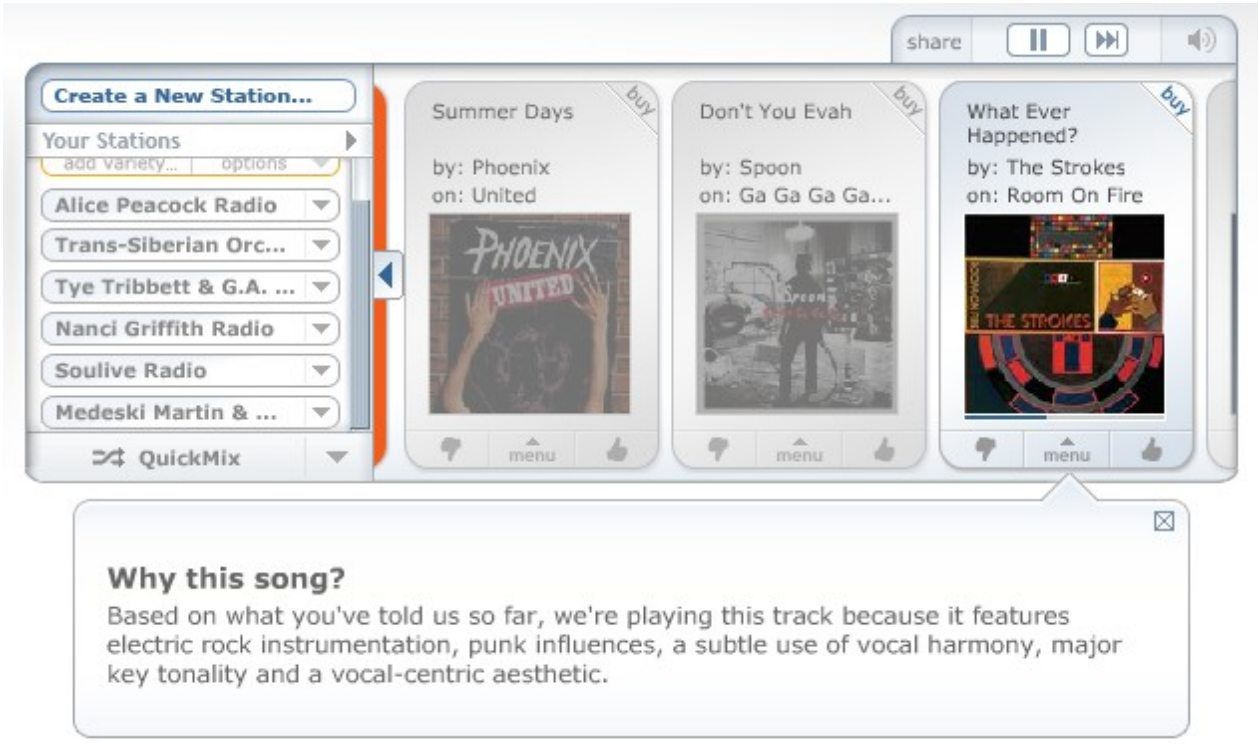
Chapter 4: Classification

In the previous chapters we talked about making recommendations by social filtering. In social filtering we harness the power of a community of people to help us make recommendations. You buy Wolfgang Amadeus Phoenix. We know that many of our customers who bought that album also bought Contra by Vampire Weekend. So we recommend that album to you. In this chapter we look at a different approach. Consider the streaming music site, Pandora. In Pandora, as many of you know, you can set up different streaming radio stations. You seed each station with an artist and Pandora will play music that is similar to that artist. I can create a station seeded with the band Phoenix. It then plays bands it thinks are similar to Phoenix—for example, it plays a tune by the Strokes. It doesn't do this with social filtering—because people who listened to Phoenix also listened to the Strokes. It plays The Strokes because the algorithm believes the Strokes are musically similar to Phoenix. In fact, we can ask Pandora why it played the Strokes:



It plays the Stokes tune *What Ever Happened* on the Phoenix station because "Based on what you told us so far, we're playing this track because it features electric rock instrumentation, punk influences, a subtle use of vocal harmony, major key tonality and a vocal-centric aesthetic."

Pandora bases its recommendation on what it calls *The Music Genome Project.* They hire professional musicians with a solid background in music theory as analysts that determine the features (they call them 'genes') of a song. These analysts are given over 150 hours of training. Once trained they spend an

average of 20-30 minutes analyzing a song to determine its genes/features. Many of these genes are technical—

- what are the beats per minute?
- is the song in a major or minor key?
- does it have block chords?
- does it have swinging sixteenths?
- how much twangy guitars does it have?

The analyst provides values for over 400 genes. Its a very labor intensive process and approximately 15,000 new songs are added per month.

> **NOTE: The Pandora algorithms are proprietary and I have no knowledge as to how they work. What follows is not a description of how Pandora works but rather an explanation of how to construct a similar system.**

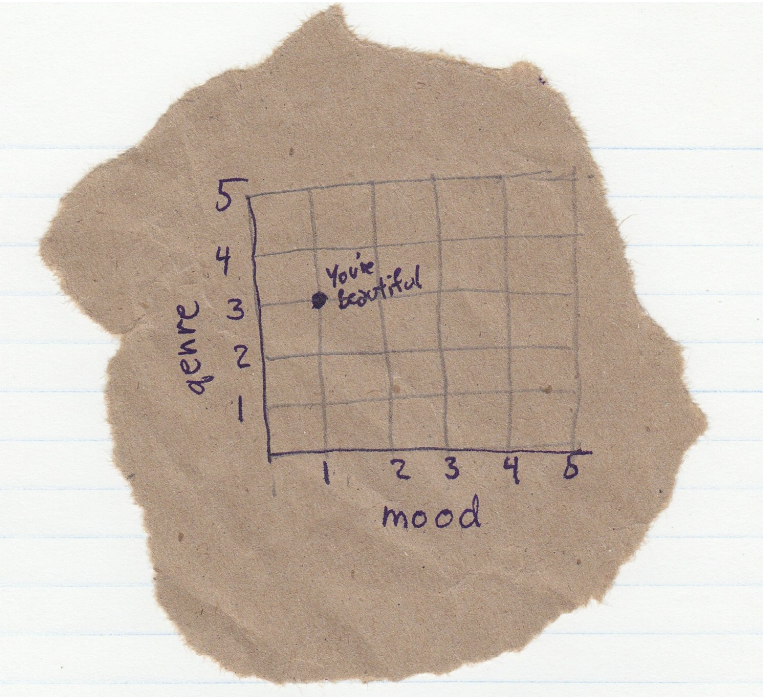
The importance of selecting appropriate values

Consider two genes that Pandora may have used: genre and mood. The values of these might look like this:

Genre	
1	Country
2	Jazz
3	Rock
4	Soul
5	Rap

Mood	
1	melancholy
2	joyful
3	passion
4	angry
5	unknown

Suppose I have a rock song that is melancholy—for example, the gag-inducing *You're Beautiful* by James Blunt. In 2D space inked quickly on a paper bag that would look as follows:

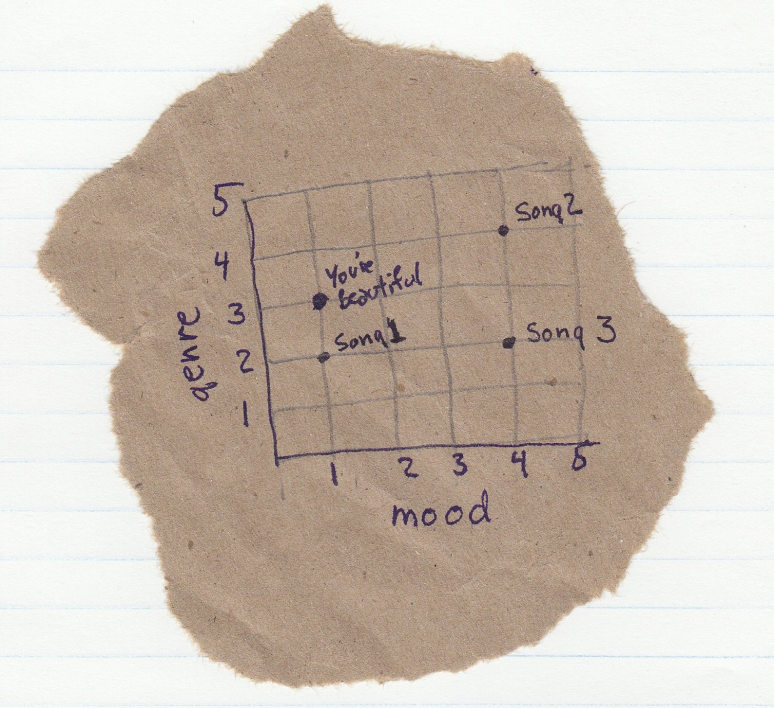


Let's say Tex just absolutely loves You're Beautiful and we would like to recommend a song to him.

Photo by Cayusa on flickr (CC license)

Let me populate our dataset with more songs. Song 1 is a jazz song that is melancholy; Song 2 is a soul

song that is angry and Song 3 is a jazz song that is angry. Which would you recommend to Tex?



I hope you see that we have a fatal flaw in our scheme. Let's take a look at the possible values for our variables again:

Genre	
1	Country
2	Jazz
3	Rock
4	Soul
5	Rap

Mood	
1	melancholy
2	joyful
3	passion
4	angry
5	unknown

If we are trying to use any distance metrics with this scheme we are saying that jazz is closer to rock than it is to soul. Or melancholy is closer to joyful than it is to angry. Even when we rearrange values

Genre	
1	Country
2	Jazz
3	Soul
4	Rap
5	Rock

Mood	
1	melancholy
2	angry
3	passion
4	joyful
5	unknown

The problem remains. No matter how we rearrange the values this won't work. This shows us that we have chosen our features poorly. We want features where the values fall along a meaningful scale. We can easily fix our genre feature by dividing it into 5 separate features—one for country, another for jazz, etc. They all can be on a 1-5 scale—how 'country' is the sound of this track—one means no hint of country to '5' means this is a solid country sound. Now the scale does mean something. If we are trying to find a song similar to one that rated a country value of '5', a song that rated a country of '4' would be closer than one of a '1'.

This is exactly how Pandora constructs its gene set. The values of most genes are on a scale of 1-5 with ½ integer increments. Genes are arranged into categories. For example, there is a musical qualities category which contains genes for *Blues Rock Qualities, Folk Rock Qualities*, and *Pop Rock Qualities* among others. Another category is instruments with genes such as *Accordian, Dirty Electric Guitar Riffs* and *Use of Dirty Sounding Organs.* Using these genes, each of which has a well-defined set of values from 1 to 5, Pandora represents each song as a vector of 400 numeric values (each song is a point in a 400 dimensional space). Now Pandora can make recommendations (that is, decide to play a song on a user-defined radio station) based on standard distance functions like those we already have seen.

Simple example

Unfortunately, I failed in finding a good dataset as an example of this approach so I created one. I have seven features each one ranging from 1-5 in ½ integer increments (I admit this isn't a very rational nor complete selection):

Amount of piano	1 indicates lack of piano; 5 indicates piano throughout and featured prominently
Amount of vocals	1 indicates lack of vocals; 5 indicates prominent vocals throughout song.
Driving beat	Combination of constant tempo, and how the drums & bass drive the beat.
Blues Influence	
Presence of dirty electric guitar	
Presence of backup vocals	
Rap Influence	

Now, using those features I rated ten tunes:

	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/ Just Got to Be	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

Thus, each tune is represented as a vector of numbers and we can use any distance function to compute the distance between tunes. For example,

Dr. Dog/Fate =	(2.5,	4,	3.5,	3,	5,	4,	1)	
Phoenix/ Lisztomania =	(2,	5,	5,	3,	2,	1,	1)	
Manhattan Distance =	0.5 +	1 +	1.5 +	0 +	3 +	3 +	0	= 9

Recall that our data for social filtering was of the format:

```
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}}
```

We can represent this current data in a similar way:

```
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/Just Got to Be": {"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, "blues": 2,
                                "guitar": 1, "backup vocals": 1, "rap": 1},
"Mike Posner":                 {"piano": 2.5, "vocals": 4, "beat": 4, "blues": 1,
                                "guitar": 1, "backup vocals": 1, "rap": 1},
"Black Eyed Peas/Rock That Body": {"piano": 2, "vocals": 5, "beat": 5, "blues": 1,
                                   "guitar": 2, "backup vocals": 2, "rap": 4},
"Lady Gaga/Alejandro":         {"piano": 1, "vocals": 5, "beat": 3, "blues": 2,
                                "guitar": 1, "backup vocals": 2, "rap": 1}}
```

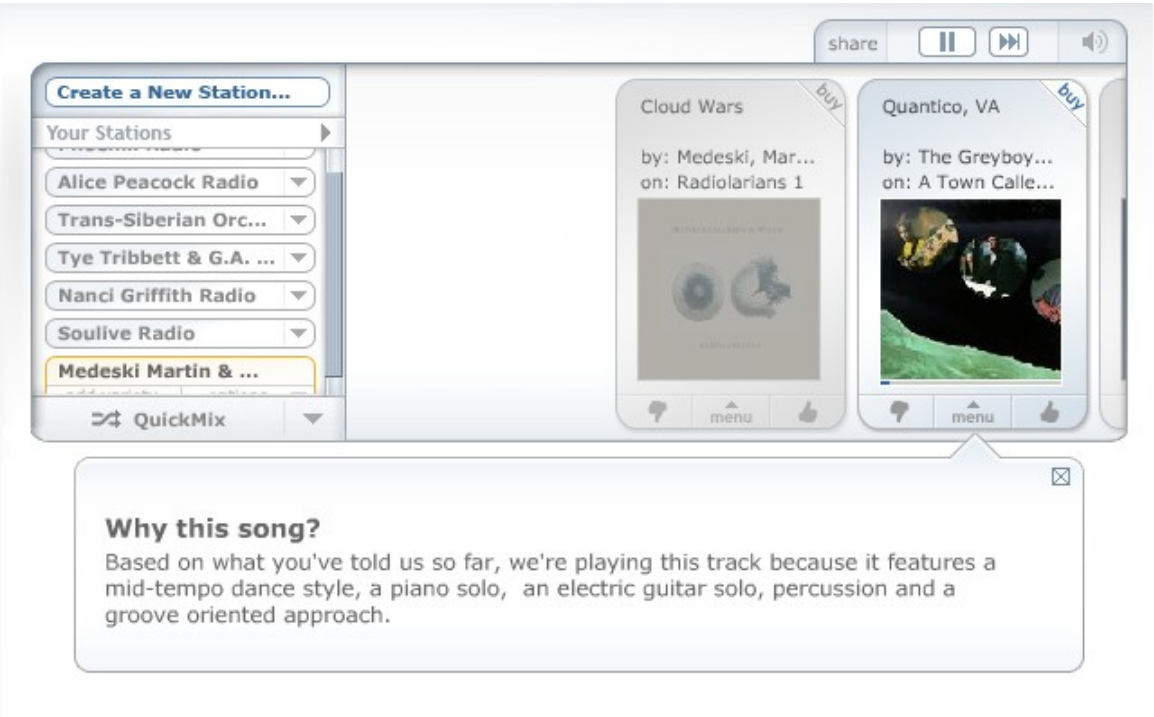
Now suppose I have a friend who says he likes the Black Keys Just Got to Be. I can plug that into my handy Manhattan distance function:

```
>>> computeNearestNeighbor('The Black Keys/Just Got to Be', 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")]
>>>
```

and I can recommend to him Heartless Bastard's *Out at Sea*. This is actually a pretty good recommendation.

NOTE:
The code for this example, as well as all examples in this book, is available on the book website http://www.guidetodatamining.com

When Pandora recommends something it explains why you might like it:



So Pandora thinks I might like Quantico, VA by the Greyboy Allstars because I like Medeski, Martin, and Wood (MMW) and MMW tunes tend to have a mid-tempo dance style, a piano solo, etc. We can do the same. My friend liked Black Keys Just Got to Be, we recommended Heartless Bastard's *Out at Sea*. We can compare the two vectors:

```
"Out at Sea":
{"piano": 1, "vocals": 5, "beat": 4, "blues": 2, "guitar": 4, "backup vocals": 1, "rap": 1},
"The Black Keys/Just Got to Be":
{"piano":1, "vocals": 4, "beat": 5, "blues": 3.5, "guitar": 5, "backup vocals": 1, "rap":1},
```

We could say something like: “We think you might like the Heartless Bastard's Out at Sea because it features vocals, a driving beat, and dirty electric guitar.

Because I had few features, and not a well-balanced set of features the other recommendations are not as compelling:

```
>>> 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')]
```

That Lady Gaga recommendation is particularly bad.

Normalization

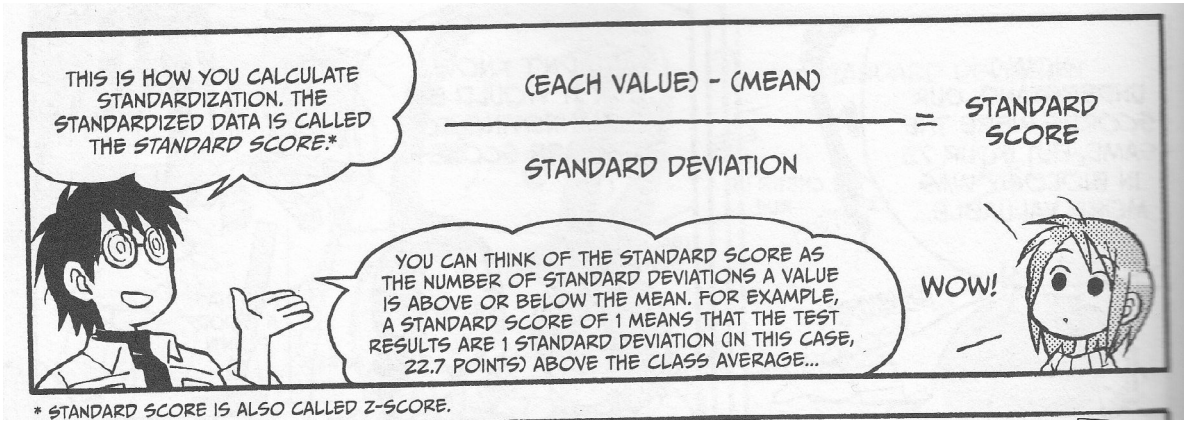
Suppose I want to add another feature to my set. This time I will add beats per minute (or pbm). This makes some sense—I might like fast beat songs or slow ballads. Now my data would look like this:

	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	100
Heartless Bastards / Out at Sea	1	5	4	2	4	1	1	130
The Black Keys/ Just Got to Be	1	4	5	3.5	5	1	1	90

As you can see this feature will wreck havoc with our distance function.---bpm dominates the calculation.

Consider another example. Suppose I represent people with two variables: age and income. In this case income dominates any distance calculation. To remove this bias we need to standardize or normalize the data. One common method of normalization involves having the values of each feature range from 0 to 1. For example, consider the piano feature. The minimum value is 1 and the max is 2.5. That makes the range 1.5. To convert each value to one in the range 0 to 1 we subtract the min from the value and divide by the range. So the value of piano for Phoenix is (2 – 1) / 1.5 = 0.66. Depending on the dataset this rough method of normalization may work well.

If you have taken a statistics course you will be familiar with more accurate methods for normalizing data. For example, my copy of *The Manga Guide to Statistics*, describes in cartoon detail how to compute the standard score



(By the way, if you like manga and are rusty with statistics—this is an awesome book!)

Standard deviation is

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

card(*x*) is the cardinality of *x*—that is, how many values there are.

Consider the following test scores

Student	Score
Amy	56
Ben	40
Jeff	38
Thomas	48
Chris	56
Stephen	36
John	60
Katherine	52
David	44
Joe	24
Katherine	46
SUM:	500

There are 11 students so the mean is 500/11 = 45.45.
The standard deviation is

$$sd = \sqrt{\frac{1120.73}{11}} = \sqrt{101.88} = 10.09$$

And the standard scores are

Student	Score	Standard Score
Amy	(56 – 45.45) / 10.09	1.05
Ben	(40 – 45.45) / 10.09	-0.54
Jeff	(38 – 45.45) / 10.09	-0.74
Thomas	(48 – 45.45) / 10.09	0.25
Chris	(56 – 45.45) / 10.09	1.05
Stephen	(36 – 45.45) / 10.09	-0.94
John	(60 – 45.45) / 10.09	1.44
Katherine	(52 – 45.45) / 10.09	0.65
David	(44 – 45.45) / 10.09	-0.14
Joe	(24 – 45.45) / 10.09	-2.13
Katherine	(46 – 45.45) / 10.09	0.05

The problem with using the standard score

The problem with the standard score is that it is greatly influenced by outliers. For example, if all the 100 employees of LargeMart make $10/hr but the CEO makes six million a year the mean hourly wage is

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

= (1000 + 2885) / 101

= $38/hr.

Not a bad average wage at LargeMart. As you can see, the mean is greatly influenced by outliers.

Because of this problem with the mean, the standard score formula is often modified.

Modified Standard Score

First, the mean in the above formula is replaced by the median (the middle value). The standard deviation is replaced with what is called the absolute standard deviation

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

where μ is the median.

To continue with our example above, the median is 46 and the absolute standard deviation is 8.18.

The modified standard scores are:

Student	Score	Modified Standard Score
Amy	(56 –46) / 8.18	1.22
Ben	(40 – 46) / 8.18	-0.73
Jeff	(38 – 46) / 8.18	-0.98
Thomas	(48 – 46) / 8.18	0.24
Chris	(56 – 46) / 8.18	1.22
Stephen	(36 – 46) / 8.18	-1.22
John	(60 – 46) / 8.18	1.71
Katherine	(52 – 46) / 8.18	0.73
David	(44 – 46) / 8.18	-0.24
Joe	(24 – 46) / 8.18	-2.69
Katherine	(46 – 46) / 8.18	0

You try

Give it a try. Compute the traditional Standard Scores and the Modified Standard Scores for the following data:

Track	Play Count	Standard Score	Modified Standard Score
Power/Marcus Miller	3		
If You Were A Bluebird/ Joe Ely	2		
Between the Lines/Sara Bareilles	17		
Pocket Calculator/Kraftwerk	1		
Europa/Santana	2		

My answer

Mean = 5; standard deviation = 9.27; median = 2.5; absolute standard deviation = 3.5

Track	Play Count	Standard Score	Modified Standard Score
Power/Marcus Miller	3	-0.22	0.14
If You Were A Bluebird/ Joe Ely	2	-0.32	-0.14
Between the Lines/Sara Bareilles	17	1.29	4.14
Pocket Calculator/Kraftwerk	1	-0.43	-0.43
Europa/Santana	2	-0.32	-0.14

To normalize or not

Normalization makes the most sense when the scale of the variables—the scales of the different dimensions—significantly varies. I've noted several examples of this above. If our variables are age, income, and college GPA, income will dominate any distance function. In this case we should normalize the data. In the other extreme consider a person giving thumbs up and thumbs down ratings to news articles on Digg. Here a vector representing a user consists of binary values (1 = thumbs up; 0 = thumbs down):

Bill = {0, 0, 0, 1, 1, 1, 1, 0, 1, 0 …

Obviously there is no need to normalize this data.

What about the Pandora case: all variables lie on a scale from 1 to 5 inclusive. Should we normalize or not? It probably wouldn't hurt the accuracy of the algorithm if we normalized, but keep in mind that there is a computational cost involved with normalizing. In this case, we might empirically compare results between using the regular and normalized data and select the best performing approach.

Back to Pandora

In the Pandora inspired example, we had each song represented by a number of attributes. If a user creates a radio station for Green Day we decide what to play based on a nearest neighbor approach. Pandora allows a user to give a particular tune a thumbs up or thumbs down rating. Let us suppose that a user gives a thumbs down for a particular song. How do we use this information?

Suppose I use 2 attributes for songs: the amount of dirty guitar and the presence of a driving beat both rated on a 1-5 scale. A user has given the thumbs up to 5 songs indicating he liked the song (and indicated on the following chart with a 'L'); and a thumbs down to 5 songs indicating he disliked the song (indicated by a 'D').

Do you think the user will like or dislike the song indicated by the star in this chart?

I am guessing you said he would like the song. We base this on the fact that the star is closer to the Ls in the chart than the Ds. We will spend the rest of this chapter describing computational approaches to this idea. The most obvious approach is to find the nearest neighbor of the star and predict that the star will share the class of the nearest neighbor. The star's nearest neighbor is an L so we would predict that the star is something the user would like.

The Python nearest neighbor classifier code

Let's use the example dataset I used earlier—ten tunes rated on 7 attributes (amount of piano, vocals, driving beat, blues influence, dirty electric guitar, backup vocals, rap influence). The chart with this data is shown on the following page.

	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/ Just Got to Be	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

Earlier in this chapter we developed a Python representation of this data:

```
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},
                                        etc.
```

Here the strings *piano, vocals, beat, blues, guitar, backup vocals*, and *rap* occur multiple times; if I have a 100,000 tunes those strings are repeated 100,000 times. I'm going to remove those strings from the representation of our data and simply use vectors:

```
#
#  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]}
```

In addition, I need to represent the thumbs up/ thumbs down ratings that users gives to songs. Because each user doesn't rate all songs (sparse data) I will go with the dictionary of dictionaries approach:

```
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"}             }
```

In order to use the new data format for items I need to revise the manhattan distance and the computeNearestNeighbor functions.

```
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 distance to item"""
   distances = []
   for otherItem in items:
      if otherItem != itemName:
         distance = manhattan(itemVector, items[otherItem])
         distances.append((distance, otherItem))
   # sort based on distance -- closest first
   distances.sort()
   return distances
```

Finally, I need to create a classify function. I want to predict how a particular user would rate an item represented by itemName and itemVector. For example:

"Enya/ Wild Child" [1, 5, 2.5, 1, 1, 5, 1]

The first thing the function needs to do is find the nearest neighbor of this Enya tune. Then it needs to

see how the user rated that nearest neighbor and predict that the user will rate Enya the same. Here's my rudimentary classify function:

```
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, itemVector, items)[0][1]
   rating = users[user][nearest]
   return rating
```

Ok. Let's give this a try. I wonder if Angelica will like Enya's Wild Child?

```
>>> classify('Angelica', 'Enya/Wild Child', [1, 5, 2.5, 1, 1, 5, 1])
'L'
```

We are predicting she will like it! Why are we predicting that?

```
>>> computeNearestNeighbor('Enya/Wild Child', [1, 5, 2.5, 1, 1, 5, 1], items)
[(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')]
>>>
```

We are predicting that Angelica will like Enya's Wild Child because that tune's nearest neighbor is Lady Gaga/Alejandro and Angelica liked that tune.

What we have done here is build a classifer—in this case, our task was to classify tunes as belonging to one of two groups—the *like* group and the *dislike* group.

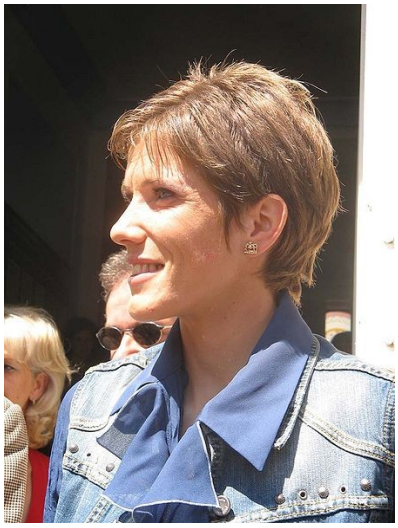
What sport?

To give you a taste of what we will be doing in a few chapters consider trying to classify world class athletes by what sport they participate in, based solely on height and weight. In the table on the next page I have a small sample dataset drawn from a variety of web sources. The gymnastic data lists some of the top participants in the 2008 Olympics.[1] The basketball players play for teams in the WNBA. The women track stars were finishers in the 2008 Olympic marathon.

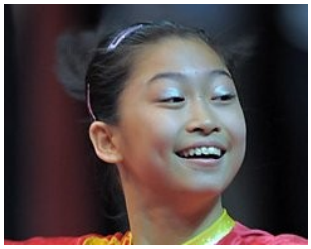
1 http://www.latimes.com/sports/olympics/la-sp-chinesegymnasts-12aug2008-g,0,7475962.graphic

Name	Sport	Age	Height	Weight
Shawn Johnson	Gymnastics	16	57	90
Li Shanshan	Gymnastics	16	57	79
Deng Linlin	Gymnastics	16	54	68
Bridget Sloan	Gymnastics	16	59	104
Nastia Liukin	Gymnastics	18	63	99
Ksenia Semenova	Gymnastics	16	54	77
Jennifer Lacy	Basketball	27	75	175
Shavonte Zellous	Basketball	24	70	155
Nakia Sanford	Basketball	34	76	200
Brittainey Raven	Basketball	22	72	162
Shanna Crossley	Basketball	26	70	155
Nikki Blue	Basketball	26	68	163
Dita Constantina	Track	40	65	105
Lisa Hunter-Galvan	Track	41	66	132
Mara Yamauchi	Track	37	64	112
Blake Russell	Track	35	66	110
Martha Komu	Track	37	65	115
Jelena Prokopcuka	Track	37	66	112

As you can see, I've included age in the table. Just scanning the data you can see that age alone is a good predictor. Try to guess the sports of these athletes[2]:



Olivera Jevtic
Age: 33



Jiang Yuyuan
Age: 16

Jayne Appel
Age: 22

2 The Samantha Peszek picture is licensed under a Creative Commons Attribution license by Wikipedia user TheBostonianLonghorn. The Olivera Jevtic picture is licensed under a Creative Commons Attribution license by Wikipedia user BokicaK. The Jayne Appel picture is licensed under a Creative Commons license by Flickr user E>mar.

The answers

Olivera Jevtic is a Serbian long-distance runner who competed in the 2008 Olympics.

Jiang Yuyyan was 16 when she won an Olympic gold medal in team gymnastics

Jayne Appel is a center for the San Antonio Stars of the Women's National Basketball Association.

Let's remove age from the picture. Here's a group of individuals I would like to classify.

Test data

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

YOUR TASK

1. Build a classifier based on the material covered in this chapter. It should use the dataset from the previous page and be able to classify the people listed in the test data table above. The output of my implementation of this is shown below. Your code may produce different output from mine.

2. The correct results are shown on the website for this book (guidetodatamining.com). Analyze the results from your program. How accurate was your program (x% accurate). If it produces an incorrect result can you give an explanation is to why it did so?

3. Does doubling number of instances in the training dataset (the table on the previous page) improve performance?

```
>>> a = classifier(athletes, sport)
>>> a.classify('Crystal Langhorne', (74, 190))
Closest to Jennifer Lacy
Crystal Langhorne: basketball
>>> a.classify('Li Shanshan', (64, 101))
Closest to Nastia Liukin
Li Shanshan: gymnastics
>>> a.classify('Kerri Strug', (57, 87))
Closest to Shawn Johnson
Kerri Strug: gymnastics
>>> a.classify('Koko Tsurumi', (55, 75))
Closest to Ksenia Semenova
Koko Tsurumi: gymnastics
>>> a.classify('Lindsay Whalen', (69, 169))
Closest to Nikki Blue
Lindsay Whalen: basketball
>>>
```

YOUR TASK #2

In this chapter I outlined an approach to developing a recommendation system that involved coming up with a set of attributes for objects and using a nearest neighbor approach to classify objects on a like/dislike scale. I used a tiny dataset consisting of 7 attributes and I rated a handful of songs.

Can you construct a better dataset? Using at least 25 attributes and rating at least 25 objects? You could create a better music dataset; a movie dataset; or whatever.

How well does the algorithm do at making recommendations for this dataset?

With My Own 2 Hands

The Pandora dataset is created by hand—experts rate songs on 400 attributes. That's labor intensive and time consuming but the results sound pretty good. Of course that is not the only way of populating a dataset relying on attributes. You could spider the web to get product attributes. For example, if I am doing a car recommendation system (*You looked at the 2010 Subaru Impreza WRX, you might also like* ....) I might spider sites like cars.com to get basic car attributes: engine size, engine HP, base price, highway MPG. You might mine textual reviews of cars to determine the values of attributes like *exhilarating power,* and *handling.* If the product type you are recommending is digital, perhaps your program can directly extract features. For example, for music you can extract beats per minute information, and other auditory attributes. There is nothing about this approach that requires hand-built data.