ERG2050 Midterm Project Reprot

Jiang Jingxin

117020119

1. Data Preprocessing

1.1. Import data

1.1.1 train data

```
spam_train = read.csv("spam_train.csv")
```

1.1.2 Test data

```
spam_test = read.csv("spam_test.csv")
```

1.2. Data transformation

1.2.1. remove index column

```
rownames(spam_train) = spam_train[,1]
spam_train = spam_train[,-1]
```

```
rownames(spam_test) = spam_test[,1]
spam_test = spam_test[,-1]
```

1.2.2. convert type

```
spam_train$capitalLong = as.numeric(spam_train$capitalLong)
spam_train$capitalTotal = as.numeric(spam_train$capitalTotal)
```

```
spam_test$capitalLong = as.numeric(spam_test$capitalLong)
spam_test$capitalTotal = as.numeric(spam_test$capitalTotal)
```

1.2.3. set class label

```
spam_train$type = as.factor(spam_train$type)
levels(spam_train$type) = c("ham","spam")
```

1.3. Data cleaning

1.3.1. missing values

No missing value detected:

```
print(paste("train:",anyNA(spam_train),
            "test:",anyNA(spam_test)))
```

```
## [1] "train: FALSE test: FALSE"
```

1.3.2. duplicated values

No redundant features detected:

```
print(paste("train:",any(duplicated(t(as.matrix(spam_train)))),
            "test:",any(duplicated(t(as.matrix(spam_test))))))
```

```
## [1] "train: FALSE test: FALSE"
```

Duplicate records occured:

```
print(paste("train:",any(duplicated(spam_train)),
            "test:",any(duplicated(spam_test))))
```

```
## [1] "train: TRUE test: TRUE"
```

It's fine to leave them alone since more frequent data should be more weighted.

1.4. Feature selection

1.4.1. zero-variance filtering

No constant feature detected:

```
print(min(diag(var(spam_train[,-58]))))
```

```
## [1] 0.005523331
```

Although there are various advanced feature selection method available, after several trials it turns out that the improvemnt in training perfromance is very limited and some even decreases.

Since we are using tree models instead of logistic regression models, it should be fine to include all the features with proper pruning which in some sense will select important features implicitly.

2. Model Fitting

2.1. Theoretic setup - Cost Sensitive Classification

$$\begin{aligned} Score &= N_{00} + N_{11} - 30N_{10} - 2N_{01} \\ &= -c_{00}N_{00} - c_{11}N_{11} + c_{10}N_{10} + c_{01}N_{01} \end{aligned}$$

$$Cost\ Matrix = \begin{array}{c|cc} & \multicolumn{2}{c}{\text{predict}} \\ \text{observed} & ham & spam \\ \hline ham & c_{00} = -1 & c_{01} = 30 \\ spam & c_{10} = 1 & c_{11} = -1 \end{array}$$

$$Objective\ Function\ \mathcal{L}(x,i) = \sum_j P(j|x)c_{ij}$$

$$\min_i \mathcal{L}(x,i) \Leftrightarrow \min \sum_j P(j|x)(c_{ij} - \alpha),\ \text{where } \alpha \in \mathbb{R}$$

$$\text{So, the cost matrix is equivalent to :} \begin{array}{c|cc} \text{observed} & ham & spam \\ \hline ham & 0 & 31 \\ spam & 2 & 0 \end{array}$$

$$Theorecial\ Threhold = \frac{c_{10} - c_{00}}{c_{10} - c_{00} + c_{01} - c_{11}}$$

2.1.1. cost matrix

```
costs = matrix(c(0,2,31,0),2,dimnames=list(c("ham", "spam"), c("ham", "spam")))
names(dimnames(costs)) <- c("observed", "predicted")
print(costs)
```

```
##         predicted
## observed ham spam
##     ham    0   31
##     spam   2    0
```

2.1.2. therotical threhold

```
th = costs[2,1]/(costs[2,1]+costs[1,2])
w = (1 - th) / th
```

2.2. Model selection

2.2.0. import library

```
library(mlr)
library(randomForestSRC)
```

2.2.1. alternatives

```
#Package `mlr`

# Learners that accept observation weights
##                        class         package  score
## 1            classif.binomial           stats  *
## 2                 classif.C50             C50  **
## 3             classif.cforest           party  o
## 4               classif.ctree           party  o
## 5            classif.cvglmnet          glmnet  *
## 6               classif.earth     earth,stats  o
## 7              classif.evtree          evtree  o
## 8          classif.extraTrees      extraTrees  *
## 9          classif.fdausc.knn         fda.usc  x
## 10           classif.gamboost          mboost  o
## 11                classif.gbm             gbm  ***
## 12           classif.glmboost          mboost  ***
## 13             classif.glmnet          glmnet  **
## 14 classif.h2o.deeplearning             h2o  o
## 15            classif.h2o.glm             h2o  o
## 16             classif.logreg           stats  *
## 17           classif.multinom            nnet  *
## 18               classif.nnet            nnet  x
## 19                classif.plr         stepPlr  o
## 20             classif.probit           stats  **
## 21  classif.randomForestSRC randomForestSRC  ****
## 22             classif.ranger          ranger  ****
## 23              classif.rpart           rpart  ***
## 24            classif.xgboost         xgboost  o

# Learners that can deal with class weights
##                            class       package  score
## 1                   classif.ksvm       kernlab  ***
## 2       classif.LiblineaRL1L2SVC     LiblineaR  -
## 3      classif.LiblineaRL1LogReg     LiblineaR  -
## 4       classif.LiblineaRL2L1SVC     LiblineaR  -
## 5      classif.LiblineaRL2LogReg     LiblineaR  -
## 6         classif.LiblineaRL2SVC     LiblineaR  -
## 7 classif.LiblineaRMultiClassSVC     LiblineaR  -
## 8           classif.randomForest randomForest  ****
## 9                    classif.svm         e1071  ***
```

After trying plenty of classifiers in package `mlr` listed above with their scores, it seems that Random Forest classifiers achieves the best performance in terms of the cost sensitive score measure.

They are: ranger(a fast implementaion of RF), randomForest and randomForestSRC(survival, regression & classification). Among the three, ranger is not so good as the other two since it give up some performance in order to speed up. By comparing randomForest and randomForestSRC, although that randomForest generate slightly better results, randomForestSRC predicts fewer spam class and the proportion of predicted spam classes in the test set using randomForestSRC is closer to the proportion of spam classes in the training set. So, it is safer to use randomForestSRC.

2.2.2. randomForestSRC

```
spam.task = makeClassifTask(data=spam_train,target="type")
lrn = makeLearner("classif.randomForestSRC")
lrn = makeWeightedClassesWrapper(lrn, wcw.weight = w)
mod = train(lrn, spam.task)
```

```
pred = predict(mod,newdata=spam_train)
confusion = table(pred$data$truth,pred$data$response)
names(dimnames(confusion)) <- c("observed", "predicted")
print(confusion)
```

```
##         predicted
## observed  ham spam
##     ham  1411    0
##     spam  163  727
```

```
(score = confusion[1,1]+confusion[2,2]-30*confusion[1,2]-confusion[2,1])
```

```
## [1] 1975
```

2.2.3. select random seed

As indicated by the name, the model has some randomness in the classification results. By setting different random seeds, it seems that the model are quite stable with training score bouncing about 1965~1990.

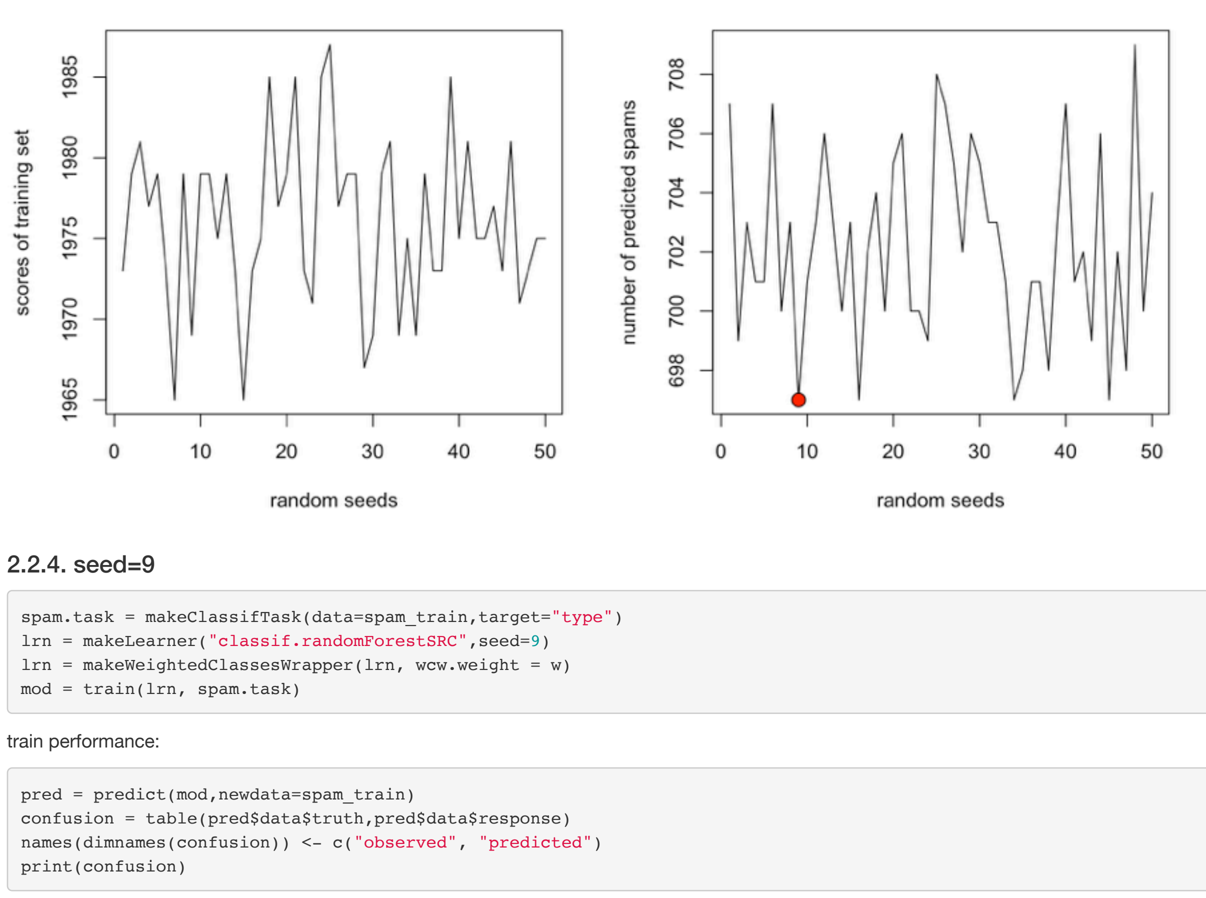
Counting the number of predicted spam classes under different seeds, it turns out that the number are also quite stable around 700.

Since there are little variation in the number of predicted spam class, the safer way is to select the random seed that generates the smallest number of predicted spams. By enumerating the random seeds in [1,50], the seed=9 is the one that results in the smallest number of predicted spams.

```
# spam.task = makeClassifTask(data=spam_train,target="type")
# scores = NULL
# numspams = NULL
# for (i in seq(1,50)) {
#   lrn = makeLearner("classif.randomForestSRC",seed=i)
#   lrn = makeWeightedClassesWrapper(lrn, wcw.weight = w)
#   mod = train(lrn, spam.task)
#   pred = predict(mod,newdata=spam_train)
#   confusion = table(pred$data$truth,pred$data$response)
#   scores = append(scores, confusion[1,1]+confusion[2,2]-30*confusion[1,2]-confusion[2,1])
#   pred = predict(mod,newdata=spam_test)
#   numspams = append(numspams,sum(pred$data$response=="spam"))
# }

# results of code above
scores = c(1973, 1979, 1981, 1977, 1979, 1973, 1965, 1979, 1969, 1979,
           1979, 1975, 1979, 1973, 1965, 1973, 1975, 1985, 1977, 1979,
           1985, 1973, 1971, 1985, 1987, 1977, 1979, 1979, 1967, 1969,
           1979, 1981, 1969, 1975, 1969, 1979, 1973, 1973, 1985, 1975,
           1981, 1975, 1975, 1977, 1973, 1981, 1971, 1973, 1975, 1975)
numspams = c(707, 699, 703, 701, 701, 707, 700, 703, 697, 701,
             703, 706, 703, 700, 703, 697, 702, 704, 700, 705,
             706, 700, 700, 699, 708, 707, 705, 702, 706, 705,
             703, 703, 701, 697, 698, 701, 701, 698, 703, 707,
             701, 702, 699, 706, 697, 702, 698, 709, 700, 704)
```

```
par(mfrow=c(1,2))
plot(1:50,scores,'l',xlab="random seeds",ylab="scores of training set")
plot(1:50,numspams,'l',xlab="random seeds",ylab="number of predicted spams")
points(9,697,bg="red",pch=21,cex=1.5)
```



2.2.4. seed=9

```
spam.task = makeClassifTask(data=spam_train,target="type")
lrn = makeLearner("classif.randomForestSRC",seed=9)
lrn = makeWeightedClassesWrapper(lrn, wcw.weight = w)
mod = train(lrn, spam.task)
```

train performance:

```
pred = predict(mod,newdata=spam_train)
confusion = table(pred$data$truth,pred$data$response)
names(dimnames(confusion)) <- c("observed", "predicted")
print(confusion)
```

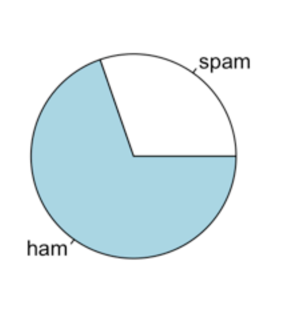
```
##         predicted
## observed  ham spam
##     ham  1411    0
##     spam  166  724
```

```
(score = confusion[1,1]+confusion[2,2]-30*confusion[1,2]-confusion[2,1])
```

```
## [1] 1969
```

prediction distribution:

```
pred = predict(mod,newdata=spam_test)
pie(c(sum(pred$data$response == "spam"),sum(pred$data$response == "ham"))/2300,
    labels=c("spam","ham"))
```




```
prediction = data.frame("index"= 1:2300,"type"=as.numeric(pred$data$response)-1)
write.csv(prediction,"prediction.csv")
```

```
spam_test$type = as.numeric(pred$data$response)-1
write.csv(spam_test,"spam_test_pred.csv")
```