Project 5 - Spam Forecasting (NLP)

Hongyu Ji, Amon Tokoro, Hengyang Lin

12/5/2018

Motivation: Many of us are already aware of spam detection which plays a significant role in the world. All of email services have a functionality of it to protect users from potential fear of frauds. With this set, given the dataset with label (spam or ham) and texts on email, we would like to explore this study field, find out the contextural tendency and build a model capable of calssifying the email.

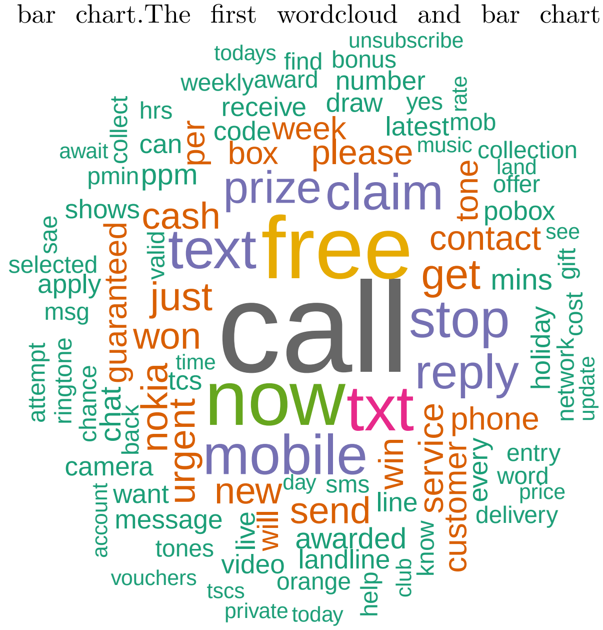
Step 0 – Data Preprocessing

Loading library

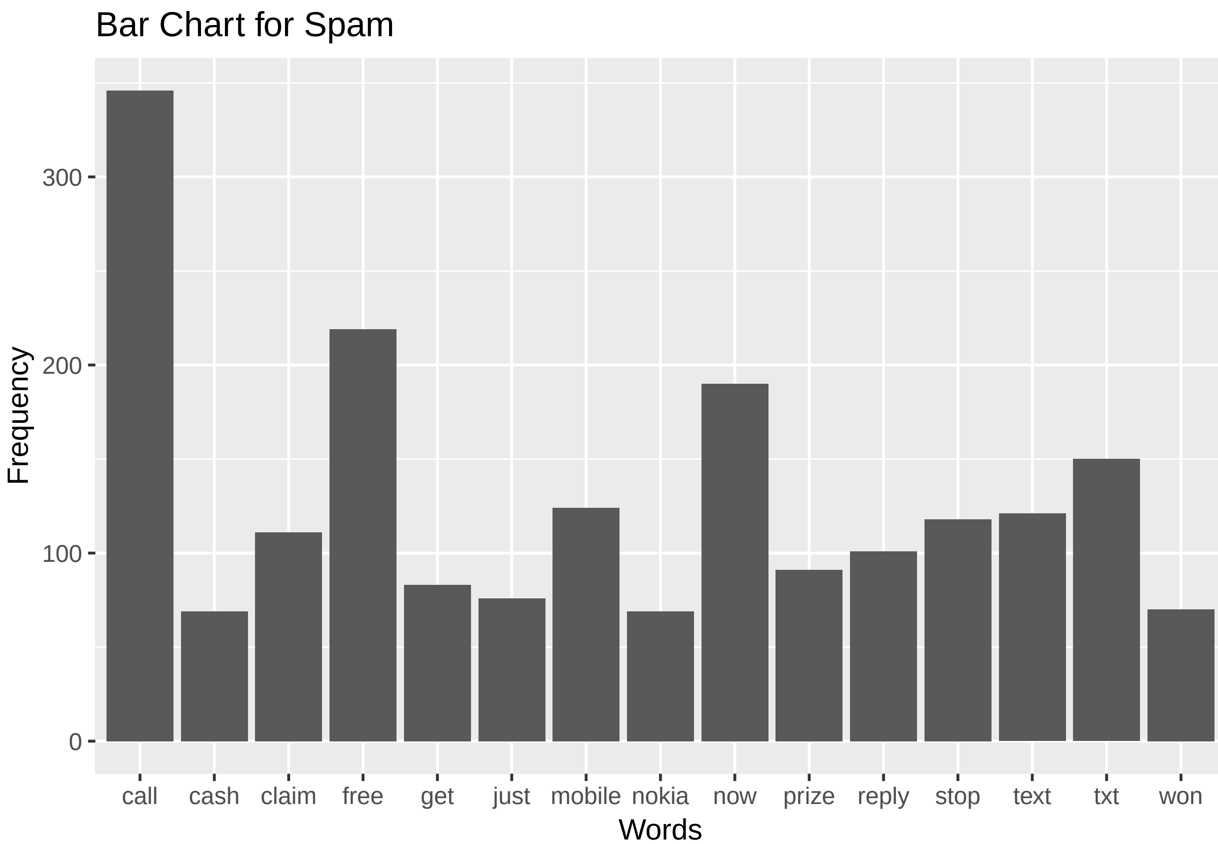
We first load all of necessary packages, and read the dataset. After that, the dataset is split into two data frame based on the labeled category.

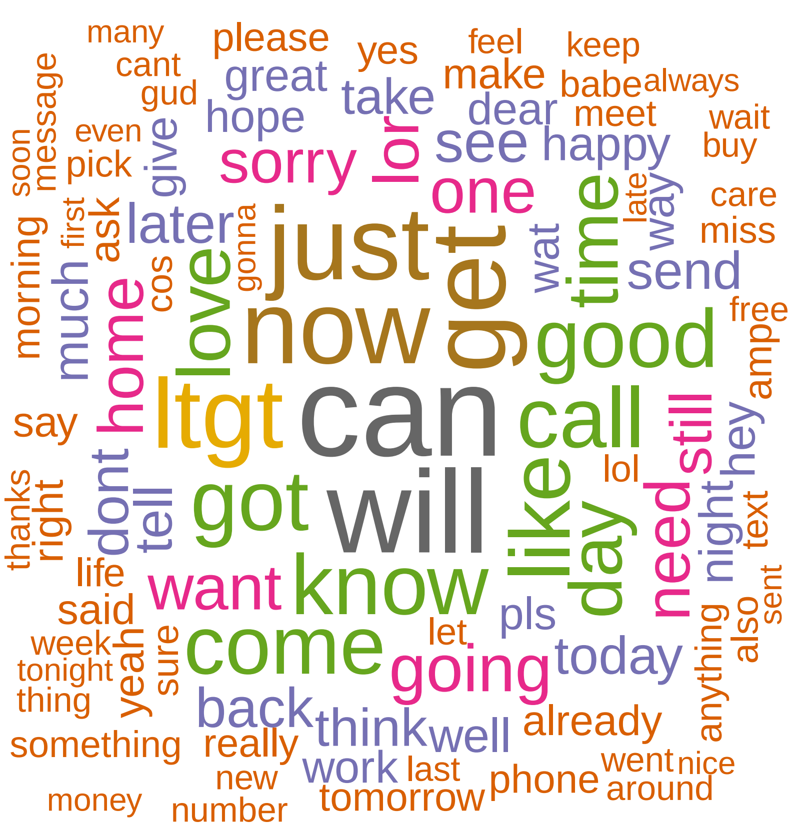
Word Cloud

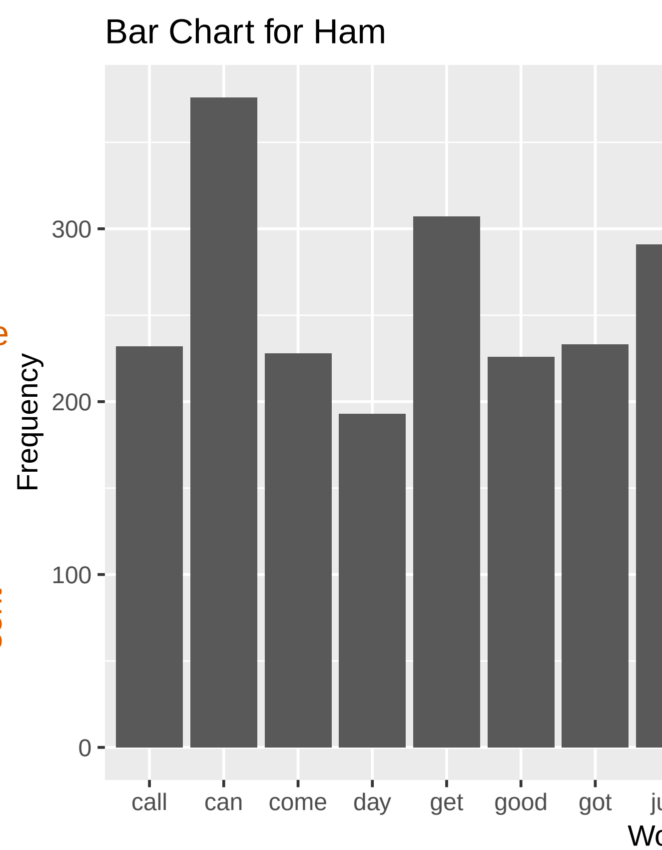
We are now intersted in the top 100 words which the most frequently appear in the messages and quantify the frequency of those words by a bar chart.The first wordcloud and bar chart



are for Spam and the second ones are for ham.

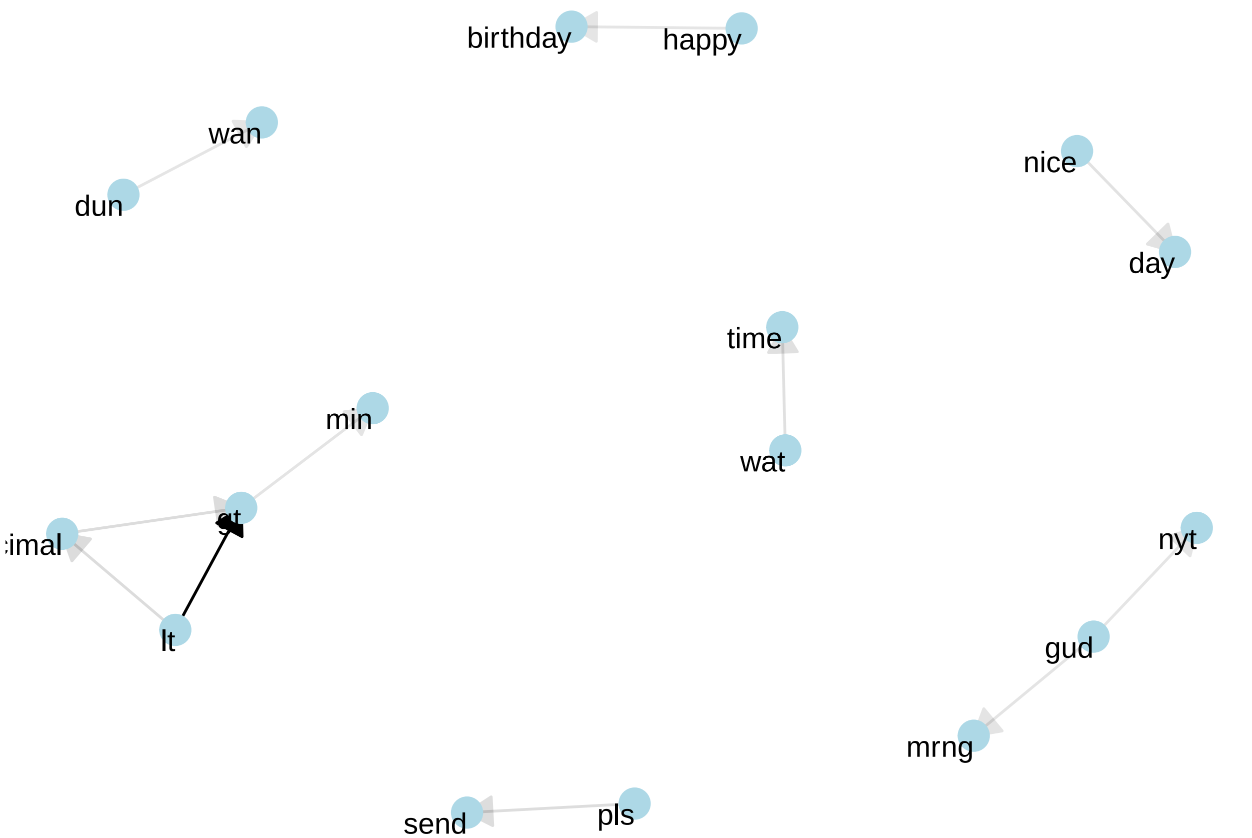

Bar Chart for Spam
Frequency
0
100
200
300
call
cash
claim
free
get
just
mobile
nokia
now
prize
reply
stop
text
txt
won
Words


many
please
yes
feel
keep
cant
great
make
babe
always
gud
take
dear
meet
wait
even
hope
buy
message
soon
pick
sorry
lor
see
happy
give
one
care
first
ask
later
late
way
miss
morning
gonna
just
wat
time
send
much
cos
love
get
free
home
now
good
amp
say
ltgt
can
call
still
hey
lol
thanks
right
dont
tell
got
will
like
night
text
life
day
need
want
know
said
pls
also
sent
week
yeah
let
anything
tonight
sure
come
going
today
thing
back
think
well
already
something
really
work
last
went
nice
new
phone
around
money
number
tomorrow

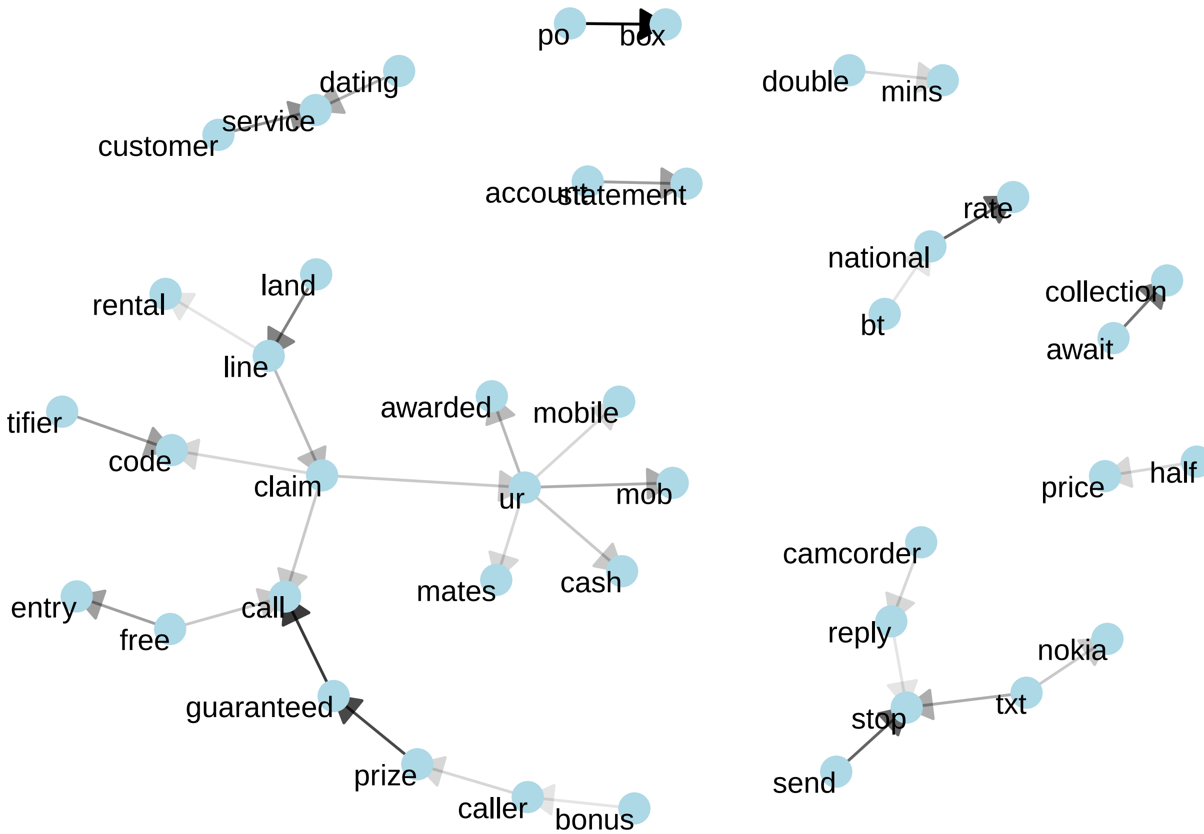

Bar Chart for Ham
Frequency
0
100
200
300
call
can
come
day
get
good
got

Bigram

```r
#Visualize paired words
kjv_bigrams(ham_df)
```




```r
kjv_bigrams(spam_df)
```



In this two chunks, we split the dataset into a training set and test set for building a model in the rest of the process.

```
spamdt <- spamdt[sample(nrow(spamdt)),]
spamdt$Message <- as.character(spamdt$Message)
msg.corpus<-corpus(spamdt$Message)
docvars(msg.corpus) <- spamdt$Category
```

```
## Document-feature matrix of: 6 documents, 1,932 features (99.1% sparse).
```

Naive Bayes

For the model creation, we first choose Naive Bayes which is considered as a baseline method for text categorization.

```
##
## Call:
## textmodel_nb.dfm(x = msg.dfm.train, y = spam.train[, 1])
##
## Distribution: multinomial; prior: uniform; smoothing value: 1; 4458 training documents; 1932 fitted
```

```
##           actual
## predicted ham spam
##      ham  945   10
##      spam  19  141
```

```
## [1] 0.9337748
```

```
## [1] 0.88125
```

```
## [1] 0.973991
```

Predictive Model

Since we have a list of reviews and the individual rating from such reviews, we can construct a predictive model to determine what are the words that are determinant in prediction of the rating of a review.

Data Preprocessing

We create a corpus and convert it into a document term matrix.

Bag of Word Matrix

Decision Tree

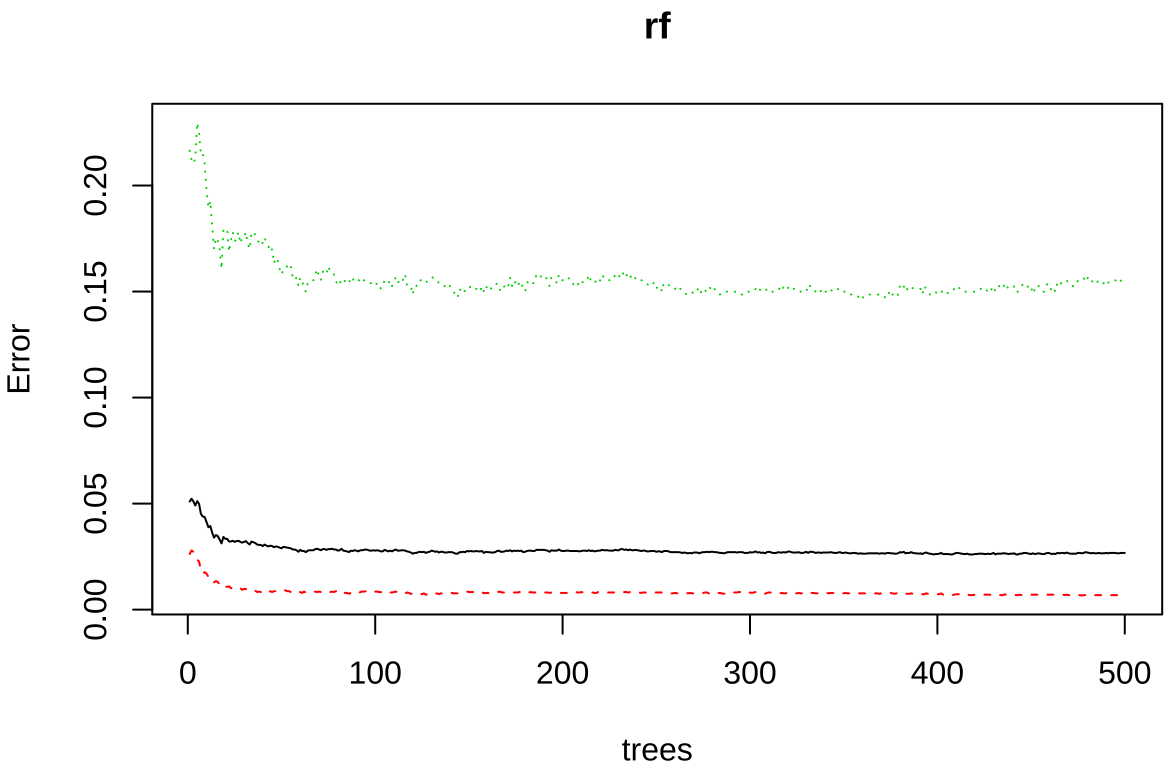
```
##          actual
## predicted  ham spam
##      ham  1395   71
##      spam   26  180
```

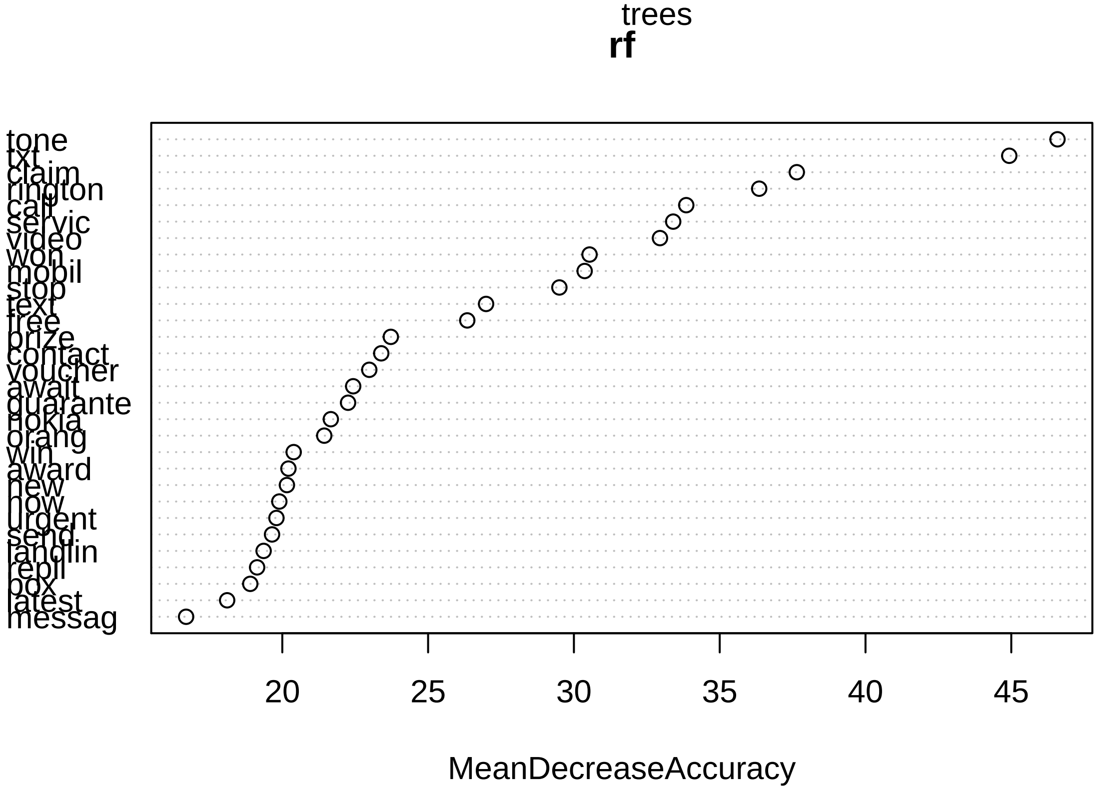
```
## [1] 0.7171315
```

```
## [1] 0.8737864
```

```
## [1] 0.9419856
```

Random Forest





%IncMse means by removing this variable, it will increase MSE by %IncMSE. RF cannot result the pos and neg correlation between predictors and response, but it tends to have higher accuracy.

```
##           actual
## predicted  ham spam
```

```
##      ham  4792  116
##      spam   33  631
```

```
## [1] 0.8447122
```

```
## [1] 0.9503012
```

```
## [1] 0.9732592
```

SVM

Linear SVM

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##   0.01   10
##
## - best performance: 0.03846154
```

From the initial svm, we saw that mse tend to be smaller as gamma decreases and as cost increases. Initial range gamma = 0:1, cost = 0:10

```
svm.best <- svm.m$best.model
svm.pred <- predict(svm.best, testset3)
#mean((svm.pred - testset3$rate)^2)
(t4 <- table(predicted=svm.pred,actual=testset3$rate))
```

```
##          actual
## predicted ham spam
##      ham  281   16
##      spam   3   34
```

```
(recall4 <- t4[4]/(t4[4] + t4[3]))
```

```
## [1] 0.68
```

```
(precision4 <- t4[4]/(t4[4] + t4[2]))
```

```
## [1] 0.9189189
```

```
mean(svm.pred == testset3$rate)
```

```
## [1] 0.9431138
```

XGBoost

```
# dtrain <- xgb.DMatrix(data = as.matrix(trainset2[,-which(names(trainset2) == 'rate')]), label = as.nu
# Dtest <- xgb.DMatrix(data = as.matrix(testset2[,-which(names(testset2) == 'rate')]), label = testset2
```

tuning

```
# params <- list(booster = "gbtree", objective = "reg:linear", eta=0.1, gamma=0, max_depth=6, min_child_
#
# xgbcv <- xgb.cv( params = params, data = dtrain, nrounds = 1000, nfold = 5, showsd = T, stratified =
```

fitting model

```
# xgb1 <- xgb.train (data = dtrain, max_depth=2,eta=1,nthread=2, nrounds = 100, watchlist = list(val=Dt
#
# xgb1 <- xgb.train(data = dtrain, nrounds = 100, watchlist = list(val=Dtest,train=dtrain), print_every_
```

prediction

```
# xgbpred <- predict_proba(xgb1,Dtest)
# #mean((xgbpred - testset2$rate)^2)
#
# mean(xgbpred == testset2$rate)
```

important variable

```
# mat <- xgb.importance (feature_names = colnames(trainset2),model = xgb1)
# xgb.plot.importance (importance_matrix = mat[1:20])
```

```
#for(depth in c(1,2,5)){
#  for(subsamp in c(0.5, 0.7, 1)){
# xgb.m <- xgboost(data = dtrain,
#                 booster = "gbtree",
#                             nrounds = 217,
#                             verbose = F,
#                             objective = "reg:linear", max.depth = 1, "eta"
#                             =2/217,subsample = 0.5)
# xg.pred <- predict(xgb.m, Dtest)
# rmse <- mean((xg.pred - Dtest$rate)^2)
# print( c(depth,subsamp, rmse))
```

```
# xg2 <- xgboost(data = dtrain ,
#         booster = "gbtree",
#         objective = "reg:linear",
#         max.depth = 2,
#         eta = 0.05,
#         nthread = 2,
#         nround = 10000,
#         min_child_weight = 1,
#         subsample = 0.5,
#         colsample_bytree = 1,
#         num_parallel_tree = 3)
```

```
# xg.pred <- predict(xg2, Dtest)
# mean(((xg.pred) - testset2[,which(names(testset2)=='rate')])^2)
```

```
# Dtest <- xgb.DMatrix(data = as.matrix(testset2[,-which(names(testset2) == 'rate')]), label = testset2
# xg.pred <- predict(xg6, Dtest)
# mean(((xg.pred) - testset2[,which(names(testset2)=='rate')])^2)
```

```
# result.xgb <- data.frame(depth = rep(NA,25), para_tree = rep(NA, 25), mse = rep(NA,25))
# k <- 1
# for(depth in c(1:5)){
#   for(paraTree in c(1:5)){
#     xgLoop <- xgboost(data = dtrain ,
#         booster = "gbtree",
#         objective = "reg:linear",
#         max.depth = 1,
#         eta = 0.05,
#         nthread = 2,
#         nround = 1200,
#         min_child_weight = 1,
#         subsample = 0.5,
#         colsample_bytree = 1,
#         num_parallel_tree = 1,
#         verbose = 0)
#     xgLoop.pred <- predict(xgLoop, Dtest)
#     xgb.mse <- mean(((xgLoop.pred) - testset2[,which(names(testset2)=='rate')])^2)
#     cat('depth:', depth, ', para_tree:', paraTree, ', MSE:', xgb.mse, '\n')
#     result.xgb$depth[k] = depth
#     result.xgb$para_tree[k] = paraTree
#     result.xgb$mse[k] = xgb.mse
#     k <- k+1
#   }
# }
```

```
# result.xgb.linear <- data.frame(depth = rep(NA,30), para_tree = rep(NA, 30), mse = rep(NA,30))
# k <- 1
# for(l in seq(0,1,0.2)){
#   for(a in seq(0,2,0.5)){
#     xgLoop <- xgboost(data = dtrain ,
#         nrounds = 1200,
#         booster = "gblinear",
#         objective = "reg:linear",
#         lambda = l,
#         alpha = a,
#         verbose = 0)
#     xgLoop.pred <- predict(xgLoop, Dtest)
#     xgb.mse <- mean(((xgLoop.pred) - testset2[,which(names(testset2)=='rate')])^2)
#     cat('lambda:', l, ', alpha:', a, ', MSE:', xgb.mse, '\n')
#     result.xgb.linear$lambda[k] = l
#     result.xgb.linear$alpha[k] = a
#     result.xgb.linear$mse[k] = xgb.mse
#     k <- k+1
#   }
# }
```

```r
performance_matrix <- as.data.frame(matrix(0, ncol = 4, nrow = 3))
names(performance_matrix) <- c("Naive Bayes", "Decision Tree", "Random Forest", "SVM")
rownames(performance_matrix) <- c("accuracy", "recall","precision")
performance_matrix[1,1] <- mean(pred==spam.test[,1])
performance_matrix[2,1] <- recall1
performance_matrix[3,1] <- precision1

performance_matrix[1,2] <- mean(testset2$rate==tree.pred)
performance_matrix[2,2] <- recall2
performance_matrix[3,2] <- precision2

performance_matrix[1,3] <- mean(dtm.df$rate == rf$predicted)
performance_matrix[2,3] <- recall3
performance_matrix[3,3] <- precision3

performance_matrix[1,4] <- mean(svm.pred == testset3$rate)
performance_matrix[2,4] <- recall4
performance_matrix[3,4] <- precision4

performance_matrix
```

```
##           Naive Bayes Decision Tree Random Forest       SVM
## accuracy    0.9739910     0.9419856     0.9732592 0.9431138
## recall      0.9337748     0.7171315     0.8447122 0.6800000
## precision   0.8812500     0.8737864     0.9503012 0.9189189
```

```r
library(knitr)
kable(performance_matrix)
```

	Naive Bayes	Decision Tree	Random Forest	SVM
accuracy	0.9739910	0.9419856	0.9732592	0.9431138
recall	0.9337748	0.7171315	0.8447122	0.6800000
precision	0.8812500	0.8737864	0.9503012	0.9189189