Final Exam_YL

Yating Liao (7636428840)

2023-05-01

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
library(tinytex)
```

```
library(NLP)
library(tm) # VCorpus( ), tm_map( ), findFreqTerms( )
# read all as character columns
df0 <- read.csv("sms.csv", stringsAsFactors = FALSE)
str(df0)
```

```
## 'data.frame':    5559 obs. of  2 variables:
##  $ type: chr  "ham" "ham" "ham" "spam" ...
##  $ text: chr  "Hope you are having a good week. Just checking in" "K..give back my thanks." "Am
```

```
df0$type = as.factor(df0$type)
str(df0)
```

```
## 'data.frame':    5559 obs. of  2 variables:
##  $ type: Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
##  $ text: chr  "Hope you are having a good week. Just checking in" "K..give back my thanks." "Am
```

```
# a)
# build a corpus (a collection of messages suitable for text mining)
sms_corpus <- VCorpus(VectorSource(df0$text))
# examine it
as.character(sms_corpus[[1]])
```

```
## [1] "Hope you are having a good week. Just checking in"
```

```r
lapply(sms_corpus[1:2], as.character)
```

```
## $`1`
## [1] "Hope you are having a good week. Just checking in"
##
## $`2`
## [1] "K..give back my thanks."
```

```r
# change all words to lowercase
sms_corpus_clean <- tm_map(sms_corpus, content_transformer(tolower))
as.character(sms_corpus_clean[[1]])
```

```
## [1] "hope you are having a good week. just checking in"
```

```r
# remove numbers
sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
# remove stop words
sms_corpus_clean <- tm_map(sms_corpus_clean, removeWords, stopwords())
# remove punctuation
sms_corpus_clean <- tm_map(sms_corpus_clean, removePunctuation)
# example of word stemming
library(SnowballC)
wordStem(c("learn", "learned", "learning", "learns"))
```

```
## [1] "learn" "learn" "learn" "learn"
```

```r
#
# replace words by stem words
sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
# eliminate unneeded whitespace
sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
# compare original with the final clean corpus
lapply(sms_corpus[1:3], as.character)
```

```
## $`1`
## [1] "Hope you are having a good week. Just checking in"
##
## $`2`
## [1] "K..give back my thanks."
##
## $`3`
## [1] "Am also doing in cbe only. But have to pay."
```

```
lapply(sms_corpus_clean[1:3], as.character)
```

```
## $`1`
## [1] "hope good week just check"
##
## $`2`
## [1] "kgive back thank"
##
## $`3`
## [1] "also cbe pay"
```

b) (10 pts.) Convert the tm object sms_corpus_clean to a Document term matrix DTM as follows. How many binary columns does the matrix has?

```
sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
sms_dtm
```

```
## <<DocumentTermMatrix (documents: 5559, terms: 6559)>>
## Non-/sparse entries: 42147/36419334
## Sparsity           : 100%
## Maximal term length: 40
## Weighting          : term frequency (tf)
```

```
dim(sms_dtm)
```

```
## [1] 5559 6559
```

```
# It has 6559 number of columns.
```

c) (10 pts.) Split the matrix into train set (first 4169 rows) and test set. Further simplify these sets by keeping words that appear at least 5 times in the data.

```
# split into train and test sets
m = 4169
sms_dtm_train <- sms_dtm[1:m, ]
sms_dtm_test <- sms_dtm[(m+1):5559, ] # m+1 = 4170
dim(sms_dtm_train)
```

```
## [1] 4169 6559
```

```
sms_train_labels <- df0[1:m, ]$type
sms_test_labels <- df0[(m+1):5559, ]$type
```

```
# vector with words appearing at least 5 times
sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
# show some of them
```

```
set.seed(2)
sample(sms_freq_words,12)
```

```
##  [1] "that"   "pay"    "rather" "happi"  "gone"   "dude"   "order"  "natur"
##  [9] "kinda"  "ignor"  "island" "rose"
```

```
# DTMs with only the frequent terms
sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]
sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
# a function that converts 1/0 to Yes/No
convert_counts = function(x) x = ifelse(x > 0, "Yes", "No")
# Use convert_counts() to the columns of the train/test sets
sms_train <- apply(sms_dtm_freq_train, 2, convert_counts)
sms_test <- apply(sms_dtm_freq_test, 2, convert_counts)
dim(sms_test)
```

```
## [1] 1390 1139
```

d) (20 pts.) Use the train set to build a Naive Bayes model. Use it to predict the test set with threshold equal to 0.50. Report the TPR and FPR.

```
# Fitting Naive Bayes model on train set
library(e1071)
NBmodel <- naiveBayes(sms_train, sms_train_labels)
# Compute predicted probabilities for test set
Probabs = predict(NBmodel, sms_test, type = "raw")
NBproba <- Probabs[,2]
head(NBproba)
```

```
## [1] 4.018928e-07 7.852061e-06 1.452211e-04 3.875736e-05 1.000000e+00
## [6] 1.489135e-04
```

```
# Set the threshold
threshold <- 0.5
# Calculate the True Positive Rate (TPR) and False Positive Rate (FPR)
NBPredict <- ifelse(NBproba >= threshold, "spam", "ham")
confus_mat_nb <- table(sms_test_labels,NBPredict)
confus_mat_nb
```

```
##                  NBPredict
## sms_test_labels   ham spam
##            ham   1201    6
##            spam    30  153
```

```
prop.table(confus_mat_nb)
```

```
##                NBPredict
## sms_test_labels         ham        spam
##            ham  0.864028777 0.004316547
##            spam 0.021582734 0.110071942
```

```
NB_TPR <- confus_mat_nb[2, 2] / sum(confus_mat_nb[2, ])
NB_FPR <- confus_mat_nb[1, 2] / sum(confus_mat_nb[1, ])
cat("True Positive Rate (TPR): ", NB_TPR, "\n")
```

```
## True Positive Rate (TPR):  0.8360656
```

```
cat("False Positive Rate (FPR): ", NB_FPR, "\n")
```

```
## False Positive Rate (FPR):  0.004971002
```

```
# Calculate the Area Under the ROC Curve (AUC)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## 载入程辑包：'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
ROC_curve <- roc(sms_test_labels, NBproba)
```

```
## Setting levels: control = ham, case = spam
```

```
## Setting direction: controls < cases
```

```
auc <- auc(ROC_curve)
cat("Area Under the ROC Curve (AUC): ", auc, "\n")
```

```
## Area Under the ROC Curve (AUC):  0.984177
```

e) (20 pts.) Change the threshold to improve the test positive accuracy rate. Report the improved TPR and FPR.

```
# loop for ROC Curve
cutoff = seq(0.001,0.92,0.001)
n = length(cutoff)
n
```

```
## [1] 920
```

```
TPR = rep(0,n)
FPR = rep(0,n)
#
for(i in cutoff)
{
NBPredict2 <- ifelse(NBproba >= i, "spam", "ham")
confusionmat_nb = as.matrix(table(sms_test_labels, NBPredict2))
j = n*i
TPR[j] = confusionmat_nb[2,2]/sum(confusionmat_nb[2, ])
FPR[j] = confusionmat_nb[1,2]/sum(confusionmat_nb[1, ])
}
#
df1 = data.frame(cutoff,TPR,FPR)
head(df1,15)
```

```
##    cutoff       TPR        FPR
## 1   0.001 0.9726776 0.10770505
## 2   0.002 0.9672131 0.08202154
## 3   0.003 0.9562842 0.07539354
## 4   0.004 0.9562842 0.06545153
## 5   0.005 0.9562842 0.05965203
## 6   0.006 0.9398907 0.05468103
## 7   0.007 0.9398907 0.05219553
## 8   0.008 0.9398907 0.04473902
## 9   0.009 0.9398907 0.04142502
## 10  0.010 0.9398907 0.03811102
## 11  0.011 0.9398907 0.03396852
## 12  0.012 0.9344262 0.03396852
## 13  0.013 0.9344262 0.03314002
## 14  0.014 0.9289617 0.03314002
## 15  0.015 0.9289617 0.03314002
```

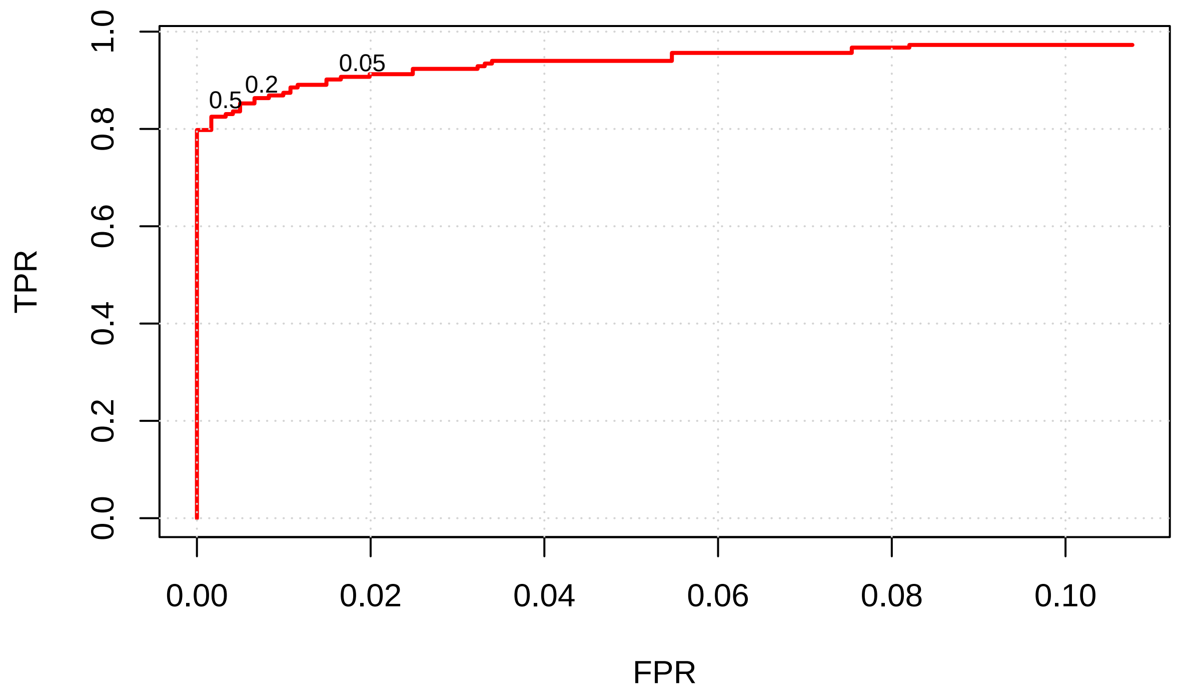
```
which(df1$cutoff == 0.50,)
```

```
## [1] 500
```

```
df1[500,]
```

```
##     cutoff       TPR         FPR
## 500    0.5 0.8306011 0.003314002
```

```r
# Plotting
plot(FPR,TPR,type = "s", col = "red", lwd = 2)
text(FPR[50],TPR[50],labels = df1$cutoff[50], pos = 3, cex = 0.75,offset = 0.15)
text(FPR[200],TPR[200],labels = df1$cutoff[200], pos = 3, cex = 0.75,offset = 0.15)
text(FPR[500],TPR[500],labels = df1$cutoff[500], pos = 3, cex = 0.75,offset = 0.15)
grid()
```



```r
Improved_TPR = df1[200,2]
cat("Improved True Positive Rate (TPR) is: ", Improved_TPR, "\n")
```

```
## Improved True Positive Rate (TPR) is:  0.863388
```

```r
Improved_FPR = df1[200,3]
cat("Improved False Positive Rate (FPR) is: ", Improved_FPR, "\n")
```

```
## Improved False Positive Rate (FPR) is:  0.007456504
```

```r
cat("So a better threshold is:",df1[200,1])
```

```
## So a better threshold is: 0.2
```

2. (30 pts.) The following US map shows the number of coronavirus cases per 100000 residents for each county as of April 21, 2020. The file usmap.csv has the relevant data (the size of the circles

shows the number of cases in each county). You may ignore the cases in the state of Louisiana. You may use Google maps or work offline. Use R to reproduce the map (continental US only, you may ignore Alaska) as close as possible You may use

```
library(readr)
library(maps)
library(ggplot2)
```

```
##
## 载入程辑包：'ggplot2'

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

```
usmap <- read_csv("usmap.csv")
```

```
## New names:
## * `` -> `...1`

## Rows: 2978 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): County
## dbl (6): ...1, FIPS, population, cases, lat, lon
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

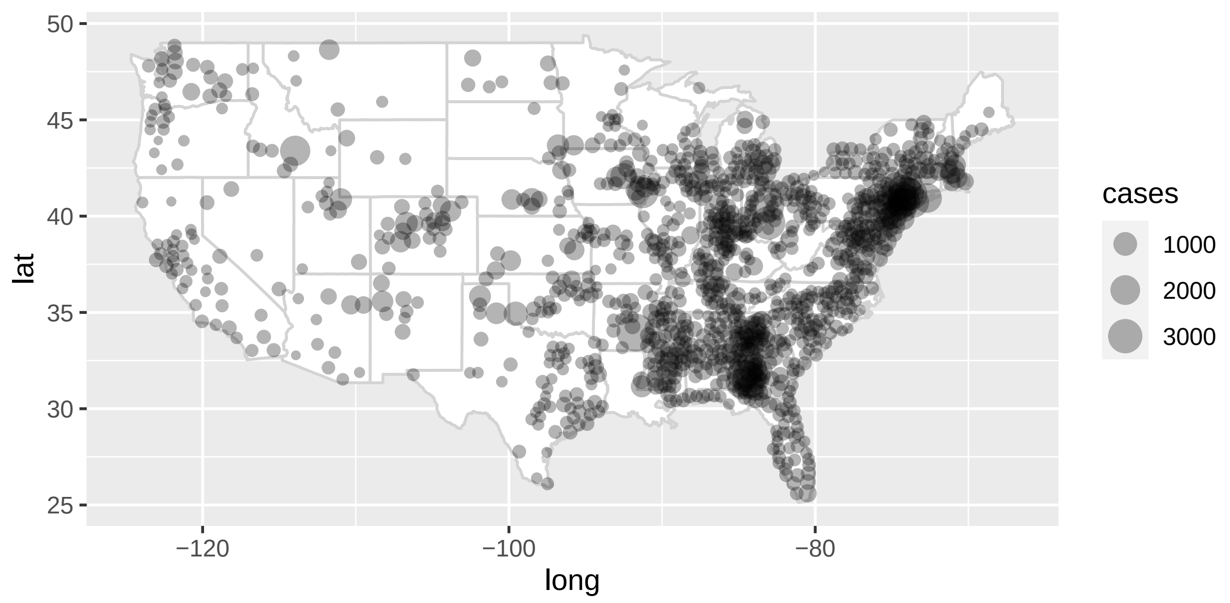
```
str(usmap) # 2978 observations with 7 variables
```

```
## spc_tbl_ [2,978 x 7] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ ...1      : num [1:2978] 1 2 3 4 5 6 7 8 9 10 ...
##  $ FIPS      : num [1:2978] 10001 10003 10005 1001 1003 ...
##  $ County    : chr [1:2978] "Kent County" "New Castle County" "Sussex County" "Autauga County"
##  $ population: num [1:2978] 180786 558753 234225 55869 223234 ...
##  $ cases     : num [1:2978] 503 1352 1317 32 132 ...
##  $ lat       : num [1:2978] 39.1 39.6 38.7 32.5 30.7 ...
##  $ lon       : num [1:2978] -75.5 -75.6 -75.3 -86.6 -87.7 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   ...1 = col_double(),
##   ..   FIPS = col_double(),
##   ..   County = col_character(),
##   ..   population = col_double(),
##   ..   cases = col_double(),
##   ..   lat = col_double(),
##   ..   lon = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
# ignore counties with small number of cases
d3 = usmap[usmap$cases > 22,]
# cases per 100000 residents
d3$cases = 100000*d3$cases/d3$population
head(d3)
```

```
## # A tibble: 6 x 7
##    ...1  FIPS County            population cases   lat   lon
##   <dbl> <dbl> <chr>                  <dbl> <dbl> <dbl> <dbl>
## 1     1 10001 Kent County           180786 278.   39.1 -75.5
## 2     2 10003 New Castle County     558753 242.   39.6 -75.6
## 3     3 10005 Sussex County         234225 562.   38.7 -75.3
## 4     4  1001 Autauga County         55869  57.3  32.5 -86.6
## 5     5  1003 Baldwin County        223234  59.1  30.7 -87.7
## 6     6  1005 Barbour County         24686 117.   31.9 -85.4
```

```
par(mar = c(10, 5, 10, 5))
states = map_data("state")
d3 = d3[d3$lon > -130,]
p <- ggplot(data = states)+geom_polygon(aes(x = long,y = lat,group = group),
                                         col = 'lightgrey', fill = 'white') +
  geom_point(data = d3, aes(x = lon, y = lat, size = cases),col = 'black',alpha = 0.3) +
  coord_quickmap()
p
```



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.