Practice Problems 4

Jordan Lian

2/28/2021

```
library(tidyverse)
```

```
## -- Attaching packages -----------------------------------------------------------------------
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts -----------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Problem 1 (50 Points)

Build an R Notebook of the SMS message filtering example in the textbook on pages 103 to 123 (data set). Show each step and add appropriate documentation. Note that the attached data set differs slightly from the one usedon the book; the number of cases differ.'

Step 1: Collecting Data

Here we are looking at different types of text (ham vs spam), and we want to develop a Naive Bayes classifier to determine what type of text fits what class.

Step 2: Exploring and Preparing the Data

```
sms_raw <- read_csv('da5030.spammsgdataset.csv')
```

```
## Parsed with column specification:
## cols(
##   type = col_character(),
##   text = col_character()
## )
```

```
sms_raw
```

```
## # A tibble: 5,574 x 2
##    type  text
##    <chr> <chr>
##  1 ham   Go until jurong point, crazy.. Available only in bugis n great world l~
##  2 ham   Ok lar... Joking wif u oni...
##  3 spam  Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Te~
##  4 ham   U dun say so early hor... U c already then say...
##  5 ham   Nah I don't think he goes to usf, he lives around here though
##  6 spam  FreeMsg Hey there darling it's been 3 week's now and no word back! I'd~
##  7 ham   Even my brother is not like to speak with me. They treat me like aids ~
##  8 ham   As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)'~
##  9 spam  WINNER!! As a valued network customer you have been selected to receiv~
## 10 spam  Had your mobile 11 months or more? U R entitled to Update to the lates~
## # ... with 5,564 more rows
```

```r
# Convert to factor
sms_raw$type <- factor(sms_raw$type)

str(sms_raw$type)
```

```
##  Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
```

```r
table(sms_raw$type)
```

```
##
##  ham spam
## 4827  747
```

Cleaning and Standardizing the Data: Here we start to remove numbers, punctuation, and uninteresting words so we can get to the bottom of what ham vs spam text is like.

```r
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```r
# Create a corpus, which is a collection of text documents
sms_corpus <- VCorpus(VectorSource(sms_raw$text))

# Inspect the corpus
inspect(sms_corpus[1:2])
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
```

```
## Content:  documents: 2
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 111
##
## [[2]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 29
```

```r
# View the text
as.character(sms_corpus[[1]])
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
```

```r
# View multiple documents
lapply(sms_corpus[1:2], as.character)
```

```
## $`1`
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
##
## $`2`
## [1] "Ok lar... Joking wif u oni..."
```

```r
# Standardize to all lowercase
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))

# Make sure tm_map() function worked
as.character(sms_corpus[[1]])
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there g
```

```r
as.character(sms_corpus_clean[[1]])
```

```
## [1] "go until jurong point, crazy.. available only in bugis n great world la e buffet... cine there g
```

```r
# Remove all the numbers from the corpus
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)

# Remove the filler words like and, if, but, etc using the stopwords() function
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())

# Remove punctuation to complete the standardization
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)

# Reduce words to their root form, a.k.a. stemming
library(SnowballC)
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

```r
# Strip the additional whitespace to finalize the data cleaning
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)

# Before cleaning
as.character(sms_corpus[[1]])
```

```
## [1] "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there
```

```r
# After cleaning
as.character(sms_corpus_clean[[1]])
```

```
## [1] "go jurong point crazi avail bugi n great world la e buffet cine got amor wat"
```

Splitting Text Documents into Words: We want to create a document term matrix (DTM) using the DocumentTermMatrix() function. a DTM is a data structure where the rows indicate documents (SMS messages) and columns indicate terms (words).

```r
# original dtm
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6592)>>
## Non-/sparse entries: 42608/36701200
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```r
# modify preprocessing steps
sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  removePunctuation = TRUE,
  stemming = TRUE
))
sms_dtm2
```

```
## <<DocumentTermMatrix (documents: 5574, terms: 6995)>>
## Non-/sparse entries: 43713/38946417
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

Create training/test datasets: We always do this, just like we have done before.

```r
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test  <- sms_dtm[4170:5559, ]

sms_train_labels <- sms_raw[1:4169, ]$type
sms_test_labels  <- sms_raw[4170:5559, ]$type

prop.table(table(sms_train_labels))
```

```
## sms_train_labels
##       ham      spam
## 0.8647158 0.1352842
```

```r
prop.table(table(sms_test_labels))
```

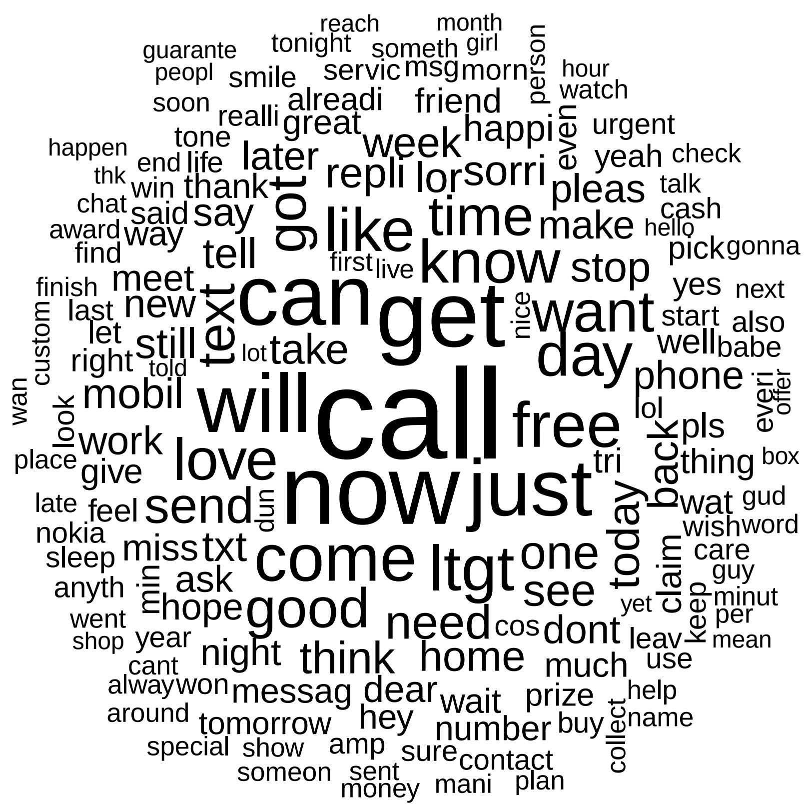
```
## sms_test_labels
##       ham      spam
## 0.8697842 0.1302158
```

Visualizing text data - word clouds: This is just a way to show the frequency of words.

```r
# Original word cloud
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

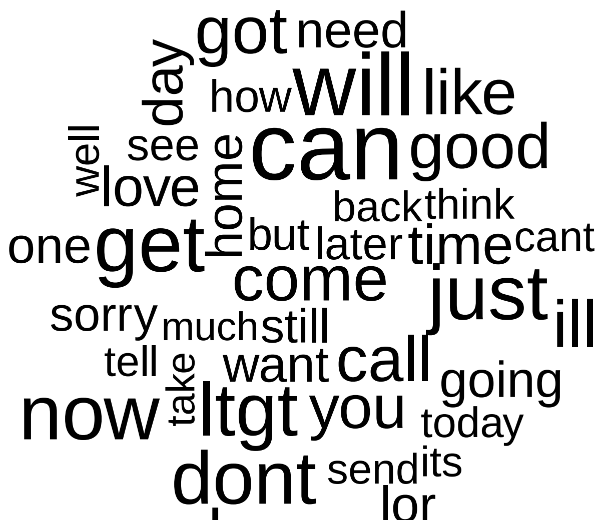
```r
wordcloud(sms_corpus_clean, min.freq = 50, random.order = FALSE)
```




```r
# Subset the different types
spam <- subset(sms_raw, type == "spam")
ham <- subset(sms_raw, type == "ham")

# Spam word cloud
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
```


```r
# Ham word cloud
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



Creating indicator features for frequent words: We want to find the frequent words we can count as a reference for the classifier.

```
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
str(sms_freq_words)
```

```
##  chr [1:1157] "£wk" "abiola" "abl" "abt" "accept" "access" "account" ...
```

```
sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]

sms_dtm_freq_train
```

```
## <<DocumentTermMatrix (documents: 4169, terms: 1157)>>
## Non-/sparse entries: 25173/4798360
## Sparsity           : 99%
## Maximal term length: 13
## Weighting          : term frequency (tf)
```

```
sms_dtm_freq_test
```

```
## <<DocumentTermMatrix (documents: 1390, terms: 1157)>>
## Non-/sparse entries: 8165/1600065
```

```
## Sparsity           : 99%
## Maximal term length: 13
## Weighting          : term frequency (tf)
```

```r
# Convert counts to yes/no strings
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}

# Apply the function to the new training and test datasets
sms_train <- apply(sms_dtm_freq_train, MARGIN = 2,
                   convert_counts)
sms_test <- apply(sms_dtm_freq_test, MARGIN = 2,
                  convert_counts)
```

Step 3, Train a model on the data

```r
library(e1071)
sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

Step 4, Evaluate model performance

We use the same stuff that we used for kNN, where we use CrossTable to look at the accuracy of the model.

```r
sms_test_pred <- predict(sms_classifier, sms_test)

# Use CrossTable from gmodels
library(gmodels)
```

```
## Warning: package 'gmodels' was built under R version 4.0.3
```

```r
CrossTable(sms_test_pred, sms_test_labels,    prop.chisq = FALSE, prop.t = FALSE,    dnn = c('predicted
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
```

```
##          ham |      1200 |        20 |      1220 |
##              |     0.984 |     0.016 |     0.878 |
##              |     0.993 |     0.110 |           |
## -------------|-----------|-----------|-----------|
##         spam |         9 |       161 |       170 |
##              |     0.053 |     0.947 |     0.122 |
##              |     0.007 |     0.890 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1209 |       181 |      1390 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Step 5, Improve model performance

This time we changed the Laplace estimator to 1, which reduced our false positives.

```r
sms_classifier2 <- naiveBayes(sms_train, sms_train_labels, laplace = 1)
sms_test_pred2 <- predict(sms_classifier2, sms_test)
CrossTable(sms_test_pred2, sms_test_labels,
           prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
           dnn = c('predicted', 'actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  1390
##
##
##              | actual
##    predicted |       ham |      spam | Row Total |
## -------------|-----------|-----------|-----------|
##          ham |      1182 |        10 |      1192 |
##              |     0.978 |     0.055 |           |
## -------------|-----------|-----------|-----------|
##         spam |        27 |       171 |       198 |
##              |     0.022 |     0.945 |           |
## -------------|-----------|-----------|-----------|
## Column Total |      1209 |       181 |      1390 |
##              |     0.870 |     0.130 |           |
## -------------|-----------|-----------|-----------|
##
##
```

Problem 2 (50 Points)

Install the requisite packages to execute the following code that classifies the built-in **iris** data using Naive Bayes. Build an R Notebook and explain in detail what each step does. Be sure to look up each function to understand how it is used.

```
library(klaR)
data(iris)

nrow(iris)
summary(iris)
head(iris)

testidx <- which(1:length(iris[, 1]) %% 5 == 0)

# separate into training and testing datasets
iristrain <- iris[-testidx,]
iristest <- iris[testidx,]

# apply Naive Bayes
nbmodel <- NaiveBayes(Species~., data=iristrain)

# check the accuracy
prediction <- predict(nbmodel, iristest[,-5])
table(prediction$class, iristest[,5])
```

Data Preparation

There isn't too much we have to do for data preparation since the data is nicely set up for us in R.

```
# Install packages and show data
library(klaR)
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
data(iris)

# Get rows, summary, and head of data
nrow(iris)
```

```
## [1] 150
```

```r
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

```r
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
# get all multiples of 150 that are divisible by 5
testidx <- which(1:length(iris[, 1]) %% 5 == 0)
testidx
```

```
##  [1]   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90  95
## [20] 100 105 110 115 120 125 130 135 140 145 150
```

```r
# separate into training and testing datasets
iristrain <- iris[-testidx,]
iristest <- iris[testidx,]
```

Naive Bayes Model

Basically for this model, we use the species as the given part of Bayes Theorem where we look for a probability given a condition. The condition here is the species. For instance, we will get the sepal length, petal length, etc given each species (setosa, versicolor, virginica).

```r
# apply Naive Bayes
library(naivebayes)
```

```
## naivebayes 0.9.7 loaded
```

```r
nbmodel <- naive_bayes(Species~., data=iristrain)
nbmodel
```

```
##
## ================================== Naive Bayes ==================================
##
##  Call:
## naive_bayes.formula(formula = Species ~ ., data = iristrain)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 0
##
## ---------------------------------------------------------------------------------
##
##  A priori probabilities:
##
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## ---------------------------------------------------------------------------------
##
##  Tables:
##
## ---------------------------------------------------------------------------------
##  ::: Sepal.Length (Gaussian)
## ---------------------------------------------------------------------------------
##
## Sepal.Length    setosa versicolor virginica
##         mean 4.9975000  5.9900000 6.6100000
##         sd   0.3675892  0.5295378 0.6647922
##
## ---------------------------------------------------------------------------------
##  ::: Sepal.Width (Gaussian)
## ---------------------------------------------------------------------------------
##
## Sepal.Width    setosa versicolor virginica
##        mean 3.4175000  2.7775000 2.9700000
##        sd   0.3960623  0.3415556 0.3081791
##
## ---------------------------------------------------------------------------------
##  ::: Petal.Length (Gaussian)
## ---------------------------------------------------------------------------------
##
## Petal.Length    setosa versicolor virginica
##         mean 1.4425000  4.3100000 5.5575000
##         sd   0.1583367  0.4850588 0.5930743
##
## ---------------------------------------------------------------------------------
##  ::: Petal.Width (Gaussian)
## ---------------------------------------------------------------------------------
##
## Petal.Width    setosa versicolor virginica
```

```
##        mean 0.2525000  1.3325000 2.0300000
##        sd   0.1109111  0.2080280 0.2355572
##
## ---------------------------------------------------------------------------------
```

Accuracy of Model

This just tracks the predictions against the actual results which are in the iristest data. We use the table function to count how many accurate predictions we have.

```
# check the accuracy
prediction <- predict(nbmodel, iristest[,-5])
prediction
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     versicolor versicolor
## [13] versicolor versicolor versicolor versicolor versicolor versicolor
## [19] versicolor versicolor virginica  virginica  virginica  versicolor
## [25] virginica  virginica  versicolor virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
table(prediction, iristest[,5])
```

```
##
## prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         2
##   virginica       0          0         8
```

It looks like setosas have a 100% accuracy, versicolors had 10/12 correct guesses, and virginicas had 8/10 correct guesses. Not bad.