Chapter 5: Further Explorations in Classification

I am going to start this chapter by returning to the athlete example from the previous chapter. In that example you built a classifier which took the height and weight of an athlete as input and classified that input by sport—gymnastics, track, or basketball. So Kerri Strug who is 4 feet 9 inches and weighs 87 pounds

```
>>> a.classify('Kerri Strug', (57, 87))
'gymnastics'
```

Is classified as a gymnast.

Training set and test set

In this example we had two different datasets. One we used to construct the classifier. This data set is called the *training set.* The other data was the list of athletes we used to evaluate the classifier. That data is called the *test set.* *Training set* and *test set* are common terms in data mining.

People in data mining never test with the data they used to train the system.

You can see why we don't use the training data for testing if we consider the nearest neighbor algorithm. If Kerri Strug from the above example, was in our training data. She at (57,87) would be nearest neighbors with herself. So if our test set was a subset of our training data we would always be 100% accurate.

10-fold cross validation

One of the better methods of evaluating data mining algorithms is something called ten-fold cross validation. With this method we have one data set which we divide randomly into 10 parts. We use 9 of those parts for training and reserve one tenth for testing. We repeat this procedure 10 times each time reserving a different tenth for testing. Here's an example. Suppose I have a dataset of 100 instances. I order the instances randomly and number them 1 through 100. The procedure is then as follows

Train the classifier using examples 11-100
Test that classifier using examples 1-10
Train the classifier using examples 1-10 and 21-100
Test that classifier using examples 11-20
Train the classifier using examples 1-20 and 31-100
Test that classifier using examples 21-30.
…

We've created ten different classifiers, which we tested on different test sets. We sum up the results to report on the accuracy.

Later we will use this method but for our informal evaluations now, we will use separate training and test sets.

The athlete classifier

How accurate is your athlete classifier? In the last chapter you computed the accuracy using a small test set. Let's give it a try with a larger test set of 23 athletes (still not a large test set). You can download this test set from the book's website at http://www.guidetodatamining.com (the page for chapter 5). The file, athleteTestSet.txt is in the following tab-separated format:

```
Crystal Langhorne   Basketball   0     74    190
Li Shanshan  Track  0     64    101
Kerri Strug  Gymnastics   0     57    87
Jaycie PhelpsGymnastics   0     60    97
```

I have a function in my classifier that takes a filename containing data in this format and uses it to evaluate my classifier. That function looks like this:

```
   def evalClassifier(self, filename):
      """evaluate test set data"""
      f = open(filename)
      total = 0
      correct = 0
      for line in f:
         elements = line.split('\t')
         if len(elements) >= 5:
            total += 1
            name = elements[0]
            sport = elements[1]
            # not using age which is elements[2]
            height = int(elements[3])
            weight = int(elements[4])
            classification = self.classify(name, (height, weight))
            if classification == sport:
               print("%s CORRECT" % name)
               correct += 1
            else:
               print("%s MISCLASSIFIED AS %s. Should be %s" % (name, classification, sport))
      print("%f correct" % (correct / total))
```

Now I will evaluate the classifier using the original training set:

```
>>> a.evalClassifier('/Users/raz/Dropbox/guide/ch5/athleteTestSet.txt')
Crystal Langhorne CORRECT
Li Shanshan MISCLASSIFIED AS Gymnastics. Should be Track
Kerri Strug CORRECT
Jaycie Phelps CORRECT
Kelly Miller CORRECT
Zhu Xiaolin CORRECT
...
Naoko Sakamoto MISCLASSIFIED AS Gymnastics. Should be Track
Lidia Simon MISCLASSIFIED AS Gymnastics. Should be Track
Malgorzata Sabanska CORRECT
...
0.875000 correct
```

So the nearest neighbor algorithm with the original dataset containing 18 examples was 87.5% accurate. Let's try a few experiments.

Adding more data

What happens when we add more data to the training set? On the book website, I have a training set containing a bit over 60 examples. The training set is in the same tab-separated format as the test set described above. Give this a try with the code you wrote at the end of the previous chapter. Here are my results.

```
>>> bigData = classifier()
>>> bigData.loadAthleteData('athletes.txt')
63 entries loaded
...
>>> bigData.eval('athleteTestSet.txt')
Crystal Langhorne CORRECT
…
Gaelle Mys MISCLASSIFIED AS Track. Should be Gymnastics
0.958333 correct
```

Perhaps not surprisingly, adding more data improves performance. With my small training set I misclassified three instances of my test set. With the larger training set I only misclassified one.

Improving the algorithm - kNN

One trivial example of a classifier is the **Rote Classifier**, which just memorizes the entire training set and only classifies an instance if that instance matches one in the training set exactly. If we only evaluated classifiers on instances in the training data, the Rote Classifier would always be 100% accurate. In real life, the rote classifier is not a good choice because there will be instances we want to classify that are not in the training set. You can view the nearest neighbor algorithm we have been working with as an extension of the rote classifier. Instead of requiring exact matches we are looking at instances that are close matches. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar in their data mining textbook[1] call this the *If it walks like a duck, quacks like a duck, and looks like a duck, then it's probably a duck* approach.

One way to improve our current nearest neighbor approach is instead of looking at one nearest neighbor we look at a number of nearest neigbors—*k* nearest neighbors or kNN. We did this when we were using the neighbor approach approach to determine a numeric rating for how well a user would like a particular song. For example, suppose we are using three nearest neighbors for our calculation, and we are trying to determine how well Ben will like the Funky Meters. Distance is measured on a 0-1 scale where 1 is an exact match and 0 is maximally distal. The three closest to Ben are

User	Distance	Rating for Funky Meters
Sally	0.8	4
Tara	0.6	5
Jade	0.6	5

We are going to apportion influence on our guess. We sum up the distances to get 2.0. Sally's share of that is 0.8/2 = 0.4; Tara's is 0.3 and Jade's is 0.3. We mulitply each user's rating by their influence and sum the results:

Our guess as to how well Ben will like the Funky Meters:

	Sally's contribution		Tara's contribution		Jade's contribution
=	.4 * 4	+	.3 * 5	+	.3 * 5

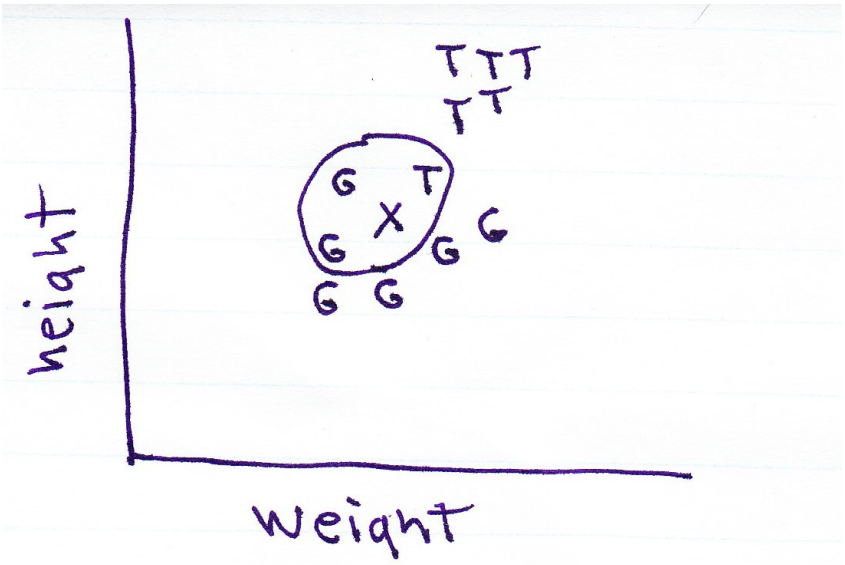
= 1.6 + 1.5 + 1.5 = 4.6

In this chapter our algorithm is not outputing a number, it's outputing a discrete classification: gymnastics, basketball, or track. The kNN algorithm for a discrete classifier is like majority voting.

1 Introduction to Data Mining. 2006. Pearson Press.

Every neighbor has the same weighted vote for classification. For example, if we are using a three nearest neighbor approach and two or three neighbors have the same classification, we will go with that classification. If every neighbor has a different classification, we will just randomly pick a classification.

Using k nearest neighbors instead of one may help smooth weirdnesses in our data. For example, suppose we have a particularly short and light weight marathoner in our training set. Suppose we want to classify the x shown in the following chart.



The three nearest neighbors to x are shown in the circle, the nearest neighbor is the T in the circle. If we were to just use a single nearest neighbor we would classify the x as 'track'. If we used the 3 nearest neighbor approach we would classify x as 'gymnastics'. Eyeballing the chart, the x does look like it belongs with the cluster of *g*s more than it does with the cluster of *t*s.

Python implementation of kNN

On the following page is my code for the k nearest neighbor algorithm

> **YAN (yet another note)**
>
> **The file containing all the code for this classifier is on the book website.**

```
01 def kNN(self, itemName, itemVector, k):
02  # first find nearest neighbor
03  tmp = {}
04  resultSet = self.computeNearestNeighbor(itemName,self.normalizeInstance(itemVector))[:k]
05  # the resultSet now contains the k nearest neighbors
06  for res in resultSet:
07     classification = self.category[res[1]]
08     if classification in tmp:
09        tmp[classification] += 1
10     else:
11        tmp[classification] = 1
12   recommendations = list(tmp.items())
13   # recommendations is a list of classes and how many times
14   # that class appeared in the nearest neighbor list (votes)
15   # i.e. [['gymnastics', 2], ['track', 1]]
16  recommendations.sort(key=lambda artistTuple: artistTuple[1], reverse = True)
17  # construct list of classes that have the largest number of votes
18  topRecommendations = list (filter(lambda k: k[1] == recommendations[0][1],
                        recommendations))
19  # if only one class has the highest number of votes return that class
20  if len(topRecommendations) == 1:
21     rating =  topRecommendations[0][0]
22  else:
23     rint = random.randint(0, len(topRecommendations) - 1)
24     rating = topRecommendations[rint][0]
25  return rating
```

Let's dissect this code a bit. A call to this function looks like:

```
>>> a.kNN('Paula Radcliffe', (63, 99), 3)
```

So here, itemname is Paula Radcliffe, the itemVector is (63, 99) representing Paula's height and weight as 5 foot 3 and 99 pounds, and k is 3 meaning that we will use the 3 nearest neighbors. On line 4 I call the function computeNearestNeighbor which finds the nearest neighbors to this itemVector.

Instead of sending itemVector itself to computeNearestNeighbor you see that I call normalizeInstance on itemVector.

Creative Commons Attribution Share Alike 3.0 license Richard Greenhill and Hugo Elias of the Shadow Robot Company

I am hoping that while you were coding the athlete problem at the end of the last chapter a lightbulb went on in your head (is that the correct expression?). Maybe you had the idea of normalizing before you started coding, or maybe you coded the whole thing, the results were disappointing, and you had this ah-ha moment.

Or, maybe you are about to have that ah-ha lightbulb moment.

I talked about normalization at the beginning of that chapter. Consider the athlete training data shown below.

Deng Linlin is both the shortest and lightest of the athletes. Nakia Sanford is both the tallest and the heaviest. The difference in weight between Deng Linlin and Nakia Sanford is 132 pounds; the difference in their height is only 22 inches. If we used raw numbers for our nearest neighbor classifier, weight would dominate the calculation.

This is a classic case of when we should normalize our training data.

Athlete Training Data

Name	Sport	Age	Height	Weight
Shawn Johnson	Gymnastics	16	57	90
Li Shanshan	Gymnastics	16	57	79
Deng Linlin	**Gymnastics**	**16**	**54**	**68**
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

So, when I created my classifier, I normalized the training data.

Line 4 in the above function:

```
04  resultSet = self.computeNearestNeighbor(itemName,self.normalizeInstance(itemVector))[:k]
```

normalizes the instance we are about to test before sending it to the computeNearestNeighbor function.

This requires me knowing the median and absolute standard deviation which I previously computed in order to normalize the training data.

ComputeNearestNeighbor returns a list of all the instances in the training set ordered by their distance to the test instance. At the very end of line 4 I have [:k] which truncates this to the k closest neighbors.

My code in lines 6-11 looks up the classification of each of these k neighbors and adds them to a dictionary named tmp.

```
06  for res in resultSet:
07     classification = self.category[res[1]]
08     if classification in tmp:
09        tmp[classification] += 1
10     else:
11        tmp[classification] = 1
```

The result might be something like [['track', 1], ['gymnastics', 2]] indicating that 1 neighbor was classified as 'track' and 2 as 'gymnastics'.

Finally, the rest of the code determines the classification to be assigned to the test instance. If there is a single classification with the highest number of votes it returns that. If there is a tie, it randomly determines a class for the tied classes.

Recall that our original nearest neighbor algorithm was 87.5% accurate:

```
>>> a.evalClassifier('/Users/raz/Dropbox/guide/ch5/athleteTestSet.txt')
Crystal Langhorne CORRECT
Li Shanshan MISCLASSIFIED AS Gymnastics. Should be Track
Kelly Miller CORRECT
Zhu Xiaolin CORRECT
...
Naoko Sakamoto MISCLASSIFIED AS Gymnastics. Should be Track
Lidia Simon MISCLASSIFIED AS Gymnastics. Should be Track
Malgorzata Sabanska CORRECT
...
0.875000 correct
```

Here is our the kNN algorithm performs with k = 3:

```
>>> a.evalKNN('/Users/raz/Dropbox/guide/ch5/athleteTestSet.txt')
Crystal Langhorne CORRECT
Li Shanshan CORRECT
Kerri Strug CORRECT
Jaycie Phelps CORRECT
Kelly Miller CORRECT
Zhu Xiaolin CORRECT
Lindsay Whalen CORRECT
…
Lidia Simon MISCLASSIFIED AS Gymnastics. Should be Track
…
Gaelle Mys CORRECT
0.958333 correct
```

Nearly 96% accurate so that is an improvement. In fact it is the same accuracy as the original algorithm when we added more data (however the instance each got incorrect is different). What happens when we use a large training set plus our new, improved kNN algorithm:

```
>>> bigData.evalKNN('/Users/raz/Dropbox/guide/ch5/athleteTestSet.txt')
Crystal Langhorne CORRECT
Li Shanshan CORRECT
…
Essence Carson CORRECT
Marion Jones CORRECT
Jade Barbosa CORRECT
1.000000 correct
```

That gives us 100% accuracy. Keep in mind that this is an easy task and our test set is small—don't expect this type of accuracy on real-world problems. Let's take a moment to dissect these results a bit.

Using the one nearest neighbor algorithm with the small training set, we were able to classify the Belgian gymnast, Gaelle Mys correctly:

```
>>> a.classify('Gaelle Mys', (58, 88))
'Gymnastics'
```

but with the larger training set we misclassified this instance.

```
>>> bigData.classify('Gaelle Mys', (58, 88))
'Track'
```

but with the larger taining set plus the kNN algorithm with k = 3 we correctly classified the instance

```
>>> bigData.kNN('Gaelle Mys', (58, 88), 3)
'Gymnastics'
```

Why is this? Contained in the larger training set is the data for Mizuki Noguchi, the winner of the 2004 Olympic women's marathon. She is 4'11" tall and weighs 88 pounds—not completely unusual for a long-distance runner but still on the short and light side. She ends up being Gaelle Mys closest neighbor and that is how Gaelle Mys gets misclassified. Now consider using the three nearest neighbors. Gaelle's three nearest neighbors are Mizuki Noguchi, Cha Jang Hwa, and Kyoho Oshima:

```
>>> n = bigData.normalizeInstance((58, 88))
>>> bigData.computeNearestNeighbor('Gaelle Mys', n)[:3]
[(0.19003115264797499, 'Mizuki Noguchi'), (0.25895770632029136, 'Cha Jong Hwa'),
(0.25895770632029136, 'Kyoho Oshima')]
```

Mizuki Noguchi	Track
Cha Jong Hwa	Gymnastics
Kyoho Oshima	Gymnastics

Cha Jang Hwa and Kyoho Oshima are gymnasts (in fact, other than for Mizuli Noguchi, all of Gaelle's ten nearest neighbors are gymnasts). Each of the three nearest neighbors gets a vote. Thus we correctly classify Gaelle Mys as participating in gymnastics. Using kNN tends to smooth out any irregularities in the training data.

More data, better algorithms, & a broken bus.

Several years ago I was at a conference in Mexico City. This conference was a bit unusual in that it alternated between a day of presentations and a day of touring (the Monach Butterflies, Inca ruins, etc). The days of touring involved riding long distances on a bus and the bus had a tendency to break down. As a result, a bunch of us PhD types spend a good deal of time standing at the side of road talking to one another as the bus was being attended to. These roadside exchanges were the highpoint of the conference for me. One of the people I talked to was a person named Eric Brill. Eric Brill is famous for developing what is called the Brill tagger, which does part-of-speech tagging. Similar to what we have been doing in the last few chapters, the Brill tagger classifies data—in this case, it classifies words by their part of speech (noun, verb, etc.). The algorithm Brill came up with was significantly better than its predecessors (and as a result Brill became famous). At the side of that Mexican road, I got to talking with Eric Brill about improving the performance of algorithms. His view is that you get more of an improvement by getting more data for the training set, than you would by improving the algorithm. In fact, he felt that if he kept the original part-of-speech tagging algorithm and just increased the size of the training data, the improvement would exceed that of his famous algorithm.

Here's another example. In various machine translation competitions, Google always places at the top. Granted that Google has a large number of very bright people developing great algorithms, much of Google's dominance is due to its enormous training sets it spidered from web.

This isn't to say that you shouldn't pick the best algorithm for the job. As we have already seen picking a good algorithm makes a significant difference. However, if you are trying to solve a practical problem (rather than publish a research paper) it might not be worth your while to spend a lot of time researching and tweaking algorithms. You will perhaps get more bang for your buck—or a better return on your time—if you concentrate on getting more data.

With that nod toward the importance of data, I will continue my path of introducing new algorithms.

Probability and Naïve Bayes

Let's continue with our athlete example. Suppose I ask you what sport Christi Thomas participates in (gymnastics, marathon running, or basketball) and I tell you she is 6 foot 4 and weighs 198 pounds. I imagine you would say basketball and if I ask you how confident you feel about your decision I imagine you would say something along the lines of *pretty darn confident.*

Now I ask you about what sport Yumiko Hara participates in; she is 5 foot 4 inches tall and weighs 95 pounds. Here I am less certain how you will answer. Let's say you say gymnastics and I ask how confident you feel about your decision. You will probably say something along the lines of *not too confident.*

With the nearest neighbor algorithms, it is difficult to quantify confidence about a classification. With classification methods based on probably—Bayesian methods—we can not only make a classification but we can make probablistic classifications—this athlete is 80% likely to be a baskball player, this patient has a 40% chance of having a heart attack in the next five years, the probabilty of rain in Las Cruces in the next 24 hours is 10%.

Nearest Neighbor approaches are called **lazy learners**. They are called this because when we give them a set of training data, they just basically save—or remember—the set. Each time it classifies an instance it goes through the entire training dataset. If we have a 100,000 music tracks in our training data, it goes through the entire 100,000 tracks each time it classifies an instance. Bayesian methods are called **eager learners**. When given a training set eager learners immediately analyze the data and build a model. When it wants to classify an instance it uses this internal model. Eager learners tend to classify instances faster than lazy learners.

The ability to make probablistic classifications, and the fact that they are eager learners are two advantanges of Bayesian methods.

Probability.

I am assuming you have some basic knowledge of probability. I flip a coin; what is the probably of it beings a 'heads'? I roll a 6 sided fair die, what is the probabily that I roll a '1'? that sort of thing. I tell

you I picked a random 19 year old and have you tell me the probabily of that person being female and without doing any research you say 50%. These are example of what is called *prior probability* and is denoted $P(h)$—the probabily of hypothesis h.

So for a coin → $P(heads) = 0.5$

for a 6-sided die → $P(1) = 1/6$

If I have an equal number of 19 yr. old male and females → $P(female) = .5$

Suppose I give you some additional information about that 19 yr. old—the person attends Rishi University in New Mexico. You do a quick Google search, see that the student body is 70% female and revise your estimate of the likelihood of the person being female to 70%.

This we denote as $P(h|d)$ —the probability of the hypothesis some some data. For example:

$P(female \mid attends\ Rishi\ U) = .7$

This is called **posterior probability.** The two other probabilities we need are $P(D)$ and $P(D|h)$. To explain these consider a different example.

Microsoft Shopping Cart

Did you know that Microsoft makes smart grocery store shopping carts?[2] Yep, they do.

You come in with your grocery store loyalty card. The cart recognizes you. It has recorded all previous purchases (as well as the purchases of everyone else in the store).

Suppose the cart software wants to determine whether to show you a targeted ad for Japanese Sensha Green Tea. It only wants to show that ad if you are likely to purchase the tea.

The cart system has accumulated the small dataset shown below from other shoppers

P(D) is the probability that some training data will be observed. For example, the probability that the zip code will be 88005 is 5/10 or 0.5.

P(88005) = 0.5

$P(D|h)$ is the probability that some data value holds given the hypothesis.

Customer ID	Zip code	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

2 Photo from Flickr user varmakup. Creative Commons Attribution NonCommercial ShareAlike 2.0 license

For example, the probability of the zip code being 88005 given that the person bough Sencha Green Tea. So I am looking at all the instances where the person bought Sensha Tea:

Customer ID	Zip code	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

There are 5 such instances. Of those, 3 of them are with the 88005 zip code

P(88005| bought Sencha Tea) = 3/5.

Bayes Theorem

Bayes Theorem describes the relationship between $P(h)$, $P(h|D)$, $P(D)$, and $P(D|h)$:

$$P(h|D)=\frac{P(D|h)\,P(h)}{P(D)}$$

This theorem is the cornerstone of all Bayesian methods. Usually in data mining we use this theorem to decide among alternative hypotheses. Given the evidence, is the person a gymnast, marathoner, or basetball player. Given the evidence, will this person buy Sencha tea, or not. To decide among alternatives we compute the probability for each hypothesis:

$$P(gymnast|D)=\frac{P(D|gymnast)\,P(gymnast)}{P(D)} \qquad P(marathon|D)=\frac{P(D|marathon)\,P(marathon)}{P(D)}$$

$$P(basketball|D)=\frac{P(D|basketball)\,P(basketball)}{P(D)}$$

Or, more abstractly

$$P(h_1|D)=\frac{P(D|h_1)\,P(h_1)}{P(D)} \qquad \ldots \qquad P(h_n|D)=\frac{P(D|h_n)\,P(h_n)}{P(D)}$$

Once we compute all these probabilities, we will pick the hypothesis with the highest probability. This is called the *maximum a posteriori* hypothesis, or h_{MAP}.

If you look at the formulas for $P(gymnast|D)$, P(*marathoner*|D), and $P(basketball\ player|D)$ you will notice that the denominators of these formulas, $P(D)$ are identical—they are independent of the hypotheses and as a result we can simplify our calculations:

$$h_{MAP}=argmax_{h\in H}\,P(h|D)$$

$$h_{MAP}=argmax_{h\in H}\frac{P(D|h)\,P(h)}{P(D)}$$

$$h_{MAP}=argmax_{h\in H}\,P(D|h)\,P(h)$$

To see how this work I will use an example from Tom Mitchell's book, *Machine Learning*. Consider a medical domain and the hypotheses are

1. the patient has a particular type of cancer
2. the patient does not have this particular type of cancer.

We know that only 0.8% of the people in the U.S. Have this form of cancer. Luckily there is a simple blood test we can do. The test is a binary one—it comes back either POS or NEG. The test returns a correct POS result 98% of the time when the disease is present; it returns a correct NEG result 97% of the time in cases when the disease is not present. To summarize this using the notion I introduced:

$P(cancer) = .008$	$P(\neg\, cancer) = .992$
$P(\text{POS} \mid cancer) = .98$	$P(\text{ NEG} \mid cancer) = .02$
$P(\text{POS} \mid \neg\, cancer) = .03$	$P(\text{ NEG} \mid \neg\, cancer) = .97$

Suppose Ann, comes into the doctor's office. A blood test for the cancer is given and the test result is POS. This is not looking good for Ann. After all, the test is 98% accurate.

Using Bayes theorem compute the likelihood that Ann has cancer, and the likelihood that she doesn't.

Give it a try!

The Answer

We are finding the maximum a posteriori probability:

$P(\text{POS} \mid cancer)P(cancer) = .98(.008) = .0078$

$P(\text{POS} \mid \neg\, cancer)\, P(\neg\, cancer) = .03(.992) = .0298$

We select h_{MAP} and classify the patient as not having cancer.

If we want to know the exact probability we can normalize these values by having them sum to 1:

$$P(cancer|POS) = \frac{.0078}{.0078 + .0298} = .21$$

Ann has a 21% chance of having cancer.

Naïve Bayes Classifier

Most of the time we have more evidence than just a single piece of data. In the Sencha tea example we had two types of evidence: zip code and whether the person purchased organic food. To compute the probability of an hypothesis given all the evidence, we simply multiply the individual probabilities. In this example

tea = person buys Sencha tea

¬ tea = person doesn't buy Sencha tea

P(88005 | tea) = probability that a tea purchaser is in the 88005 zip code

etc.

Customer ID	Zip code	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

To compute

P(tea | lives in 88005 and buys organic food) we simply multiply the probabilities

$P(tea \mid lives\ in\ 88005\ and\ buys\ organic\ food) =$

$P(88005 \mid tea)\ P(organic\ food \mid tea)\ P(tea) = .6(.8)(.5) = .24$

$P(\neg\ tea \mid lives\ in\ 88005\ and\ buys\ organic\ food) =$

$P(88005 \mid \neg\ tea)\ P(organic\ food \mid \neg\ tea)\ P(\neg\ tea) = .4(.25)(.5) = .05$

So a person who lives in the trendy 88005 zip code area and buys organic food is more likely to buy Sencha Green tea than not. So let's display the Green Tea ad on the shopping cart display!

Here's how Stephen Baker describes the smart shopping cart technology:

> … here's what shopping with one of these carts might feel like. You grab a cart on the way in and swipe your loyalty card. The welcome screen pops up with a shopping list. It's based on patterns of your last purchases. Milk, eggs, zucchini, whatever. Smart systems might provide

you with the quickest route to each item. Or perhaps they'll allow you to edit the list, to tell it, for example, never to promote cauliflower or salted peanuts again. This is simple stuff. But according to Accenture's studies, shoppers forget an average of 11 percent of the items they intend to buy. If stores can effective remind us of what we want, it means fewer midnight runs to the convenience store for us and more sales for them.

Baker. 2008. P49.

The Numerati.

I've mentioned this book by Stephen Baker several times. I highly encourage you to read this book. The paperback is only $10 at its a good late night read.

i100, i500

Let's say we are trying to help iHealth, a company that sells wearable exercise monitors that come in two models, increasing in functionality: the i100 and the i500.

iHealth100: heart rate, GPS (to compute miles per hour, etc), wifi to automatically connect to iHealth website to upload data.

iHealth500: i100 features + pulse oximetry (oxygen in blood) + free 3G connection to iHealth website

They sell these online and they hired us to come up with a recommendation system for their customers. To get data to build our system when someone buys a monitor, we ask them to fill out the questionaire. Each question in the questionaire relates to an attribute. First, we ask them what their main reason is for starting an exercise program and have them select among three options: health, appearance or both. We ask them what their currect exercise level is: sedentary, moderate, or active. We ask them how motivated they are: moderate or aggressive. And finally we ask them if they are comfortable with using technological devices. Our results are shown in the table on the next page.

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

Using the naïve Bayes method, which model would you recommend to a person whose

- main interest is health
- current exercise level is moderate
- is moderately motivated
- and is comfortable with technological devices

Try doing this on your own before turning the page. If you get stuck, turn the page for clues.

Clue

Ok. So we want to compute

P(i100 | health, moderateExercise, moderateMotivation, techComfortable)

and

P(i500 | health, moderateExercise, moderateMotivation, techComfortable)

and pick the model with the highest probability.

Let me lay out what we need to do for the first one:

P(i100 | health, moderateExercise, moderateMotivation, techComfortable) =

P(health|i100) P(moderateExercise|i100) P(moderateMotivated|i100) P(techComfortable|i100)P(i100)

So here is what we need to first compute

P(health|i100) = 1/6

P(moderateExercise|i100) =

P(moderateMotivated|i100) =

P(techComfortable|i100) =

P(i100) = 6 / 15

That was my clue. Now hopefully you can figure out the example

P(health|i100) = 1/6

P(moderateExercise|i100) = 1/6

P(moderateMotivated|i100) = 5/6

P(techComfortable|i100) = 2/6

P(i100) = 6 / 15

P(i100| evidence) = .167 * .167 * .833 * .333 * .4 = .00309

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

Thus, in this case we would recommend the iHealth500. Since (.01975 > .00309)

Python code

Let's do this in Python.

I am going to represent the training data as lists:

```
data  = [['i100', 'both', 'sedentary', 'moderate', 'yes'],
         ['i100', 'both', 'sedentary', 'moderate', 'no'],
         ['i500', 'health', 'sedentary', 'moderate', 'yes'],
         etc.
```

As you can see, I put the category of the instance first. There is no compelling reason for this position, but it just makes the code simpler if I can assume a standard spot for the category. I am going to divide the code into two general parts: the part where I train the classifier, and the part that actually classifies instances. First the training.

Training

The output of the training needs to be

- a set of prior probabilties—for example, the P(i100) = .4 above.
- A set of conditional probabilties—for example P(health| i100)

The prior probabilities can be represented as a simple dictionary I will call self.prior.

The conditional probabilities are a bit more complex. My way of doing this—and there are probably better methods—is to associate a set of conditional probabilities with each class.

self.conditionals = {i100: *conditional probabilities associated with i100*, etc.

The training data can be seen as being in columns:

```
data  = [['i100', 'both', 'sedentary', 'moderate', 'yes'],
         ['i100', 'both', 'sedentary', 'moderate', 'no'],
         ['i500', 'health', 'sedentary', 'moderate', 'yes'],
```

lobbing off the initial category column we have the first row as

```
['both', 'sedentary', 'moderate', 'yes']
```

For each column I am going to compute the probability for each value of that column. For example, the first column in this example represent the answer to the question *What is the reason for getting this health monitor: you want to improve your health, your appearance, or both?* So, for example, I am going to compute the probability

P(column1='both' | i100)

My final representation for the conditional probabilities are as follows

```
{'i500': [{'health': 0.4444444444444442, 'appearance': 0.3333333333333331,
           'both': 0.2222222222222221},
          {'active': 0.4444444444444442, 'sedentary': 0.2222222222222221,
           'moderate': 0.3333333333333331},
          {'aggressive': 0.6666666666666663, 'moderate': 0.3333333333333331},
          {'yes': 0.6666666666666663, 'no': 0.3333333333333331}],
 'i100': [{'both': 0.5, 'health': 0.1666666666666666, 'appearance': 0.3333333333333331},
          {'active': 0.3333333333333331, 'sedentary': 0.5,
           'moderate': 0.1666666666666666},
          {'moderate': 0.8333333333333337, 'aggressive': 0.1666666666666666},
          {'yes': 0.3333333333333331, 'no': 0.6666666666666663}]}
```

So you can better disect this structure I color coded the data associated with each column: column 1, column 2, column 3, column 4.

With that prologue, here is the code for the training method:

```
class Bayes:

   def __init__(self, data):
      # here I am assuming the first column of the data is the class.
      self.data = data
      self.prior = {}
      self.conditional = {}

   def train(self):
      """train the Bayes Classifier
      basically a lot of counting"""
      total = 0
      classes = {}
      counts = {}
      # determine size of a training vector
      size = len(self.data[0])
      #
      #   iterate through training instances
      for instance in self.data:
         total += 1
         category = instance[0]
         classes.setdefault(category, 0)
         counts.setdefault(category, {})
         classes[category] += 1
         # now process each column in instance
         col = 0
         for columnValue in instance[1:]:
```

```
            col += 1
            tmp = {}
            if col in counts[category]:
               tmp = counts[category][col]
            if columnValue in tmp:
               tmp[columnValue] += 1
            else:
               tmp[columnValue] = 1
            counts[category][col] = tmp
         # ok. done counting. now compute probabilities
         #
         # first prior probabilities
         #
         for (category, count) in classes.items():
            self.prior[category] = count / total
         # now compute conditional probabilities
         for (category, columns) in counts.items():
            tmp = {}
            for (col, valueCounts) in columns.items():
               tmp2 = {}
               for (value, count) in valueCounts.items():
                  tmp2[value] = count / classes[category]
               tmp[col] = tmp2
            #convert tmp to vector
            tmp3 = []
            for i in range(1, size):
               tmp3.append(tmp[i])
            self.conditional[category] = tmp3
```

Now on to the classification method. This method will take a vector argument and return the classification (the category with the highest probability).

```
def classify(self, instance):
   categories = {}
   for (category, vector) in self.conditional.items():
      prob = 1
      for i in range(len(vector)):
         colProbability = .0000001
         if instance[i] in vector[i]:
            # get the probability for that column value
            colProbability = vector[i][instance[i]]
         prob = prob * colProbability
      prob = prob * self.prior[category]
      categories[category]  = prob
   cat = list(categories.items())
   cat.sort(key=lambda catTuple: catTuple[1], reverse = True)
   return(cat[0])
```

And finally, here is the code at work:

```
>>> b = Bayes(data)
>>> b.classify(['health', 'moderate', 'moderate', 'yes'])
('i500', 0.019753086419753083)
>>> b.classify(['appearance', 'moderate', 'moderate', 'no'])
('i100', 0.012345679012345678)
```

As you can see, our code gets the same answer for the first one as when we computed it by hand.

Problems

The training set is just a sample of instances drawn from the larger universe. ihealth may have sold thousands of health monitors but we only have 15 instances in our training set. Even large training sets are typically just a small portion of the entire population. For example, The National Institutes of Health's Women's Health Initiative study gathered data on 17,000 menopausal women. That seems like a large number but there are over 50 million menopausal women in the U.S. So what I am about to say is not limited to the trivial 15 instance datasets we've been playing with but is applicable to most datasets.

However, to illustrate the problem, I will use our existing health monitor training set. Suppose we add another feature to the training set, principal aerobic exercise with values like running, cycling, etc. and suppose that about one out of 200 people that buy a health monitor are principly scullers.

So, the true probabily of health monitor buyers who are scullers is .005. So how many instances of scullers are we likely to find in our 15 instance training set?

If you answered ZERO that would be correct. So the probability of recommending an i100 for a moderate active, moderately motivated, tech-savy sculler who is interested in buying a monitor for health reasons is also zero. The zero occurrences of sculling dominates our calculation and the probabilities of an i100 and i500 are both zero. We can fix our estimate of probability by changing our calculation slightly.

If we are trying to calculate something like P(health | i500) our calculation has been:

$$P(health|i500) = \frac{the\ number\ of\ instances\ that\ both\ have\ the\ health \wedge i500}{the\ total\ number\ of\ instances\ of\ i500}$$

For expository ease let me simplify this by using shorter variable names

$$P(x|y) = \frac{n_c}{n}$$

Here n is the total number of instances of the class y in the training set; n_c is the total number of instances of class y that have the value x.

The problem we have is when n_c equals zero. We can eliminate this problem by changing the formula to[3]

$$P(x|y) = \frac{n_c + mp}{n+m}$$

m is a constant called the equivalent sample size. The method for determining the value of m varies. For now I will use the number of different values that attribute takes. For example, the values for the first column are *health*, *appearance*, and *both* so I will use an m of 3. p is the prior estimate of the probability. Often we assume uniform probability. For example, in this case what is the probability of randomly picking *health* out of a hat containing *health*, *appearance*, and *both.* That is 1/3 so p in this case is 1/3

Let me quickly go through an example to show how this works—it is identical to our previous example except I deleted the last row of the table representing our training set.

3 This formula is from p179 of *Machine Learning* by Tom Mitchell.

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

Using the naïve Bayes method, which model would you recommend to a person whose

- main interest is health
- current exercise level is moderate
- is moderately motivated
- and is comfortable with technological devices

I want to compute the probability of selecting an i100. As part of this I need to compute P(health|i100). In this case, there are zero occurrences of people who buy the i100 and say it is for health reasons. Using our original formula, that makes p(health|i100) = 0, and, a result, the probability that a person whose main interest is health, whose current exercise level is moderate, who is moderately motivated and is comfortable with technological devices is zero. With our new formula we get

$$P(health|i100) = \frac{n_c + mp}{n + m} = \frac{0 + \frac{1}{3} \times 3}{5 + 3} = \frac{1}{8} = .125$$

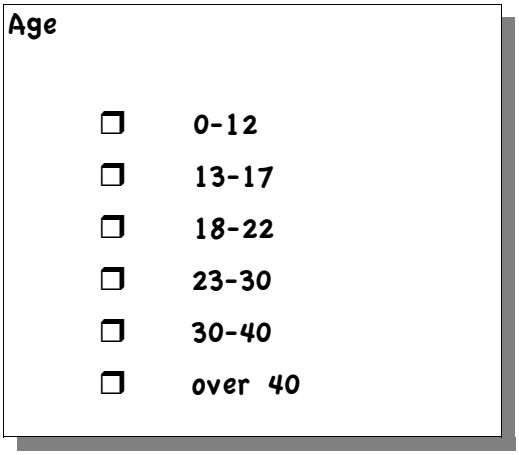
We will see another example of why this improved formula is important in the next chapter.

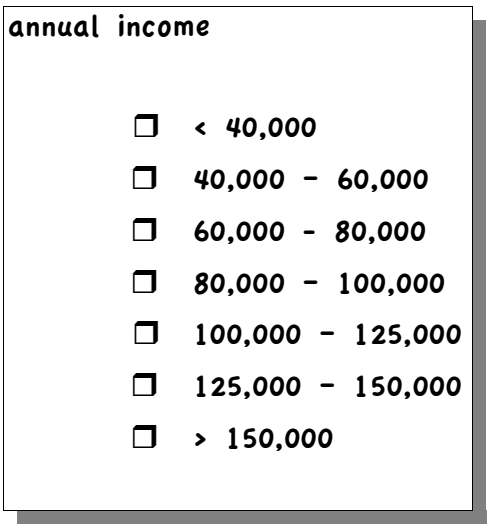
Numbers

You probably noticed that I changed from numerical data which I used in all the nearest neighbor approaches I discussed to using categorical data for the naïve Bayes formula. For Bayesian approahes we count things—how many occurrences are there of people who are sedentary—and it may not be initially obivious how to count things that are on a scale—for example, something like grade point average. There are two approaches.

Making categories

One solution is to make categories by discretizing the continuous attribute. You often see this on websites. For example:





Examples like this are quite common. Once the attributes are discretized, we can use Bayesian methods.

Method 2: Gaussian distributions

The next method of dealing with a continuous scale in a Bayesian method is to use what is called a probability density function. Consider the example we have been using with an added attribute of income:

Main Interest	Current Exercise Level	How Motivated	Comfortable with tech. Devices?	Income	Model #
both	sedentary	moderate	yes	60k	i100
both	sedentary	moderate	no	75k	i100
health	sedentary	moderate	yes	90k	i500
appearance	active	moderate	yes	125k	i500
appearance	moderate	aggressive	yes	100k	i500
appearance	moderate	aggressive	no	90k	i100
health	moderate	aggressive	no	220k	i500
both	active	moderate	yes	85k	i100
both	moderate	aggressive	yes	80k	i500
appearance	active	aggressive	yes	110k	i500
both	active	aggressive	no	95k	i500
health	active	moderate	no	80k	i500
health	sedentary	aggressive	yes	70k	i500
appearance	active	moderate	no	70k	i100

Let's think of the typical purchaser of an i500, our awesome, premire device. If I were to ask you to describe this person you might give me the average income:

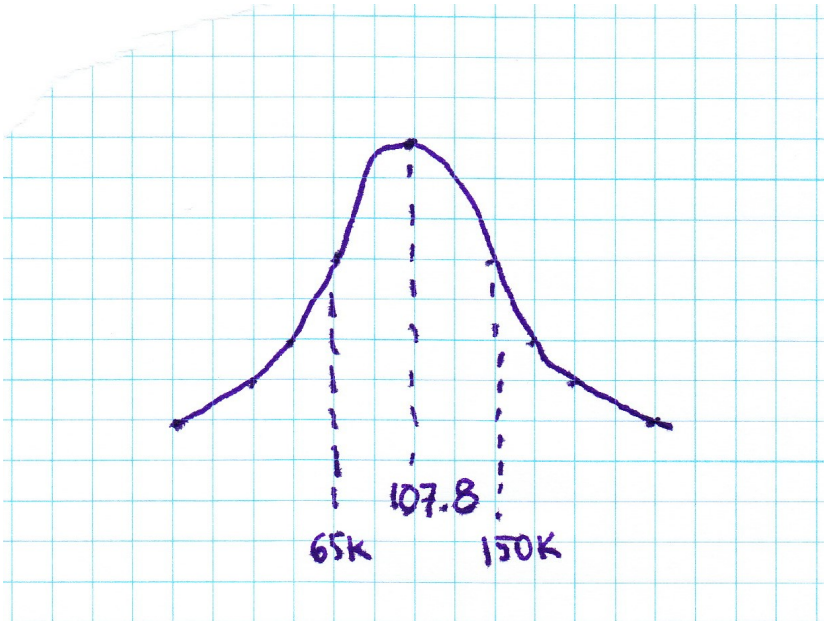
$$mean = \frac{90+125+100+220+80+110+95+80+70}{9} = \frac{970}{9} = 107.8$$

And perhaps after reading chapter 4 you might give the standard deviation:

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

Recall that the standard deviation describes the range of scattering. If all the values are bunched up around the mean, the standard deviation is small; if the values are scattered the standard deviation is large. In this example, the standard deviation of the income of i500 users is 42.69.

You probably have heard terms such as normal distribution, bell curve, and Gaussian distribution. Gaussian distribution is just a high falutin term for normal distribution. The function that describes this distribution is called the Gaussian function or bell curve. Most of the time the Numerati (aka data miners) assume attributes follow a Gaussian distribution. What is means is that about 68% of the instances in a Gaussian distribution fall within 1 standard deviation of the mean.



In our case, 68% of the i500 purchases earn between 65k and 150k. If I asked you if you thought P(100k| i500) (the likelihood that an i500 purchaser earns 100k) was, you might think that's pretty likely. If I asked you what you thought the likelihood of P(20k| i500) was, you might think it was pretty unlikely. To formalize this, we are going to use the mean and standard deviation to compute this probability as follows:

$$P(x_i|y_j)=\frac{1}{\sqrt{2\pi}\sigma_{ij}}\exp^{\frac{-(x_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

Let's dissect that. To compute something like P(100k| i500) represented as $P(x_i \mid y_j)$ …

The σ_{ij} is the standard deviation. σ_{ij}^2 is called the variance. μ_{ij} is the mean. So for the example p(100k| i500):

$$P(100\text{k}|i500)=\frac{1}{\sqrt{2\pi}(42.69)}\exp^{\frac{-(100-107.8)^2}{2(42.69)^2}}$$

$$P(100\text{k}|i500)=\frac{1}{77.31159}\exp^{\frac{-(60.84)}{3644.88}}=0.01315$$

So what is the likelihood of someone buying an i500 who makes 100k, exercies moderately, is moderately motivated, is comfortable with technology, and who wants the device for lealth reasons is

p(100|i500)		P(health|i500)		P(moderateEx|i500)		P(modMotiv|i500)		P(techComfy|i500)		P(i500)		
.01315	*	.444	*	.333	*	.333	*	.667	*	.6	=	.000259

Yeah, but why is it called naïve?

What enables us to multiple probabilities together is the fact that the events these probabilities represent are independent.

What is the probability I flip a coin and it is heads and then I roll a die and it is 6?

½ * 1/6 = .08333

Okay so I won't put my money on that event.

Let's say I alter a deck of cards keeping the black cards as in but only retaining the face cards for the red suits. What is the probability of selecting a face card?

P(facecard) = 12/32 = .375

The probability of selecting a red card?

P(red) = 6 /32 = .1875

What is the probability of selecting a single card that is both red and a face card?

Here we don't multiply probabilities:

P(red and facecard) = P(red) * P(facecard) = .375 * .185 = .0703

Here is what our common sense tells us. The chance of picking a red card is .1875. But if we pick a red card it is 100% likely it will be a facecard. So it seems that the probability of picking a card that is both red and a facecard is .1875.

Or we can start a different way. The probability of picking a facecard is .375. The way the deck is arranged half the facecards are red. So the probability of picking a card that is both red and a facecard is .375 * .5 = .1875.

Here we cannot multiply probabilities together because the attributes are not independent—if we pick red the probability of a facecard changes—and vice versa.

In many if not most real world data mining problems there are attributes that are not independent.

Consider the athlete data. Here we had 2 attributes weight and height. Weight and height are not independent. The taller you get the more likely you will be heavier.

Suppose I have attributes zip code, income, and age. These are not independent. Certain zipcodes have big bucks houses others consist of trailer parks. Palo Alto zipcodes may be dominated by 20-somethings—Arizona zipcodes may be dominated by retirees.

Think about the music attributes—things like amount of distorted guitar (1-5 scale), amount of classical violin sound. Here many of these attributes are not independent. If I have a lot of distorted guitar sound, the probability of having a classical violin sound decreases.

Suppose I have a dataset consisting of blood test results. Many of these values are not independent. For example, there are multiple thyroid tests including free

T4 and TSH. There is an inverse relationship between the values of these two tests.

Think about cases yourself. For example, consider attributes of cars. Are they independent? Attributes of a movie? Amazon purchases?

So, for Bayes to work we need to use attributes that are independent, but most real-world problems violate that condition. What we are going to do is just to assume that they are indepdent! We are using the magic wand of sweeping things under the rug—and ignore this problem. We call it **naïve Bayes** because we are naively assuming independence even though we know it is not. It turns out that naïve Bayes works really, really, well even with this naïve assumption.

A Challenge

Forthcoming ...