A3_classifying_text_spam

A3_classifying_text_spam

library package

```
options(warn = -1)
library(rpart)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidyverse)

## -- Attaching packages ----------------------------------------------
------------- tidyverse 1.3.0 --

## √ ggplot2 3.3.0     √ purrr   0.3.4
## √ tibble  3.0.1     √ stringr 1.4.0
## √ tidyr   1.1.0     √ forcats 0.5.0
## √ readr   1.3.1

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

library(ggplot2)
```

1. Use rpart to fit and prune (if necessary) a tree predicting spam/non-spam from the common word counts in the wordmatrix matrix.

prepare data

```
load("./A3_datasets/spam.rda")
word.df = data.frame(wordmatrix)
word.df$is_spam = df$is_spam
word.df$is_spam = factor(word.df$is_spam, levels = c(TRUE, FALSE))
```

Get train set and test set

```
# Divide training set and test set
set.seed(0501)
train_test=sample(2,nrow(word.df),replace = T,prob = c(0.75,0.25))
word.train<-word.df[train_test==1,]
word.test<-word.df[train_test==2,]
```

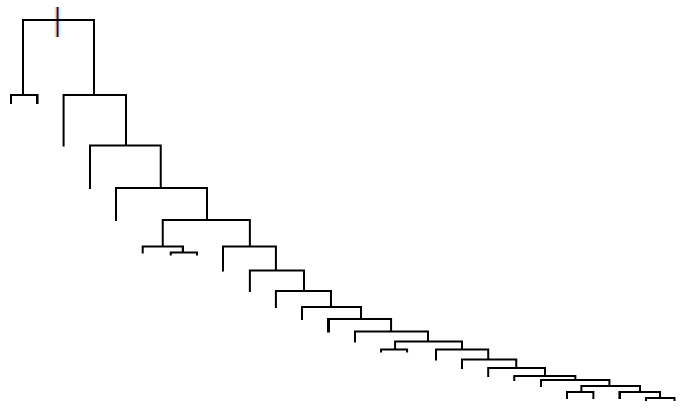
Use rpart to fit a tree

```
mytree <-rpart(is_spam~.,data = word.train, cp = 1e-4)
mytree

## n= 4188
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##       1) root 4188 565 FALSE (0.13490926 0.86509074)
##         2) w_Call>=0.5 120  19 TRUE (0.84166667 0.15833333)
##           4) w_me< 0.5 103   6 TRUE (0.94174757 0.05825243) *
##           5) w_me>=0.5 17   4 FALSE (0.23529412 0.76470588) *
##         3) w_Call< 0.5 4068 464 FALSE (0.11406096 0.88593904)
##           6) w_www>=0.5 58   1 TRUE (0.98275862 0.01724138) *
##           7) w_www< 0.5 4010 407 FALSE (0.10149626 0.89850374)
##            14) w_claim>=0.5 48   0 TRUE (1.00000000 0.00000000) *
##            15) w_claim< 0.5 3962 359 FALSE (0.09061080 0.90938920)
##              30) w_Txt>=0.5 42   3 TRUE (0.92857143 0.07142857) *
##              31) w_Txt< 0.5 3920 320 FALSE (0.08163265 0.91836735)
##                62) w_mobile>=0.5 51  11 TRUE (0.78431373 0.21568627)

##                 124) w_your>=0.5 20   0 TRUE (1.00000000 0.00000000)
 *
##                 125) w_your< 0.5 31  11 TRUE (0.64516129 0.35483871)

##                   250) w_>=3.5 14   2 TRUE (0.85714286 0.14285714) *
##                   251) w_< 3.5 17   8 FALSE (0.47058824 0.52941176)
*
##                63) w_mobile< 0.5 3869 280 FALSE (0.07237012 0.927629
88)
##                 126) w_service>=0.5 27   1 TRUE (0.96296296 0.037037
04) *
##                 127) w_service< 0.5 3842 254 FALSE (0.06611140 0.933
88860)
##                   254) w_FREE>=0.5 26   1 TRUE (0.96153846 0.0384615
4) *
##                   255) w_FREE< 0.5 3816 229 FALSE (0.06001048 0.9399
8952)
##                     510) w_PO>=0.5 17   0 TRUE (1.00000000 0.0000000
0) *
##                     511) w_PO< 0.5 3799 212 FALSE (0.05580416 0.9441
```

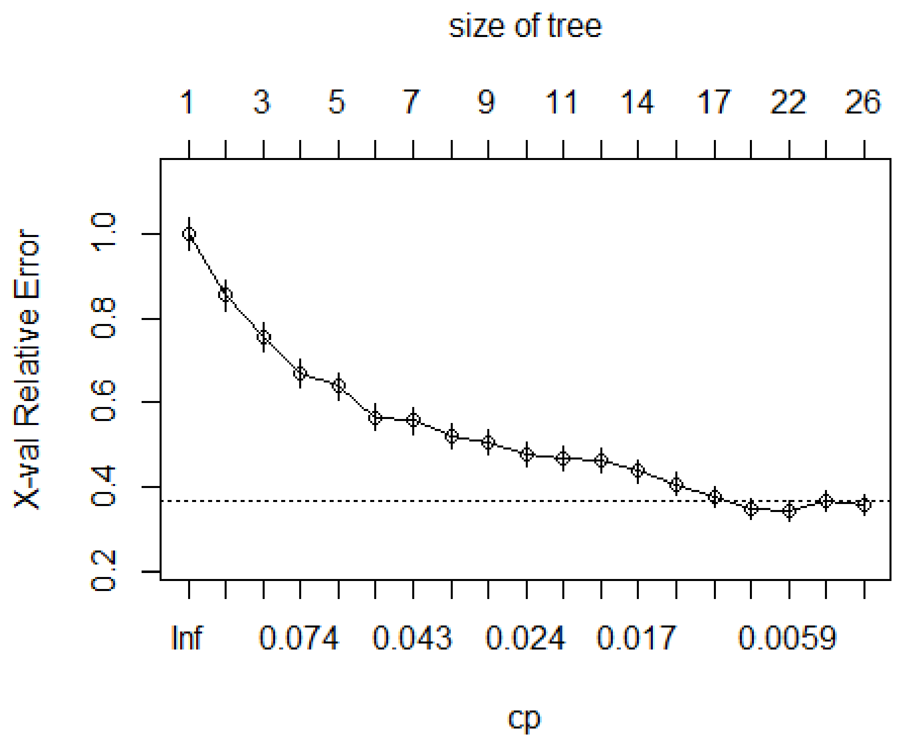
```
9584)
##                        1022) w_com>=0.5 24   5 TRUE (0.79166667 0.2083
3333) *
##                        1023) w_com< 0.5 3775 193 FALSE (0.05112583 0.9
4887417)
##                          2046) w_18>=0.5 13   0 TRUE (1.00000000 0.000
00000) *
##                          2047) w_18< 0.5 3762 180 FALSE (0.04784689 0.
95215311)
##                            4094) w_landline>=0.5 11   0 TRUE (1.000000
00 0.00000000) *
##                            4095) w_landline< 0.5 3751 169 FALSE (0.045
05465 0.95494535)
##                              8190) w_Reply>=0.5 23   7 TRUE (0.6956521
7 0.30434783)
##                               16380) w_the< 0.5 16   2 TRUE (0.8750000
0 0.12500000) *
##                               16381) w_the>=0.5 7   2 FALSE (0.2857142
9 0.71428571) *
##                              8191) w_Reply< 0.5 3728 153 FALSE (0.0410
4077 0.95895923)
##                               16382) w_Free>=0.5 12   0 TRUE (1.000000
00 0.00000000) *
##                               16383) w_Free< 0.5 3716 141 FALSE (0.037
94403 0.96205597)
##                                 32766) w_To>=0.5 16   4 TRUE (0.750000
00 0.25000000) *
##                                 32767) w_To< 0.5 3700 129 FALSE (0.034
86486 0.96513514)
##                                   65534) w_co>=0.5 8   0 TRUE (1.00000
000 0.00000000) *
##                                   65535) w_co< 0.5 3692 121 FALSE (0.0
3277356 0.96722644)
##                                    131070) w_txt>=0.5 12   3 TRUE (0.7
5000000 0.25000000) *
##                                    131071) w_txt< 0.5 3680 112 FALSE 
(0.03043478 0.96956522)
##                                      262142) w_contact>=0.5 15   6 TRU
E (0.60000000 0.40000000) *
##                                      262143) w_contact< 0.5 3665 103 F
ALSE (0.02810368 0.97189632)
##                                        524286) w_Please>=0.5 35  14 FA
LSE (0.40000000 0.60000000)
##                                         1048572) w_message>=0.5 7   0 
TRUE (1.00000000 0.00000000) *
##                                         1048573) w_message< 0.5 28   7
 FALSE (0.25000000 0.75000000) *
##                                        524287) w_Please< 0.5 3630  89 
FALSE (0.02451791 0.97548209)
##                                         1048574) w_STOP>=0.5 7   1 TRU
```

```
E (0.85714286 0.14285714) *
##                                        1048575) w_STOP< 0.5 3623  83 
FALSE (0.02290919 0.97709081)  
##                                          2097150) w_Text>=0.5 7   2 T
RUE (0.71428571 0.28571429) *
##                                          2097151) w_Text< 0.5 3616  7
8 FALSE (0.02157080 0.97842920) *

#check the complexity parameters
printcp(mytree)

## 
## Classification tree:
## rpart(formula = is_spam ~ ., data = word.train, cp = 1e-04)
## 
## Variables actually used in tree construction:
##  [1] w_         w_18       w_Call     w_claim    w_co       w_com     

##  [7] w_contact  w_Free     w_FREE     w_landline w_me       w_messag
e
## [13] w_mobile   w_Please   w_PO       w_Reply    w_service  w_STOP    

## [19] w_Text     w_the      w_To       w_txt      w_Txt      w_www     

## [25] w_your    
## 
## Root node error: 565/4188 = 0.13491
## 
## n= 4188 
## 
##            CP nsplit rel error  xerror     xstd
## 1  0.14513274      0   1.00000 1.00000 0.039130
## 2  0.09911504      1   0.85487 0.85487 0.036586
## 3  0.08495575      2   0.75575 0.75575 0.034659
## 4  0.06371681      3   0.67080 0.67080 0.032860
## 5  0.05132743      4   0.60708 0.63894 0.032146
## 6  0.04424779      5   0.55575 0.56460 0.030384
## 7  0.04247788      6   0.51150 0.55752 0.030208
## 8  0.03008850      7   0.46903 0.52035 0.029263
## 9  0.02477876      8   0.43894 0.50619 0.028892
## 10 0.02300885      9   0.41416 0.47611 0.028081
## 11 0.01946903     10   0.39115 0.46903 0.027886
## 12 0.01858407     11   0.37168 0.46195 0.027688
## 13 0.01592920     13   0.33451 0.43717 0.026984
## 14 0.01415929     14   0.31858 0.40708 0.026095
## 15 0.01061947     16   0.29027 0.37699 0.025166
## 16 0.00663717     17   0.27965 0.34867 0.024251
## 17 0.00530973     21   0.25310 0.34336 0.024074
## 18 0.00088496     23   0.24248 0.36814 0.024884
## 19 0.00010000     25   0.24071 0.35929 0.024599
```

```r
plot(mytree,margin = 0.01)
```



```r
plotcp(mytree)
```



Our first tree has a long tail, almost all branches in the same direction. The long tail means there exist some key word to classify the spam, such as, “Call”, “me”.

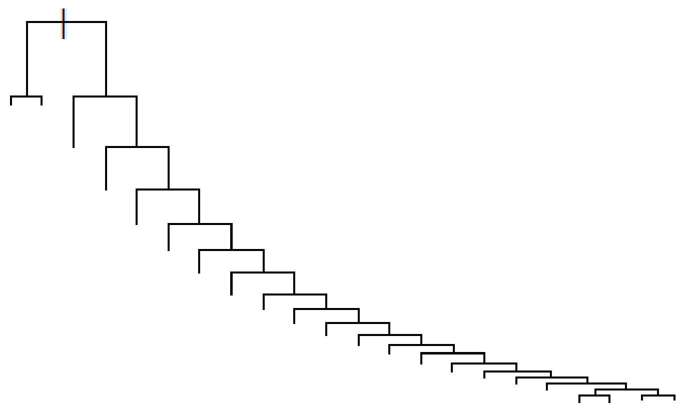
According to the cp plot, we have the smallest cp in no.22 split. The following splits is not useful. We need do a little pruning. ### The confusion matrix of first tree.

```
predictions_1<-predict(mytree,word.test,type = "class")
cm_1 = table(word.test$is_spam,predictions_1)
cm_1

##        predictions_1
##         TRUE  FALSE
##   TRUE   145    37
##   FALSE   20  1184

accuracy_F_1=cm_1[4]/(cm_1[3]+cm_1[4])
accuracy_T_1=cm_1[1]/(cm_1[1]+cm_1[2])
accuracy_T_1

## [1] 0.8787879

accuracy_F_1

## [1] 0.969697
```

The FALSE accuracy (specificity) is about 87%. The TRUE accuracy (specificity) is about 96%. ### Pruning

We find the smallest xerror split to do pruning.

```
mytree.cp<-mytree$cptable[which.min(mytree$cptable[,"xerror"]),"CP"]
mytree.cp

## [1] 0.005309735

prune.tree<-prune(mytree,cp=mytree.cp)
plot(prune.tree,margin = 0.01)
```



```
#text(prune.tree,all = T,use.n = T)
```

Because the tree is not complex, the shape of the tree doesn't change much after pruning.

Then let's use test set to get a confusion matrix to see how accuracy the tree is.

The confusion matrix of the prune tree.

```
pred_prune<-predict(prune.tree,word.test,type = "class")
cm_prune = table(word.test$is_spam,pred_prune)
cm_prune

##        pred_prune
##         TRUE  FALSE
##   TRUE   146    36
##   FALSE   21  1183

accuracy_F_prune=cm_prune[4]/(cm_prune[3]+cm_prune[4])
accuracy_T_prune=cm_prune[1]/(cm_prune[1]+cm_prune[2])
accuracy_T_prune

## [1] 0.8742515

accuracy_F_prune

## [1] 0.9704676
```

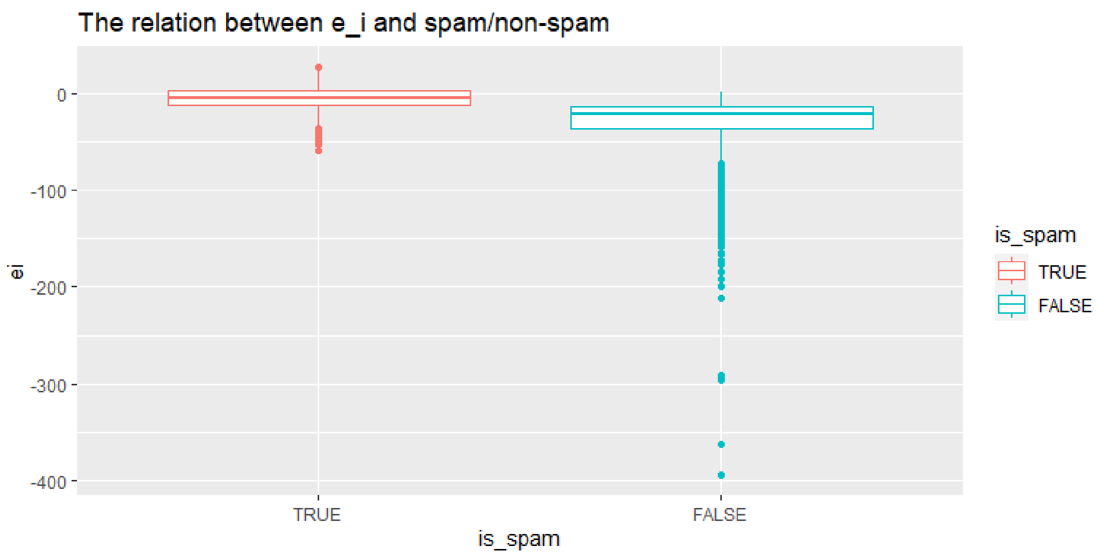
The FALSE accuracy (specificity) is about 88%, better than the first tree. The TRUE accuracy (specificity) is about 97%, better than the first tree. So the pruning does improve the first tree, but not much.

2. A 'Naïve Bayes' classifier

```
y_i <- word.df %>% filter(is_spam == "TRUE") %>% select(-is_spam) %>% c
olSums()
n_i <- word.df %>% filter(is_spam == "FALSE") %>% select(-is_spam) %>%c
olSums()
e_i  = log(y_i+1) - log(n_i+1)
nb <- t(t(wordmatrix) * e_i) %>% rowSums()
summary(nb)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -394.13  -34.16  -19.68  -25.55  -11.05   26.93
```

The greater the ei of one text, the more words appear in spam, and the smaller the ei, the more words appear in non-spam.

```
word.df$ei = nb
qplot(is_spam, ei,data = word.df, geom= "boxplot", col = is_spam) +
  ggtitle("The relation between e_i and spam/non-spam")
```



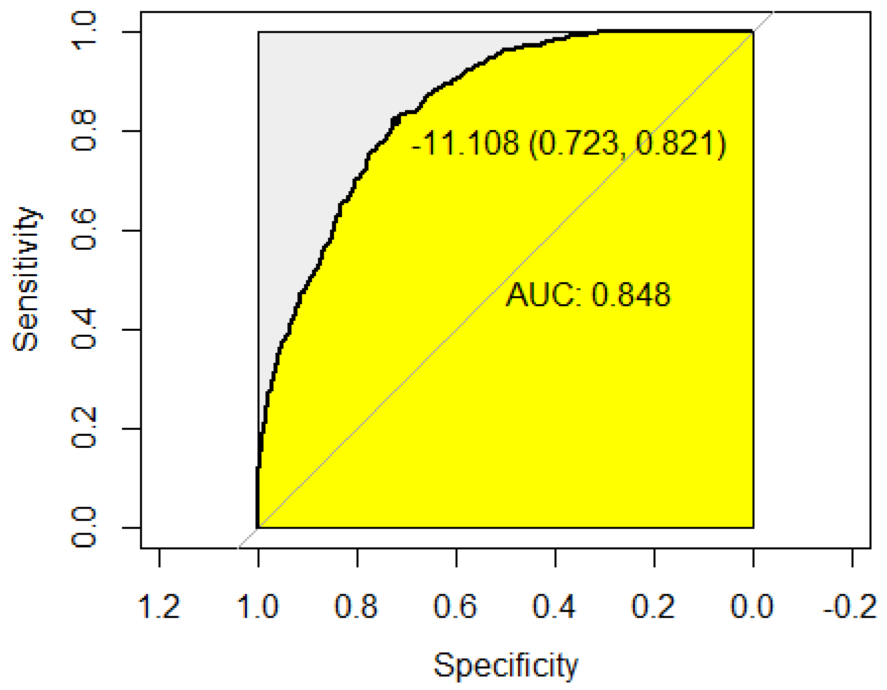
It seems spam has higher e_i than non-spam.

Construct a naïve Bayes classifier and choose the threshold so the proportion of spam predicted is the same as the proportion observed.

First we use ROC curve to get a initial threshold value.

```
library(pROC)

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

```
##
## Attaching package: 'pROC'

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

roc<-roc(word.df$is_spam,word.df$ei )

## Setting levels: control = TRUE, case = FALSE

## Setting direction: controls > cases

plot(roc,print.auc=T,auc.polygon=T,max.auc.polygon=T,auc.polygon.col="y
ellow",print.thres=T)
```

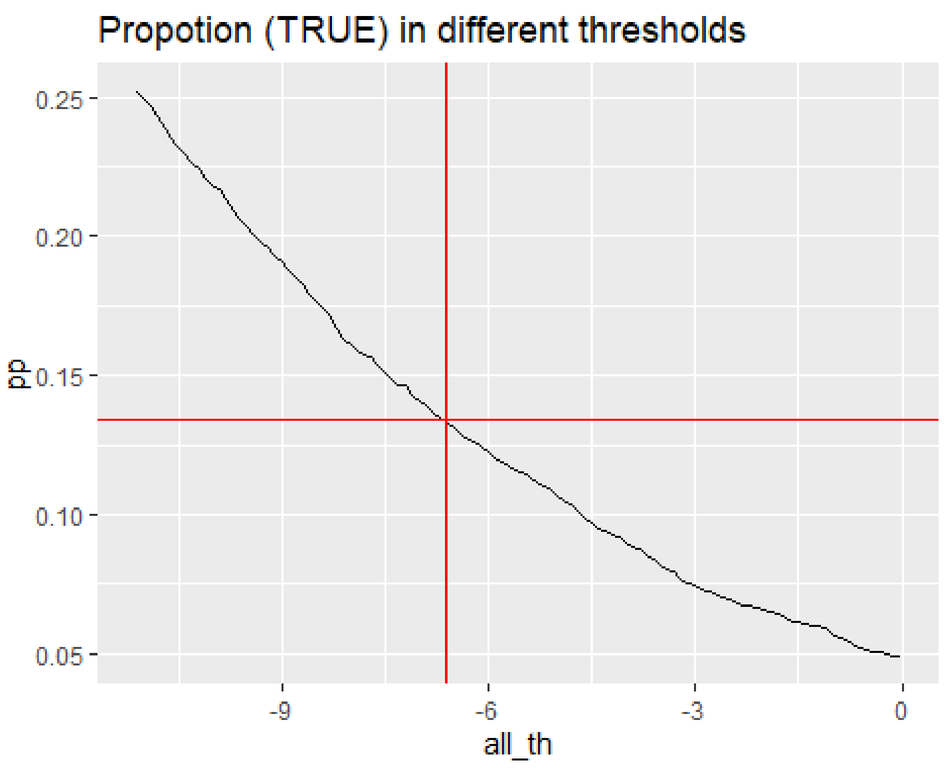


```
th =  -11.108
```

The initial threshold = -11.108. The threshold need to smaller than 0. Use this initial threshold range to find the threshold so the proportion of spam predicted is the same as the proportion observed.

```
a = summary(word.df$is_spam)
a

##  TRUE FALSE
##   747  4827
```

```
p_actual = a[1]/(a[2]+a[1])
p_actual

##      TRUE
## 0.1340151

pred=rep(0,nrow(word.df))
all_th = seq(th,0, 0.1)
pp = rep(0,length(all_th))
for (t in all_th) {
  for (k in c(1:nrow(word.df))) {
  if (word.df$ei[k] >= t) {
    pred[k] = "TRUE"
  }else{
    pred[k] = "FALSE"
  }

  }
  b = table(pred)
  p_pred = b[2]/(b[2]+b[1])
  pp[which(all_th==t)] = p_pred
}
th.df = cbind(data.frame(all_th),data.frame(pp))
head(th.df)

##    all_th        pp
## 1 -11.108 0.2518837
## 2 -11.008 0.2495515
## 3 -10.908 0.2466810
## 4 -10.808 0.2434517
## 5 -10.708 0.2389666
## 6 -10.608 0.2355579

pplot = ggplot() +geom_line(aes(x=all_th,y=pp))+ ggtitle("Propotion (TR
UE) in different thresholds")

pplot + geom_hline(aes(yintercept=p_actual), col = "red") + geom_vline
(aes(xintercept= all_th[which.min(abs(pp- p_actual))]), col = "red")
```



```
my_threshold =  all_th[which.min(abs(pp- p_actual))]
my_threshold

## [1] -6.608
```

The required threshold = -6.608, that means if e_i of a text greater than -6.608, the text is a spam.

confusion matrix

```
cm_e = table(word.df$is_spam,factor(word.df$ei >= my_threshold, levels
= c(TRUE, FALSE)))
cm_e

##
##          TRUE  FALSE
##   TRUE    414    333
##   FALSE   329   4498

accuracy_F_e=cm_e[4]/(cm_e[3]+cm_e[4])
accuracy_T_e=cm_e[1]/(cm_e[1]+cm_e[2])
accuracy_T_e

## [1] 0.5572005

accuracy_F_e

## [1] 0.9310702
```

The FALSE accuracy (specificity) is about 93.1%, good. The TRUE accuracy (specificity) is about 5.5%, bad. Though we have found the threshold that make the proportion of spam predicted is the same as the proportion observed, the accuracy of this cutoff is not good.

3.Why is spam/non-spam accuracy likely to be higher with this dataset than in real life? What can you say about the generalisability of the classifier to particular populations of text users?

From the UCI website: A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University.

A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received.

A list of 450 SMS ham messages collected from Caroline Tag's PhD Finally, we have incorporated the SMS Spam Corpus v.0.1 Big. It has 1,002 SMS ham messages and 322 spam messages

According to the website, most data is from Singapore, it is mainly written in Singaporean English. Only a small part of data set is from UK in native English.

If we use English words to construct our classifier, it makes sense that the spam/non-spam accuracy likely to be higher with this data set than in real life.

Because in real life, not everyone uses English like Singaporean do. The classifier is good for Singaporean but not good for other English users. So I don't think the generalisability of the classifier is good.