5. Naive Bayes

5.1 Overview

The mathematical foundations of Naive Bayes go back to the 18th Century and the mathematician and minister, Thomas Bayes, who formalized this probabilistic equation that bears his name. Bayes theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \quad aka: \quad \textbf{posterior} = \frac{\textbf{likelihood} \times \textbf{prior}}{\textbf{marginal}} \tag{5.1}$$

Let's consider the above equation in terms of the Titanic data.

$$P(survived|data) = \frac{P(data|survived)P(survived)}{P(data)} \tag{5.2}$$

The quantity $P(data|survived)$ is called the **likelihood**. It is calculated from the training data by determining the joint probabilities of Survived and the Data. It quantifies how likely it is that we would see the data given the Survived instances. The quantity $P(survived)$ is called the **prior** and the distribution of this data is also learned from the training set. The denominator $P(data)$ is used to normalize the fraction to a probability in the range 0 to 1. It is also called the marginal. The quantity of the left is called the **posterior** and it will be the probability of the positive class for a given observation.

5.2 Naive Bayes in R

We will apply the naive Bayes algorithm to the same Titanic data set as we applied logistic regression in the previous chapter. The same steps were used for data cleaning and creating the train/test split, so we will skip those here. In the code below we see that we first load package e1071 which contains the naiveBayes() function. Then we call Naive Bayes on the training data using a formula the same as other algorithms we have used. When we type the name of the model we have built, the information below the code is output.

Code 5.2.1 — Naive Bayes. Requires package e1071.

```
library(e1071)
nb1 <- naiveBayes(survived~., data=train)
nb1
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        0         1
0.6146789 0.3853211

Conditional probabilities:
   pclass
Y           1         2         3
  0 0.1409619 0.1840796 0.6749585
  1 0.4047619 0.2222222 0.3730159

   sex
Y                female      male
  0 0.0000000 0.1708126 0.8291874
  1 0.0000000 0.6746032 0.3253968

   age
Y       [,1]     [,2]
  0 29.81689 12.31094
  1 28.91601 13.70471
```

Earlier we stated that the prior and likelihood data was calculated from the training set. The output above shows this. The prior for Survived, called A-priori above, is .61 not-Survived and .39 Survived. The likelihood data is shown in the output as conditional probabilities. For discrete variables, this is a breakdown by survived/not for each possible value of the attribute. For continuous data like age we are given the mean and standard deviation for the two classes. Notice also the reference at the top of the output about laplace. More about that in a later section.

In the output above there are 2 discrete variables, pclass and sex. Notice that each row sums to 1, indicating that they are probabilities. The probabilities of survival=1 for the 3 classes are 40%, 22%, and 37% respectively. The probability of surviving for females was 67% compared to 33% for males. The age variable is continuous. The mean for not surviving is 29.8 and for surviving is 28.9. These values are very close, so just looking at age alone does not tell us much.

We calculated accuracy on the test set the same as in the logistic regression example. Whereas the logistic regression got 79% accuracy, the naive Bayes algorithm achieved 81% accuracy. One final thing to note about this example is that we can extract the raw probabilities from the predictions as shown next. If we want to compute accuracy, however, we need to use type="class", as shown in the Quick Reference at the end of the chapter.

Code 5.2.2 — Raw Probabilities. With type="raw".

```
p2_raw <- predict(nb1, newdata=test, type="raw")
head(p2_raw, n=3)
```

```
             0         1
[1,] 0.1312656 0.8687344
[2,] 0.1194605 0.8805395
[3,] 0.1324441 0.8675559
```

Check Your Understanding 5.1 — Pima Diabetes Data. In the previous example we saw that naive Bayes slightly outperformed logistic regression. Let's compare the two algorithms on a different data set. Try the following:

- Load the PimaIndiansDiabetes2 data set from package mlbench into variable df.
- Use str() to familiarize yourself with the data and attach the data.
- Use sapply() to get counts of NAs per column.
- Fix the NAs as follows. Replace NAs in triceps and insulin with the mean value of the column. Then remove final NAs with complete.cases().
- Divide the data into a 80/20 train/test split using seed 1234.
- Create a logistic regression model on the training data with diabetes as the target and all other columns as predictors.
- Evaluate on the test data. What is your accuracy?
- Create a naive Bayes model on the training data.
- Evaluate on the test data. What is your accuracy?

This data set had extremely large numbers of NAs for insulin and triceps, almost a third to one half of the data. By replacing these NAs with the mean of each column we could have diminished their predictive power. What if instead of replacing with the mean, we replace with the class-conditional mean? First we check to see if there is much difference in the means when diabetes is positive versus negative:

```
n <- which(df$diabetes=="pos")
mean(df$insulin[n], na.rm=TRUE)  # 206.8
mean(df$insulin[-n], na.rm=TRUE) # 130
```

Yes, the numbers seem quite different. Let's run the algorithms again on updated data:

- Reload the data into df.
- Replace NAs for insulin and triceps with the class conditional means. An example is shown below of one of the four lines you will need.
- Redivide into train and test using the same indices as before.
- Run logistic regression and naive Bayes on this updated data.
- Where your results significantly different? Which models do you prefer and why?

Here is one of the four lines of code you will need to write to replace NAs on triceps and insulin with class-conditional means:

```
df$triceps[which(df$diabetes=="pos" & is.na(df$triceps))] <-
        mean(df$triceps[n], na.rm=TRUE)
```

■

5.3 Probability Foundations

This book assumes that readers have had a prior course on probability but we will review some key concepts here. Random variables, often denoted by capital letters such as X, can be discrete or continuous. The probability of two variables X and Y is called the *joint* distribution, determined jointly by X and Y, $P(X,Y)$. The *conditional* distribution of P(X|Y) is given by:

$$P(X|Y) = \frac{P(X,Y)}{P(Y)} \tag{5.3}$$

Two important probability rules are the product rule and the sum rule. The product rule says that the joint probability of A and B can be calculated by multiplying the conditional probability of A given B by the probabiity of B.

$$p(A,B) = p(A|B)p(B) \tag{5.4}$$

The sum rule says that we can calculate the probability of A by finding the joint probability with B and summing over all possible values of b.

$$p(A) = \sum_b p(A,B) = \sum_b p(A|B=b)P(B=b) \tag{5.5}$$

The chain rule lets us take the joint probability of many variables as follows:

$$p(X_{1:D}) = p(X_1)p(X_2|X_1)p(X_3|X_2,X_1)...p(X_D|X_{1:D-1}) \tag{5.6}$$

The expectation of a random variable is also known as the mean, or first moment. The expectation of a discrete random variable is:

$$E(X) = \sum_i X_i P(X_i) \tag{5.7}$$

So the expected value of a fair die is 3.5:

$$E(X) = 1\times\frac{1}{6} + 2\times\frac{1}{6} + 3\times\frac{1}{6} + 4\times\frac{1}{6} + 5\times\frac{1}{6} + 6\times\frac{1}{6} \tag{5.8}$$

The variance of a distribution is also called its second moment, and is represented by σ^2. When we take it's square root, we have the standard deviation.

$$\sigma^2 = E[(X-\mu)^2] \tag{5.9}$$

The log trick is often used when multiplying probabilities to prevent underflow and possibly multiplying by 0. Instead of multiplying the probabilities, the log trick says to add the log of the probabilities.

5.4 Probability Distributions

There are a few probability distributions that occur frequently in Bayesian approaches so it would be a good idea to review them here. Many of these are discussed also in context of their conjugate prior. Conjugate distributions are in the same family.

5.4.1 Bernouli, Binomial, and Beta Distributions

These distributions concern binary variables representing such things as the flip of a coin, wins and losses, and so forth. Our example will be shooting baskets for practice, where x=1 means that we made a basket and x=0 means that we missed. Let's say that I am shooting hoops for the first time and I made 2 baskets out of 10 tries. Given this data, my probability of making a basket is 0.2. The Bernouli distribution describes binary outcomes like this example. The Bernouli distribution has a parameter, μ, which specifies the average probability of the positive class.

$$Bernoulli(x|\mu) = \mu^x(1-\mu)^{1-x} \tag{5.10}$$

This gives us the probability that x is 1: $p(x=1) = 0.2^1 * .8^0 = 0.2$. And the probability that x is 0: $p(x=0) = 0.2^0 * .8^1 = 0.8$.

The Bernoulli distribution is a special case of the binomial distribution in which the number of trials, N = 1. Now suppose we run our Bernouli trial N=100 times, that is, I shoot 100 baskets. Let X be the random variable which represents the number of baskets made. The binomial distribution of X where I made k baskets in N trials has the following probability mass function (pmf):

$$Binomial(k|N,\mu) = \binom{N}{k}\mu^k(1-\mu)^{N-k} \tag{5.11}$$

Let's let k=20 for our 100 trials. Will the outcome of the binomial be 0.2?

$$Binomial(20|100,0.2) = \binom{100}{20}0.2^{20}(1-0.2)^{80} = 0.09930021 \tag{5.12}$$

No, it is not because there are many ways we can get 20 baskets out of 100 trials, 100 choose 20, to be exact. You can get out your calculator to confirm that or use r command `dbinom(20, 100, 0.2)`.

What is the expected mean of our 100 trials? Our expected value will be $N\mu$ which in our case is $100 * 0.2 = 20$. Let's derive these values using R. First we make a vector of possible values for k, the number of baskets made. We could make anywhere from 0 to 100 baskets. Then we multiply each k by its probability and add them together following Equation 5.7 above for the mean of a discrete distribution. E is 20, as we expected. The plot in Figure 5.1 shows the expected value at the center with the variance, calculated as $N\mu(1-\mu)$, which is 16 in our example. If you `sum(dbinom(k, 100, 0.2))` you get 1.0 of course because this represents the total probability space.

Code 5.4.1 — Basketball. A Binomial Distribution.

```
k <- 0:100  # possible number of baskets for 100 tries
E <- sum(k * dbinom(k, 100, 0.2))
v <- 100 * .2 * .8
plot(k, dbinom(k, 100, 0.2))
```

Now suppose that I got really lucky when I shot my first 3 hoops and made all 3 baskets. This gives me $\mu = 1.0$. It's unlikely I can keep this probability over the long haul. In fact, small sample sizes serve poorly as prior estimates of probabilities. Instead of a prior μ we really need a prior distribution over μ. We want this prior distribution to be a *conjugate* of our binomial distribution, meaning that we want it to be proportional to $\mu^x(1-\mu)^{1-x}$. The beta distribution is a good choice:

$$Beta(\mu|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\mu^{a-1}(1-\mu)^{b-1} \tag{5.13}$$

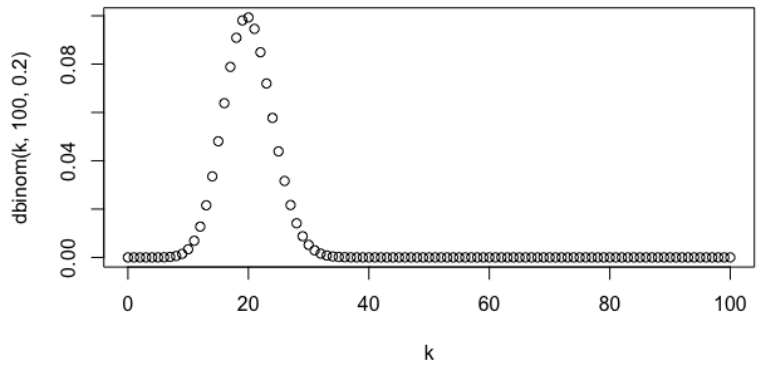

Figure 5.1: Binomial Distribution for 0.2

In the above equation, the first term with the gammas serves as a normalizing constant to make sure that the total probability integrates to 1. The gamma function is commonly used in probability, and is an extension of the factorial function to the real numbers: $\Gamma(n) = (n-1)!$ and is also extended to complex numbers. The parameters a and b in our example will be the number of baskets made and missed, respectively. Beta distributions are commonly used in Bayesian approaches to represenent prior knowledge of a parameter. The gamma function is defined as follows for positive real numbers or complex numbers with a positive real portion:

$$\Gamma(x) \equiv \int_0^\infty u^{x-1} e^{-u} du \tag{5.14}$$

Let's look at the beta distribution for our example in R. We use the rbeta() function to create 100 random beta samples with shape parameters 20 and 80. Then we plot this curve as the black line in Figure 5.2. Now suppose I take 15 more shots and make 5 of them. This will make a=30 and b=85. This updated curve is shown in red in Figure 5.2. The code below shows how to create the random beta samples with `rbeta(n, shape1, shape2)`. What will x look like? it is a vector of 100 random numbers drawn from a beta distribution with parameters a=20 and b=80. The mean will be 0.2 with the min around 0.1 and the max around 0.3. The code then draws the original curve in black and the updated curve in red. The `par(new=TRUE))` is used when you want to plot over an existing plot.

Code 5.4.2 — Basketball. A Beta Distribution.

```
x <- rbeta(100, 20, 80)
curve(dbeta(x, 20, 80), xlab=" ", ylab=" ", xlim=c(0.1,0.6),ylim=c(0,10))
par(new=TRUE)
curve(dbeta(x, 30, 85), xlab=" ", ylab=" ", xlim=c(0.1,0.6),ylim=c(0,10),
    col="red")
```

In the code and plot above, we updated the original black curve by adding baskets to a and misses to b. Our new probability given our data will be:

$$p(x = 1|D) = \frac{a+m}{a+b+m+l} \tag{5.15}$$

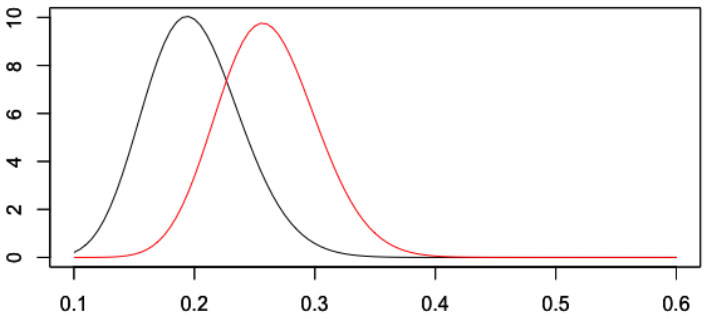

Figure 5.2: Beta Distribution for a=20, b=80

where m represents the number of new baskets and l represents the number of new losses.

The equation above is a Bayesian estimate. In contrast, note that the MLE estimate is simply m/N. As m and l approach infinity, the Bayesian estimate converges to the MLE. For a finite data set, the posterior probability with be between the prior and the MLE.

5.4.2 Multinomial and Dirichlet Distributions

We can extend the binomial distribution to the case where we have variables that are not binary but can take on more than 2 values. This is a multinomial distribution. The probability mass function of a multinomial distribution is:

$$Multinomial(m_1, m_2, ..., m_k|N, \mu) = \left(\frac{N}{m_1!m_2!...m_k!}\right)\prod_{k=1}^{K}\mu_k^{m_k} \tag{5.16}$$

Where k indexes over the number of classes, K, and each of the m_i represent the probability of that class, with the sum of all $m_i = 1$.

The iris data is an example of a multinomial distribution. There are 3 classes, and the data set has 50 examples of each class, an even distribution. If we want to put the 150 flowers in 3 boxes (classes) with even probability of being in each class, we could use the following R command.

Code 5.4.3 — Multinomial. Iris Example

```
rmultinom(n=10, size=150, prob=c(1/3, 1/3, 1/3))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   56   44   51   55   54   47   49   60   61    42
[2,]   36   48   54   44   42   46   58   50   40    61
[3,]   58   58   45   51   54   57   43   40   49    47
```

The output above shows us 10 vectors, left to right, because we said "n=10". Our size is 150 for each of our iris flowers, and they have an even distribution. What we see in each column are distributions of the 150 flowers. Each column sums to 150. What we see in each row are the number of flowers in each box (class). The mean values for the 3 classes are 54, 50.1, and 45.9.

If we selected 6 flowers at random, what is the probability that there will be 1 flower from class 1, 2 flowers from class 2, and 3 flowers from class 3?

Code 5.4.4 — Multinomial. Use dmultinom() for probabilities.

```
dmultinom(x=c(1,2,3), prob=c(1/3, 1/3, 1/3))
[1] 0.08230453
# check:
factorial(6)/(factorial(3)*factorial(2)*factorial(1))*
    0.333333^1*0.333333^2*0.333333^3
[1] 0.08230403
```

The Dirichlet distribution is the conjugate prior of the multinomial distribution. The Dirichlet distribution has k parameters, α, one for each class. So instead of X being 0 or 1, it can take on k values. In the following, α_0 is the sum of all alphas. The Dirichlet distribution:

$$Dir(\mu|\alpha) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)...\Gamma(\alpha_k)} \prod_{k=1}^{K} \mu_k^{\alpha_{k-1}} \tag{5.17}$$

Consider a magic bag containing balls of K=3 colors: red, blue, yellow. For each of N draws, you place the ball back in the bag with an *additional* ball of the same color. As N approaches infinity, the colored balls in the magically expanded bag will be $Dir(\alpha_1, \alpha_2, \alpha_3)$.

You can think of the Dirichlet distribution as a multinomial factory.

Code 5.4.5 — Dirichlet. Output Distribution.

```
library(MCMCpack) # for function rdirichlet()
m <- rdirichlet(10, c(1, 1, 1))
m
              [,1]      [,2]       [,3]
 [1,] 0.015740801 0.3900641 0.59419507
 [2,] 0.295649733 0.3622780 0.34207224
 [3,] 0.464984547 0.4516325 0.08338300
 [4,] 0.365099590 0.3074731 0.32742729
 [5,] 0.065993901 0.2832624 0.65074371
 [6,] 0.252786635 0.6786473 0.06856608
 [7,] 0.049175200 0.4904748 0.46034997
 [8,] 0.297815089 0.2121868 0.48999815
 [9,] 0.005201826 0.3076536 0.68714457
[10,] 0.326711959 0.4160060 0.25728208
mean(m[,1])
[1] 0.2139159
> mean(m[,2])
[1] 0.3899679
> mean(m[,3])
[1] 0.3961162
```

We asked for 10 distributions with our alphas all equal to 1. The sum of every row, which is every distribution, is 1.0. The mean of the columns for these 10 examples are 0.2, 0.38, and 0.39. When run with 1000 examples the means were 0.34, 0.33 and 0.32.

5.4.3 Gaussian

The Gaussian or normal distribution is used for quantitative variables. It has two parameters which define its shape: the mean μ and the variance σ^2. The probability density function is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big) \tag{5.18}$$

Let's look at a plot of a few Gaussians to see how the mean and variance influence the shape. All 3 distributions are generated with the dnorm() function, and all have a mean of 0. Different means would shift the curves right or left. The three curves have different standard deviations.

Code 5.4.6 — Gaussians. Normal distributions.

```
curve( dnorm(x,0,1), xlim=c(-4,4), ylim=c(0,1) )
curve( dnorm(x,0,2), add=T, col='blue' )
curve( dnorm(x,0,.5), add=T, col='orange' )

legend(par('usr')[1], par('usr')[4], xjust=0,
       c('(0,1)', '(0,2)', '(0,0.5)'),
       lwd=c(1,1,1), # line width
       col=c(par('fg'), 'blue', 'orange'))
```

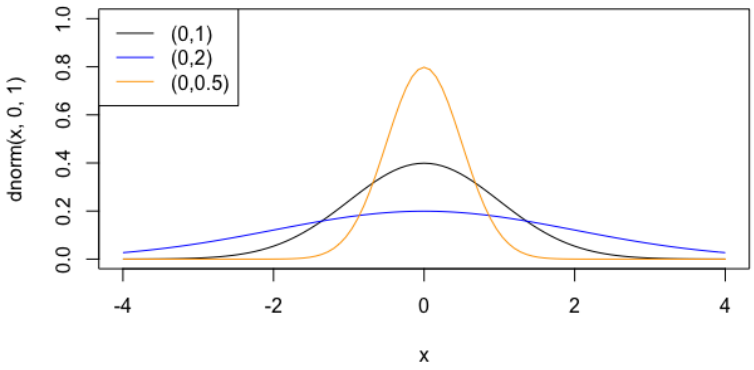

Figure 5.3: Gaussian Distributions

The pdf of the Gaussian above is for a single variable x. The Gaussian can be extended to a D-dimensional vector **x** in which case it is called a multivariate Gaussian and has the pdf shown below where μ is now a vector of means, Σ is a DxD covariance matrix, $|\Sigma|$ is the determinant.

$$f(x) = \frac{1}{\sqrt[D/2]{2\pi}\sqrt{|\Sigma|}} exp\Big(-\frac{(x-\mu)^T(x-\mu)}{2\Sigma}\Big) \tag{5.19}$$

5.5 The Algorithm

Calculating joint probabilities with the chain rule above would be mathematically intractable. The simplifying assumption of the naive Bayes algorithm is that each of the predictors is independent.

Therefore:

$$p(X_1, X_2, ...X_D|Y) = \prod_{i=1}^{D} p(X_i|Y) \tag{5.20}$$

The naive assumption of the independence of the predictors is typically not true, but perhaps surprisingly, naive Bayes works well and almost a universal baseline machine learning algorithm.

The algorithm requires a single pass over the data to estimate parameters. There are two sets of parameters to estimate from the data and two ways we can estimate the from the test data. The two methods are maximum likelihood estimates (MLE) and maximum apriori (MAP). MLE simply involves counting instances in the training data while MAP additionally makes some estimates based on prior distributions of the data. The two sets of parameters are counts for the probability of each class, and parameters for each predictor.

The first set of parameters to estimate is the probability of each class. If we estimate this with MLE, we just calculate the number of observations in each class. The estimate for class c will be the count of observations with class c divided by the number of observations:

$$MLE_c = \frac{|Y = y_c|}{|N|} \tag{5.21}$$

Estimating parameters for predictors depends upon their type. Binary features can use the mean of the Bernoulli distribution to get probabiities for each class. For discrete variables with more than 2 categories, the mean of the multinoulli distribution for each category can be used. Parameters for quantitative predictors are estimated from the mean and variance of the Gaussian distribution.

The MLE for the likelihood of a predictor given the class is also achieved by counting the data for discrete predictors. For predictor X_i and class c:

$$\hat{\theta}_{ic} = \frac{|X_{ic}|}{|N_c|} \tag{5.22}$$

It is possible that a given predictor for a given class may have 0 observations. In this case the estimate is 0 which is a problem given the multiplication of predictor likelihoods. An approach that eliminates this problem is smoothing, which involves adding a little to the numerator and denominator:

$$\hat{\theta}_{ic} = \frac{|X_{ic}| + l}{|N_c| + lm} \tag{5.23}$$

The value added to the denominator, m, represents the number of categories of X_i. If l=1 it is called Laplace or add-one smoothing. If we let l be a larger value this corresponds to a MAP estimate for the likelihood. We can add smoothing in a similar way to the MLE for the prior, in effect making it a MAP estimate.

For continuous variables, the mean and standard deviation of the predictor can be estimated but we would really like these values as they are associated with each class. Therefore separate mean, μ, and standard deviation, σ^2, values are computed by class. This is called a Gaussian naive Bayes classifier.

$$\hat{\theta_{ic}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.24}$$

5.6 Applying the Bayes Theorem

There are a lot of misconceptions about probabilities, even among educated people. Let's walk through an example to illustrate some of the subtleties.

A test for cancer has a sensitivity of 80% which means that if you have cancer, it will be positive with p=0.8. If your test is positive this does not mean you have an 80% chance of having cancer. Other data points we need: the probability of a false positive rate for the test is 10%, and the overall probability of having cancer is 0.4%. By plugging in the numbers into the theorem we say that the probability of actually having cancer given this positive test is 3.1%.

$$p(cancer|positive\_test) = \frac{0.8 * 0.004}{0.8 * 0004 + 0.1 * 0.996} = 0.031 \tag{5.25}$$

5.7 Handling Text Data

Previously all of our data has been numeric. However, machine learning with text is a hot topic in the field of Natural Language Processing. That's a separate course but we will talk a little about how text can be handled numerically now and revisit this in the deep learning chapter. Any kind of natural language processing involves a lot of preprocessing. In a bag of words model, the text is tokenized, meaning that it is divided into individual words. The sentence structure is lost but just looking at counts of words can be informative. Additional preprocessing may involve removing punctuation, numbers, and making all text lowercase. Further, stop words are often removed. Stopped words are common words that glue a sentence together but don't add much content. In this sentence, the stop words might be: *in, this, the, might, be*. Finally, the vocabulary size may be eliminated since rare words don't have predictive value.

Often a document-term matrix is created in which each row represents a document and each column represents a term in the vocabulary. The intersection gives the count of a specific term in a given document. This is a sparse matrix in that each document has relatively few words out of the total vocabulary. Sometimes it is converted to a binary matrix with counts being replaced by 1 indicating the presense of that word in the document.

5.8 Naive Bayes v. Logistic Regression

Next we look at classification data set in package mlbench. This package collects several real-world and artificial data sets for benchmarking. You can see the list of data sets by typing `data(package="mlbench")` at the console. We will use the BreastCancer data set, sometimes called the Wisconsin breast cancer data since the data originated from clinical practice at the Univeristy of Madison Wisconsin hospital. This data set has 669 observations with 11 columns. Column 1 is an ID that will be ignored later, columns 2-10 are factors specifying information gleaned from bioposies. The final column is the label: benign or malignant.

5.8.1 Compare to Logistic Regression

In the notebook available online we first divide into 80% train and 20% test data, first removing the Id column. Then we perform logistic regression and get 92% accuracy on predicting benign or malignant on the test data.

5.8.2 Balance or Unbalanced Data

Of the 669 observations in the breast cancer data set, approximately 64% are benign to 36% malignant. This is a reasonably balanced data set. We can easily use the summary() function to get the numbers: `summary(BreastCancer$Class)` gives us 458 benign and 241 malignant.

Although breast cancer is rare in the general population, the data is fairly balanced, probably because patients going for a biopsy are likely to have suspicious lumps or other symptoms, and therefore be more likely to have cancer than the general population.

Why do we care about how balanced the data set it? An unbalanced data set is when one class is represented by a low percentage of observations. This can happen commonly in data gathered towards anomaly detection such as credit card fraud, or detecting rare diseases. If you had an extremely unbalanced data set, like 90% of one class and only 10% in another, a classifier could achieve 90% accuracy just by guessing the dominant class each time. Many classifiers will not take into consideration the underlying distribution of the classes and the model they learn reflects the skewed nature of the data presented to them.

What can we do about unbalanced data? If it is possible to get more data, then that is the best option. If getting more data is not possible, then resampling techniques can be used. These could involve oversampling the minority class, essentially duplicating some of these observations, and/or undersampling the majority class by using only a subset of them.

The accuracy of a classifier on test data can be further examined by breaking it down into sensitivity and specificity.

5.8.3 Accuracy, Sensitivity and Specificity

When we use the table() function to examine our results we see the breakdown of values. This table is called a *confusion matrix* in machine learning. It breaks down the results into 4 categories:

- TP true positive observations are positive and were classified as positive
- TN true negative observations are negative and were classified as negative
- FP false positive observations are negative but were classified as positive
- FN false negative observations are positive but were classified as negative

Look at the following sample output:

```
table(pred1, test$Class)
pred1 benign malignant
    1     73         7
    2      4        50
```

This result is from the logistic regression classifier. If we interpret malignant as the positive class, the table shows that TN=73, TP=50. These are on the left-to-right diagonal. The table shows that FN=4 and FP=7. That is the specificity and sensitivity are:

$$specificity = \frac{TP}{TP+FN} = \frac{50}{50+7} = 0.877 \tag{5.26}$$

$$sensitivity = \frac{TN}{TN+FP} = \frac{73}{73+4} = 0.948 \tag{5.27}$$

The specificity, the true negative rate was lower than the sensitivity, the true positive rate, but they were both close to 1 which is ideal. For this example, the classifier correctly identified benign cases 88% of the time and correctly identified malignant cases 95% of the time. The `caret` package can compute the confusion matrix and associated statistics for us as shown next. First we load the package then use the confusionMatrix() function in that package. The first argument is the predictions and the second is the actual values. In this data set, malignant is coded as 2 and benign as 1. We pass the integer representations for test$Class. The third argument tells the function to interpret "2", malignant, as the positive class.

Code 5.8.1 — Confusion Matrix. Using the caret package.

```
library(caret)
confusionMatrix(pred1, as.integer(test$Class), positive="2")
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1  2
         1 73  7
         2  4 50

               Accuracy : 0.9179
                 95% CI : (0.8579, 0.9583)
    No Information Rate : 0.5746
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8309
 Mcnemar's Test P-Value : 0.5465

            Sensitivity : 0.8772
            Specificity : 0.9481
         Pos Pred Value : 0.9259
         Neg Pred Value : 0.9125
             Prevalence : 0.4254
         Detection Rate : 0.3731
   Detection Prevalence : 0.4030
      Balanced Accuracy : 0.9126

       'Positive' Class : 2
```

5.8.4 Naive Bayes Model

Next we build a Naive Bayes model on the same train and test data. Then we make predictions and output a confusion matrix. In the first Naive Bayes example in this chapter we used the naiveBayes() function with a typical R function as the first argument. There is another way to set up the model as shown in this code example. In this alternative use of the function the first argument is the training columns and the second argument is the training labels. The code below also shows the predictions and the call to the confusionMatrix() function. Notice this time in the call to that function we did not convert the test$Class column to integer. This is because the naiveBayes function output benign or malignant instead of 1 or 2. If we had left the as.integer() matrix in place we would receive the following error message: *The data must contain some levels that overlap the reference*. Error messages can be hard to track down so a good start is to work backwards from the point of the error, to make sure the data looks the way you expect it to.

Code 5.8.2 — Naive Bayes. On the Breast Cancer Data.

```
nb1 <- naiveBayes(train[,-10], train[,10])
pred2 <- predict(nb1, newdata=test[,-10], type="class")
confusionMatrix(pred2, test$Class, positive="malignant")
```

The output for the Naive Bayes confusion matrix is shown next. The accuracy for Naive Bayes

was 96% compared to 92% for the logistic regression model. Specificity was less than 1 percente higher for the Naive Bayes mode but sensitivity was 3 percentage points higher. This is because the logistic regression model had 7 false positive observations and the Naive Bayes only had 1. Notice that the Kappa score for Naive Bayes is higher as well. Recall from the previous chapter than the Kappa statistic adjusts for the probability of getting the correct prediction by chance.

```
Confusion Matrix and Statistics

           Reference
Prediction  benign malignant
  benign        78         1
  malignant      4        57

               Accuracy : 0.9643
                 95% CI : (0.9186, 0.9883)
    No Information Rate : 0.5857
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.927
 Mcnemar's Test P-Value : 0.3711

            Sensitivity : 0.9828
            Specificity : 0.9512
         Pos Pred Value : 0.9344
         Neg Pred Value : 0.9873
             Prevalence : 0.4143
         Detection Rate : 0.4071
   Detection Prevalence : 0.4357
      Balanced Accuracy : 0.9670

       'Positive' Class : malignant
```

5.8.5 Generative v. Discriminative Classifiers

In this example Naive Bayes outperformed logistic regression. It is important to keep in mind that these are two quite different classifiers. Logistic regression directly estimates the parameters of P(Y|X). This is called a *discriminative classifier*. Naive Bayes directly estimates parameters for P(Y) and P(X|Y). This is called a *generative classifier*. If the naive Bayes independence assumptions hold, and the number of training examples grows towards infinity, the naive Bayes and logistic regression converge toward similar classifiers. In general, Naive Bayes will do better with small data sets and logistic regression will do better as the size of the data grows. Naive Bayes has higher bias but lower variance than logistic regression. In this example, if you look at the output of the summary() function for the logistic regression model online you will see dozens of predictors because each of the 9 predictor columns are broken down into their factors. None of these almost 100 predictors achieved a low p-value and many factor levels were not included in the model. Further, five of the 9 predictor columns are ordinal factors which not only classify different values but there is an order to the values. The logistic regression function may have been overwhelmed by the sheer number of factors and levels whereas the simpicity of the Naive Bayes approach may have worked in its favor.

5.9 Naive Bayes from Scratch

In order to understand Naive Bayes on a deeper level, we will explore creating the algorithm from scratch. This will be applied to the Titanic data. The notebook online first loads and cleans the data, then runs the naiveBayes() function on the data. We will compare the by-scratch results to the results from the algorithm.

5.9.1 Probability Tables

Here are the probability tables from the naiveBayes() function:

```
A-priori probabilities:
df[, 2]
       0        1
0.618029 0.381971

Conditional probabilities:
       pclass
df[, 2]         1         2         3
      0 0.1520396 0.1953028 0.6526576
      1 0.4000000 0.2380000 0.3620000

       sex
df[, 2]  female      male
      0 0.1569839 0.8430161
      1 0.6780000 0.3220000

       age
df[, 2]     [,1]     [,2]
      0 29.94757 12.22384
      1 28.78417 13.92003
```

Calculating the prior (apriori) probabilities of survived or perished is easy. It is simply dividing the counts of surived/perished by the total number of observations, as shown here:

```
apriori <- c(
  nrow(df[df$survived=="0",])/nrow(df),
  nrow(df[df$survived=="1",])/nrow(df)
  )
print("Prior probability, survived=no, survived=yes:")
apriori
[1] 0.618029 0.381971
```

5.9.2 Conditional Probability for Discrete Data

The conditional probability tables are also quite simple to create for qualitative data. Following Equation 5.21, we have a count for each level of each predictor. First we get counts for survived.

```
# get survived counts for no and yes
count_survived <- c(
  length(df$survived[df$survived=="0"]),
```

```
  length(df$survived[df$survived=="1"])
)
# likelihood for pclass
lh_pclass <- matrix(rep(0,6), ncol=3)
for (sv in c("0", "1")){
  for (pc in c("1","2","3")) {
    lh_pclass[as.integer(sv)+1, as.integer(pc)] <-
      nrow(df[df$pclass==pc & df$survived==sv,]) /
       count_survived[as.integer(sv)+1]
  }
}
          [,1]      [,2]      [,3]
[1,] 0.1520396 0.1953028 0.6526576
[2,] 0.4000000 0.2380000 0.3620000
```

We do a similar calculation for the likelihood of sex.

```
# likelihood for sex
lh_sex <- matrix(rep(0,4), ncol=2)
for (sv in c("0", "1")){
  for (sx in c(2, 3)) {
    lh_sex[as.integer(sv)+1, sx-1] <-
      nrow(df[as.integer(df$sex)==sx & df$survived==sv,]) /
       count_survived[as.integer(sv)+1]
  }
}
lh_sex
          [,1]      [,2]
[1,] 0.1569839 0.8430161
[2,] 0.6780000 0.3220000
```

5.9.3 Likelihood for Continuous Data

To calculate the likelihood for age we first need the mean and variance.

```
age_mean <- c(0, 0)
age_var <- c(0, 0)
for (sv in c("0", "1")){
    age_mean[as.integer(sv)+1] <-
      mean(df$age[df$survived==sv])
    age_var[as.integer(sv)+1] <-
      var(df$age[df$survived==sv])
}


age_mean
[1] 29.94757 28.78417
> age_var
[1] 149.4223 193.7673
```

Now we plug these values into Equation 5.23. We will write a function to calculate this.

```
calc_age_lh <- function(v, mean_v, var_v){
  # run like this: calc_age_lh(6, 25.9, 138)
  1 / sqrt(2 * pi * var_v) * exp(-((v-mean_v)^2)/(2 * var_v))
}
```

5.9.4 Putting it All Together

Now we need a function to caculate Bayes' theorem for us.

```
calc_raw_prob <- function(pclass, sex, age) {
  # pclass=1,2,3  sex=1,2   age=numeric
  num_s <- lh_pclass[2, pclass] * lh_sex[2, sex] * apriori[2] *
      calc_age_lh(age, age_mean[2], age_var[2])
  num_p <- lh_pclass[1, pclass] * lh_sex[1, sex] * apriori[1] *
      calc_age_lh(age, age_mean[1], age_var[1])
  denominator <- lh_pclass[2, pclass]  * lh_sex[2, sex] *
   calc_age_lh(age, age_mean[2], age_var[2]) * apriori[2] +
      lh_pclass[1, pclass]  * lh_sex[1, sex] *
      calc_age_lh(age, age_mean[1], age_var[1]) * apriori[1]
  return (list(prob_survived <- num_s / denominator,
          prob_perished <- num_p / denominator))
}
}
```

Let's call this function for the first 5 test observations.

```
for (i in 1:5){
  raw <- calc_raw_prob(test[i,1], as.integer(test[i,3]), test[i,4])
  print(paste(raw[2], raw[1]))
}
```

5.9.5 Results

The following is the prediction:

```
[1] "0.134219499226771 0.865780500773229"
[1] "0.119544295476936 0.880455704523064"
[1] "0.135715780701606 0.864284219298394"
[1] "0.267316737470339 0.732683262529661"
[1] "0.649768435768306 0.350231564231694"
```

Below is the raw probabilities from the Naive Bayes model. Notice they are the same.

```
> pred[1:5,]
             0         1
[1,] 0.1342195 0.8657805
[2,] 0.1195443 0.8804557
[3,] 0.1357158 0.8642842
[4,] 0.2673167 0.7326833
[5,] 0.6497684 0.3502316
```

The above code detailed how the algorithm works: counting and simple math. The code was not written in the most efficient way for R but in the way that should be the most clear to computer science students.

5.10 Summary

Naive Bayes is a dependable classifier that is often used as a baseline in machine learning papers that are exploring more sophisticated algorithms that are expected to outperform Naive Bayes.

Stengths and weakness of naive Bayes:

Strengths

- Works well with small data sets
- Easy to implement
- Handles high dimensions well

Weaknesses

- May be outperformed by other classifiers for larger data sets
- Guesses are made for values in the test set that did not occur in the training data
- If the predictors are not independent, the naive assumption that they are may limit the performance of the algorithm

5.10.1 New Terminology in this Chapter

This chapter used a lot of terminology from probability theory:

- likelihood v. probability
- prior probability v. posterior probability
- Bayes Theorem
- conditional probability
- joint probability
- marginal probability
- expected values, mean and variance

In addition we reviewed several probability distributions by family:

- Bernouli, binomial and beta distributions
- Multinomial and Dirichlet distributions
- Gaussian distributions

Finally, there are a few terms related to techniques:

- MLE maximum likelihood estimate
- MAP maximum apriori estimate
- Laplace smoothing

5.10.2 Quick Reference

Reference 5.10.1 Build a Naive Bayes Model

```
library(e1071)
# method one: use a formula
nb_model <- naiveBayes(formula, data=train)
```

Reference 5.10.2 Build a Naive Bayes Model

```
# method two: X, Y
nb_model <- naiveBayes(predictor_cols, target_col, data=train
```

Reference 5.10.3 Predict

```
raw <- predict(model, newdata=test, type="raw")
pred <- predict(model, newdata=test, type="class"
```

Reference 5.10.4 Confusion Matrix

```
library(caret)
confusionMatrix(predictions, test$target, positive="2")
```

5.10.3 Labs

Problem 5.1 — Classification on the Abalone Data. Try the following:

- Re-run your code from the Chapter 4 Lab on classifying the Abalone data using Logistic regression, or follow those instructions to create the model.
- Create a naive Bayes model on the same train/test split.
- Compare the performance of the two algorithms.
- Compare the confusion matrix tables of the two algorithms. What do you observe? Was one better at predicting true positives versus true negatives? Why might this be important?

Problem 5.2 — Classification on the Heart Data. Try the following:

- Re-run your code from the Chapter 4 Lab on classifying the Heart data using Logistic regression, or follow those instructions to create the model.
- Create a naive Bayes model on the same train/test split.
- Compare the performance of the two algorithms.
- Compare the confusion matrix tables of the two algorithms. What do you observe? Was one better at predicting true positives versus true negatives? Why might this be important? How many NAs were predicted for each algorithm?

Problem 5.3 — Classification on the Sonar Data. Try the following:

- Load the Sonar data set from package mlbench. Research this data set and write a brief description of the columns.
- Divide the data into 80-20 train-test.
- Create a logistic regression model of the data. What is the accuracy?
- Create a naive Bayes model of the data. Compare the accuracy to the logistic regression model.
- Compare the confusion matrix tables for each. Discuss what you find.

5.10.4 Exploring Concepts

Problem 5.4 Compare how logistic regression makes classification predictions compared to naive Bayes.

Problem 5.5 Briefly summarize why logistic regression is called a discriminative classifier and naive Bayes is called a generative classifier.

Problem 5.6 What is the naive assumption in Naive Bayes?

5.10.5 Going Further

Tom Mitchell's classic book, *Machine Learning* Chapter 3 discusses Naive Bayes and Logistic Regression. Tom Mitchell has provided free access to this chapter here:

`http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf`

A well-regarded paper comparing discriminative and generative classifiers by Andrew Ng and Michael Jordan can be found here:

`https://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf`