Individual Coursework Submission Form

Specialist Masters Programme

Surname: Haixiang	**First Name: Yan**
MSc in: Quantitative Finance	**Student ID number: 190054621**
Module Code: SMM636	
Module Title: Machine Learning	
Lecturer: Zhu, Rui	**Submission Date: 03 April 2020**

Declaration:

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the coursework instructions and any other relevant programme and module documentation. In submitting this work I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

We acknowledge that work submitted late without a granted extension will be subject to penalties, as outlined in the Programme Handbook. Penalties will be applied for a maximum of five days lateness, after which a mark of zero will be awarded.

Marker's Comments (if not being marked on-line):

Deduction for Late Submission:

Final Mark: %

SMM636 Machine Learning (PRD2 A 2019/20)
Individual Assignment

Yan Haixiang
March 2020

Table of Contents

1. Introduction

This ML assignment explores the robust applications of Support Vector Machine, Decision Tree, as well as Random Forest with the OJ data downloaded from ISLR package. Under each of the three sections, the model is trained with data representing 70% of the entire dataset, while the remaining 30% is reserved for model testing. For each model trained, a test error rate is computed as an indicator of the reliability of that particular model. Besides, as you will see below, with different assumptions of parameters and training methods (especially evident in SVM where three methods, namely liner, radial, and polynomial) applied, the final results differ. The results are plotted for better visualization.

2. Support Vector Machine (SVM) and ROC Plots

2.1 Test error rates with different methods and tuned parameter

2.1.1 SVM (kernel = linear)

For different cost values tuned, which in this case cost is set to 0.01, 0.10, 1.00, and 10.00 respectively, different error rate results. As shown below, when cost is set to 10.00, we have the lowest error rate (0.1680).

```
- Detailed performance results:
   cost     error dispersion
1  0.01 0.1746667 0.05162065
2  0.10 0.1733333 0.04532462
3  1.00 0.1733333 0.04988877
4 10.00 0.1680000 0.04836078
```

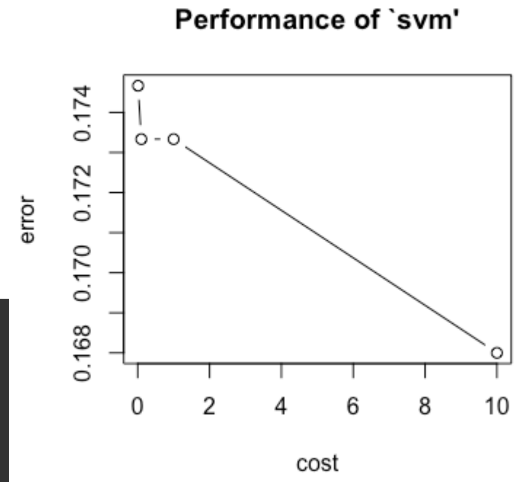


Figure 1, 2 – Test error rates under linear SVM model

2.1.2 SVM (kernel = radial)

Radial is mostly used kernel for SVM. As shown below, when cost is set to 1.00, we have the lowest error rate (0.1853).

```
- Detailed performance results:
   cost     error dispersion
1  0.01 0.3893333 0.05989291
2  0.10 0.2666667 0.06942444
3  1.00 0.1853333 0.04283762
4 10.00 0.2040000 0.05931294
```

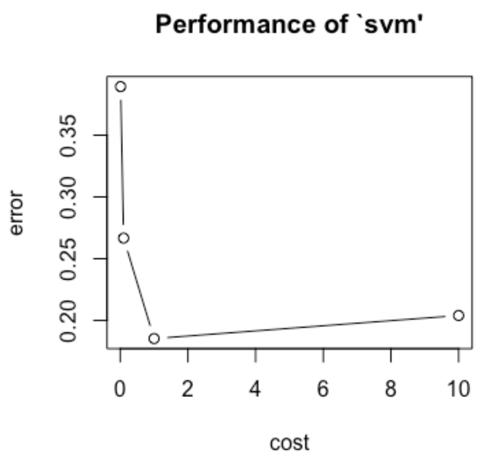


Figure 3, 4 – Test error rates under radial SVM model

2.1.3 SVM (kernel = polynomial, degree = 2)
Under this kernel, when cost is set to 10.00, we have the lowest error rate (0.1733).

```
- Detailed performance results:
   cost     error dispersion
1  0.01 0.3893333 0.03760746
2  0.10 0.3266667 0.02828427
3  1.00 0.2066667 0.05154406
4 10.00 0.1733333 0.04661373
```

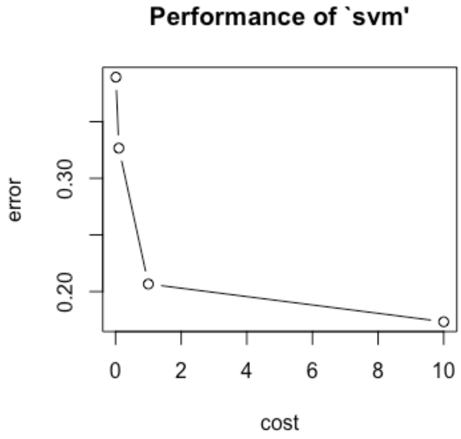


Figure 5, 6 – Test error rates under radial SVM model

In summary, although different kernel models have different optimal cost value for which its error rate is lowest, the general pattern observed is that when cost value increases (at least until 10.00), the error rate decreases.

Besides, it is worth noticing that for each time the model is run, the error rate results are likely to be different. One solution to get a reliable single value with great predictability capacity is to increase the number of times the models are run. Due to time limit of this assignment, this is not further elaborated.

2.2 ROC plots and interpretations

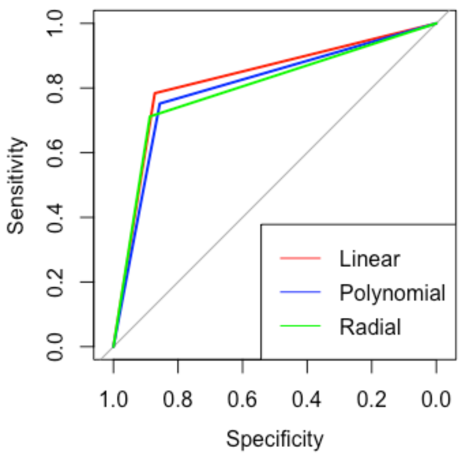


Figure 7 – ROC plotting for three models under SVM

Receiver operating characteristics (ROC) curve serves as a graphical representation of sensitivity and specificity with different classification thresholds. As shown above, the three methods have similar ROC plots and thus areas under curve (AUC), with the liner model slightly better than the other two. Lastly, for all three models, the performance is up to standard, as the AUC is significant.

Besides, non-linear models (such as radial and polynomial) are more likely to result in overfitting problems as they tend to interpret the existing data well but when comes to prediction capabilities, they are disadvantaged.

3. *Decision Tree, Random Forest, and ROC Plots*

3.1 Decision tree

3.1.1 Cross validation (10-fold, repeat 3 times)

Before apply cross-validation to increase prediction accuracy, the number of terminal nodes is 9. *However, w*hen applying a 10-fold, 3-time cross validation on the tree model, the optimal tree size is reduced to 4 *(Appendix A)*. The accuracy rate with different cp values are present below:

```
CART

750 samples
 17 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 675, 675, 675, 675, 675, 675, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.01598174  0.7964753  0.5741827
  0.03253425  0.7902233  0.5586241
  0.48287671  0.6986713  0.2906605

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01598174.
```

Figure 8 – Cross-validation Accuracy Rates

The optimal cp value is 0.01598174, with an accuracy rate of 0.7964753.

3.1.2 Tree pruning with terminal node of 5

Despise the results from cross-validation above, I further prune the number of terminal nodes to 5, here is the shape of the pruned tree and corresponding accuracy rates.

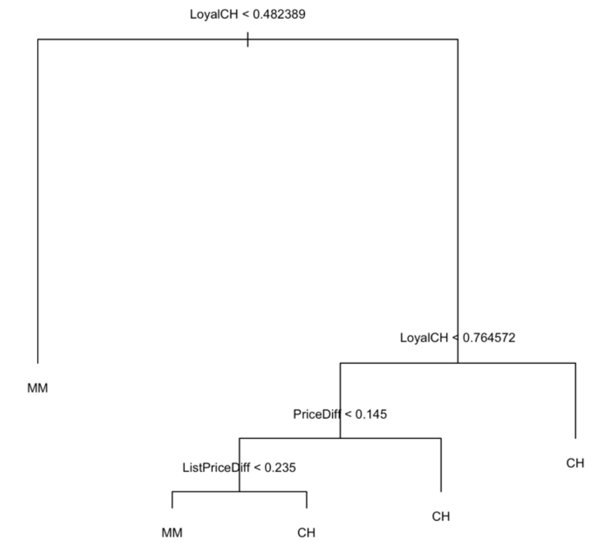


Figure 9 – Tree pruning with number of terminal nodes set to 5

The test error rate of the pruned tree is 0.184375, which is a slight improvement over the cross-validation method above. However, this error rate may not be the lowest for all the possible tree size we can choose. The optimal number of terminal nodes, though not the goal of this assignment, is further explored below.

3.1.3 Additional findings – optimal tree size

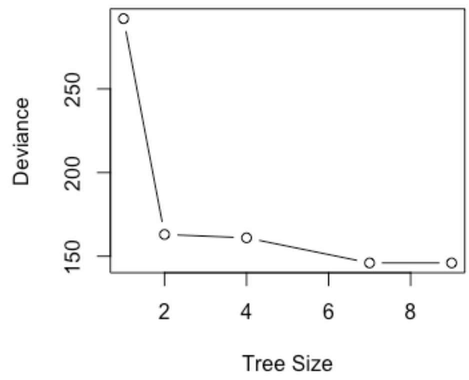


Figure 10 – Optimal Tree Size

The deviation data is set as the indicator to choose the optimal tree size, As shown The deviation value is selected as the parameter to choose the optimal tree size. As shown above, when tree size is 7 or 9, the deviation is at its lowest, which is around 150. Thus a better selection of pruned tree should have a terminal nodes of size 7 or 9, suppose all else are equal.

3.2 Random forest

3.2.1 Test error rate

Under this section, the parameter "mtry" is set to 1, 2, 3, 4, 5, 6, each with a different accuracy rate, which are illustrated below:

```
Random Forest

750 samples
 17 predictor
  2 classes: 'CH', 'MM'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 750, 750, 750, 750, 750, 750, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  1     0.7522585  0.4475725
  2     0.7943541  0.5588451
  3     0.8004569  0.5747206
  4     0.7999495  0.5754432
  5     0.7987759  0.5729798
  6     0.7970948  0.5702088

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 3.
```

Figure 11 – Random forest test error rate

When mtry = 3, the accuracy is the highest at 0.8004589. It is worth noticing that for each time the model is run, the results are likely to be different. For this particular run, so long the mtry value is within 2 to 6, the accuracy rates show similar patterns.

3.2.2 Variable importance

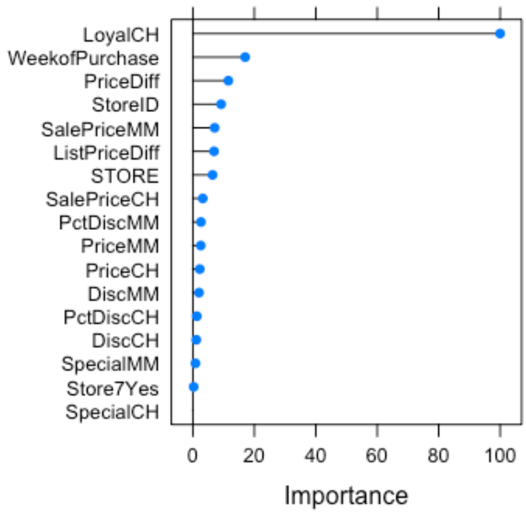


Figure 12 – Variable Importance

Based upon the result that when mtry = 3 the accuracy rate is the best, I plot the variable importance graph to present the variables that influence the prediction outcome the most. As shown in the above graph, the variable "Loyal CH" dominates the chart as the best predicting variable, followed by "Week of Purchase", "Price Difference", and "Store ID", which are of the second tier in terms of importance.

3.3 ROC plots and interpretations

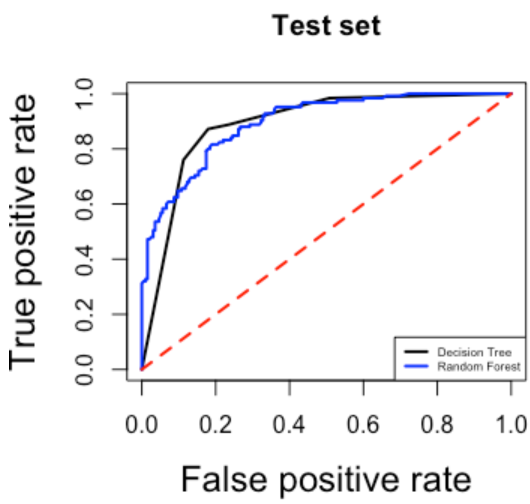


Figure 13 – ROC for decision tree and random forest

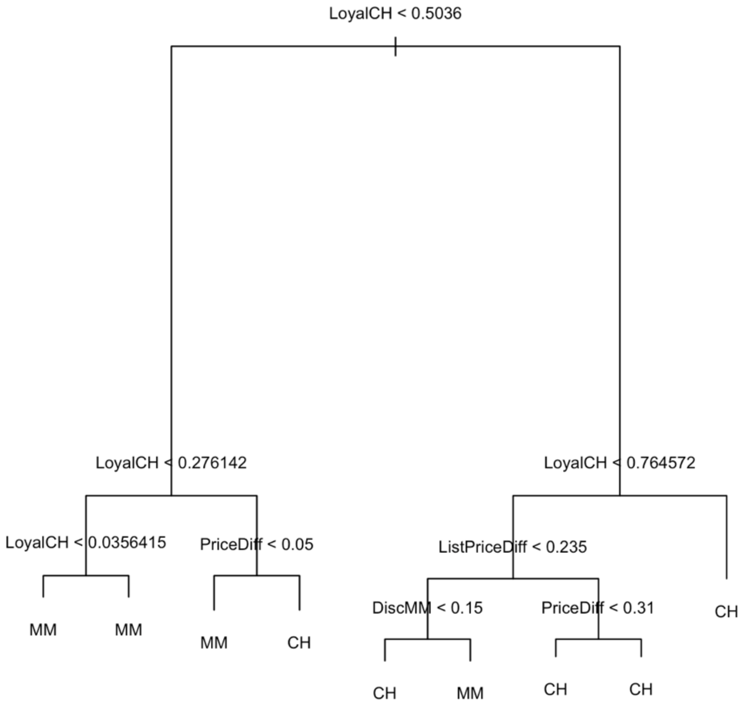
As shown above, both models perform well in this set of data, with significant AUC value, significantly above the benchmark line (red-dotted line), with the Decision Tree model has a AUC of 0.8896821 and Random Forest model has a value of 0.8948923.

4. Conclusion

All of the above models have a satisfying prediction capability with accuracy rate concentrating within 0.77 to 0.82. However, this results may be impacted by the particular set of data chosen. For further research, such limitations could be addressed.

5. *Appendix*

Appendix A – Decision Tree without pruning




```
Classification tree:
tree(formula = Purchase ~ ., data = train)
Variables actually used in tree construction:
[1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "DiscMM"
Number of terminal nodes:  9
Residual mean deviance:  0.7418 = 549.6 / 741
Misclassification error rate: 0.164 = 123 / 750
```

Appendix B – Coding Scripts

```r
### R INDIVIDUAL COURSEWORK SVM+DTRF_Haixiang
## Question 1
# install packages
install.packages("ISLR")
install.packages("e1071")
install.packages("caret")
install.packages("pROC")
install.packages("ROCR")
library(ISLR)
library(caret)
library(e1071)
library(tree)
library(pROC)
library(randomForest)
library(ROCR)


# load data
data(OJ)
attach(OJ)

# data tidiness
OJ=OJ[complete.cases(OJ), ]     #check if there is any missing data
OJ

# split data with 70% training and 30% test data
trainIndex <- createDataPartition(OJ$Purchase, p=0.7,list=FALSE, times = 1)

train<-OJ[trainIndex,]
test<-OJ[-trainIndex,]
trainlabel<-OJ[trainIndex]
testlabel<-OJ[-trainIndex]

# specify the cost vector
set.seed(1234)
cost<-c(0.01,0.1,1,10)

########################################################################
#(1) fit support vector classifier
set.seed(122)
svm_linear = tune(svm,Purchase~.,data=train,
                 ranges=list(cost=cost),kernel='linear')

# summaty of accuracy rates with corresponding cost values
summary(svm_linear)
# plot
plot(svm_linear)

########################################################################
#(2) fit support vector machine with radial kernel
svm_radial = tune(svm,Purchase~.,data=train,
```

```
                ranges=list(cost=cost),kernel='radial',gamma=0.25)

# summaty of accuracy rates with corresponding cost values
summary(svm_radial)
#plot
plot(svm_radial)

##########################################################################
#(3) fit support vector machine with polynomial kernel
svm_poly = tune(svm,Purchase~.,data=train,
                ranges=list(cost=cost),kernel='polynomial',degree=2)   #specify cost and degree parameters

# summaty of accuracy rates with corresponding cost values
summary(svm_poly)
#plot
plot(svm_poly)

##########################################################################
#(4) compute test prediction and accuracy rate for each of the above SVM methods
pred1 = predict(svm_linear$best.model,test,decision.values=TRUE)
testrate1 = mean(test$Purchase != pred1)
pred2 = predict(svm_radial$best.model,test,decision.values=TRUE)
testrate2 = mean(test$Purchase != pred2)
pred3  = predict(svm_poly$best.model,test,decision.values=TRUE)
testrate3 = mean(test$Purchase != pred3)

# ROC function
rocplot =function (pred , truth , order,...){
  predob = prediction (pred , truth,label.ordering = order)
  perf = performance (predob,"tpr", "fpr")
  plot(perf,...)}

#source("rocplot.R")

# ROC plotting
roc_linear <- roc(response = test$Purchase, predictor =as.numeric(pred1))
roc_radial <- roc(response = test$Purchase, predictor =as.numeric(pred2))
roc_poly <- roc(response = test$Purchase, predictor =as.numeric(pred3))

plot(roc_linear,col = c("red"))
plot(roc_radial,col = c("blue"), add=TRUE)
plot(roc_poly,col = c("green"), add=TRUE)

# set up legend details
legend("bottomright", legend = c("Linear","Polynomial","Radial"), lty = c(1), col = c("red","blue","green"))

##########################################################################
#(5) decision tree
# install.packages
install.packages("tree")
```

```
library("tree")
install.packages("randomForest")
library("randomForest")
install.packages("gbm")
library("gbm")
install.packages("rpart")
library("rpart")
install.packages("e1071")
library("e1071")
install.packages("rpart.plot")
library("rpart.plot")
install.packages("AUC")
library(AUC)

# (5.1) train the model without cross validation or pruning
set.seed(13243)
OJ_tree = tree(Purchase ~., train)
summary(OJ_tree)

# plot the tree without cross validation or pruning
plot(OJ_tree)
text(OJ_tree,pretty=1,cex=0.8)

# evaluate the error rate
testrate_dt = mean(predict(OJ_tree, test, type='class')!=test$Purchase)
testrate_dt

# (5.2) train the model with cross-validation (10-fold, repeat 3 times)
fitcontrol=trainControl(method = "repeatedcv",
                        number = 10,
                        repeats = 3)
set.seed(1)
OJ_tree_cv=train(train[,-1],
                  train[,1],
                  method = "rpart",
                  trControl = fitcontrol)
OJ_tree_cv

# see the prediction result
pred_dt_cv = predict(OJ_tree_cv, test)

# plot the tree after cross validation
plot(OJ_tree_cv$finalModel)
text(OJ_tree_cv$finalModel,pretty=1,cex=.8)

# (5.3) find the optimal tree size
#tree pruning
OJ_cv = cv.tree(OJ_tree,FUN=prune.misclass)
plot(OJ_cv)
text(OJ_cv,pretty=1,cex=0.8)
```

```r

# visualize the best tree size by examining deviation
plot(OJ_cv$size, OJ_cv$dev, type = "b", xlab = "Tree Size", ylab = "Deviance")

# prune the tree according to the best tree size based on above analysis
prunedTree_best = prune.misclass(OJ_tree,best = OJ_cv$size[which.min(OJ_cv$dev)])
pred_dt_best = predict(prunedTree_best, test)
testrate_dt_best = mean(predict(prunedTree_best, test, type='class')!=test$Purchase)
testrate_dt_best

# (5.4) prune the tree with terminal nodes of 5
prunedTree_five = prune.misclass(OJ_tree,best = 5)
pred_dt_five = predict(prunedTree_five, test)
testrate_dt_five = mean(predict(prunedTree_five, test, type='class')!=test$Purchase)
testrate_dt_five

# plot the pruned tree with terminal nodes of 5
plot(prunedTree_five)
text(prunedTree_five,pretty=1,cex=0.8)

##########################################################################
#(6) random forest
# train the model under random forest method
set.seed(2)
mtryGrid=expand.grid(mtry=c(1,2,3,4,5,6))
OJ_rf=train(Purchase~.,data=train,method="rf",
            metric="Accuracy",
            tuneGrid=mtryGrid)
OJ_rf

# prediction with random forest
pred_rf = predict(OJ_rf, test, type='prob')
pred_rf

#variable importance
#importance(OJ_rf)
varImp(OJ_rf)

#plot variable importance
plot(varImp(OJ_rf))

##########################################################################
#(7) ROC plot for decision tree and random forest
dt_roc = rocplot(pred_dt_five[,2],test[,1],order=c("CH","MM"),col="black",lwd=2,cex.lab=1.5,cex.axis=1.5,main="Test set")
rf_roc = rocplot(pred_rf[,2],test[,1],order=c("CH","MM"),add=TRUE,col="blue",lwd=2,cex.lab=1.5,cex.axis=1.5)

# AUC values
auc(roc(pred_dt_five[,2],test[,1]))
auc(roc(pred_rf[,2],test[,1]))

# set up legend details
legend("bottomright",
       legend = c("Decision Tree","Random Forest"),
       col=c("black","blue"),cex=0.5,lty=1,lwd=2)

# set the axis scales
x=seq(0,1,0.01); y=x

# add reference line
lines(x,y,lwd =2, col =" red",lty=2)

```