two things. First, fi
nding the nearest
                    ## neighbor to this new cluster. Second
, building a new
                    ## neighbors list by merging the neighb
ors lists of item1
                    ## and item2. If the distance between i
tem1 and element 23
                    ## is 2 and the distance betweeen item2
 and element 23 is 4
                    ## the distance between element 23 and
the new cluster will
                    ## be 2 (i.e., the shortest distance).
                    ##

                    minDistance = 99999
                    nearestPair = ()
                    nearestNeighbor = ''
                    merged = {}
                    nNeighbors = nextOne[2][2]
                    for (key, value) in topOne[2][2].items(
):
                       if key in nNeighbors:
```

```
                        if nNeighbors[key][1] < value[1]
:
                            dist =  nNeighbors[key]
                        else:
                            dist = value
                        if dist[1] < minDistance:
                            minDistance =  dist[1]
                            nearestPair = dist[0]
                            nearestNeighbor = key
                        merged[key] = dist

                if merged == {}:
                    return curCluster
                else:
                    self.queue.put( (minDistance, self.c
ounter,
                                     [curCluster, neares
tPair, merged]))
                    self.counter += 1




def printDendrogram(T, sep=3):
    """Print dendrogram of a binary tree.  Each tree nod
e is represented by a
    length-2 tuple. printDendrogram is written and provi
ded by David Eppstein
    2002. Accessed on 14 April 2014:
    http://code.activestate.com/recipes/139422-dendrogra
m-drawing/ """

    def isPair(T):
        return type(T) == tuple and len(T) == 2

    def maxHeight(T):
        if isPair(T):
            h = max(maxHeight(T[0]), maxHeight(T[1]))
        else:
            h = len(str(T))
        return h + sep

    activeLevels = {}

    def traverse(T, h, isFirst):
```

```
        if isPair(T):
            traverse(T[0], h-sep, 1)
            s = [' ']*(h-sep)
            s.append('|')
        else:
            s = list(str(T))
            s.append(' ')

        while len(s) < h:
            s.append('-')

        if (isFirst >= 0):
            s.append('+')
            if isFirst:
                activeLevels[h] = 1
            else:
                del activeLevels[h]

        A = list(activeLevels)
        A.sort()
        for L in A:
            if len(s) < L:
                while len(s) < L:
                    s.append(' ')
                s.append('|')

        print (''.join(s))

        if isPair(T):
            traverse(T[1], h-sep, 0)

    traverse(T, maxHeight(T), -1)



filename = '//Users/raz/Dropbox/guide/data/dogs.csv'

hg = hClusterer(filename)
cluster = hg.cluster()
printDendrogram(cluster)
```

هنگامی که این کد را اجرا می‌کنم، نتیجه‌ی زیر را می‌گیرم:

که با چیزی که به صورت دستی محاسبه کردیم مطابقت دارد.

حالا شما امتحان کنید!

غلات صبحانه

در وب‌سایتِ این کتاب (guidetodatamining.com و chistio.ir) فایلی وجود دارد که شامل اطلاعات غذایی حدود ۷۷ صبحانه‌ی غلات شامل:

اسم غله

کالری هر بار مصرف

پروتوئین (به گرم)

چربی (به گرم)

نمک (به میلی‌گرم)

فیبر (به گرم)

کربوهیدرات (به گرم)

قند (به گرم)

پتاسیم (به میلی‌گرم)
ویتامین‌ها (درصد از RDA)
است. آیا می‌توانید، خوشه‌بندی سلسله‌مراتبی را بر روی این داده‌ها انجام دهید؟
کدام یک از غلات بیشترین شباهت را به Trix دارد؟
کدام‌یک بیشترین شباهت را به Muesili Raisins & Almonds دارند؟

مجموعه‌ی داده، در اصل متعلق به دانشگاه کارنی ملون (Carnegie Mellon) است.
http://lib.stat.cmu.edu/DASL/Datafiles/Cereals.html

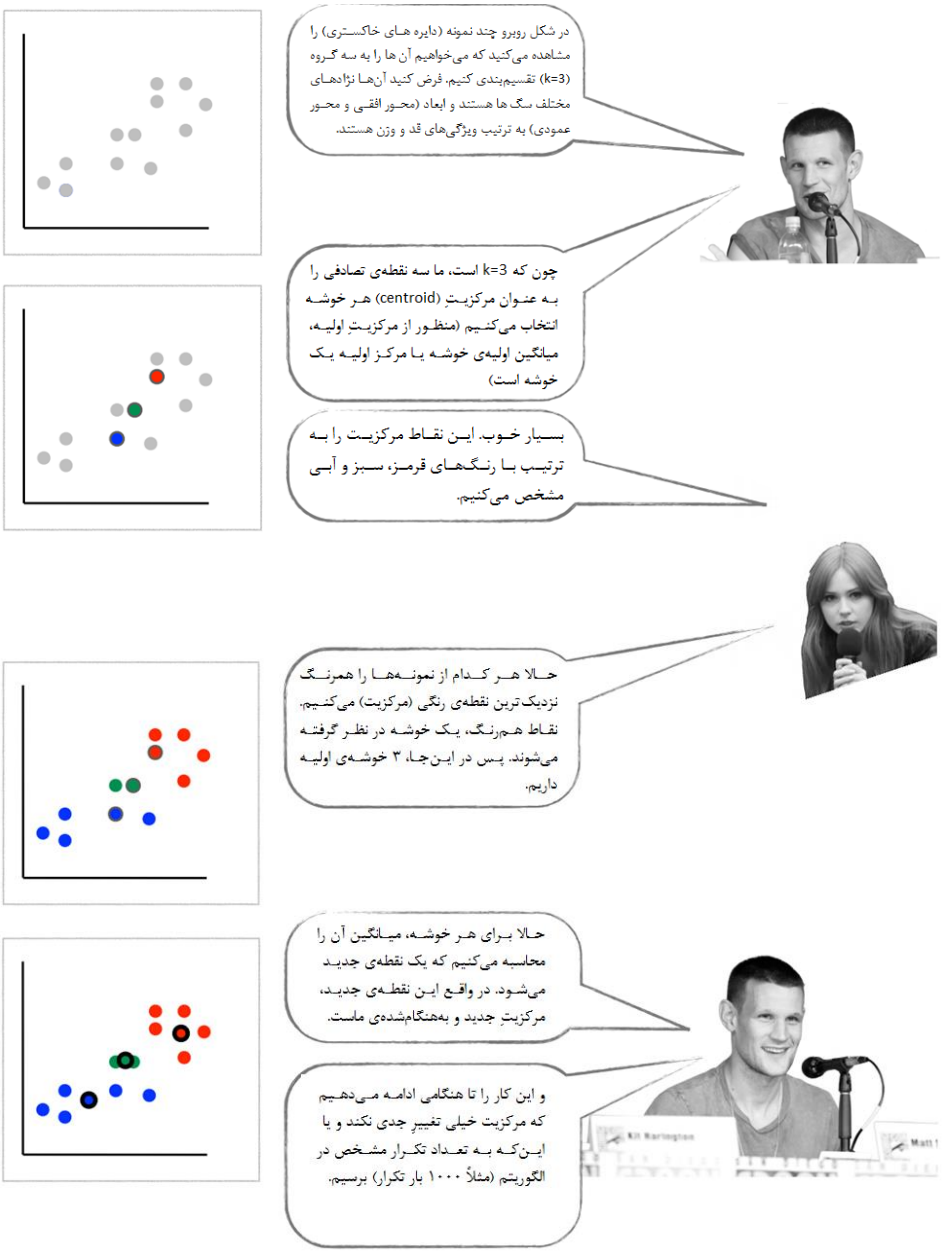
شما امتحان کنید – راه حل
برای اجرای خوشه‌بندی بر روی این مجموعه‌ی داده، فقط بایستی که نام فایل را از dogs.csv به cereal.csv تغییر دهیم. در زیر خلاصه‌ای از نتیجه را می‌بینید:

```
Mueslix Crispy Blend ------------------------------------------------------------------+
                                                                                       |--+
Muesli Raisins & Almonds ------------------------------------------------------------+  |
                                                                                       |--+
Muesli Peaches & Pecans --------------------------------------------------------------+
...
Lucky Charms ----------+
                       |--+
Fruity Pebbles --+     |  |
                 |--+  |  |
Trix ------------+  |  |  |
                    |--+  |
Cocoa Puffs -----+  |     |
                 |--+     |
Count Chocula ---+        |
```

مشاهده می‌کنید که Trix بیشتر به Fruity Pebbles شباهت دارد. همچنین به طور واضحی Muesli Raisins & Almonds به Muesli Peaches & Pecans شباهت دارند.

این هم از خوشه‌بندی سلسله مراتبی! (که به نظر ساده بود)

خوشه‌بندی K-means



پس الگوریتم k-means به صورت زیر کار می‌کند:

۱. تعداد k نمونه‌ی تصادفی را انتخاب می‌کنیم که به عنوان مرکزیت‌های (centroids) اولیه باشند

۲. موارد ۳ و ۴ را تکرار می‌کنیم:

۳. هر کدام از نمونه‌ها را به نزدیک‌ترین مرکزیت، متصل (همرنگ) می‌کنیم. این کار باعثِ ایجادِ k خوشه می‌شود

۴. مرکزیت‌ها را به‌هنگام می‌کنیم و آن‌ها را به جایی انتقال می‌دهیم که میانگین آن خوشه باشد

۵. تا وقتی که مرکزیت‌ها تغییر زیادی نکنند

اجازه بدهید، این کار را با یک مثال انجام دهیم. نقاط زیر را در نظر بگیرید (برای مثال فرض کنید این‌ها نقاط موجود در محورهای x و y در تصویر بالا)

```
(1, 2)
(1, 4)
(2, 2)
(2, 3)
(4, 2)
(4, 4)
(5, 1)
(5, 3)
```

می‌خواهیم این نمونه‌ها را به دو خوشه (گروه) تقسیم‌بندی کنیم

مرحله‌ی ۱ از الگوریتم بالا: تعداد k نمونه‌ی تصادفی را برای مرکزیت‌های اولیه انتخاب می‌کنیم

فرض کنید ما به صورت تصادفی، گزینه‌های (1, 4) را به عنوان centroid 1 و (4, 2) را به عنوان centroid 2 انتخاب کنیم

مرحله‌ی ۳: هر کدام از نمونه‌ها را به نزدیک‌ترین مرکزیت (centroid) متصل می‌کنیم

برای اتصالِ هر کدام از نمونه‌ها به نزدیک‌ترین مرکزیت، می‌توانیم از یک معیارِ فاصله که قبلاً یاد گرفتیم استفاده کنیم. برای ساده‌سازی، اجازه دهید از فاصله‌ی منهتن استفاده کنیم.

point	فاصله تا centroid 1 یعنی (1, 4)	فاصله تا centroid 2 یعنی (4, 2)
(1, 2)	2	3
(1,4)	0	5
(2, 2)	3	2
(2, 3)	2	3
(4, 2)	5	0
(4, 4)	3	2
(5, 1)	7	2
(5, 3)	5	2

بر اساس این جدول، نقاطِ مختلف را به خوشه‌های زیر تخصیص می‌دهیم.

خوشه‌ی ۱

```
CLUSTER 1
(1, 2)
(1, 4)
(2, 3)
```

خوشه‌ی ۲

```
CLUSTER 2
(2, 2)
(4, 2)
(4, 4)
(5, 1)
(5, 3)
```

مرحله‌ی ۴: مرکزیت‌ها (centroids) را به‌هنگام می‌کنیم

مرکزیت‌های جدید را با توجه به میانگینِ هر خوشه محاسبه می‌کنیم. میانگینِ محور x برای خوشه‌ی ۱ به صورت زیر محاسبه می‌شود:

(1 + 1 + 2) / 3 = 4/3 = 1.33

و برای ۷، میانگین به صورت زیر می‌شود:

(2 + 4 + 3) / 3 = 9/3 = 3

پس برای خوشه‌ی ۱، مرکزیت جدید برابر (1.33, 3) می‌شود.

به همین‌صورت، مرکزیت جدید برای خوشه‌ی ۲ نیز برابر (4, 2.4) می‌شود

مرحله‌ی ۵: تا زمانی که مرکزیت‌ها نسبت به مرحله‌ی قبلی تغییر خاصی نکرده باشند:
مرکزیت‌های قبلی، برابر (1, 4) و (4, 2) بودند و مرکزیت‌های جدید (1.33, 3) و (4, 2.4).
همان‌طور که ملاحظه می‌کنید، مرکزیت‌ها تغییر کرده‌اند، پس الگوریتم را ادامه می‌دهیم.
مرحله‌ی ۳: هر کدام از نمونه‌ها را به نزدیک‌ترین مرکزیت (centroid) متصل می‌کنیم
دوباره، فاصله‌ی منهتن را (این‌بار برای مرکزیت‌های جدید) محاسبه می‌کنیم:

point	فاصله تا centroid 1 یعنی (1.33, 3)	فاصله تا centroid 2 یعنی (4, 2.4)
(1, 2)	1.33	3.4
(1, 4)	1.33	4.6
(2, 2)	1.67	2.4
(2, 3)	0.67	2.6
(4, 2)	3.67	0.4
(4, 4)	3.67	1.6
(5, 1)	5.67	2.4
(5, 3)	3.67	1.6

و بر اساس این فاصله، هر کدام از نقاط را به مرکزیت‌های جدید، تخصیص می‌دهیم:

```
CLUSTER 1
(1, 2)
(1, 4)
(2, 2)
(2, 3)
```

```
CLUSTER 2
(4, 2)
(4, 4)
(5, 1)
(5, 3)
```

مرحله‌ی ۴: به‌هنگام‌سازی مرکزیت‌ها
مرکزیت جدید برای خوشه‌ی ۱ – CLUSTER 1: (1.5, 2.75)

مرکزیت جدید برا خوشه‌ی ۲ – CLUSTER 2: (4.5, 2.5)

مرحله‌ی ۵: تا وقتی که مرکزیت‌ها تغییری نکنند

مرکزیت‌ها تغییر کردند، پس تکرار را ادامه می‌دهیم

مرحله‌ی ۳: هر کدام از نمونه‌ها را به نزدیک‌ترین مرکزیت (centroid) متصل می‌کنیم

دوباره، فاصله‌ی منهتن:

point	فاصله تا centroid 1 یعنی (1.5, 2.75)	فاصله تا centroid 2 یعنی (4.5, 2.5)
(1, 2)	1.25	4.0
(1, 4)	1.75	5.0
(2, 2)	1.25	3.0
(2, 3)	0.75	3.0
(4, 2)	3.25	1.0
(4, 4)	3.75	2.0
(5, 1)	5.25	2.0
(5, 3)	3.75	1.0

تخصیص نقاط به مرکزیت‌های جدید:

```
CLUSTER 1
(1, 2)
(1, 4)
(2, 2)
(2, 3)
```

```
CLUSTER 2
(4, 2)
(4, 4)
(5, 1)
(5, 3)
```

مرحله‌ی ۴: به‌هنگام‌سازی مرکزیت‌ها

مرکزیتِ هر کدام از خوشه‌ها را با به دست آوردن میانگین هر خوشه محاسبه می‌کنیم.

مرکزیت خوشه‌ی ۱ – CLUSTER 1: (1.5, 2.75)

مرکزیت خوشه‌ی ۲ – CLUSTER 2: (4.5, 2.5)

مرحله‌ی ۵: تا وقتی که مرکزیت‌ها تغییری نکرده باشند

مرکزیت‌های جدیدِ به‌هنگام‌شده، همانند مرحله‌ی قبل هستند، پس الگوریتم، **همگرا (converge)** شده است و می‌توانیم همین‌جا الگوریتم را متوقف کنیم. خوشه‌های نهایی به صورت زیر هستند:

```
CLUSTER 1
(1, 2)
(1, 4)
(2, 2)
(2, 3)
```

```
CLUSTER 2
(4, 2)
(4, 4)
(5, 1)
(5, 3)
```

هنگامی که مرکزیت‌ها تغییری نکنند، توقف خواهیم کرد. مانند این است که بگوییم، هیچ نقطه‌ای از یک خوشه، به خوشه‌ی دیگر منتقل نشده است. منظورمان از همگرا شدن (converge) همین است.

در هنگام اجرای این الگوریتم، مرکزیت‌ها از مکانِ اولیه‌ی خود به سمت یک مکان پایانی حرکت می‌کنند. اکثر این تغییر در چند دور اول رخ می‌دهد و معمولاً مرکزیت‌ها در دورهای آخر، تغییرات کمی دارند.

این بدان معنی است که الگوریتم k-means خوشه‌های خوبی را در ابتدا می‌سازد و تکرارهای بعدی، فقط اصلاحات جزئی هستند.

به خاطر این رفتارِ الگوریتم، می‌توانیم تا حد زیادی سرعت اجرای الگوریتم را با ریلکس (relax) کردنِ شرایطِ خاتمه، کم کنیم. یعنی به جای این‌که بگوییم «هیچ نقطه‌ای از یک خوشه به خوشه‌ی دیگر در تکرار بعدی تغییر نکند»، می‌توانیم بگوییم «کمتر از ۱ درصد از نقاط از یک خوشه به خوشه‌ی دیگر در تکرار بعدی تغییر کنند». و البته که این روشِ مرسومی است!

برای شما متخصصان علم کامپیوتر

الگوریتم k-means یک نمونه از روش بیشینه‌سازی انتظار (Expectation Maximization) یا همان EM است. این روش، یک روش تکراری (iterative) است که بین دو فاز جابه‌جا می‌شود. ابتدا با یک تخمین اولیه برخی از پارامترها شروع می‌کنیم. در

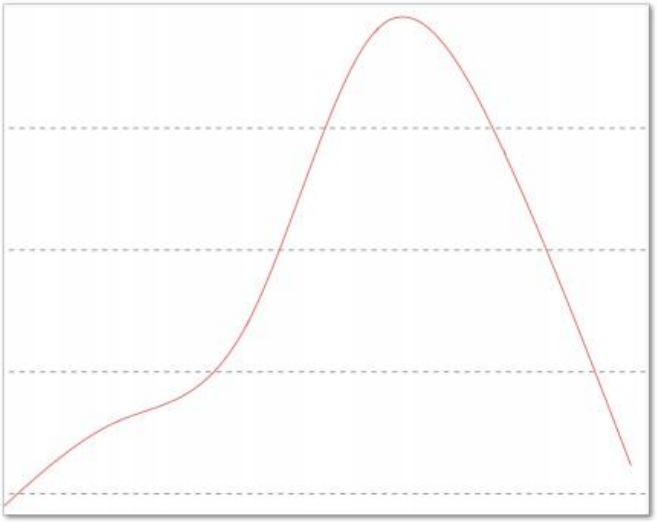
مورد k-means، این‌کار با تخمین مرکزیت‌ها شروع می‌شود. در فاز انتظار (E)، ما از این تخمین استفاده می‌کنیم تا نقاط را به خوشه‌هایی که انتظارشان را داریم، تخصیص دهیم. در فاز بیشینه‌سازی (M)، ما از این مقادیر در فاز انتظار استفاده می‌کنیم تا بتوانیم به وسیله‌ی آن‌ها مرکزیت‌ها را تخمین بزنیم. اگر می‌خواهید بیشتر در مورد روش EM بدانید، ویکی‌پدیا، https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm محل خوبی برای شروع است.

تپه‌نوردی (Hill Climbing)

اگر اجازه دهید، می‌خواهم به صورت خلاصه، خوشه‌بندی k-means و توضیحاتش را متوقف کرده و به الگوریتم‌های تپه‌نوردی بپردازم. فرض کنید هدفِ ما این باشد که بخواهیم به قله‌ی کوهی برسیم، و این‌کار را با استفاده از الگوریتم زیر انجام دهیم:

از یک نقطه‌ی تصادفی در کوهستان شروع می‌کنیم
خط زیر را تکرار می‌کنیم
به جهتی قدم می‌گذاریم که باعث بیشترین شدن ارتفاعِ ما از سطح زمین بشود
تا هنگامی که هیچ جهتی موجود نباشد که با رفتن به آن جهت، به ارتفاع بیشتری برسیم (یعنی قله باشد)

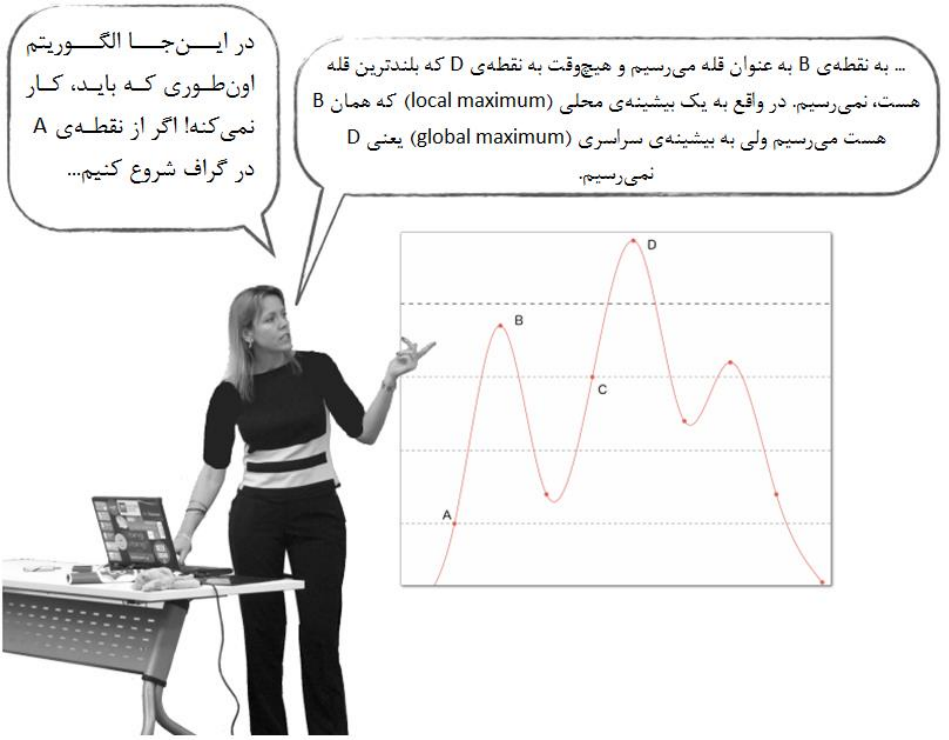
که به نظر الگوریتمی منطقی می‌رسد.
این الگوریتم را با توجه به شکلِ کوه مانند زیر در نظر بگیرید:



همان‌طور که می بینید، مهم نیست که از کجا شروع کنیم، یعنی از هر جا که شروع کنیم، با استفاده از الگوریتم بالا، می‌توانیم به قله برسیم.

و اگر به صورت گراف به این قضیه نگاه کنیم، بدون توجه به این‌که از کجای گراف شروع کنیم، به **بیشترین مقدار (قله)** می‌رسیم.

حالا فرض کنید، با همین الگوریتم بخواهیم گرافِ زیر را پیمایش کنیم:



پس این نسخه‌ی ساده از الگوریتمِ تپه‌نوردی، تضمین نمی‌کند که ما به نقطه‌ی بهینه برسیم.

الگوریتم خوشه‌بندیِ k-means هم شبیه به همین است. تضمینی وجود ندارد که الگوریتم بتواند تقسیم‌بندی‌های بهینه را برای ساخت خوشه‌ها انجام دهد. ولی چرا؟

به خاطر این‌که در شروعِ الگوریتم، ما نقاط شروع را به صورت تصادفی انتخاب کردیم، مانند این‌که در شکلِ بالا، نقطه‌ی A را به عنوان نقطه‌ی شروع در نظر بگیریم. پس، بر اساس این

نقطه‌ی شروع، ما سعی می‌کنیم خوشه‌ها را به صورت محلی، بهینه کنیم (مانند نقطه‌ی B در شکل بالا).

خوشه‌های پایانی، بسیار بستگی به نقاط شروع اولیه دارند. با وجود این، الگوریتم k-means، خوشه‌های خوبی را تولید می‌کند.

معیار SSE و پراکندگی

برای بررسی و مشخص کردنِ کیفیتِ مجموعه‌ای از خوشه‌ها، می‌توانیم از مجموع مربعات خطا (SSE) استفاده کنیم. به این خطا، پراکندگی (scatter) هم می‌گویند و به این صورت محاسبه می‌شود: برای هر کدام از نقطه‌ها (یعنی نمونه‌ها) ما فاصله‌ی آن نقطه را نسبت به مرکزیتِ (centroid) خوشه محاسبه کرده و به توان ۲ می‌رسانیم (مربع می‌کنیم)، و تمامی این فاصله‌ها را (برای هر نقطه یک فاصله) که به توان ۲ رسیده است، با هم جمع می‌کنیم. به صورت رسمی‌تر:

$$SSE = \sum_{i=1}^{k} \sum_{x \in c_i} dist(c_i, x)^2$$

بیایید این فرمول را واکاوی کنیم. در اولین علامت مجموع ($\sum$) بر روی خوشه‌ها تکرار را انجام می‌دهیم (یعنی به ازای هر خوشه). پس در ابتدا **i** برابر **خوشه‌ی ۱** است، بعد از آن به **خوشه‌ی ۲** می‌رسد و همین‌طور تا **خوشه‌ی k**. علامت مجموع ($\sum$) بعدی، تکرار را بر روی هر کدام از نقاطِ آن خوشه انجام می‌دهد – مانند این است که بگوییم، برای هر نقطه‌ی x در خوشه‌ی i. در این‌جا dist، می‌تواند هر کدام از معیارهای فاصله باشد (مثلاً منهتن یا اقلیدسی). پس ما فاصله‌ی هر نقطه‌ی x را نسبت به مرکزیت خوشه‌ی i (یعنی c_i) به دست می‌آوریم و آن‌را به توان ۲ رسانده و حاصلِ این‌کار را با مجموعِ کل، جمع می‌کنیم.
فرض کنید الگوریتم k-means را دومرتبه بر روی یک مجموعه‌ی داده، اجرا کنیم و برای هر اجرا، یک سری نقاطِ اولیه‌ی متفاوت انتخاب کنیم. آیا مجموعِ خوشه‌هایی که در اجرای اول ساخته شد، بهتر از مجموع خوشه‌هایی بود که در اجرای دوم ساخته شد؟ برای پاسخ به این سوال، ما مقدار SSE را برای هر دو اجرا محاسبه می‌کنیم. مجموعه‌ای که کمترین میزان SSE را داشته باشد، خوشه‌بندیِ بهتری ارائه کرده است.

وقت کد زدنه!

در زیر کد k-means پایه رو مشاهده می‌کنید

```
import math
import random


"""
Implementation of the K-means algorithm
for the book A Programmer's Guide to Data Mining"
http://www.guidetodatamining.com

"""

def getMedian(alist):
    """get median of list"""
    tmp = list(alist)
    tmp.sort()
    alen = len(tmp)
    if (alen % 2) == 1:
        return tmp[alen // 2]
    else:
        return (tmp[alen // 2] + tmp[(alen // 2) - 1]) /
 2


def normalizeColumn(column):
    """normalize the values of a column using Modified S
tandard Score
    that is (each value - median) / (absolute standard d
eviation)"""
    median = getMedian(column)
    asd = sum([abs(x - median) for x in column]) / len(c
olumn)
    result = [(x - median) / asd for x in column]
    return result


class kClusterer:
    """ Implementation of kMeans Clustering
    This clusterer assumes that the first column of the
data is a label
    not used in the clustering. The other columns contai
n numeric data
    """

    def __init__(self, filename, k):
        """ k is the number of clusters to make
        This init method:
```

```
            1. reads the data from the file named filenam
e
            2. stores that data by column in self.data
            3. normalizes the data using Modified Standar
d Score
            4. randomly selects the initial centroids
            5. assigns points to clusters associated with
 those centroids
        """
        file = open(filename)
        self.data = {}
        self.k = k
        self.counter = 0
        self.iterationNumber = 0
        # used to keep track of % of points that change
cluster membership
        # in an iteration
        self.pointsChanged = 0
        # Sum of Squared Error
        self.sse = 0
        #
        # read data from file
        #
        lines = file.readlines()
        file.close()
        header = lines[0].split(',')
        self.cols = len(header)
        self.data = [[] for i in range(len(header))]
        # we are storing the data by column.
        # For example, self.data[0] is the data from col
umn 0.
        # self.data[0][10] is the column 0 value of item
 10.
        for line in lines[1:]:
            cells = line.split(',')
            toggle = 0
            for cell in range(self.cols):
                if toggle == 0:
                    self.data[cell].append(cells[cell])
                    toggle = 1
                else:
                    self.data[cell].append(float(cells[c
ell]))

        self.datasize = len(self.data[1])
```

```
        self.memberOf = [-1 for x in range(len(self.data
[1]))]
        #
        # now normalize number columns
        #
        for i in range(1, self.cols):
                self.data[i] = normalizeColumn(self.data
[i])

        # select random centroids from existing points
        random.seed()
        self.centroids = [[self.data[i][r]  for i in ran
ge(1, len(self.data))]
                          for r in random.sample(range(
len(self.data[0])),
                                                 self.k)
]
        self.assignPointsToCluster()



    def updateCentroids(self):
        """Using the points in the clusters, determine t
he centroid
        (mean point) of each cluster"""
        members = [self.memberOf.count(i) for i in range
(len(self.centroids))]
        self.centroids = [[sum([self.data[k][i]
                                for i in range(len(self.
data[0]))
                                if self.memberOf[i] == c
entroid])/members[centroid]
                          for k in range(1, len(self.da
ta))]
                         for centroid in range(len(self
.centroids))]



    def assignPointToCluster(self, i):
        """ assign point to cluster based on distance fr
om centroids"""
        min = 999999
        clusterNum = -1
        for centroid in range(self.k):
            dist = self.euclideanDistance(i, centroid)
```

```
            if dist < min:
                min = dist
                clusterNum = centroid
        # here is where I will keep track of changing po
ints
        if clusterNum != self.memberOf[i]:
            self.pointsChanged += 1
        # add square of distance to running sum of squar
ed error
        self.sse += min**2
        return clusterNum

    def assignPointsToCluster(self):
        """ assign each data point to a cluster"""
        self.pointsChanged = 0
        self.sse = 0
        self.memberOf = [self.assignPointToCluster(i)
                         for i in range(len(self.data[1]
))]



    def euclideanDistance(self, i, j):
        """ compute distance of point i from centroid j"
""
        sumSquares = 0
        for k in range(1, self.cols):
            sumSquares += (self.data[k][i] - self.centro
ids[j][k-1])**2
        return math.sqrt(sumSquares)

    def kCluster(self):
        """the method that actually performs the cluster
ing
        As you can see this method repeatedly
            updates the centroids by computing the mean
point of each cluster
            re-assign the points to clusters based on th
ese new centroids
        until the number of points that change cluster m
embership is less than 1%.
        """
        done = False

        while not done:
            self.iterationNumber += 1
```

```
            self.updateCentroids()
            self.assignPointsToCluster()
            #
            # we are done if fewer than 1% of the points
 change clusters
            #
            if float(self.pointsChanged) / len(self.memb
erOf) <  0.01:
                done = True
        print("Final SSE: %f" % self.sse)

    def showMembers(self):
        """Display the results"""
        for centroid in range(len(self.centroids)):
             print ("\n\nClass %i\n========" % centroid)
             for name in [self.data[0][i]  for i in rang
e(len(self.data[0]))
                          if self.memberOf[i] == centroi
d]:
                 print (name)

##
## RUN THE K-MEANS CLUSTERER ON THE DOG DATA USING K = 3
###
# change the path in the following to match where dogs.c
sv is on your machine
km = kClusterer('../../data/dogs.csv', 3)
km.kCluster()
km.showMembers()
```

مانند کدِ خوشه‌بندیِ سلسله‌مراتبی، ما داده‌ها را به صورت ستونی ذخیره می‌کنیم. همان مثالِ نژاد سگ‌ها را تصور کنید. اگر ما داده‌ها را در یک صفحه‌ی گسترده (مانند اکسل) نمایش دهیم، چیزی شبیه به شکل زیر خواهد شد (ستون‌های قد (height) و وزن (weight) نرمال‌شده هستند. ستون breed هم نژاد سگ‌ها را نشان می‌دهد):

breed	height	weight
Border Collie	0	-0.1455
Boston Terrier	-0.7213	-0.873
Brittany Spaniel	-0.3607	-0.4365
Bullmastiff	1.2623	2.03704
German Shepherd	0.9016	0.81481
...	...	...

اگر بخواهیم داده‌ها را به پایتون انتقال دهیم، احتمالاً لیستی شبیه به لیست زیر می‌سازیم:

```
data = [ data for the Border Collie,
         data for the Boston Terrier,
         ... ]
```

و برای مشخص کردنِ فرمت داده‌ها به صورت کامل، مانند زیر عمل می‌کنیم:

```
data = [ ['Border Collie',  0, -0.1455],
         ['Boston Terrier', -0.7213, -0.873],
         ... ]
```

پس در این‌جا، داده‌ها را به صورت سطری ذخیره کردیم. این نحوه‌ی نمایش شهودی‌تر به نظر می‌رسد، ما هم از همین نحوه‌ی نمایش در سراسر این کتاب استفاده کرده‌ایم. البته که می‌توانیم داده‌ها را به **صورت اول ستون (column first)** نیز ذخیره کنیم:

```
data = [ column 1 data,
         column 2 data,
         column 3 data ]
```

پس برای مثال نژاد سگ‌ها:

```
data = [ ['Border Collie', 'Boston Terrier', 'Brittany Spaniel', ...],
         [ 0, -0.7213, -0.3607, ...],
         [-0.1455, -0.7213, -0.4365, ...],
         ... ]
```

این همان‌کاری است که برای خوشه‌بندیِ سلسله‌مراتبی انجام دادیم و این‌جا برای k-means هم همین‌کار را تکرار کردیم. مزیت این روش این‌است که پیاده‌سازیِ توابعِ ریاضی را ساده‌تر می‌کند. این سادگی را می‌توان در دو تابع اولِ در کدِ بالا مشاهده کرد که دو تابع getMedian و normalizeColumn هستند. چون در این حالت، داده‌ها را به صورت ستونی ذخیره کردیم، این توابع، یک سری لیست ساده را به عنوان ورودی دریافت می‌کنند.

```
>>> normalizeColumn([8, 6, 4, 2])
[1.5, 0.5, -0.5, -1.5]
```

تابعِ constructor (یعنی همان تابع __init__)، نام فایل (filename) و مقدار k را به عنوان ورودی گرفته، داده‌ها را از فایل می‌خواند و این داده‌ها را به صورت ستونی ذخیره می‌کند. این تابع، داده‌ها را توسط تابع normalizeColumn نرمال‌سازی می‌کند. تابع normalizeColumn در واقع پیاده‌سازی امتیاز استاندارد اصلاح‌شده (modified standard score) است. در آخر، k آیتم از این داده‌ها را انتخاب می‌کند، و به عنوان مرکزیت‌های اولیه قرار داده و هر نقطه را به خوشه‌ی مربوط، براساس فاصله‌ی آن نقطه تا

مرکزیت اولیه، تخصیص می‌دهد. این تخصیص توسط تابع assignPointsToCluster انجام می‌شود.

تابعِ kCluster در اصل عملیات خوشه‌بندی را انجام می‌دهد. این تابع، خوشه‌بندی را با فراخوانیِ چندباره‌ی تابع updateCentroids انجام می‌دهد. تابع updateCentroids میانگینِ هر خوشه را محاسبه کرده و تابع assignPointsToCluster را تا زمانی که کمتر از یک درصدِ داده‌ها تغییر کنند، فراخوانی می‌کند. تابع showMembers نتایج را نمایش می‌دهد.

با اجرای کد بر روی داده‌های نژاد سگ‌ها، نتایج زیر حاصل می‌شود:

```
Final SSE: 5.243159


Class 0
========
Bullmastiff
Great Dane

Class 1
========
Boston Terrier
Chihuahua
Yorkshire Terrier

Class 2
========
Border Collie
Brittany Spaniel
German Shepherd
Golden Retriever
Portuguese Water Dog
Standard Poodle
```

خب! برای این مجموعه‌ی داده‌ی کوچک، الگوریتمِ خوشه‌بندیِ ما خیلی خوب عمل کرد.

شما امتحان کنید

الگوریتم خوشه‌بندیِ k-means بر روی داده‌های غلات صبحانه با k=4 چگونه عمل می‌کند؟
آیا sweet cerealها با هم به یک خوشه تعلق می‌گیرند (Cap’n’Crunch, Cocoa Puffs, Froot Loops, Lucky Charms)؟
آیا bran cerealها، با هم در یک خوشه قرار می‌گیرند (100% Bran, All-Bran, All-Bran with Extra Fiber, Bran Chex)؟
محصول Cheerios با چه چیزهایی هم خوشه می‌شود؟
همچنین الگوریتم خوشه‌بندی را بر روی داده‌های مصرفِ اتومبیل بر اساس مایل (auto mpg) با مقادیر مختلف برای k=8 انجام دهید.
آیا این الگوریتمِ خوشه‌بندی، انتظارات شما را برای خوشه‌بندیِ اتومبیل‌ها برآورده می‌کند؟

شما امتحان کنید – راه حل
الگوریتم خوشه‌بندیِ k-means بر روی داده‌های غلات صبحانه با k=4 چگونه عمل می‌کند؟
نتایجی که شما به دست آورده‌اید، ممکن است با نتایج من فرق کند، ولی چیزی که من نتیجه گرفته‌ام به صورت زیر است.

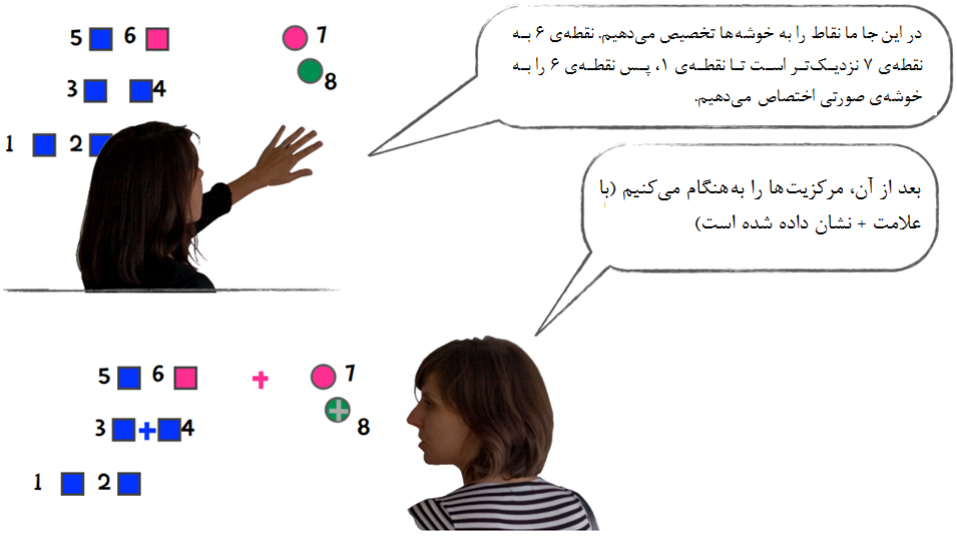
آیا sweet cerealها با هم به یک خوشه تعلق می‌گیرند (Cap’n’Crunch, Cocoa Puffs, Froot Loops, Lucky Charms)؟
بله، تمامی این‌ها (به همراه Count Chocula، Fruity Pebbles و بقیه) در یک خوشه قرار گرفته‌اند.

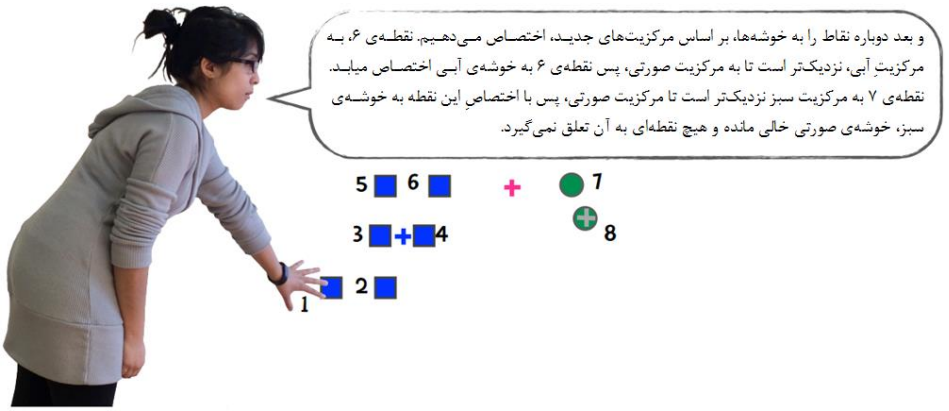
آیا bran cerealها، با هم در یک خوشه قرار می‌گیرند (100% Bran, All-Bran, All-Bran with Extra Fiber, Bran Chex)؟
دوباره، بله! همچنین در این خوشه Raisin Bran و Fruitful Bran هم قرار داشتند.

محصول Cheerios با چه چیزهایی هم خوشه می‌شود؟
این محصول معمولاً با Special K هم‌خوشه می‌شود.

همچنین الگوریتم خوشه‌بندی را بر روی داده‌های مصرف اتومبیل بر اساس مایل (auto mpg) با مقادیر مختلف برای k=8 انجام دهید.
آیا این الگوریتمِ خوشه‌بندی، انتظارات شما را برای گروه‌بندی اتومبیل‌ها برآورده می‌کند؟
الگوریتمِ خوشه‌بندی، به نظر منطقی و خوب آمد ولی در موارد نادری احتمالاً یک یا چند خوشه، خالی هستند.

این مثال از Tolga Can را ببینید:
http://www.ceng.metu.edu.tr/~tcan/ceng465_f1314/Schedule/KMeansEmpty.html





در مجموع، اگر تعداد k را برای الگوریتمِ k-means برابر عددی خاص قرار دهیم، به این معنا نیست که الگوریتم خوشه‌بندیِ k-mean، دقیقاً همین تعداد خوشه‌ی غیر خالی را برای ما می‌سازد. البته این قضیه می‌تواند چیز خوبی باشد. به داده‌های بالا نگاه کنید، به نظر می‌رسد واقعاً داده‌ها در دو گروه قرار دارند و تلاشِ ما برای سه گروه کردن آن‌ها، بی‌ثمر بوده است. حالا فرض کنید ۱۰۰۰ نمونه داریم که می‌خواهیم به ۱۰ گروه تقسیمشان کنیم و هنگامی که الگوریتمِ خوشه‌بندی را بر روی این داده‌ها اجرا می‌کنیم، دو گروه از این ده گروه، خالی می‌مانند. این خالی ماندنِ خوشه‌ها، شاید بتواند چیزی در مورد ساختار اصلی داده‌ها به ما بگوید. شاید داده‌ها به صورت ذاتی به ۱۰ گروه تقسیم‌بندی نمی‌شوند و ما می‌بایستی که گروه‌بندی‌های دیگری را در نظر بگیریم (مثلاً تلاش کنیم که داده‌ها را به جای ۱۰ گروه، در ۸ گروه، خوشه‌بندی کنیم).

به بیانی دیگر، برخی اوقات هنگامی که ما ۱۰ خوشه را برای الگوریتم در نظر می‌گیریم، منظورمان ۱۰ خوشه‌ی غیر خالی است. اگر واقعاً به دنبال این قضیه هستیم، نیازداریم الگوریتم را تغییر دهیم تا بتواند خوشه‌های خالی را پیدا کند. هنگامی که یک خوشه‌ی خالی پیدا شد، الگوریتم، مرکزیتِ (centroid) این خوشه را جابه‌جا می‌کند. یک حالت این است که این مرکزیت را به دورترین نمونه نسبت به مرکزیتِ متناظرِ خودش انتقال دهد. (در مثال بالا، هنگامی که ما خوشه‌ی صورتی را خالی میافتیم، مرکزیت خوشه‌ی صورتی را به نقطه‌ی ۱ می‌بردیم، چون ۱ دورترین نقطه از مرکزیت متناظر خودش است). برای همین، من فاصله را نسبت به حالت‌های زیر محاسبه کرده‌ام:

نقطه‌ی ۱ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۲ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۳ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۴ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۵ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۶ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۷ به مرکزیت خوشه‌ی خودش
نقطه‌ی ۸ به مرکزیت خوشه‌ی خودش

و نقطه‌ای را انتخاب کردم که دورترین نقطه به مرکز خوشه‌ی خودش باشد (که خوشه‌ی ۱ شد) و بعد خوشه‌ی خالی (صورتی) را به آن نقطه انتقال دادم.

به نظرتون بهتر نمی‌شه اگه الگوریتم k-mean سریع‌تر و با دقتِ بالاتری ارائه بشه؟
با تغییری نسبت به الگوریتم k-mean به الگوریتم جدیدی به اسم **k-mean++** می‌رسیم.
حتی اسمش هم جدیدتر، بهتر، سریع‌تر و با دقت بیشتری به نظر می‌رسه – یک k-mean توربوشارژ

k-means++

در قسمت قبلی، ما الگوریتم k-mean را به صورت اصلی و بدون تغییر، یعنی همان چیزی که در چند دهه قبل، توسعه یافته بود، پیاده‌سازی کردیم و آن را بر روی مجموعه داده‌های مختلف آزمایش کردیم. همان‌طور که دیدیم، پیاده‌سازیِ آن ساده بود و نتیجه‌ی خوبی هم می‌داد. این الگوریتم، هنوز هم پراستفاده‌ترین الگوریتم خوشه‌بندی در جهان است. البته که کاستی‌هایی دارد. یکی از ضعف‌های اساسیِ k-means همان اولین قدمی است که به صورت «تصادفی» تعداد k داده را به عنوان مرکزیت داده‌ها انتخاب می‌کند. همان‌طور که از تاکیدم بر روی کلمه‌ی «تصادفی» متوجه شدید، قسمت تصادفیِ ماجرا، مشکل‌ساز است. به خاطر این‌که در این انتخاب تصادفی است، گاهی اوقات مرکزیت‌های اولیه بسیار عالی هستند و به خوشه‌های تقریباً بهینه‌ای ختم می‌شوند. اما برخی اوقات، با انتخابِ تصادفی، این مرکزیت‌های اولیه ضعیف انتخاب می‌شوند و خوشه‌هایی با کیفیتِ پایین و غیر بهینه

می‌سازند. الگوریتم k-means++ این نقطه ضعف را با تغییر روش، برای انتخاب مرکزیت‌های اولیه برطرف کرده است. بقیه‌ی چیزها در مورد k-means، مانند قبل هستند.

روش k-means++ - انتخاب مرکزیت‌های اولیه

۱. در ابتدا، مجموعه‌ی مرکزیت‌های اولیه خالی است

۲. یک نقطه را به عنوان مرکزیت، به صورت تصادفی مانند قبل از میان داده‌ها انتخاب می‌کنیم

۳. تا هنگامی که k مرکزیتِ اولیه داریم

الف. فاصله‌ی D را برای هر نقطه (نمونه) نسبت به نزدیک‌ترین مرکزیت پیدا می‌کنیم. این فاصله را D(dp) برای هر نقطه، نام‌گذاری می‌کنیم

با یک احتمال متناسب با مقدار D(dp) یک نقطه‌ی تصادفی انتخاب می‌کنیم که مرکزیت جدید شود و این مرکزیت جدید را به مجموعه‌ی مرکزیت‌ها اضافه می‌کنیم

ج. این کار را تکرار می‌کنیم

حالا اجازه دهید عبارت «**با یک احتمال متناسب با مقدار** D(dp) یک نقطه تصادفی انتخاب می‌کنیم که مرکزیت جدید شود» را واکاوی کنیم و ببینیم که منظورمان از این قسمت چه بوده است. برای این‌کار، یک مثال ساده می‌آورم. فرض کنید ما وسط این فرآیند باشیم. مثلاً ما تا الان در الگوریتم دو مرکزیت (centroid) اولیه را انتخاب کرده‌ایم و می‌خواهیم بعدی را انتخاب کنیم. پس ما در در مرحله‌ی ۳ – الف از الگوریتم بالا هستیم. فرض کنید ما ۵ مرکزیت دیگر داریم که می‌خواهیم آن‌ها را انتخاب کنیم در حالی که فاصله‌های نقاط نسبت به دو مرکزیتِ انتخاب شده به صورت زیر است:

	Dc1	Dc2
dp1	5	7
dp2	9	8
dp3	2	5
dp4	3	7
dp5	5	2

منظور از Dc1، فاصله تا مرکزیت ۱ و منظور از Dc2، فاصله تا مرکزیت ۲ است. عبارت dp1 به معنای نقطه‌ی ۱ است.

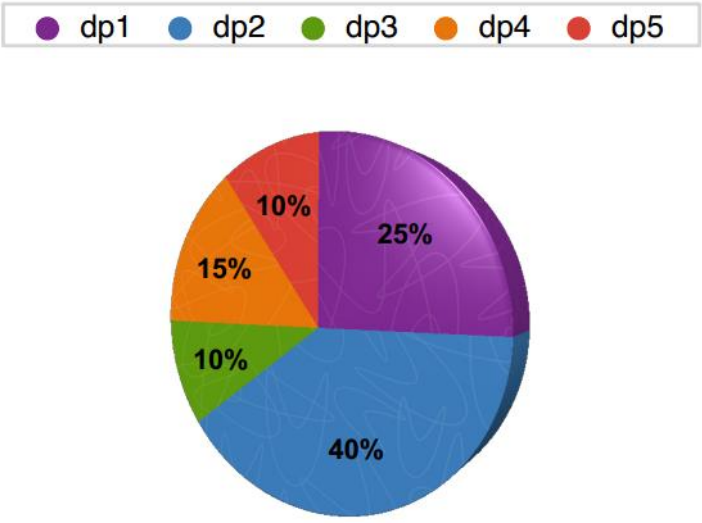
در مرحله‌ی ۳ – الف گفته شده که بایستی نزدیک‌ترین فاصله را انتخاب کنیم. پس به صورت زیر عمل می‌کنیم:

	closest
dp1	5
dp2	8
dp3	2
dp4	3
dp5	2

حالا می‌خواهیم اعداد را طوری تغییر دهیم که جمع آن‌ها ۱ شود (من این را وزن (weight) می‌نامم). برای این کار ما اعدادِ اولیه را با هم جمع می‌کنیم. در این مورد، جمع برابر ۲۰ می‌شود، و حالا هر کدام از اعداد را بر ۲۰ تقسیم می‌کنیم. نتیجه به صورت زیر است:

	weight
dp1	0.25
dp2	0.40
dp3	0.10
dp4	0.15
dp5	0.10
sum	1.00

این طور تصور کنید که این‌داده‌ها مثل یک چرخ، مانند شکل زیر هستند:



ما این چرخ را می‌چرخانیم، می‌بینیم که کجا می‌ایستد و همان‌جا را به عنوان مرکزیت جدید انتخاب می‌کنیم. پس متوجه شدید که هر چه قدر عددِ نوشته شده برای آن نقطه،

بزرگتر باشد، احتمال انتخاب آن هم بیشتر است. منظور ما هم از «با یک **احتمال متناسب با مقدار D(dp)** یک نقطه تصادفی انتخاب می‌کنیم که مرکزیت جدید شود» همین بود.
حالا اجازه دهید همین ایده را در پایتون پیاده‌سازی کنیم. فرض کنید مجموعه‌ای از چندتایی‌ها (tuples) همراه با وزنشان داریم:

```
data = [("dp1", 0.25), ("dp2", 0.4), ("dp3", 0.1), ("dp4", 0.15), ("dp5", 0.1)]
```

تابع roulette در زیر، یک نقطه را با توجه به **احتمالِ متناسب با وزنِ** آن، انتخاب می‌کند:

```
import random
random.seed()
def roulette(datalist):
      i = 0
      soFar = datalist[0][1]
      ball = random.random()
      while soFar < ball:
            i += 1
            soFar += datalist[i][1]
      return datalist[i][0]
```

اگر این تابع، با نسبت‌هایی که داریم انتخاب را انجام دهد، به این معنی است که مثلاً اگر ۱۰۰ مرتبه انتخاب را انجام دهیم، ۲۵ مرتبه نقطه‌ی dp1، 40 مرتبه نقطه‌ی dp2، 10 مرتبه نقطه‌ی dp3، 15 مرتبه نقطه‌ی dp4 و ۱۰ مرتبه نقطه‌ی dp5 انتخاب می‌شود. بیایید ببینیم آیا این فرضیه درست است یا خیر؟

```
import collections
results = collections.defaultdict(int)
for i in range(100):
    results[roulette(data)] += 1
print results
```

```
{'dp5': 11, 'dp4': 15, 'dp3': 10, 'dp2': 38, 'dp1': 26}
```

خب! تابع ما، نقاط را به صورت تقریبی با نسبتِ درستی انتخاب کرد.

پس به طور خلاصه ایده‌ی اصلی در k-means++ به این صورت است که، با این‌که نقاطِ اولیه را به صورتِ تصادفی انتخاب می‌کنیم، ترجیح می‌دهیم مرکزیت‌ها از هم فاصله داشته باشند.

کد بزنید

آیا می‌توانید الگوریتم k-means++ را در پایتون پیاده‌سازی کنید؟

دوباره، تنها تفاوت پیاده‌سازی قبلی ما در روش k-means و این روش (k-means++) در انتخاب مرکزیت‌های اولیه است. یک کپی از کد اصلی k-means بگیرید و آن را تغییر دهید. کد اصلیِ ما که مرکزیت‌های اولیه را می‌ساخت در خط زیر بود:

```
self.centroids = [[self.data[i][r] for i in range(1, len(self.data))]
for r in random.sample(range(len(self.data[0])),
self.k)]
```

اجازه دهید این قسمت را به صورت زیر تغییر دهیم:

```
self.selectInitialCentroids()
```

وظیفه‌ی شما این است که این تابع را بنویسید!

خوش بگذره!

کد بزنید – راه حل

این نسخه‌ای است که من برای تابع selectInialCentroids پیاده‌سازی کرده‌ام:

```
def distanceToClosestCentroid(self, point, centroidLis
t):
    result = self.eDistance(point, centroidList[0])
    for centroid in centroidList[1:]:
        distance = self.eDistance(point, centroid)
        if distance < result:
            result = distance
    return result


def selectInitialCentroids(self):
    """implement the k-means++ method of selecting
    the set of initial centroids"""
    centroids = []
    total = 0
    # first step is to select a random first centroid
    current = random.choice(range(len(self.data[0])))
    centroids.append(current)
    # loop to select the rest of the centroids, one at
 a time
    for i in range(0, self.k - 1):
```

```
        # for every point in the data find its distanc
e to
        # the closest centroid
        weights = [self.distanceToClosestCentroid(x, c
entroids)
                   for x in range(len(self.data[0]))]
        total = sum(weights)
        # instead of raw distances, convert so sum of
weight = 1
        weights = [x / total for x in weights]
        #
        # now roll virtual die
        num = random.random()
        total = 0
        x = -1
        # the roulette wheel simulation
        while total < num:
            x += 1
            total += weights[x]
        centroids.append(x)
    self.centroids = [[self.data[i][r]  for i in range
(1, len(self.data))]
                      for r in centroids]
```

خلاصه

خوشه‌بندی تماماً در مورد کشف کردن است. با این وجود، مثال‌های ساده‌ای که در این فصل زدیم، ممکن است این ایده‌ی اصلی یعنی همان «کشف کردن» را پنهان کرده باشد. با این همه، ما بدون کمکِ کامپیوتر هم می‌دانیم که چگونه محصولاتِ غلاتِ صبحانه را خوشه‌بندی کنیم – مثلاً طبیعتاً محصولاتِ sugary cereals و healthy cereals با هم هستند. همچنین می‌دانیم که چگونه بایستی مدل‌های مختلفِ اتومبیل را خوشه‌بندی کنیم – مثلاً اتومبیل Ford F150 به دسته‌بندیِ کامیون‌ها (truck) تعلق می‌گیرد و Mazda Miata در دسته‌ی اتومبیل‌های ورزشی (sports)، و اتومبیل Honda Civic در دسته‌ی مصرف سوختِ بهینه (fuel efficient) قرار می‌گیرد. ولی حالا فرض کنید بخواهیم مسئله‌ای را حل کنیم که در آن «کشف کردن» مهم و اولویت‌دار باشد.

هنگامی که یک سرچ در اینترنت انجام می‌دهیم در واقع با یک لیست بسیار بلندی از نتایج روبرو می‌شویم. برای مثال، هنگامی که سرچی در سایتِ گوگل بر روی کلمه‌ی « carbon sequestration» انجام دادم، بیش از ۲٫۸ میلیون نتیجه به دست آوردم. برخی از محققین مزیت خوشه‌بندیِ این نتایج را بررسی کرده‌اند. به جای این لیست بسیار بلند که در مورد کلمه‌ی carbon sequestration باشد، مجموعه‌هایی مانند « carbon sequestration in freshwater wetlands» و «carbon sequestration in forests» را نیز مشاهده می‌کنید.

تیم Josh Gotbaum یک مصاحبه‌ی گسترده با ۳۰۰۰ نفر انجام داد که از آن‌ها در مورد ارزش‌هایشان سوالاتی پرسید. هنگامی که این تیم، خوشه‌ها را بررسی کرد، به پنج گروه رسید و به هر کدام از آن‌ها یک نام و توضیح اختصاص داد:

۱. بازکننده فرصت برای دیگران
۲. تعامل‌کننده با اجتماع
۳. مستقل
۴. متوجه فامیل
۵. مدافع عدالت

بعد از آن، این تیم برای هر کدام از این گروه‌ها، کمپین‌های تبلیغاتی راه‌اندازی کردند.
از کتاب The Numerati از Stephen Baker

ما تا به حال، فقط دو الگوریتم خوشه‌بندی را یاد گرفتیم: خوشه‌بندی سلسله‌مراتبی و خوشه‌بندی k-means. در چه شرایطی باید از روش سلسله‌مراتبی و در چه شرایطی بایستی از روش k-mean استفاده کنیم؟

سوال خوبی بود!

مزیت روش k-means این است که این روش ساده‌تر است و سرعت اجرای بالاتری دارد. این روش، یک راه حل عالی برای کارهای عمومی است. همچنین این روش، راه حلی خوب برای این است که توسط آن بتوانید در مراحل اولیه، داده‌ها را به صورت اکتشافی بررسی کنید حتی اگر در نهایت به الگوریتمی دیگر مهاجرت کنید. با این‌حال، k-means، داده‌های پرت (outlier) را به خوبی شناسایی نمی‌کند. البته که می‌توانیم این نقطه ضعف را با شناسایی و حذف داده‌های پرت، رفع کنیم.

اوکی! گرفتم. در مورد خوشه‌بندی سلسله‌مراتبی چه؟

یکی از موارد استفاده‌ی الگوریتم خوشه‌بندی سلسله‌مراتبی، هنگامی است که بخواهیم یک رده‌بندی (taxonomy) یا سلسله‌مراتب (hierarchy) از داده‌ها بسازیم. این سلسله مراتبِ ساخته شده، ممکن است حاوی اطلاعاتِ بیشتری نسبت به خوشه‌بندی تخت (flat) باشد. البته که این روش، خیلی هم بهینه (باتوجه به سرعت اجرا و حافظه‌ی اشغال‌شده) نیست.

عالیه!

شاید بهتر باشه که آن‌ها را بر روی یک مجموعه داده‌ی جدید، آزمایش کنم.

انرون (Enron)

ممکن است Enron و Enron Scandal را به خاطر بیاورید. در زمان اوج خودش، Enron یک شرکت انرژیِ بسیار بزرگ با درآمد بیش‌از ۱۰۰ میلیارد دلار بود. این شرکت در آن زمان، ۲۰۰۰۰ کارمند داشت (درآمد ماکروسافت بعد از آن فقط ۲۲ میلیارد دلار بود). Enron به دلیل سستیِ سیستماتیک و قطعی‌ها که شامل ایجادِ کمبودِ مصنوعیِ

انرژی و در نتیجه خاموشی برق کالیفرنیا شد، ورشکسته شد و عده‌ای راهی زندان شدند. برای مشاهده‌ی مستندی در مورد Enron می‌توانید فیلم زیر را در نتفیلیکس و آمازون پرایم جستجو کنید:

Enron: The Smartest Guys in the Room

ممکن است فکر کنید که «خب، این ماجراهای شرکت Enron جالبه، ولی چه ربطی به داده‌کاوی داره؟»

برای اطلاعات بیشتر در مورد پایگاه‌داده‌ی Enron، وب‌سایت ویکی‌پدیا را نگاه کنید:
https://en.wikipedia.org/wiki/Enron_Corpus
همچنین نگاهی به صفحه‌ی زیر بیندازید:
/https://www.cs.cmu.edu/~./enron
این مجموعه‌ی داده، منبع بزرگی برای محققان در حوزه‌های مختلف است.

حالا ما می‌خواهیم یک قسمت کوچک از مجموعه‌ی داده‌ی Enron را خوشه‌بندی کنیم. برای مجموعه‌ی آزمایشیِ کوچکمان، من این اطلاعات که «چه کسی به چه کسی ایمیل زده است» را واکشی کردم و در جدولی مانند جدول زیر برای نمایش قرار دادم (هر کدام از سطرها یا ستون‌ها اسم افراد هستند):

	Kay	Chris	Sara	Tana	Steven	Mark
Kay	0	53	37	6	0	12
Chris	53	0	1	0	2	0
Sara	37	1	0	1144	0	962
Tana	6	0	1144	0	0	1201
Steven	0	0	2	0	0	0
Mark	12	0	962	1201	0	0

در مجموعه‌ی داده‌ای که در وب‌سایتمان قرار دادیم، من این اطلاعات را برای ۹۰ نفر، استخراج کردم. فرض کنید می‌خواهیم افراد را طوری خوشه‌بندی کنیم که بتوانیم ارتباطات میان این افراد را کشف کنم.

تحلیل لینک (Link Analysis)

یک زیرحوزه در داده‌کاوی به اسم تحلیل لینک (link analysis) وجود دارد که کاملاً مناسبِ این دست از مسئل است (یعنی مسئله‌ای که در آن بخواهیم ارتباطات بین ورودی‌ها را ارزیابی کنیم) و الگوریتم‌های مخصوصی برای مسائل نیز به وجود آمده است.

> **شما امتحان کنید**
>
> **آیا می‌توانید خوشه‌بندیِ سلسله‌مراتبی را بر روی مجموعه‌ی ایمیل‌های Enrol انجام دهید؟**
>
> **می‌توانید داده‌ها را از وب‌سایت ما (guidetodatamining.com و chistio.ir) دانلود نمایید. البته احتمالاً بایستی برای تطبیق با این مسئله کدتان را کمی تغییر دهید.**
>
> **خوش بگذره!**

شما امتحان کنید – راه حل

در مجموعه‌ی داده‌ای که در وب‌سایت ما فراهم شده است، من این داده‌ها را برای ۹۰ نفر استخراج کرده‌ام.

ما افراد را بر اساس شباهتِ مکاتباتِ ایمیل خوشه‌بندی می‌کنیم. مثلاً اگر مکاتبات ایمیلیِ من، با Ann، Ben و Clara باشد، و اکثر مکاتبات شما هم با همین افراد رخ دهد، این خود می‌تواند نشانه‌ای باشد که ما در گروه یکسانی هستیم. ایده‌ی کلی چیزی شبیه به زیر است:

(بین) between ->	Ann	Ben	Clara	Dongmei	Emily	Frank
my emails (ایمیل‌های من)	127	25	119	5	1	6
your emails (ایمیل‌های شما)	172	35	123	7	3	5

به دلیل این‌که سطرهای ما شبیه به هم هستند، با هم در یک خوشه قرار می‌گیریم. البته مشکل هنگامی به وجود می‌آید که ما ستون‌هایمان را به این صورت اضافه می‌کنیم:

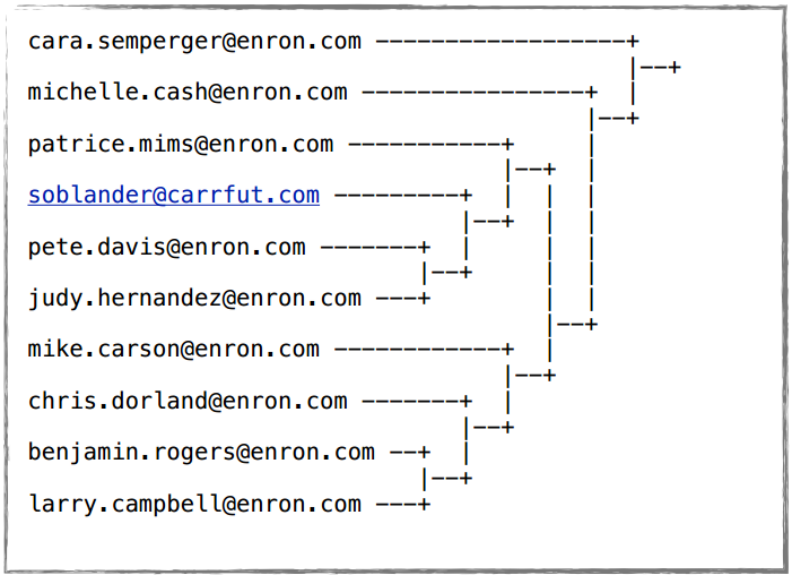
(بین) between ->	me	you	Ann	Ben	Clara	Dongmei	Emily	Frank
my emails (ایمیل‌های من)	2	190	127	25	119	5	1	6
your emails (ایمیل‌های شما)	190	3	172	35	123	7	3	5

با نگاه به به ستونِ «من (me)»، شما با من ۱۹۰ مرتبه مکاتبه داشته‌اید ولی من فقط ۲ مرتبه برای خودم ایمیل فرستاده‌ام. ستونِ «شما (you)» هم همین‌طور است. حالا اگر سطرهای خودمان را با هم مقایسه کنیم، خیلی به هم شبیه نمی‌شود. قبل از این‌که ستون‌های «me» و «you» را به جدول بالا اضافه کنم، فاصله‌ی اقلیدسی برابر ۴۶ بود و بعد از اضافه کردنِ آن‌ها، ۲۶۹ شد! برای این‌که این مشکل رفع شود، هنگامی که فاصله‌ی اقلیدسی را برای دو نفر محاسبه می‌کنم، ستون‌ها را برای آن دو نفر حذف می‌کنم. این نیازمند تغییر کمی در فرمول است:

```
def distance(self, i, j):
    #enron specific distance formula
    sumSquares = 0
    for k in range(1, self.cols):
```

```
        if (k != i) and (k != j) :
            sumSquares += (self.data[k][i] - self.data[k
][j])**2
    return math.sqrt(sumSquares)
```

در تصویر پایین، یک زیردرخت از نتایج را مشاهده می‌کنید:



همچنین الگوریتم k-means++ را بر روی این مجموعه‌ی داده با k=8 انجام دادم. در زیر چند گروهی که k-mean++ توانسته کشف کند را مشاهده می‌کنید:

```
Class 5
========
chris.germany@enron.com
scott.neal@enron.com
marie.heard@enron.com
leslie.hansen@enron.com
mike.carson@enron.com

Class 6
========
sara.shackleton@enron.com
mark.taylor@enron.com
susan.scott@enron.com

Class 7
========
tana.jones@enron.com
louise.kitchen@enron.com
mike.grigsby@enron.com
david.forster@enron.com
m.presto@enron.com
```

این نتایج شگفت‌انگیز است. مثلاً گروهِ ۵ (Class 5) شامل چند نفر از تجار است. Chris Germany و Leslie Hansen تاجر هستند. Scott Neal قائم‌مقام تجارت است. Marie Heard یک قاضی است. Mike Carson یکی از مدیران تجارت در شمال شرقی است. همچنین اعضای گروه ۷ نیز جالب توجه است. تمام چیزی که در مورد Tana Jones می‌دانم این است که یک مدیر و Louise Kitchen مدیر تجارت آنلاین بود. Mike Grigsby قائم‌مقام گاز طبیعی بود. David Forster هم یک قائم‌مقام در حوزه‌ی تجارت بود. Kevin Presto (m.presto) نیز یک قائم‌مقام و یک تاجرِ ارشد بود.


الگوهای زیاد در مجموعه داده‌های Enron موجود هستند که بسیاری از آن‌ها مخفی بوده و نیازمند کاوشند. آیا می‌توانید آن‌ها را پیدا کنید؟ مجموعه‌ی داده‌ی کامل را دانلود کرده و امتحانش کنید! (به من هم بگویید که چه چیزهایی پیدا کردید)


یا این‌که الگوریتم‌های خوشه‌بندی را بر روی مجموعه داده‌های دیگر آزمایش کنید. یادتان باشد که تمرین، باعث رشد می‌شود


و البته، تبریک می‌گویم. شما به آخر این فصل رسیدید!