Chapter 8
Probabilistic Learning: Classification Using Naive Bayes

The introduction to Chap. 7 presented the types of machine learning methods and described lazy classification for numerical data. What about nominal features or textual data? In this Chapter, we will begin to explore some classification techniques for categorical data. Specifically, we will (1) present the Naive Bayes algorithm; (2) review its assumptions; (3) discuss Laplace estimation; and (4) illustrate the Naive Bayesian classifier on a Head and Neck Cancer Medication case-study.

Later, in Chap. 20, we will also discuss text mining and natural language processing of unstructured text data.

8.1 Overview of the Naive Bayes Algorithm

Start by reviewing the basics of probability theory and Bayesian inference.

Bayes classifiers use training data to calculate an observed probability of each class based on all the features. The probability links feature values to classes like a map. When labeling the test data, we utilize the feature values in the test data and the "map" to classify our test data with the most likely class. This idea seems simple but the corresponding algorithmic implementations might be very sophisticated.

The best scenario of accurately estimating the probability of an outcome-class map is when all features in Bayes classifiers attribute to the class simultaneously. The Naive Bayes algorithm is frequently used for text classifications. The maximum *a posteriori* assignment to the class label is based on obtaining the conditional probability density function for each feature given the value of the class variable.

I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_8

8.2 Assumptions

Naive Bayes is named for its "naive" assumptions. Its most important assumption is that all of the features are *equally important* and *independent.* This rarely happens in real world data. However, sometimes even when the assumptions are violated, Naive Bayes still performs fairly accurately, particularly when the number of features p is large. This is why the Naive Bayes algorithm may be used as a powerful text classifier.

There are interesting relations between QDA (Quadratic Discriminant Analysis), LDA (Linear Discriminant Analysis), and Naive Bayes classification. Additional information about LDA and QDA is available online (http://wiki.socr.umich.edu/index.php/SMHS_BigDataBigSci_CrossVal_LDA_QDA).

8.3 Bayes Formula

Let's first define the set-theoretic Bayes formula. We assume that B_i's are mutually exclusive events, for all $i = 1, 2, \ldots, n$, where n represents the number of features. If A and B are two events, the Bayes conditional probability formula is as follows:

$$Posterior\ Probability = \frac{likelihood \times Prior\ Probability}{Marginal\ Likelihood}$$

Symbolically,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

When $B_i^{'}s$ represent a partition of the event space, $S = \cup B_i$ and $B_i \cap B_j = \varnothing\ \forall\, i \neq j$. So we have:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B|B_1) \times P(B_1) + P(B|B_2) \times P(B_2) \ldots + P(B|B_n) \times P(B_n)}.$$

Now, let's represent the Bayes formula in terms of classification using observed features. Having observed n features, F_i, for each of K possible `class` outcomes, C_k. The Bayesian model may be reformulate to make it more tractable using the Bayes' theorem, by decomposing the conditional probability.

$$P(C_k \mid F_1, \ldots, F_n) = \frac{P(F_1, \ldots, F_n|C_k)P(C_k)}{P(F_1, \ldots, F_n)}.$$

In the above expression, only the numerator depends on the class label, C_k, as the values of the features F_i are observed (or imputed) making the denominator constant. Let's focus on the numerator.

The numerator essentially represents the `joint probability model`:

$$P(F_1, \ldots, F_n|C_k)P(C_k) = \underbrace{P(F_1, \ldots, F_n, C_k)}_{\text{joint model}}$$

Repeatedly using the chain rule and the definition of conditional probability simplifies this to:

$$\begin{aligned}
&P(F_1, \ldots, F_n, C_k) = P(F_1|F_2, \ldots, F_n, C_k) \times P(F_2, \ldots, F_n, C_k) = \\
&= P(F_1|F_2, \ldots, F_n, C_k) \times P(F_2|F_3, \ldots, F_n, C_k) \times P(F_3, \ldots, F_n, C_k) = \\
&= P(F_1|F_2, \ldots, F_n, C_k) \times P(F_2|F_3, \ldots, F_n, C_k) \times P(F_3|F_4, \ldots, F_n, C_k) \\
&\quad \times P(F_4, \ldots, F_n, C_k) = \\
&\qquad\qquad = \ldots = \\
&= P(F_1|F_2, \ldots, F_n, C_k) \times P(F_2|F_3, \ldots, F_n, C_k) \times P(F_3|F_4, \ldots, F_n, C_k) \times \cdots \\
&\quad \times P(F_n|C_k) \times P(C_k)
\end{aligned}$$

Note that the "naive" qualifier in the *Naive Bayes classifier* name is attributed to the oversimplification of the conditional probability. Assuming each feature F_i is conditionally statistical independent of every other feature F_j, $\forall j \neq i$, given the category C_k, we get:

$$P(F_i|F_{i+1}, \ldots, F_n, C_k) = P(F_i|C_k).$$

This reduces the joint probability model to:

$$P(F_1, \ldots, F_n, C_k) = P(F_1|C_k) \times P(F_2|C_k) \times P(F_3|C_k) \times \cdots \times P(F_n|C_k) \times P(C_k)$$

Therefore, the joint model is:

$$P(F_1, \ldots, F_n, C_k) = P(C_k) \prod_{i=1}^{n} P(F_i|C_k)$$

Essentially, we express the probability of class level L given an observation, represented as a set of *independent features* F_1, F_2, ..., F_n. Then the posterior probability that the observation is in class L is equal to:

$$P(C_L|F_1, \ldots, F_n) = \frac{P(C_L) \prod_{i=1}^{n} P(F_i|C_L)}{\prod_{i=1}^{n} P(F_i)},$$

where the denominator, $\prod_{i=1}^{n} P(F_i)$, is a scaling factor that represents the marginal probability of observing all features jointly.

For a given case $X = (F_1, F_2, \ldots, F_n)$, i.e., given vector of *features*, the naive Bayes classifier assigns the **most likely class** $\hat{C}$ by calculating $\frac{P(C_L)\prod_{i=1}^{n} P(F_i|C_L)}{\prod_{i=1}^{n} P(F_i)}$ for all class labels L, and then assigning the class $\hat{C}$ corresponding to the maximum posterior probability. Analytically, $\hat{C}$ is defined by:

$$\hat{C} = \arg\max_{L} \frac{P(C_L)\prod_{i=1}^{n} P(F_i|C_L)}{\prod_{i=1}^{n} P(F_i)}.$$

As the denominator is static for L, the posterior probability above is maximized when the numerator is maximized, i.e., $\hat{C} = \text{argmax}_L P(C_L)\prod_{i=1}^{n} P(F_i|C_L)$.

The contingency table below illustrates schematically how the Bayesian, marginal, conditional, and joint probabilities may be calculated for a finite number of features (columns) and classes (rows).

Features/ Classes	$\boldsymbol{F_1}$	$\boldsymbol{F_2}$	...	$\boldsymbol{F_n}$	Total
$\boldsymbol{C_1}$	...	...	...	...	Marginal $\boldsymbol{P(C_1)}$
$\boldsymbol{C_2}$	...	...	...	Joint $P(C_2, F_n)$	...
...	...	...	...	...	...
$\boldsymbol{C_L}$	Conditional $P(F_1 \mid C_L) = \frac{P(F_1, C_L)}{P(C_L)}$	...	...	...	...
Total		Marginal $\boldsymbol{P(F_2)}$	...	...	$\boldsymbol{N}$

In the DSPA Appendix, we provide additional technical details, code, and applications of Bayesian simulation, modeling and inference.

8.4 The Laplace Estimator

If at least one $P(F_i|\, C_L) = 0$, then $P(C_L|\, F_1, \ldots, F_n) = 0$, which means the probability of being in this class is zero. However, $P(F_i|\, C_L) = 0$ could be result from a random chance in picking the training data.

One of the solutions to this scenario is **Laplace estimation**, also known as Laplace smoothing, which can be accomplished in two ways. One is to add small number to each cell in the frequency table, which allows each class-feature combination to be at least one in the training data. Then $P(F_i|\, C_L) > 0$ for all i. Another strategy is to add some small value, ϵ, to the numerator and denominator when calculating the posterior probability. Note that these small perturbations of the denominator should be larger than the changes in the numerator to avoid trivial (0) posterior for another class.

8.5 Case Study: Head and Neck Cancer Medication

8.5.1 Step 1: Collecting Data

We utilize the Inpatient Head and Neck Cancer Medication data for this case study, which is the case study 14 in our data archive.

Variables:

- **PID:** coded patient ID.
- **ENC_ID:** coded encounter ID.
- **Seer_stage:** SEER cancer stage (0 = In situ, 1 = Localized, 2 = Regional by direct extension, 3 = Regional to lymph nodes, 4 = Regional (both codes 2 and 3), 5 = Regional, NOS, 7 = Distant metastases/systemic disease, 8 = Not applicable, 9 = Unstaged, unknown, or unspecified). See: http://seer.cancer.gov/tools/ssm.
- **Medication_desc:** description of the chemical composition of the medication.
- **Medication_summary:** brief description about medication brand and usage.
- **Dose:** the dosage in the medication summary.
- **Unit:** the unit for dosage in the Medication_summary.
- **Frequency:** the frequency of use in the Medication_summary.
- **Total_dose_count:** total dosage count according to the Medication_summary.

8.5.2 Step 2: Exploring and Preparing the Data

Let's load our data first.

```
hn_med<-read.csv("https://umich.instructure.com/files/1614350/download?downl
oad_frd=1", stringsAsFactors = FALSE)
str(hn_med)

## 'data.frame':    662 obs. of  9 variables:
##  $ PID              : int  10000 10008 10029 10063 10071 10103 1012 1013
5 10136 10143 ...
##  $ ENC_ID           : int  46836 46886 47034 47240 47276 47511 3138 4773
9 47744 47769 ...
##  $ seer_stage       : int  1 1 4 1 9 1 1 1 9 1 ...
##  $ MEDICATION_DESC  : chr  "ranitidine" "heparin injection" "ampicillin/
sulbactam IVPB UH" "fentaNYL injection UH" ...
##  $ MEDICATION_SUMMARY: chr  "(Zantac) 150 mg tablet oral two times a day"
"5,000 unit subcutaneous three times a day" "(Unasyn) 15 g IV every 6 hours"
"25 - 50 microgram IV every 5 minutes PRN severe pain\nMaximum dose 200 mcg
Per PACU protocol" ...
##  $ DOSE             : chr  "150" "5000" "1.5" "50" ...
##  $ UNIT             : chr  "mg" "unit" "g" "microgram" ...
##  $ FREQUENCY        : chr  "two times a day" "three times a day" "every
6 hours" "every 5 minutes" ...
##  $ TOTAL_DOSE_COUNT : int  5 3 11 2 1 2 2 6 15 1 ...
```

Change the `seer_stage` (cancer stage indicator) variable into a factor.

```
hn_med$seer_stage <- factor(hn_med$seer_stage)
str(hn_med$seer_stage)

##  Factor w/ 9 levels "0","1","2","3",..: 2 2 5 2 9 2 2 2 9 2 ...

table(hn_med$seer_stage)

##
##   0   1   2   3   4   5   7   8   9
##  21 265  53  90  46  18  87  14  68
```

Data Preparation: Processing Text Data for Analysis

As you can see, the `medication_summary` contains a great amount of text. We should do some text mining to prepare the data for analysis. In R, the `tm` package is a good choice for text mining.

```
# install.packages("tm", repos = "http://cran.us.r-project.org")
# requires R V.3.3.1 +
```

The first step for text mining is to convert text features (text elements) into a `corpus` object, which is a collection of text documents.

```
hn_med_corpus<-Corpus(VectorSource(hn_med$MEDICATION_SUMMARY))
print(hn_med_corpus)
```

After we construct the `corpus` object, we could see that we have 662 documents. Each document represents an encounter (e.g., notes on medical treatment) for a patient.

```
inspect(hn_med_corpus[1:3])

## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 3
##
## [1] (Zantac) 150 mg tablet oral two times a day
## [2] 5,000 unit subcutaneous three times a day
## [3] (Unasyn) 15 g IV every 6 hours

hn_med_corpus[[1]]$content

## [1] "(Zantac) 150 mg tablet oral two times a day"

hn_med_corpus[[2]]$content

## [1] "5,000 unit subcutaneous three times a day"

hn_med_corpus[[3]]$content

## [1] "(Unasyn) 15 g IV every 6 hours"
```

There are unwanted punctuation and other symbols in the corpus document that we want to remove. We use the `tm::tm_map()` function for the cleaning.

```
corpus_clean <- tm_map(hn_med_corpus, tolower)
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
corpus_clean <- tm_map(corpus_clean, removeNumbers)
# corpus_clean <- tm_map(corpus_clean, PlainTextDocument)
```

The above lines of code changed all the characters to lower case, removed all punctuations and extra white spaces (typically created by deleting punctuations), and removed numbers (we could also convert the corpus to plain text).

```
inspect(corpus_clean[1:3])

## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 3
##
## [1] zantac  mg tablet oral two times a day
## [2]  unit subcutaneous three times a day
## [3] unasyn  g iv every  hours

corpus_clean[[1]]$content

## [1] "zantac  mg tablet oral two times a day"

corpus_clean[[2]]$content

## [1] " unit subcutaneous three times a day"

corpus_clean[[3]]$content

## [1] "unasyn  g iv every  hours"
```

The `DocumentTermMatrix()` function can tokenize the medication summary into words. It can count frequent terms in each document in the corpus object.

```
hn_med_dtm<-DocumentTermMatrix(corpus_clean)
```

Data Preparation: Creating Training and Test Datasets

Just like in Chap. 7, we need to separate the dataset into training and test subsets. We have to subset the raw data with other features, the corpus object and the document term matrix.

```
set.seed(12)
# 80% training + 20% testing

subset_int <- sample(nrow(hn_med),floor(nrow(hn_med)*0.8))

hn_med_train<-hn_med[subset_int, ]
hn_med_test<-hn_med[-subset_int, ]
hn_med_dtm_train<-hn_med_dtm[subset_int, ]
hn_med_dtm_test<-hn_med_dtm[-subset_int, ]
corpus_train<-corpus_clean[subset_int]
corpus_test<-corpus_clean[-subset_int]

# hn_med_train<-hn_med[1:562, ]
#hn_med_test<-hn_med[563:662, ]
# hn_med_dtm_train<-hn_med_dtm[1:562, ]
# hn_med_dtm_test<-hn_med_dtm[563:662, ]
#corpus_train<-corpus_clean[1:562]
#corpus_test<-corpus_clean[563:662]
```

Let's examine the distribution of **seer stages** in the training and test datasets.

```
prop.table(table(hn_med_train$seer_stage))

##
##          0          1          2          3          4          5
## 0.03024575 0.38374291 0.08317580 0.14555766 0.06616257 0.03402647
##          7          8          9
## 0.13421550 0.02268431 0.10018904

prop.table(table(hn_med_test$seer_stage))

##
##          0          1          2          3          4          5
## 0.03759398 0.46616541 0.06766917 0.09774436 0.08270677 0.00000000
##          7          8          9
## 0.12030075 0.01503759 0.11278195
```

We can separate (dichotomize) the seer_stage into two categories:

- *No stage* or *early stage* cancer, and
- *later stage* cancer.

```
hn_med_train$stage<-hn_med_train$seer_stage %in% c(4, 5, 7)
hn_med_train$stage<-factor(hn_med_train$stage, levels=c(F, T), labels = c("e
arly_stage", "later_stage"))
hn_med_test$stage<-hn_med_test$seer_stage %in% c(4, 5, 7)
hn_med_test$stage<-factor(hn_med_test$stage, levels=c(F, T), labels = c("ear
ly_stage", "later_stage"))
prop.table(table(hn_med_train$stage))

## early_stage later_stage
##   0.7655955   0.2344045

prop.table(table(hn_med_test$stage))

## early_stage later_stage
##   0.7969925   0.2030075
```

Visualizing Text Data: Word Clouds

A word cloud can help us visualize text data. More frequent words would have larger fonts in the figure, while less common words appear in smaller fonts. There is a `wordcloud` package in R that is commonly used for creating these figures (Figs. 8.1, 8.2, 8.3).

```
# install.packages("wordcloud", repos = "http://cran.us.r-project.org")
library(wordcloud)

wordcloud(corpus_train, min.freq = 40, random.order = FALSE)
```

The `random.order=FALSE` option made more frequent words appear in the middle. `min.freq=40` option sets the cutoff word frequency to be at least 40 times in the corpus object. Therefore, the words must be appear in at least 40 medication summaries to be shown on the graph.

We can also visualize the difference between early stages and later stages using this type of graph (Figs. 8.2 and 8.3).

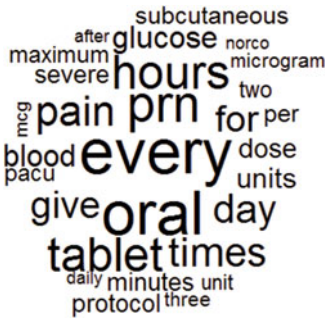

Fig. 8.1 A wordle diagram representing the common terms (frequency exceeding 40) in the head and neck (H&N) text corpus

Fig. 8.2 Wordle plot of the common tersm included in the medical treatment summary corpus of the **early stage** cancer patients

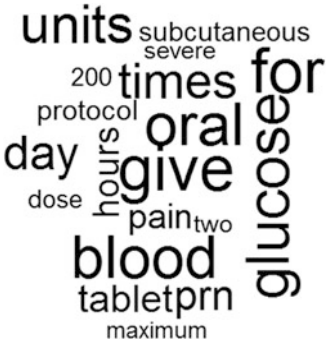

Fig. 8.3 Wordle plot of the common terms included in the medical treatment summary corpus of the **later stage** cancer patients (compare to Fig. 8.2)

```
early<-subset(hn_med_train, stage=="early_stage")
later<-subset(hn_med_train, stage=="later_stage")
wordcloud(early$MEDICATION_SUMMARY, max.words = 20)
wordcloud(later$MEDICATION_SUMMARY, max.words = 20)
```

We can see that the frequent words are somewhat different in the medication summary between early stage and later stage patients.

Data Preparation: Creating Indicator Features for Frequent Words

For simplicity, we utilize the medication summary as the only feature to classify cancer stages. You may recall that in Chap. 7 we used features for classifications. *In this study, we are going to make the frequencies of words into features.*

```
summary(findFreqTerms(hn_med_dtm_train, 5))

##    Length     Class      Mode
##       103 character character

hn_med_dict <-as.character(findFreqTerms(hn_med_dtm_train, 5))
hn_train <-DocumentTermMatrix(corpus_train, list(dictionary=hn_med_dict))
hn_test <-DocumentTermMatrix(corpus_test, list(dictionary=hn_med_dict))
```

The above code limits the document term matrix with words that have appeared in at least five different documents. This created 103 features for us to use.

The Naive Bayes classifier trains on data with categorical features. Thus, we need to transform our word count features into categorical data. A way to do this is to change the count into an indicator of whether this word appears. We can create a function of our own to deal with this.

```
convert_counts <- function(x) {
   x <- ifelse(x > 0, 1, 0)
   x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
   return(x)
}
```

An important statement is `x<-ifelse(x>0, 1, 0)`. This is saying that if we have an `x` that is greater than 0, we assign value 1 to it, otherwise the value is set to 0.

Now let's apply our own function `convert_counts()` on each column (`MARGIN=2`) of the training and testing datasets.

```
hn_train <- apply(hn_train, MARGIN = 2, convert_counts)
hn_test <- apply(hn_test, MARGIN = 2, convert_counts)
```

So far, we successfully created indicators for words that appeared at least in five different documents in the training data.

8.5.3 Step 3: Training a Model on the Data

The package we will use for Naive Bayes classifier is called `e1071`.

```
# install.packages("e1071", repos = "http://cran.us.r-project.org")
library(e1071)
```

The function `NaiveBayes()` has following components:

```
m<-naiveBayes(train, class, laplace=0)
```

- **train**: data frame containing numeric training data (features)
- **class**: factor vector with the class for each row in the training data.
- **laplace**: positive double controlling Laplace smoothing; default is zero and disables Laplace smoothing.

Let's build our classifier first.

```
hn_classifier <- naiveBayes(hn_train, hn_med_train$stage)
```

Then, we can use the classifier to make predictions using `predict()`. Recall that when we presented the AdaBoost example in Chap. 3, we saw the basic mechanism of machine-learning training, prediction and assessment.

The function `predict()` has the following components:

```
p<-predict(m, test, type="class")
```

- m: classifier trained by `NaiveBayes()`
- test: test data frame or matrix
- type: either `"class"` or `"raw"` specifies whether the predictions should be the most likely class value or the raw predicted probabilities.

```
hn_test_pred<-predict(hn_classifier, hn_test)
```

8.5.4 Step 4: Evaluating Model Performance

Similarly to the approach in Chap. 7, we use cross table to compare predicted class and the true class of our test dataset.

```
library(gmodels)
CrossTable(hn_test_pred, hn_med_test$stage)

##    Cell Contents
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
## Total Observations in Table:  133
##              | hn_med_test$stage
## hn_test_pred | early_stage | later_stage |   Row Total |
## -------------|-------------|-------------|-------------|
##  early_stage |          90 |          24 |         114 |
##              |       0.008 |       0.032 |             |
##              |       0.789 |       0.211 |       0.857 |
##              |       0.849 |       0.889 |             |
##              |       0.677 |       0.180 |             |
## -------------|-------------|-------------|-------------|
##  later_stage |          16 |           3 |          19 |
##              |       0.049 |       0.190 |             |
##              |       0.842 |       0.158 |       0.143 |
##              |       0.151 |       0.111 |             |
##              |       0.120 |       0.023 |             |
## -------------|-------------|-------------|-------------|
## Column Total |         106 |          27 |         133 |
##              |       0.797 |       0.203 |             |
```

It may be worth skipping forward to Chap. 14, where we present a summary table for the key measures used to evaluate the performance of binary tests, classifiers, or predictions.

The prediction accuracy:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} = \frac{93}{133} = 0.7.$$

From the cross table we can see that our prediction accuracy is $\frac{93}{133} = 0.70$. However, the *later stage* classification only has three counts. This might be due to the $P(F_i|\, C_L) \sim 0$ problem that we discussed above.

8.5.5 Step 5: Improving Model Performance

After setting `laplace=15`, the accuracy goes up to 76%. Although this is a small improvement in terms of accuracy, we have a better chance of detecting *later stage* patients.

```
hn_classifier <- naiveBayes(hn_train, hn_med_train$stage, laplace = 15)
hn_test_pred<-predict(hn_classifier, hn_test)
CrossTable(hn_test_pred, hn_med_test$stage)

##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
## Total Observations in Table:  133
##
##              | hn_med_test$stage
## hn_test_pred | early_stage | later_stage |   Row Total |
## -------------|-------------|-------------|-------------|
##  early_stage |          99 |          25 |         124 |
##              |       0.000 |       0.001 |             |
##              |       0.798 |       0.202 |       0.932 |
##              |       0.934 |       0.926 |             |
##              |       0.744 |       0.188 |             |
## -------------|-------------|-------------|-------------|
##  later_stage |           7 |           2 |           9 |
##              |       0.004 |       0.016 |             |
##              |       0.778 |       0.222 |       0.068 |
##              |       0.066 |       0.074 |             |
##              |       0.053 |       0.015 |             |
## -------------|-------------|-------------|-------------|
## Column Total |         106 |          27 |         133 |
##              |       0.797 |       0.203 |             |
## -------------|-------------|-------------|-------------|
```

8.5.6 Step 6: Compare Naive Bayesian against LDA

As mentioned earlier, Naive Bayes with normality assumption is a special case of Discriminant Analysis. It might be interesting to compare the results against LDA.

```
library(MASS)
df_hn_train = data.frame(lapply(as.data.frame(hn_train),as.numeric), stage =
hn_med_train$stage)
df_hn_test = data.frame(lapply(as.data.frame(hn_test),as.numeric), stage = h
n_med_test$stage)

hn_lda <- lda(data=df_hn_train, stage~.)

# hn_pred = predict(hn_lda, df_hn_test[,-104])
hn_pred = predict(hn_lda, df_hn_test)
CrossTable(hn_pred$class, df_hn_test$stage)

##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
## Total Observations in Table:  133
##
##                | df_hn_test$stage
## hn_pred$class | early_stage | later_stage |   Row Total |
## --------------|-------------|-------------|-------------|
##   early_stage |          66 |          22 |          88 |
##               |       0.244 |       0.957 |             |
##               |       0.750 |       0.250 |       0.662 |
##               |       0.623 |       0.815 |             |
##               |       0.496 |       0.165 |             |
## --------------|-------------|-------------|-------------|
##   later_stage |          40 |           5 |          45 |
##               |       0.477 |       1.872 |             |
##               |       0.889 |       0.111 |       0.338 |
##               |       0.377 |       0.185 |             |
##               |       0.301 |       0.038 |             |
## --------------|-------------|-------------|-------------|
##  Column Total |         106 |          27 |         133 |
##               |       0.797 |       0.203 |             |
## --------------|-------------|-------------|-------------|
```

Here, Naive Bayes outperforms LDA classifier in terms of the overall accuracy. However, LDA has a lower type II error $\left(\frac{22}{133}\right)$, which is clinically important in order to avoid misdiagnosing later-stage cancer patients as early stage.

In later chapters, we will step deeper into the space of classification problems and see more sophisticated approaches.

8.6 Practice Problem

In the previous case study, we classified the patients with `seer_stage` of "not applicable"(`seer_stage=8`) and "unstaged, unknown or unspecified"(`seer_stage=9`) as no cancer or early cancer stages. Let's remove these two categories and replicate the Naive Bayes classifier case study again.

```
hn_med1<-hn_med[!hn_med$seer_stage %in% c(8, 9), ]
str(hn_med1); dim(hn_med1)

## 'data.frame':    580 obs. of  9 variables:
##  $ PID              : int  10000 10008 10029 10063 10103 1012 10135 10143
10152 10184 ...
##  $ ENC_ID           : int  46836 46886 47034 47240 47511 3138 47739 47769
47800 47938 ...
##  $ seer_stage       : Factor w/ 9 levels "0","1","2","3",..: 2 2 5 2 2 2
2 2 7 2 ...
##  $ MEDICATION_DESC   : chr  "ranitidine" "heparin injection" "ampicillin/
sulbactam IVPB UH" "fentaNYL injection UH" ...
##  $ MEDICATION_SUMMARY: chr  "(Zantac) 150 mg tablet oral two times a day"
"5,000 unit subcutaneous three times a day" "(Unasyn) 15 g IV every 6 hours"
"25 - 50 microgram IV every 5 minutes PRN severe pain\nMaximum dose 200 mcg
Per PACU protocol" ...
##  $ DOSE              : chr  "150" "5000" "1.5" "50" ...
##  $ UNIT              : chr  "mg" "unit" "g" "microgram" ...
##  $ FREQUENCY         : chr  "two times a day" "three times a day" "every
6 hours" "every 5 minutes" ...
##  $ TOTAL_DOSE_COUNT  : int  5 3 11 2 2 2 6 1 24 2 ...

## [1] 580   9
```

Now we have only 580 observations. We can either use the first 480 of them as the training dataset and the last 100 as the test dataset, or select 80–20 (training-testing) split, and evaluate the prediction accuracy when `laplace=1`?

We can use the same code for creating the classes in training and test dataset. Since the `seer_stage=8 or 9` is not in the data, we classify `seer_stage=0, 1, 2 or 3` as "early_stage" and `seer_stage=4, 5 or 7` as "later_stage".

```
hn_med_train1$stage<-hn_med_train1$seer_stage %in% c(4, 5, 7)
hn_med_train1$stage<-factor(hn_med_train1$stage, levels=c(F, T), labels = c(
"early_stage", "later_stage"))
hn_med_test1$stage<-hn_med_test1$seer_stage %in% c(4, 5, 7)
hn_med_test1$stage<-factor(hn_med_test1$stage, levels=c(F, T), labels = c("e
arly_stage", "later_stage"))
prop.table(table(hn_med_train1$stage))

##
## early_stage later_stage
##   0.7392241   0.2607759

prop.table(table(hn_med_test1$stage))

##
## early_stage later_stage
##   0.7413793   0.2586207
```

Use terms that have appeared in five or more documents in the training dataset to build the document term matrix.

```
##    Length     Class      Mode
##       112 character character

##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  116
##
##
##                | hn_med_test1$stage
## hn_test_pred1 | early_stage | later_stage |   Row Total |
## --------------|-------------|-------------|-------------|
##   early_stage |          84 |          28 |         112 |
##               |       0.011 |       0.032 |             |
##               |       0.750 |       0.250 |       0.966 |
##               |       0.977 |       0.933 |             |
##               |       0.724 |       0.241 |             |
## --------------|-------------|-------------|-------------|
##   later_stage |           2 |           2 |           4 |
##               |       0.314 |       0.901 |             |
##               |       0.500 |       0.500 |       0.034 |
##               |       0.023 |       0.067 |             |
##               |       0.017 |       0.017 |             |
## --------------|-------------|-------------|-------------|
##  Column Total |          86 |          30 |         116 |
##               |       0.741 |       0.259 |             |
## --------------|-------------|-------------|-------------|
```

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} = \frac{86}{116} = 0.74.$$

Try to reproduce these results with some new data from the list of our Case-Studies.

8.7 Assignments 8: Probabilistic Learning: Classification Using Naive Bayes

8.7.1 Explain These Two Concepts

- Bayes Theorem
- Laplace Estimation

8.7.2 Analyzing Textual Data

Load the SOCR 2011 US Job Satisfaction data. The last column (`Description`) contains free text about each job. Notice that spaces are replaced by underscores, __. Mine the text field and suggest some the meta-data analytics.

- Convert the textual meta-data into a corpus object.
- Triage some of the irrelevant punctuation and other symbols in the corpus document, change all text to lower case, etc.
- Tokenize the job descriptions into words. Examine the distributions of `Stress_Category` and `Hiring_Potential`.
- Classify the job stress into two categories.
- Generate a word cloud to visualize the job description text.
- Graphically visualize the difference between low and high Stress_Category graph.
- Transform the word count features into categorical data
- Ignore those low frequency words and report the sparsity of your categorical data matrix with or without delete those low frequency words.
- Apply the Naive Bayes classifier to original matrix and lower dimension matrix. What do you observe?
- Apply and compare LDA and Naive Bayes classifiers with respect to the error, specificity and sensitivity.

References

Kidwell , David A. (2013) *Lazy Learning*, Springer Science & Business Media, ISBN 9401720533, 9789401720533

Aggarwal, Charu C. (ed.) (2015) *Data Classification: Algorithms and Applications*, Chapman & Hall/CRC, ISBN 1498760589, 9781498760584