sharpen your pencil 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 ←

There were 6 occurrences of people buying i100s and only one of those people had a main interest of 'health'

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

sharpen your pencil solution

First we compute

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

which equals the product of all these terms:

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

so

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

Now we compute

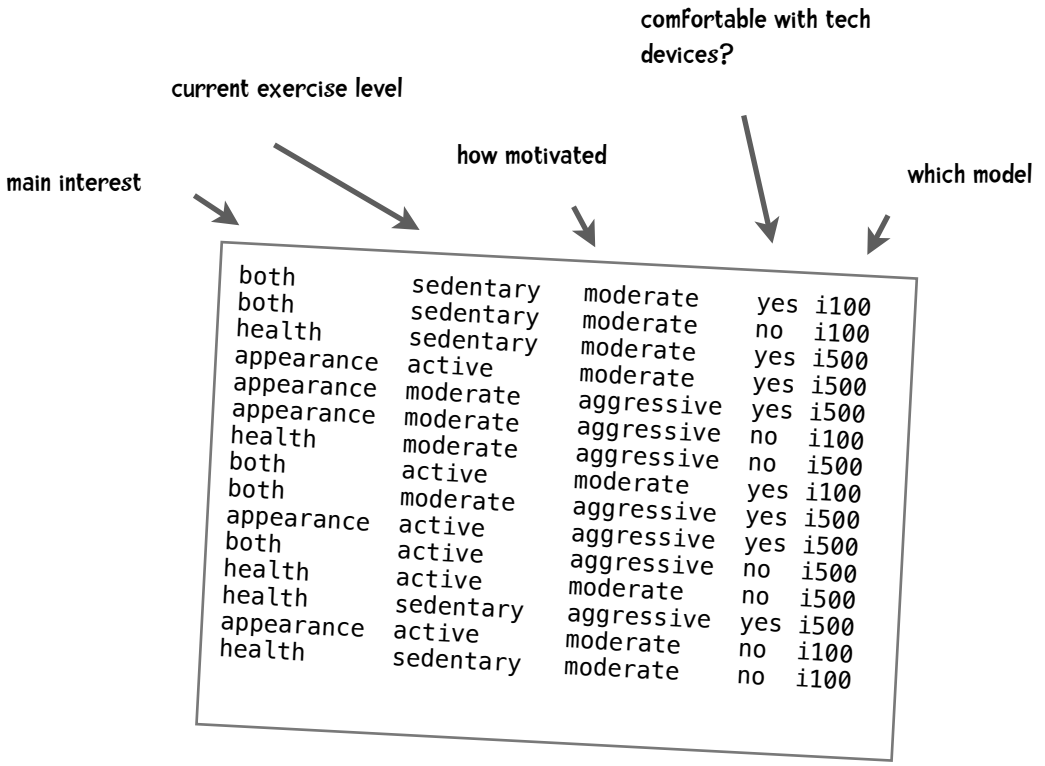
P(i500 | health, moderateExercise, moderateMotivation, techComfortable)

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**

Doing it in Python

Great! Now that we understand how a Naïve Bayes Classifier works let us consider how to implement it in Python. The format of the data files will be the same as that in the previous chapter, a text file where each line consists of tab-separated values. For our iHealth example, the data file would look like the following:



Shortly we will be using an example with substantially more data and I would like to keep the ten-fold cross validation methods we used in code from the previous chapter. Recall that that method involved dividing the data into ten buckets (files). We would train on nine of them and test the classifier on the remaining bucket. And we would repeat this ten times; each time withholding a different bucket for testing. The simple iHealth example, with only 15 instances, was designed so we could work through the Naïve Bayes Classifier method by hand. With only 15 instances it seems silly to divide them into 10 buckets. The ad hoc, not very elegant solution we will use, is to have ten buckets but all the 15 instances will be in one bucket and the rest of the buckets will be empty.

The Naïve Bayes Classifier code consists of two components, one for training and one for classifying.

Training

The output of training needs to be:

- a set of prior probabilities—for example, P(i100) = 0.4
- a set of conditional probabilities—for example, P(health|i100) = 0.167

I am going to represent the set of prior probabilities as a Python dictionary (hash table):

```
self.prior = {'i500': 0.6, 'i100': 0.4}
```

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.

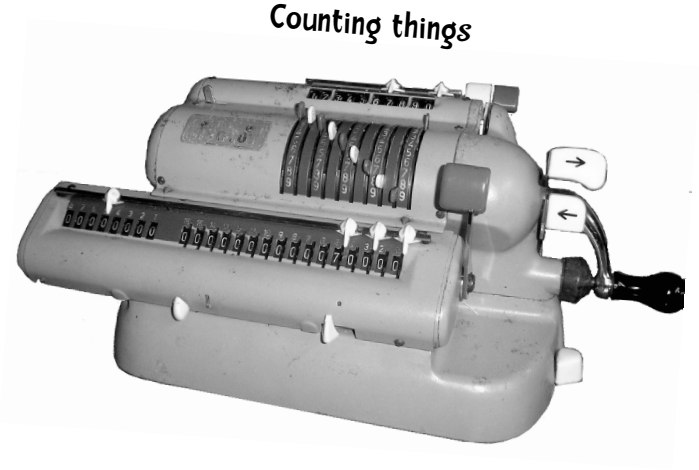
```
{'i500': {1: {'appearance': 0.3333333333333, 'health': 0.4444444444444,
              'both': 0.2222222222222},
          2: {'sedentary': 0.2222222222222, 'moderate': 0.333333333333,
              'active': 0.4444444444444444},
          3: {'moderate': 0.333333333333, 'aggressive': 0.66666666666},
          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}}}
```

The 1, 2, 3, 4 represent column numbers. So the first line of the above is "the probability of the value of the first column being 'appearance' given that the device is i500 is 0.333."

The first step in computing these probabilities is simply to count things. Here are the first few lines of the input file:

```
both        sedentary   moderate    yes i100
both        sedentary   moderate    no  i100
health      sedentary   moderate    yes i500
appearance  active      moderate    yes i500
```

Counting things



Yet again I am going to use dictionaries. One, called, `classes`, which will count the occurrences of each class or category. So, after the first line `classes` will look like

```
{'i100': 1}
```

After the second line:

```
{'i100': 2}
```

And after the third:

```
{'i500': 1, 'i100': 2}
```

Prior probability

After I process all the data, the value of `classes` is

```
{'i500': 9, 'i100': 6}
```

To obtain the prior probabilities I simply divide those number by the total number of instances.

Conditional probability

To determine the conditional probabilities I am going to count the occurrences of attribute values in the different columns in a dictionary called `counts`. and I am going to maintain separate counts for each class. So, in processing the string 'both' in the first line, `counts` will be:

```
{'i100': {1: {'both': 1}}
```

and at the end of processing the data, the value of counts will be

```
{'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}}}
```

So, in the first column of the i100 instances there were 2 occurrences of 'appearance', 1 of 'health' and 3 of 'both'. To obtain the conditional probabilities we divide those numbers by the total number of instances of that class. For example, there are 6 instances of i100 and 2 of them had a value of 'appearance' for the first column, so

P('appearance'|i100) = 2/6 = .333

With that background here is the Python code for training the classifier (remember, you can download this code at guidetodatamining.com).

```
# _____________________________________________________________________

class BayesClassifier:
    def __init__(self, bucketPrefix, testBucketNumber, dataFormat):

        """ a classifier will be built from files with the bucketPrefix
        excluding the file with textBucketNumber. dataFormat 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 = {}

        # 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 = []
            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 instance
            col = 0
            for columnValue in vector:
                col += 1
                counts[category].setdefault(col, {})
                counts[category][col].setdefault(columnValue,0)
                counts[category][col][columnValue] += 1
```

```
        #
        # 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(col, {})
                  for (attrValue, count) in valueCounts.items():
                      self.conditional[category][col][attrValue] = (
                          count / classes[category])
```

That's it for training! No Complex math. Just basic counting!!!

Classifying

Okay, we have trained the classifier. Now we want to classify various instances. For example, which model should we recommend for someone whose primary interest is health, and who is moderately active, moderately motivated, and is comfortable with technology:

```
c.classify(['health', 'moderate', 'moderate', 'yes'])
```

For this we need to compute

$$h_{MAP} = \arg\max_{h \in H} P(D \mid h)P(h)$$

When we did this by hand we computing the probability of each hypothesis given the evidence and we simply translate that method to code:

```
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[category][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])
```

And when I try the code I get the same results we received when we did this by hand:

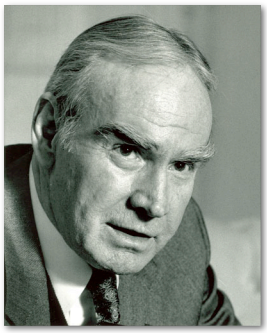
```
>>c = Classifier("iHealth/i", 10, "attr\tattr\tattr\tattr\tclass")
>>print(c.classify(['health', 'moderate', 'moderate', 'yes'])
i500
```

Republicans vs. Democrats

Let us look at a new data set, the Congressional Voting Records Data Set, available from the Machine Learning Repository (http://archive.ics.uci.edu/ml/index.html). It is available in a form that can be used by our programs at http://guidetodatamining.com. The data consists of the voting record of United States Congressional Representatives. The attributes are how that representative voted on 16 different bills.

Attribute Information:

1. Class Name: 2 (democrat, republican)
2. handicapped-infants: 2 (y,n)
3. water-project-cost-sharing: 2 (y,n)
4. adoption-of-the-budget-resolution: 2 (y,n)
5. physician-fee-freeze: 2 (y,n)
6. el-salvador-aid: 2 (y,n)
7. religious-groups-in-schools: 2 (y,n)
8. anti-satellite-test-ban: 2 (y,n)
9. aid-to-nicaraguan-contras: 2 (y,n)
10. mx-missile: 2 (y,n)
11. immigration: 2 (y,n)
12. synfuels-corporation-cutback: 2 (y,n)
13. education-spending: 2 (y,n)
14. superfund-right-to-sue: 2 (y,n)
15. crime: 2 (y,n)
16. duty-free-exports: 2 (y,n)
17. export-administration-act-south-africa: 2 (y,n)



The file consists of tab separated values:

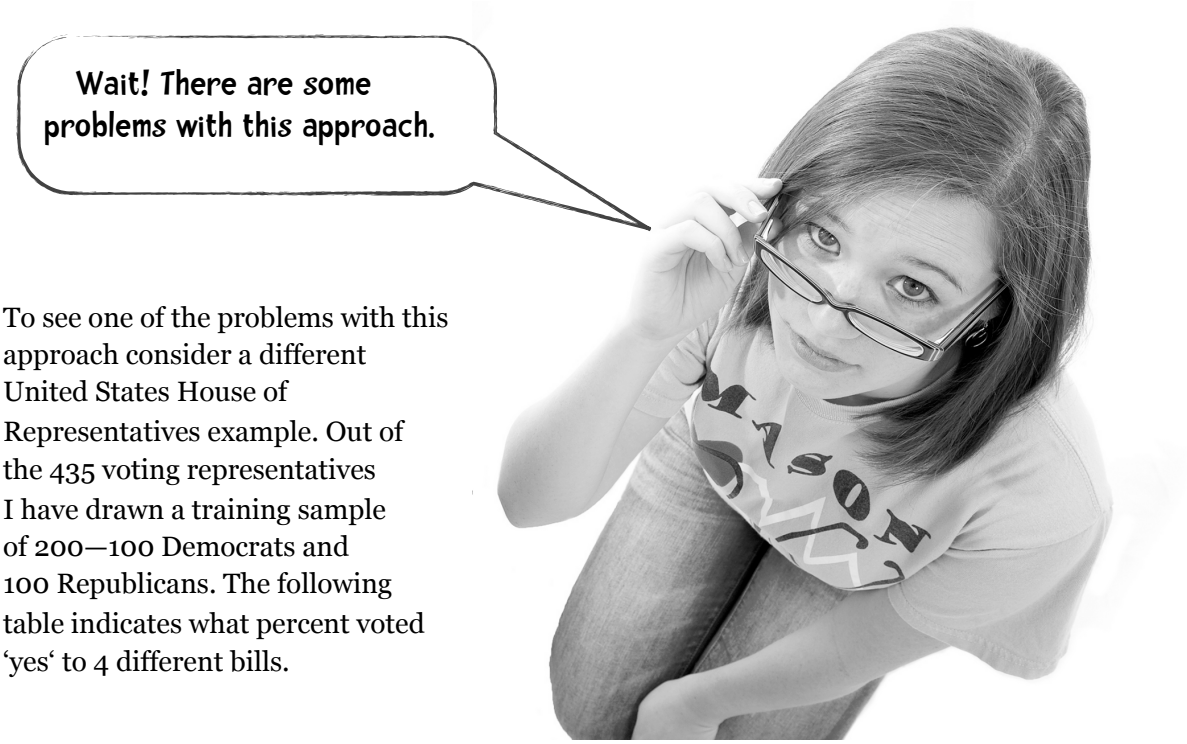
democrat	y	n	y	n	n	n	y	y	y	n	n	n	n	n	y	y
democrat	y	y	y	n	n	n	y	y	y	y	n	n	n	n	y	y
democrat	y	y	y	n	n	n	y	y	n	n	n	n	n	y	n	y
republican	y	y	y	n	n	y	y	y	y	y	n	n	n	n	n	y

Our Naïve Bayes Classifier works fine with this example (the format string says that the first column is to be interpreted as the class of the instance and the rest of the columns are to be interpreted as attributes):

```
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
```

That's great!



To see one of the problems with this approach consider a different United States House of Representatives example. Out of the 435 voting representatives I have drawn a training sample of 200—100 Democrats and 100 Republicans. The following table indicates what percent voted 'yes' to 4 different bills.

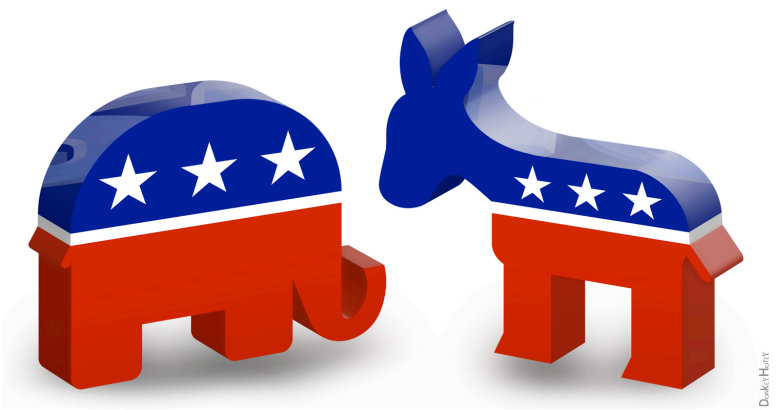
	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Republican	0.99	0.01	0.99	0.5
Democrat	0.01	0.99	0.01	1.0

% voting 'yes'

This table shows that 99% of Republicans in the sample voted for the CISPA (Cyber Intelligence Sharing and Protection Act), only 1% voted for the Reader Privacy Act, 99% voted for Internet Sales Tax and 50% voted for the Internet Snooping Bill. (I made up these numbers and they do not reflect reality.) We pick a U.S. Representative who wasn't in our sample, Representative X, who we would like to classify as either a Democrat or Republican. I added how that representative voted to our table:

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Republican	0.99	0.01	0.99	0.5
Democrat	0.01	0.99	0.01	1.0
Rep. X	N	Y	N	N

Do you think the person is a Democrat or Republican?



I would guess Democrat. Let us work through the example step-by-step using Naïve Bayes. The prior probabilities of *P*(*Democrat*) and *P*(*Republican*) are both 0.5 since there are 100 Republicans and 100 Democrats in the sample. We know that Representative X voted 'no' to CISPA and we also know

P(Republican|C=no) = 0.01 and P(Democrat|C=no) = 0.99

where C = CISPA. And with that bit of evidence our current P(h|D) probabilities are

h=	p(h)	P(C=no|h)				P(h|D)
Republican	0.5	0.01				0.005
Democrat	0.5	0.99				0.495

Factoring in Representative X's 'yes' vote to the Reader Privacy Act and X's 'no' to the sales tax bill we get:

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

If we normalize these probabilities:

$$P(Democrat \mid D) = \frac{0.485}{0.485 + 0.0000005} = \frac{0.485}{0.4850005} = 0.99999$$

So far we are 99.99% sure Representative X is a Democrat.

Finally, we factor in Representative X's 'no' vote on the Internet Snooping Bill.

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

Whoops. We went from 99.99% likelihood that X was a Democrat to 0%. This is so because we had 0 occurrences of a Democrat voting 'no' for the snooping bill. Based on these probabilities we predict that person X is a Republican. This goes against our intuition!

Estimating Probabilities

The probabilities in Naïve Bayes are really **estimates** of the true probabilities. True probabilities are those obtained from the entire population. For example, if we could give a cancer test to everyone in the entire population, we could, for example, get the true probability of the test returning a negative result given that the person does not have cancer. However, giving the test to everyone is near impossible. We can estimate that probability by selecting a random representative sample of the population, say 1,000 people, giving the test to them, and computing the probabilities. Most of the time this gives us a very good estimate of the true probabilities, but when the true probabilities are very small, these estimates are likely to be poor. Here is an example. Suppose the true probability of a Democrat voting no to the Internet Snooping Bill is 0.03—$P(S=no|Democrat) = 0.03$.

Brain Calisthenics

Suppose we try to estimate these probabilities by selected a sample of 10 Democrats and 10 Republicans. What is the most probable number of Democrats in the sample that voted no to the snooping bill?

☐0 ☐1 ☐2 ☐3

Brain Calisthenics—answer

Suppose we try to estimate these probabilities by selected a sample of 10 Democrats and 10 Republicans. What is the most probable number of Democrats in the sample that voted no to the snooping bill?

0

So based on the sample P(S=no|Democrat) = 0.

As we just saw in the previous example, when a probability is 0 it dominates the Naïve Bayes calculation—it doesn't matter what the other values are. Another problem is that probabilities based on a sample produce a biased underestimate of the true probability.

Fixing this.

If we a trying to calculate something like P(S=no|Democrat) our calculation has been

$$P(\text{S=no}|\text{Democrat}) = \frac{\text{the number that both are Democrats and voted no on the snooping bill.}}{\text{total number of Democrats}}$$

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

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

Here n is the total number of instances of 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:

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

This formula is from p179 of the book "Machine Learning" by Tom Mitchell.

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, there are 2 values for how a person voted on the snooping bill, *yes*, or *no*. So I will use an m of 2. p is the prior estimate of the probability. Often we assume uniform probability. For example, what is the probability of someone voting no to the snooping bill knowing nothing about that person? 1/2. So p in this case is 1/2.

Let's go through the previous example to see how this works. First, here are tables showing the vote:

Republican Vote

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Yes	99	1	99	50
No	1	99	1	50

Democratic Vote

	CISPA	Reader Privacy Act	Internet Sales Tax	Internet Snooping Bill
Yes	1	99	1	100
No	99	1	99	0

The person we are trying to classify voted no to CISPA. First we compute the probability that he's a Republican given that vote. Our new formula is

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

n is the number of Republicans, which is 100 and n_c is the number of Republicans who voted no to CISPA, which is 1. m is the number of values for the attribute "how they voted on CISPA", which is 2 (*yes* or *no*). So plugging those number into our formula

$$P(cispa = no \mid republican) = \frac{1 + 2(.5)}{100 + 2} = \frac{2}{102} = 0.01961$$

We follow the same procedure for a person voting no to CISPA given they are a Democrat.

$$P(cispa = no \mid democrat) = \frac{99 + 2(.5)}{100 + 2} = \frac{100}{102} = 0.9804$$

With that bit of evidence our current P(h|D) probabilities are

h=	p(h)	P(C=no|h)				P(h|D)
Republican	0.5	0.01961				0.0098
Democrat	0.5	0.9804				0.4902

Factoring in Representative X's 'yes' vote to the Reader Privacy Act and X's 'no' to the sales

sharpen your pencil

Finish this problem and classify the individual as either a Republican or Democrat.

Recall, he voted no to Cispa, yes to the Reader Privacy act, and no both to the sales tax and snooping bills.

sharpen your pencil -answer

Finish this problem and classify the individual as either a Republican or Democrat.

Recall, he voted no to CISPA, yes to the Reader Privacy act, and no both to the Internet sales tax and snooping bills.

The calculations for the next 2 columns mirror those we did for the CISPA vote. The probability that this person voted no to the snooping bill given that he is a Republican is

$$P(s = no \mid republican) = \frac{50 + 2(.5)}{100 + 2} = \frac{51}{102} = 0.5$$

and that he voted no given that he is a Democrat:

$$P(s = no \mid democrat) = \frac{0 + 2(.5)}{100 + 2} = \frac{1}{102} = 0.0098$$

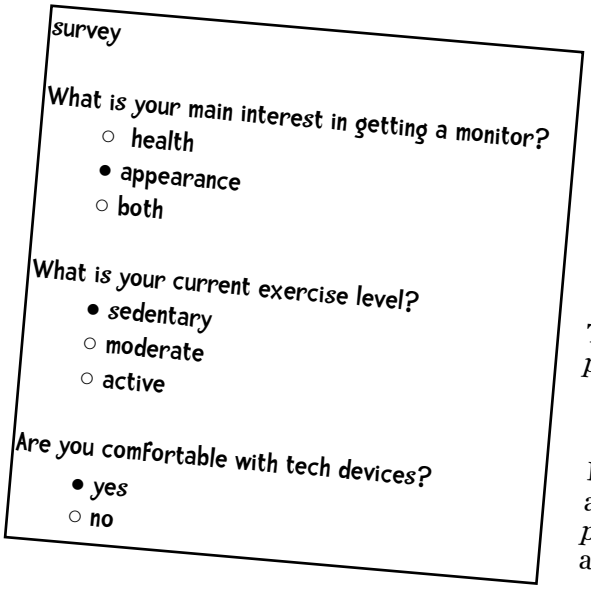
Multiplying those probabilities together gives us

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

So unlike the previous approach we would classify this individual as a Democrat. This matches our intuitions.

A clarification

For this example, the value of m was 2 for all calculations. However, it is not the case that m remains necessarily constant across attributes. Consider the health monitor example discussed earlier in the chapter. The attributes for that example included:



For this attribute, $m = 3$ since the attribute can take one of 3 values (health, appearance, both). If we assume uniform probabilities, then $p = 1/3$ since the probability of the attribute being any one of the values is

This attribute also has $m = 3$ and $p = 1/3$

For this attribute, $m = 2$ since the attribute can take one of 2 values and $p = 1/2$ since the probability of the attribute being any one of those is 1/2

Let us say the number of the people surveyed who own the i500 monitor is 100 (this is n). The number of people who own a i500 and are sedentary is 0 (n_c). So, the probability of someone being sedentary given they own an i500 is

$$P(sedentary \mid i500) = \frac{n_c + mp}{n + m} = \frac{0 + 3(.333)}{100 + 3} = \frac{1}{103} = 0.0097$$

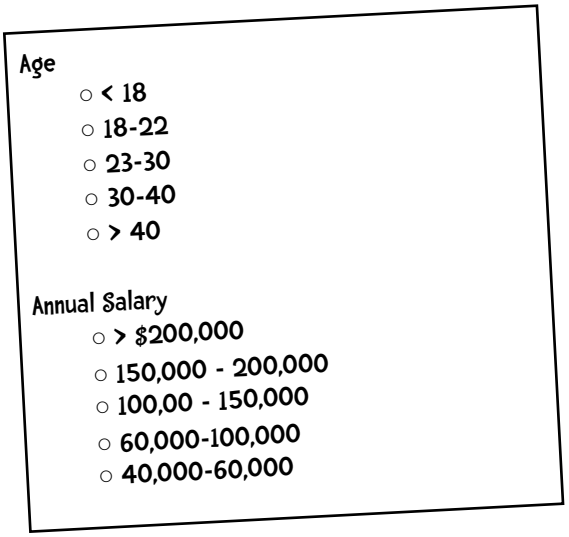
Numbers

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. By "categorical data" we mean that the data is put into discrete categories. For example, we divide people up in how they voted for a particular bill and the people who voted 'yes' go in one category and the people who voted 'no' go in another. Or we might categorize musicians by the instrument they play. So all saxophonists go in one bucket, all drummers in another, all pianists in another and so on. And these categories do not form a scale. So, for example, saxophonists are not 'closer' to pianists than they are to drummers. Numerical data is on a scale. An annual salary of $105,000 is closer to a salary of $110,000 than it is to one of $40,000.

For Bayesian approaches we count things—how many occurrences are there of people who are sedentary—and it may not be initially obvious how to count things that are on a scale—for example, something like grade point average. There are two approaches.

Method 1: Making categories

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



Once we have this information divided nicely into discrete values, we can use Naïve Bayes exactly as we did before.

Method 2: Gaussian distributions!

The terms "Gaussian Distribution" and "Probability Density Function" sound cool, but they are more than good phrases to know so you can impress your friends at dinner parties. So what do they mean and how they can be used with the Naïve Bayes method? 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 (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

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

$$mean = \frac{90+125+100+150+100+120+95+90+85}{9} = \frac{955}{9} = 106.111$$

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

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

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

sharpen your pencil

What is the income standard deviation of the people who bought the i500? (those values are shown in the column below)

Income (in $1,000)
90
125
100
150
100
120
95
90
85

sharpen your pencil - solution

What is the standard deviation of the income of the people who bought the i500? (those values are shown in the column above)

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

$\sum$ = 3638.889

$$sd = \sqrt{\frac{3638.889}{9}}$$

$$= \sqrt{404.321} = 20.108$$

Population standard deviation and sample standard deviation.

The formula for standard deviation that we just used is called the population standard deviation. It is called that because we use this formula when we have data on the entire population we are interested in. For example, we might give a test to 500 students and then compute the mean and standard deviation. In this case, we would use the population standard deviation, which is what we have been using. Often, though, we do not have data on the entire population. For example, suppose I am interested in the effects of drought on the deer mice in Northern New Mexico and as part of that study I want the average (mean) and standard deviation of their weights. In this case I am not going to weigh every mouse in Northern New Mexico. Rather I will collect and weigh some representative sample of mice.

For this sample, I can use the standard deviation formula I used above, but there is another formula that has been shown to be a better estimate of the entire population standard deviation. This formula is called the **sample standard deviatio**n and it is just a slight modification of the previous formula:

$$sd = \sqrt{\frac{\sum_i (x_i - \overline{x})^2}{card(x) - 1}}$$

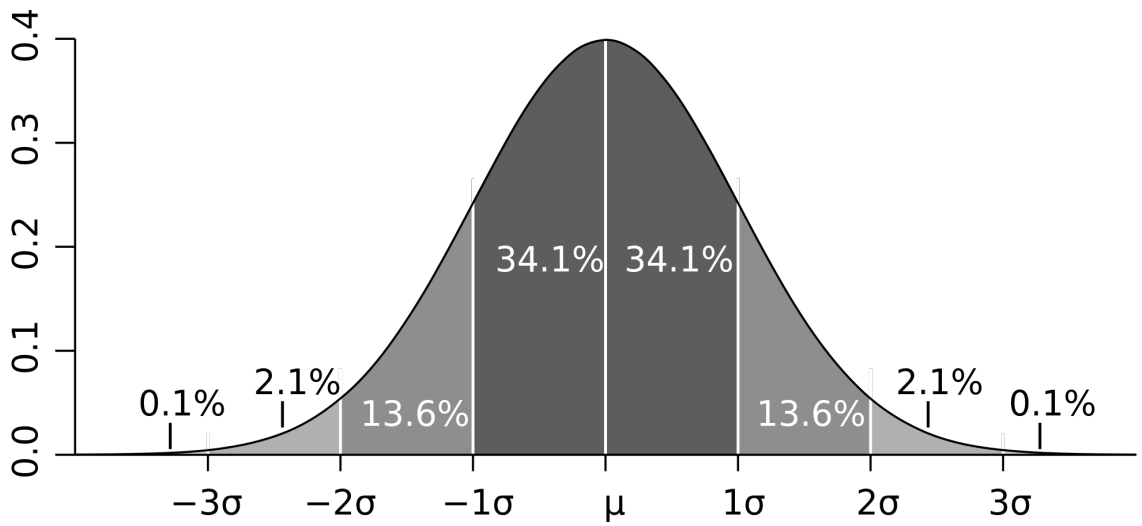
The sample standard deviation of the income example is

$$sd = \sqrt{\frac{3638.889}{9-1}} = \sqrt{\frac{3638.889}{8}}$$

$$= \sqrt{454.861} = 21.327$$

For the rest of this chapter we will be using sample standard deviation.

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 and 95% of the instances fall within 2 standard deviations of the mean:



In our case, the mean was 106.111 and the sample standard deviation was 21.327. So 95% of the people who purchase an i500 earn between $42,660 and $149,770. If I asked you if you thought P(100k| i500) —the likelihood that an i500 purchaser earns $100,000—was, you might think that's pretty likely. If I asked you what you thought the likelihood of P(20k| i500)—the likelihood that an i500 purchaser earns $20,000—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:

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

Maybe putting the formula in a bigger font makes it look simpler!

Everytime I type a complex looking formula into this book, I feel the need to say something like "don't panic." It could be that none of you readers panic and I am just the one panicking.

However, let me say this. Data mining has professional terminology and formulas. Before you dive into data mining you might think "those things look difficult." But after you study, even for a short time, these formulas become nothing special. It is just a matter of working through the formula out step-by-step.

Let's jump right into dissecting this formula so we can see how simple it really is. Let us say we are interested in computing P(100k|i500), the probability that a person earns $100,000 (or 100k) given they purchased an i500. A few pages ago we computed the average income (mean) of people who bought the i500. We also computed the sample standard deviation. These values are shown on the following page. In Numerati speak, we represent the mean with the Greek letter μ (mu) and the standard deviation as σ (sigma).

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

$\mu_{ij} = 106.111$
$\sigma_{ij} = 21.327$
$x_i = 100$

Let's plug these values into the formula:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{2\pi}(21.327)} e^{\frac{-(100-106.111)^2}{2(21.327)^2}}$$

and do some math:

$$P(x_i \mid y_j) = \frac{1}{\sqrt{6.283}(21.327)} e^{\frac{-(37.344)}{909.68}}$$

and more math:

$$P(x_i \mid y_j) = \frac{1}{53.458} e^{-0.0411}$$

The *e* is a mathematical constant that is the base of the natural logarithm. It's value is approximately 2.718.

$$P(x_i \mid y_j) = \frac{1}{53.458}(2.718)^{-0.0411} = (0.0187)(0.960) = 0.0180$$

So the probability that the income of a person who bought the i500 is $100,000 is 0.0180.

sharpen your pencil

In the table below I have the horsepower ratings for cars that get 35 miles per gallon. I would like to know the probability of a Datsun 280z having 132 horsepower given it gets 35 miles per gallon.

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 = _____

sharpen your pencil -solution - part 1

In the table below I have the horsepower ratings for cars that get 35 miles per gallon. I would like to know the probability of a Datsun 280z having 132 horsepower given it gets 35 miles per gallon.

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} = 72.875

σ_{ij} = 9.804

x_i =132

$$\sigma = \sqrt{\frac{(65-\mu)^2 +(66-\mu)^2 + (74-\mu)^2 + (88-\mu)^2 + (65-\mu)^2 + (88-\mu)^2 + (70-\mu)^2 + (67-\mu)^2}{7}}$$

$$\sigma = \sqrt{\frac{672.875}{7}} = \sqrt{96.126} = 9.804$$

sharpen your pencil -solution - part 2

In the table below I have the horsepower ratings for cars that get 35 miles per gallon. I would like to know the probability of a Datsun 280z having 132 horsepower given it gets 35 miles per gallon.

μ_{ij} = 72.875

σ_{ij} = 9.804

x_i =132

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

$$P(132hp \mid 35mpg) = \frac{1}{\sqrt{2\pi}(9.804)} e^{\frac{-(132-72.875)^2}{2(9.804)^2}}$$

$$= \frac{1}{\sqrt{6.283}(9.804)} e^{\frac{-3495.766}{192.237}} = \frac{1}{24.575} e^{-18.185}$$

$$= 0.0407(0.00000001266)$$

$$= 0.0000000005152$$

Ok. it is extremely unlikely that a Datsun 280z, given that it gets 35 miles to the gallon has 132 horsepower. (but it does!)

A few implementation notes.

In the training phase for Naive Bayes, we will compute the mean and sample standard deviation of every numeric attribute. Shortly, we will see how to do this in detail.

In the classification phase, the above formula can be implemented with just a few lines of Python:

```
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
```

Let's test this with the examples we did above:

```
>>>pdf(106.111, 21.327, 100)
0.017953602706962717

>>>pdf(72.875, 9.804, 132)
5.152283971078022e-10
```

Whew! Time for a break!

Python Implementation

Training Phase

The Naïve Bayes method relies on prior and conditional probabilities. Let's go back to our Democrat/Republican example. Prior probabilities are the probabilities that hold before we have observed any evidence. For example, if I know there are 233 Republicans and 200 Democrats, then the prior probability of some arbitrary member of the U.S. House of Representatives being a Republican is

$$P(republican) = \frac{233}{433} = 0.54$$

This is denoted $P(h)$. Conditional Probability $P(h|D)$ is the probability that h holds given that we know D, for example, $P(democrat|bill1Vote=yes)$. In Naïve Bayes, we flip that probability and compute P(D|h)—for example, $P(bill1Vote=yes|democrat)$.

In our existing Python program we store these conditional probabilities in a dictionary of the following form:

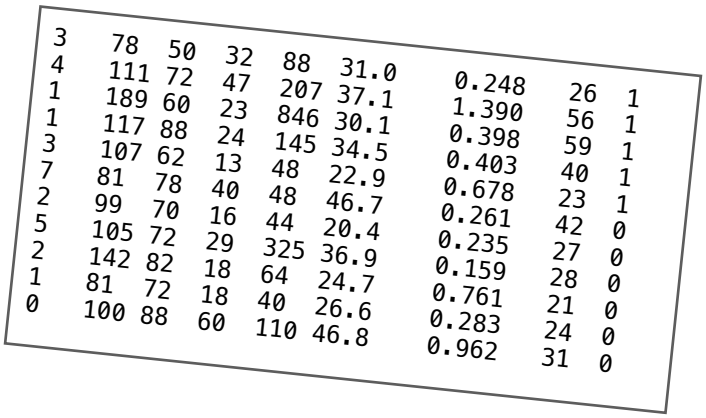
```
{'democrat': {'bill 1': {'yes': 0.333, 'no': 0.667},
              'bill 2': {'yes': 0.778, 'moderate': 0.222}}

 'republican': {'bill 1': {'yes': 0.811, 'no': 0.189},
                'bill 2': {'yes': 0.250, 'no': 0.750}}}
```

So the probability that someone voted yes to bill 1 given that they are a Democrat ($P(bill\ 1=yes|democrat)$) is 0.333.

We will keep this data structure for attributes whose values are discrete values (for example, 'yes', 'no', 'sex=male', 'sex=female'). However, when attributes are numeric we will be using the probability density function and we need to store the mean and sample standard deviation for that attribute. For these numeric attributes I will use the following structures:

```
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}}
```

As before, each instance is represented by a line in a data file. The attributes of each instances are separated by tabs. For example, the first few lines of a data file for the Pima Indians Diabetes Data set might be:



The columns represent, in order, the number of times pregnant, plasma glucose concentration, blood pressure, triceps skin fold thickness, serum insulin level, body mass index, diabetes pedigree function, age, and a '1' in the last column represents that they developed diabetes and a '0' they did not.

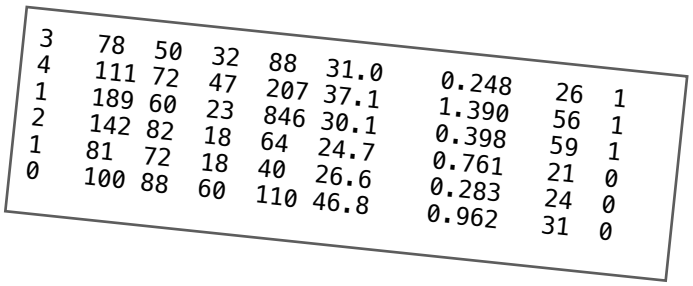
Also as before, we are going to represent how the program should interpret each column by use of a format string, which uses the terms

- `attr` identifies columns that should be interpreted as non-numeric attributes, and which will use the Bayes methods shown earlier in this chapter.
- `num` identifies columns that should be interpreted as numeric attributes, and which will use the Probability Density Function (so we will need to compute the mean and standard deviation during training).
- `class` identifies the column representing the class of the instance (what we are trying to learn)

In the Pima Indian Diabetes data set the format string will be

```
"num    num     num     num    num    num    num    num    class"
```

To compute the mean and sample standard deviation we will need some temporary data structures during the training phase. Again, let us look at a small sample of the Pima data set.



```
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
```

The last column represents the class of each instance. So the first three individuals developed diabetes and that last three did not. All the other columns represent numeric attributes. of which we need to compute the mean and standard deviation for each of the two classes. To compute the mean for each class and attribute I will need to keep track of the running total. In our existing code we already keep track of the total number of instances. I will implement this using a dictionary:

```
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}
```

So for class 1, the column 1 total is 8 (3 + 4 + 1), the column 2 total is 378, etc.

For class 0, the column 1 total is 3 (2 + 1 + 0), the column 2 total is 323 and so on.

For standard deviation, we will also need to keep the original data, and for that we will use a dictionary in the following format:

`numericValues`

```
{'1': 1: [3, 4, 1], 2: [78, 111, 189], ...},
{'0': {1: [2, 1, 0], 2: [142, 81, 100]}
```

I have added the code to create these temporary data structures to the `__init__()` method of our Classifier class as shown below:

```
import math

class Classifier:
    def __init__(self, bucketPrefix, testBucketNumber, dataFormat):

        """ a classifier will be built from files with the bucketPrefix
        excluding the file with textBucketNumber. dataFormat 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 numeric
        counts = {}
        # totals used for attributes that are numereric
        # we will use these to compute the mean and sample 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 attribute of the instance
    col = 0
    for columnValue in vector:
        col += 1
        counts[category].setdefault(col, {})
        counts[category][col].setdefault(columnValue, 0)
        counts[category][col][columnValue] += 1
    # process numeric attributes
    col = 0
    for columnValue in nums:
        col += 1
        totals[category].setdefault(col, 0)
        #totals[category][col].setdefault(columnValue, 0)
        totals[category][col] += columnValue
        numericValues[category].setdefault(col, [])
        numericValues[category][col].append(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(col, {})
                  for (attrValue, count) in valueCounts.items():
                      self.conditional[category][col][attrValue] = (
                          count / classes[category])
        self.tmp =  counts
        #
        # now compute mean and sample standard deviation
        #
```

code it

Can you add the code to compute the means and standard deviations? Download the file naiveBayesDensityFunctionTraining.py from guidetodatamining.com.

Your program should produce the data structures ssd and means:

```
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, ...}}
```

code it solution

Here is my solution:

```
        #
        # 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[category]
        # 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(SumOfSquareDifferences
                                            / (classes[category]  - 1))
```

The file containing this solution is naiveBayesDensityFunctionTrainingSolution.py at our website.

code it 2

Can you revise the classify method so it uses the probability density function for numeric attributes? The file to modify is naiveBayesDensityFunctionTemplate.py. Here is the original classify method:

```
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[category][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
        # return the category with the highest probability
        #print(results)
        return(max(results)[1])
```

code it 2 - solution

Can you revise the classify method so it uses the probability density function for numeric attributes? The file to modify is naiveBayesDensityFunctionTemplate.py.

Solution:

```
  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[category][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
            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)) * ePart)
                col += 1
            results.append((prob, category))
        # return the category with the highest probability
        #print(results)
        return(max(results)[1])
```

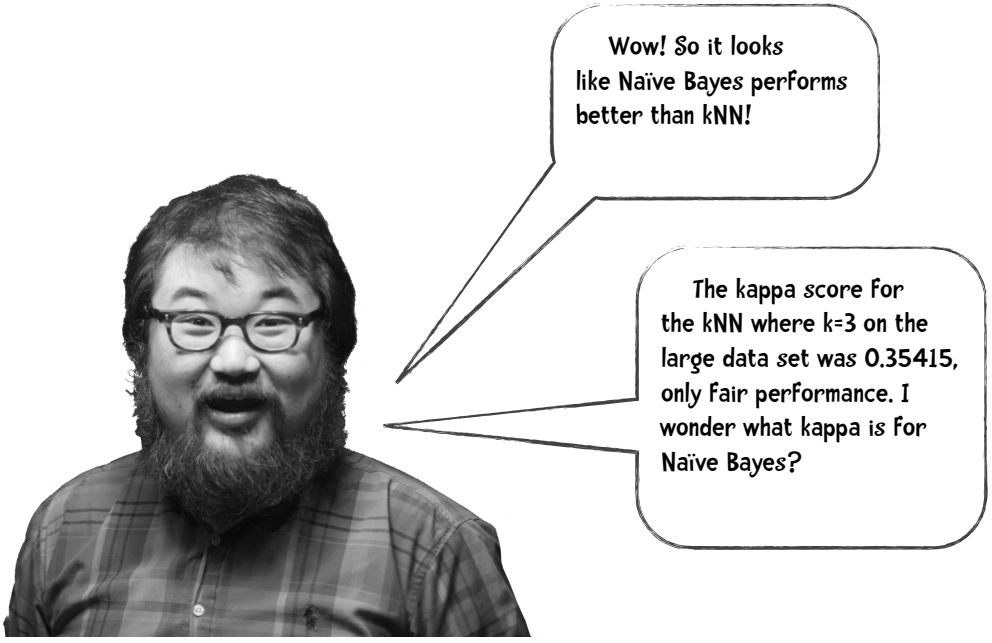
Is this any better than the Nearest Neighbor Algorithm?

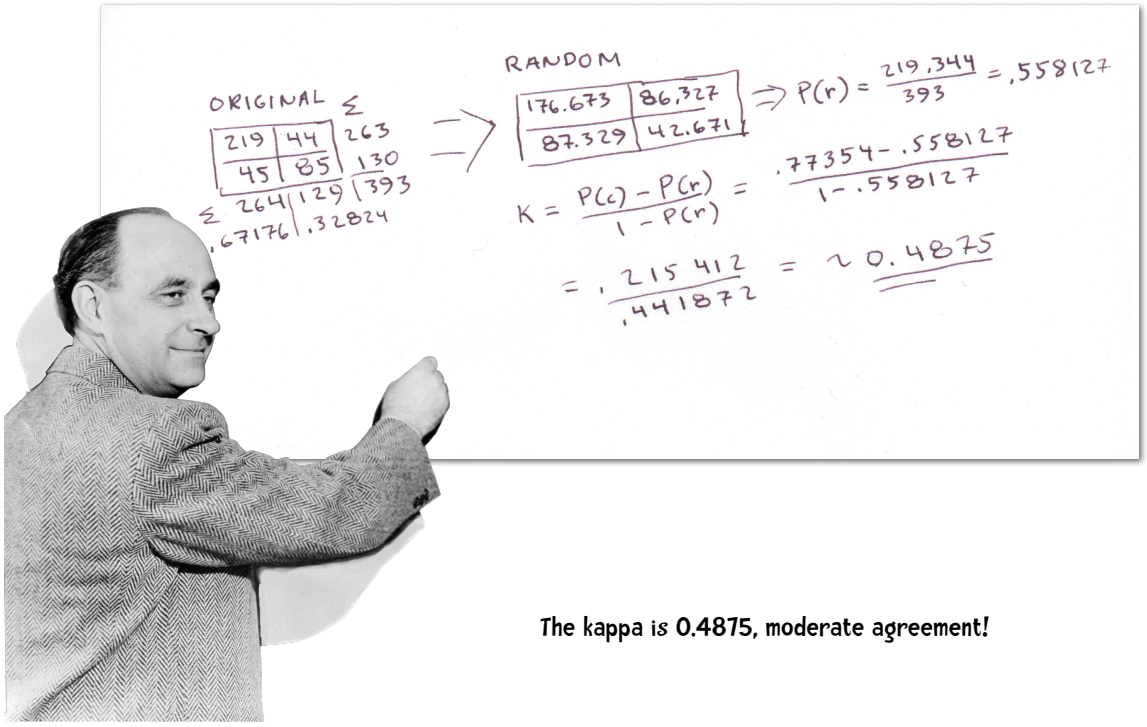
In Chapter 5 we evaluated how well the k Nearest Neighbor algorithm did with both the total Pima data set and a subset. Here are those results:

	pimaSmall	pima
k=1	59.00%	71.247%
k=3	61.00%	72.519%

Here are the results when we use Naïve Bayes with these two data sets:

	pimaSmall	pima
Bayes	72.000%	77.354%





The kappa is 0.4875, moderate agreement!

So for this example, Naïve Bayes is better than k

Advantages of Bayes

- simple to implement (just counting things)
- need less training data than many other methods
- a good method to use if you want something that performs well and has good performance times.

Main disadvantage of Bayes:

It cannot learn interactions among features. For example, it cannot learn that I like foods with cheese and I like foods with rice but I do not like foods with both

Advantages of kNN

- simple to implement.
- does not assume the data has any particular structure—a good thing!
- large amount of memory needed to store the training set.

kNN

k Nearest Neighbors is a reasonable choice when the training set is large. kNN is extremely versatile and used in a large number of fields including recommendation systems, proteomics (the study of the entire protein set of an organism), the interaction among proteins, and image classification.

What enables us to multiple probabilities together is the fact that the events these probabilities represent are independent. For example, consider a game where we flip a coin and roll a die. These events are independent meaning what we roll on the die does not depend on whether we flip a heads or tails on the coin. And, as I just said, if events are independent we can determine their joint probability (the probability that they both occurred) by multiplying the individual probabilities together. So the probability of getting a heads and rolling a 6 is

$$P(heads \wedge 6) = P(heads) \times P(6) = 0.5 \times \frac{1}{6} = 0.08333$$

Let's say I alter a deck of cards keeping all the black cards (26 of them) but only retaining the face cards for the red suits (6 of them). That makes a 32 card deck. What is the probability of selecting a face card?

$$P(facecard) = \frac{12}{32} = 0.375$$

The probability of selecting a red card is

$$P(red) = \frac{6}{32} = 0.1875$$

What is the probability of selecting a single card that is both red and a face card? Here we do not multiply probabilities. We **do not** do

$$P(red \wedge facecard) = P(red) \times P(facecard) = 0.375 \times 0.185 = 0.0703$$

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 face card. So it seems that the probability of picking a card that is both red and a face card is .1875.

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

Here we cannot multiply probabilities together because the attributes are not independent—if we pick red the probability of a face card 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 independent! We are using the magic wand of sweeping things under the rug™—and ignoreing this problem. We call it *naïve Bayes* because we are naïvely assuming independence even though we know it is not. It turns out that naïve Bayes works really, really, well even with this naïve assumption.

code it

Can you run the naïve Bayes code on our other data sets? For example, our kNN algorithm was 53% accurate on the auto MPG data set. Does a Bayes approach produce better results?

```
tenfold("mpgData/mpgData", "class attr  num  num  num  num  comment")
```

?????

Classifying unstructured text

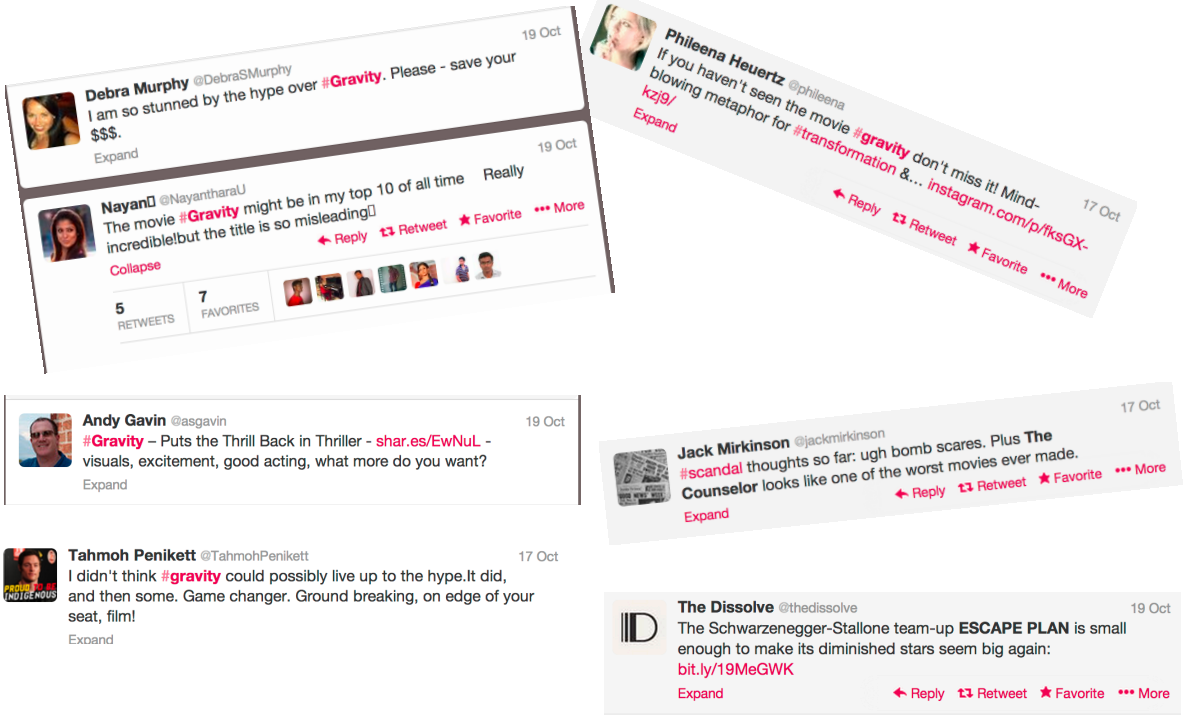
In previous chapters we've looked at recommendation systems that have people explicitly rate things with star systems (5 stars for Phoenix), thumbs-up/thumbs-down (Inception--thumbs-up!), and numerical scales. We've looked at implicit things like the behavior of people—did they buy the item, did they click on a link. We have also looked at classification systems that use attributes like height, weight, how people voted on a particular bill. In all these cases the information in the datasets can easily be represented in a table.

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

This type of data is called "structured data"—data where instances (rows in the tables above) are described by a set of attributes (for example, a row in a table might describe a car by a set of attributes including miles per gallon, the number of cylinders and so on). Unstructured data includes things like email messages, twitter messages, blog posts, and newspaper articles. These types of things (at least at first glance) do not seem to be neatly represented in a table.

For example, suppose we are interested in determining whether various movies are good or not good and we want to analyze Twitter messages:



We, as speakers of English can see that Andy Gavin likes *Gravity*, since he said "puts the thrill back in thriller" and "good acting." We know that Debra Murphy seems not so excited about the movie since she said "save your $$$." And if someone writes "I wanna see Gravity sooo bad, we should all go see it!!!" that person probably likes the movie even though they used the word *bad*.

Suppose I am at my local food co-op and see something called Chobani Greek Yogurt. It looks interesting but is it any good? I get out my iPhone, do a google search and find the following from the blog "Woman Does Not Live on Bread Alone":

Chobani nonfat greek yogurt.

Have you ever had greek yogurt? If not, stop reading, gather your keys (and a coat if you live in New York) and get to your local grocery. Even when nonfat and plain, greek yogurt is so thick and creamy, I feel guilty whenever I eat it. It is definitely what yogurt is MEANT to be. The plain flavor is tart and fantastic. Those who can have it, try the honey version. There's no sugar, but a bit of honey for a taste of sweetness (or add your own local honey-- local honey is good for allergies!). I must admit, even though I'm not technically supposed to have honey, if I've had a bad day, and just desperately need sweetness, I add a teaspoon of honey to my yogurt, and it's SO worth it. The fruit flavors from Chobani all have sugar in them, but fruit is simply unnecessary with this delicious yogurt. If your grocery doesn't carry the Chobani brand, Fage (pronounced Fa-yeh) is a well known, and equally delicious brand.

Now, for Greek yogurt, you will pay about 50 cents to a dollar more, and there are about 20 more calories in each serving. But it's worth it, to me, to not feel deprived and saddened over an afternoon snack!

http://womandoesnotliveonbreadalone.blogspot.com/2009/03/sugar-free-yogurt-reviews.html

Is that a positive or negative review for Chobani? Even based on the second sentence: *If not, stop reading, gather your keys … and get to your local grocery store,* it seems that this will be a positive review. She describes the flavor as *fantastic* and calls the yogurt *delicious*. It seems that I should buy it and check it out. I will be right back…

An automatic *system* for determining positive and negative texts.

John, that looks like a positive tweet for Gravity!

Let's imagine an automatic system that can read some text and decide whether it is a positive or negative report about a product. Why would we want such a system? Suppose there is a company that sells health monitors, they might want to know about what people are saying about their products. Are what people say mostly positive or negative? They release an ad campaign for a new product. Are people favorable about the product (*Man, I sooo want this!*) or negative (*looks like crap*). Apple has a press conference to talk about the iPhone problems. Is the resulting press coverage positive? A Senate candidate delivers a major policy speech—do the political bloggers view it favorably? So an automatic system does sound useful.