lab2block1

Prudhvi Peddmallu

23 April 2019

deviance and gene and optimal tree depth and naive bayes classification of roc curves, TPR,FPR,NOn-parametric bootstrap and parametric bootstrap,lossmatric,Principle components.,decesion tree

Assignment 2. Analysis of credit scoring

1. Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.
2. Fit a decision tree to the training data by using the following measures of impurity

a. Deviance
b. Gini index and report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.

3. Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.
4. Use training data to perform classification using Naïve Bayes and report the confusion matrices and misclassification rates for the training and for the test data. Compare the results with those from step 3.
5. Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle $\hat{Y} = 1 if p(Y = good/X > \pi$, otherwise $\hat{Y} = 0$ π=0.05,0.1,0.15,.0.9,0.95. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?
6. Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix: predicted observed good 0 1 bad 10 0 and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
library(readxl)
library(tree)
library(ggplot2)
library(MASS)
library(e1071)
library(boot)
```

2.1divideing the data into test train and validation

Import the data to R and divide into training/validation/test as 50/25/25:

```
creditscore <- readxl::read_xls ("creditscoring.xls")

creditscore1<-read_excel("creditscoring.xls")
```

```
creditscore$good_bad <- as.factor(creditscore$good_bad)


n=dim(creditscore)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
cs_train=creditscore[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
cs_valid=creditscore[id2,]


id3=setdiff(id1,id2)
cs_test=creditscore[id3,]
```

The data was loaded and was divided into training, test and validation samples.

2.2Fit a decision tree-deviance & geni

```
fit_train=tree(good_bad ~., data=cs_train, split = "deviance")

#plot(fit_train)
#text(fit_train, pretty = 0)
#summary(fit_train)
Yfit=predict(fit_train, newdata=cs_test, type="class")
mc_table <- table(cs_test$good_bad,Yfit)
misclass_rate <- 1-sum(diag(mc_table))/sum(mc_table)
fit_train_gini=tree(good_bad ~., data=cs_train, split = "gini")
#summary(fit_train_gini)
Yfit_gini=predict(fit_train_gini, newdata=cs_test, type="class")
mc_table_gini <- table(cs_test$good_bad,Yfit_gini)
misclass_rate_gini <- 1-sum(diag(mc_table_gini))/sum(mc_table_gini)
```

2.3optimal tree depth

Q-Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.

```
fit_tree=tree(good_bad ~., data=cs_train, split = "deviance")

trainScore=rep(0,15)# because we have 15 leaves intial tree
validscore=rep(0,15)
for(i in 2:15) {
  prunedTree=prune.tree(fit_tree,best=i)
  pred=predict(prunedTree, newdata=cs_valid, type="tree")
  trainScore[i]=deviance(prunedTree)
```

```
  validscore[i]=deviance(pred)
}
df <- data.frame("Tree" = seq(2,15), "Test Score" = validscore[2:15], "Train Score" = trainScore[2:15])
ggplot(df, aes(x=Tree))+geom_point(aes(y=df$Test.Score))+geom_line(aes(y=df$Test.Score), colour = "Red")
  geom_point(aes(y=df$Train.Score))+geom_line(aes(y=df$Train.Score), colour = "Blue")+
  ylab("Train / Validation Score")+xlab("Number of Trees")+ggtitle("Deviance vs Number of Leaves")


optimalTree=prune.tree(fit_tree, best=4)

plot(optimalTree)
title(main = "Optimal Tree")
text(optimalTree)

Yfit_test=predict(optimalTree, newdata=cs_test, type="class")
mc_table_test <- table(cs_test$good_bad,Yfit_test)

misclass_rate <- 1-sum(diag(mc_table_test))/sum(mc_table_test)

cat("Misclassification table\n\n")
mc_table_test
cat("\nMisclassification rate for test data = ", misclass_rate )
```

2.4naivebayes-misclassificate and confusion matrix

```
fit_nb=naiveBayes(good_bad~., data=cs_train, split = "deviance")

Yfit_nb_train = predict(fit_nb, newdata=cs_train)
Yfit_nb_test=predict(fit_nb, newdata=cs_test)

cat("Confusion matrix for train data\n\n")
 table(cs_train$good_bad,Yfit_nb_train)

cat("\n\n Confusion matrix for test data\n\n")
table(cs_test$good_bad,Yfit_nb_test)

m_rate_train <- mean(cs_train$good_bad!=Yfit_nb_train)
m_rate_test <- mean(cs_test$good_bad!=Yfit_nb_test)

cat("\n Misclassification rate for train data = ",m_rate_train)
cat("\n\n Misclassification rate for test data = ",m_rate_test)
```

2.5-tpr-fpr

```
Yfit_nb=predict(fit_nb, newdata=cs_test, type = "raw")
Yfit_tree=predict(optimalTree, newdata=cs_test)


TPR_tree = c()
```

```
FPR_tree = c()
TPR_nb = c()
FPR_nb = c()

pi <- seq(from = 0.05,to = 0.95,by = 0.05)

for(i in 1:length(pi)){
  #optimal tree
  pred_tree <- ifelse(Yfit_tree[,2] > pi[i],"good","bad")
  mc_table_tree <- table(cs_test$good_bad,pred_tree)
  if(ncol(mc_table_tree)==1){
    if(colnames(mc_table_tree)=="good"){
      mc_table_tree <- cbind("bad"=c(0,0), mc_table_tree)
    }else{
      mc_table_tree <- cbind(mc_table_tree,"good"=c(0,0))
    }
  }
  TPR_tree <- c(TPR_tree,mc_table_tree["good","good"]/sum(mc_table_tree["good",]))
  FPR_tree <- c(FPR_tree,mc_table_tree["bad","good"]/sum(mc_table_tree["bad",]))


  #naivebayes
  pred_nb <- ifelse(Yfit_nb[,2] > pi[i],"good","bad")
  mc_table_nb <- table(cs_test$good_bad,pred_nb)
  if(ncol(mc_table_nb)==1){
    if(colnames(mc_table_nb)=="good"){
      mc_table_nb <- cbind("bad"=c(0,0), mc_table_nb)
    }else{
      mc_table_nb <- cbind(mc_table_nb,"good"=c(0,0))
    }
  }
  TPR_nb <- c(TPR_nb,mc_table_nb["good",2]/sum(mc_table_nb["good",]))
  FPR_nb <- c(FPR_nb,mc_table_nb["bad",2]/sum(mc_table_nb["bad",]))

}


plot(FPR_tree,TPR_tree,xlab = "FPR", ylab = "TPR")
title(main = "Optimal Tree")
lines(FPR_tree,TPR_tree)


plot(FPR_nb,TPR_nb, xlab = "FPR", ylab = "TPR")
title(main = "Naive Bayes")
lines(FPR_nb,TPR_nb)
```

2.6-naive bayes for-loss matrix

Naïve Bayes classification as it was in step 4 but use the following loss matrix: predicted observed good 0 1 bad 10 0

```r
fit_nb=naiveBayes(good_bad~., data=cs_train, type = "prob")
Yfit_nb_train_1 <- predict(fit_nb, newdata=cs_train)
mtable_train_1 <- table(Yfit_nb_train_1,cs_train$good_bad)

Yfit_nb_train = predict(fit_nb, newdata=cs_train, type = "raw")
Yfit_nb_train <- as.data.frame(Yfit_nb_train)
output <- rep(NA, nrow(Yfit_nb_train))
Yfit_nb_train <- cbind(output,Yfit_nb_train)

for(i in 1:nrow(Yfit_nb_train)){
  if((Yfit_nb_train[i,"good"]*1) > (Yfit_nb_train[i,"bad"]*10))
  {
    Yfit_nb_train[i,"output"] <- "good"
  }else{
    Yfit_nb_train[i,"output"] <- "bad"
  }
}


Yfit_nb_test=predict(fit_nb, newdata=cs_test, type = "raw")
Yfit_nb_test_1 <- as.data.frame(Yfit_nb_test)
output <- rep(NA, nrow(Yfit_nb_test_1))
Yfit_nb_test_1 <- cbind(output,Yfit_nb_test_1)

for(i in 1:nrow(Yfit_nb_test_1)){
  if((Yfit_nb_test_1[i,"good"]*1) > (Yfit_nb_test_1[i,"bad"]*10))
  {
    Yfit_nb_test_1[i,"output"] <- "good"
  }else{
    Yfit_nb_test_1[i,"output"] <- "bad"
  }
}

mc_table_train <- table(cs_train$good_bad,Yfit_nb_train$output)
mc_table_test <- table(cs_test$good_bad,Yfit_nb_test_1$output)
m_rate_train <- 1-sum(diag(mc_table_train))/sum(mc_table_train)
m_rate_test <- 1-sum(diag(mc_table_test))/sum(mc_table_test)
```

2-ass-2group code

```r
#library
library(readxl)
library(tree)
```

```r
#Reading the data
private_enterprise <-read_excel("creditscoring.xls")
private_enterprise<-na.omit(private_enterprise)
private_enterprise$good_bad = as.factor(private_enterprise$good_bad)
```

deviance train decesion tree

```
fit_model_deviance_train=tree(formula = good_bad~., data=train,split = "deviance")
predection_trainD <-predict(fit_model_deviance_train,newdata=train,
                            type="class")
confusionmatrix_trainD<-table( train$good_bad , predection_trainD)
diagonal_trainD<-diag(confusionmatrix_trainD)
summ_trainD<-sum(confusionmatrix_trainD)
misclassificationrate_trainD<-1-(sum(diagonal_trainD)/sum(summ_trainD))
misclassificationrate_trainD
plot(fit_model_deviance_train)
text(fit_model_deviance_train, pretty=0)
fit_model_deviance_train
summary(fit_model_deviance_train)
```

gini train decesion tree do for test

```
fit_model_gini_train=tree(formula = good_bad~., data=train,split = "gini")
predection_trainG <-predict(fit_model_gini_train,newdata=train,
                            type="class")
confusionmatrix_trainG<-table( train$good_bad , predection_trainG)
diagonal_trainG<-diag(confusionmatrix_trainG)
summ_trainG<-sum(confusionmatrix_trainG)
misclassificationrate_trainG<-1-(sum(diagonal_trainG)/sum(summ_trainG))
misclassificationrate_trainG
plot(fit_model_gini_train)
text(fit_model_gini_train, pretty=0)
fit_model_gini_train
summary(fit_model_gini_train)
```

2.2-gini test decesion tree

```
fit_model_gini_test=tree(formula = good_bad~., data=test,split = "gini")
predection_testG <-predict(fit_model_gini_test,newdata=test,
                            type="class")
confusionmatrix_testG<-table( test$good_bad , predection_testG)
diagonal_testG<-diag(confusionmatrix_testG)
summ_testG<-sum(confusionmatrix_testG)
misclassificationrate_testG<-1-(sum(diagonal_testG)/sum(summ_testG))
misclassificationrate_testG
plot(fit_model_gini_test)
text(fit_model_gini_test, pretty=0)
fit_model_gini_test
summary(fit_model_gini_test)
```

2.3tree depth train

```
#fit the model on train
fit_tree_depth_train=tree(good_bad~., data=train)
trainScore_dp=rep(2,15)
testScore_dp=rep(2,15)
for(i in 2:15) {
  #pruneing on test
  prunedTree=prune.tree(fit_tree_depth_train,best=i)
  pred=predict(prunedTree, newdata=test, type="tree")
  trainScore_dp[i]=deviance(prunedTree)
  testScore_dp[i]=deviance(pred)
}
plot(2:15, trainScore_dp[2:15], type="b", col="red", ylim=c(0,700))
points(2:15, testScore_dp[2:15], type="b", col="blue")
#finding the tree depth on validation
finalTree_leaf=prune.tree(fit_tree_depth_train, best=12)
Yfit_leaf=predict(finalTree_leaf, newdata=valid, type="class")
#table of the tree
table(valid$good_bad,Yfit_leaf)
#depth of tree
nodes_traindp_opt<-as.numeric(rownames(finalTree_leaf$frame))
depth_tree_train<<-max(tree:::tree.depth(nodes_traindp_opt))
depth_tree_train
```

2.4naive's bayes

```
#libraries for navies
library(MASS)
library(e1071)
```

naive bayes train

```
#navi bayes train
fitmodel_navitrain=naiveBayes(good_bad~., data=train)
predection_trainN <-predict(fitmodel_navitrain,newdata=train,
                            type="class")
confusionmatrix_trainN<-table( train$good_bad , predection_trainN)
diagonal_trainN<-diag(confusionmatrix_trainN)
summ_trainN<-sum(confusionmatrix_trainN)
misclassificationrate_trainN<-1-(sum(diagonal_trainN)/sum(summ_trainN))
misclassificationrate_trainN
```

naive bayes for test

```
#navi bayes test
fitmodel_navitest=naiveBayes(good_bad~., data=test)
predection_testN <-predict(fitmodel_navitrain,newdata=test,
                           type="class")
confusionmatrix_testN<-table( test$good_bad , predection_testN)
diagonal_testN<-diag(confusionmatrix_testN)
summ_testN<-sum(confusionmatrix_testN)
misclassificationrate_testN<-1-(sum(diagonal_testN)/sum(summ_testN))
misclassificationrate_testN
```

2.6-loss matrix naive bayes train

```
naive_train_loss=naiveBayes(good_bad~., data=train,type="prob")
predict_naivetrain_loss<-predict(naive_train_loss, newdata=train)
mtable_naivetrain_loss<- table(predict_naivetrain_loss,train$good_bad)
Ynaive_nb_train = predict(naive_train_loss, newdata=train, type = "raw")

Ynaive_nb_train <- as.data.frame(Ynaive_nb_train)
output <- rep(NA, nrow(Ynaive_nb_train))
Ynaive_nb_train <- cbind(output,Ynaive_nb_train)


for(i in 1:nrow(Ynaive_nb_train)){
if((Ynaive_nb_train[i,"good"]*1) > (Ynaive_nb_train[i,"bad"]*10))
  {
    Ynaive_nb_train[i,"output"] <- "good"
  }else{
    Ynaive_nb_train[i,"output"] <- "bad"
  }}

Ynaivetest=predict(naive_train_loss, newdata=test, type = "raw")
Ynaivetest1 <- as.data.frame(Ynaivetest)
output <- rep(NA, nrow(Ynaivetest1))
Ynaivetest1 <- cbind(output,Ynaivetest1)


for(i in 1:nrow(Ynaivetest1)){
  if((Ynaivetest1[i,"good"]*1) > (Ynaivetest1[i,"bad"]*10))
  {
    Ynaivetest1[i,"output"] <- "good"
  }else{
    Ynaivetest1[i,"output"] <- "bad"
  }
}


missclasification_tabletrainls <- table(train$good_bad,Ynaive_nb_train$output)
mcisclassification_tabletestls <- table(test$good_bad,Ynaivetest1$output)
misrate_trainls <- 1-sum(diag(missclasification_tabletrainls))/sum(missclasification_tabletrainls)
misrate_testls <- 1-sum(diag(mcisclassification_tabletestls))/sum(mcisclassification_tabletestls)
```

```
cat("Confusion matrix for train data \n")
missclasification_tabletrainls
cat("\n\nConfusion matrix for test data \n")
mcisclassification_tabletestls

cat("\n Misclassification rate for train data = ",misrate_trainls)
cat("\n\n Misclassification rate for test data = ",misrate_testls)
```

2-ass-2-method-AN

2.1

```
set.seed(12345)
credit_data <- read.xlsx("creditscoring.xls", sheetName = "credit")
credit_data$good_bad <- as.factor(credit_data$good_bad)
n=NROW(credit_data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=credit_data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=credit_data[id2,]
id3=setdiff(id1,id2)
test=credit_data[id3,]
```

2.2

```
set.seed(12345)
# Create a decision tree model
credit_tree_deviance <- tree(good_bad~., data=train, split = c("deviance"))
credit_tree_gini <- tree(good_bad~., data=train, split = c("gini"))
# Visualize the decision tree with rpart.plot
summary(credit_tree_deviance)
summary(credit_tree_gini)
# predicting on the test dataset to get the misclassification rate.
predict_tree_deviance <- predict(credit_tree_deviance, newdata = test, type = "class")
predict_tree_gini <- predict(credit_tree_gini, newdata = test, type = "class")
conf_tree_deviance <- table(test$good_bad, predict_tree_deviance)
names(dimnames(conf_tree_deviance)) <- c("Actual Test", "P2redicted Test")
caret::confusionMatrix(conf_tree_deviance)
conf_tree_gini <- table(test$good_bad, predict_tree_gini)
names(dimnames(conf_tree_gini)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_tree_gini)
```

Analysis: On the Training dataset model with 'deviance' had a misclassfication rate of 21.05% while the model with 'gini' split had the misclassification rate of 23.68%. For the test dataset we see that the model with 'deviance' type of split has a accuracy of 73.2% or misclassifiaction rate of 26.8%, we see that to predict

'good' the accuracy is 89.08% but for predicting bad its just 36.84%. Thus our our model is heavily baised towards predicting cases as 'good'. For the test dataset we see that the model with 'gini' type of split has a accuracy of 63.6% or misclassifiaction rate of 36.4%, we also see that to predict 'good' the accuracy is 81.03% but for predicting bad its just 18.97%. Thus our model is heavily baised towards predicting cases as 'good' even more than the model which uses 'deviance' to split variable. Both our models would lead to many bad loan applicants to be given loans which is never a good thing, however among the model the one using 'deviance' mode for split is better by 9.6%. Thus we will select model using 'deviance' for further model building.

2.3

```
set.seed(12345)
credit_tree <- tree(good_bad~., data=train, split = c("deviance"))
credit_tree_purned_train <- prune.tree(credit_tree, method = c("deviance"))
credit_tree_purned_valid <- prune.tree(credit_tree, newdata = valid ,method = c("deviance"))
result_train <- cbind(credit_tree_purned_train$size,
credit_tree_purned_train$dev, "Train")
result_valid <- cbind(credit_tree_purned_valid$size,
credit_tree_purned_valid$dev, "Valid")
result <- as.data.frame(rbind(result_valid, result_train))
colnames(result) <- c("Leaf", "Deviance", "Type")
result$Leaf <- as.numeric(as.character(result$Leaf))
result$Deviance <- as.numeric(as.character(result$Deviance))
# plot of deviance vs. number of leafs
ggplot(data = result, aes(x = Leaf, y = Deviance, colour = Type)) +
geom_point() + geom_line() +
ggtitle("Plot of Deviance vs. Tree Depth (Shows Deviance least at 4)")
# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=4)
plot(credit_tree_sniped)
text(credit_tree_sniped)
# misclassification rate for best pruned tree
result_prune_test <- predict(credit_tree_sniped, newdata = test, type = "class")
conf_prune_tree_test <- table(test$good_bad, result_prune_test)
names(dimnames(conf_prune_tree_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_prune_tree_test)
```

Analysis: Choosing optimal depth tree we get that '4' as the best depth. The variables used in the best tree are- duration, history, savings. From the tree structure we can see that the following variables are best to split on, 'savings' < 2.5 then 'duration' < 43.5 and 'history' < 1.5. The accuracy on the model trained on 'train' dataset is accuracy 73.2% and misclassifiaction 26.8% while on the 'test' dataset accuracy is 74.4%, thus the misclassification rate is 25.6%. We see that model predicts 'good' applicants very well (accuracy of 96.55%) while it classifies 'bad' applicant way badly (accuracy is 23.68%). Thus this model would be very bad for the business and would likely to run the business bankrupt.

2.4

```
#Fitting the Naive Bayes model
credit_naive_model = naiveBayes(good_bad ~., data=train)
#Prediction on the dataset
```

```
predict_naive_train = predict(credit_naive_model, newdata=train, type = "class")
predict_naive_test = predict(credit_naive_model, newdata=test, type = "class")
conf_naive_train <- table(train$good_bad, predict_naive_train)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
conf_naive_test <- table(test$good_bad, predict_naive_test)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
```

Analysis: For the train dataset using NaiveBayes method we get accuracy 70% or misclassification of 30%, here we alsonotice that the accuracy of class 'bad' is 64.63% while for class 'good' is 72.24%, thus the model is morebalanced in predicting, thus its still baised in predict one class over the other. For the test dataset using NaiveBayes method we get accuracy 68.4% or misclassification of 31.6%, herewe also notice that the accuracy of class 'bad' is 60.53% while for class 'good' is 71.84%, thus the model isalmost the same compared to train.Compared to step3, we see that for the 'train' dataset the optimal tree has accuracy of 73.2% while it is74.4% on the 'test' dataset. For the NaiveBayes model, accuracy on the 'train' dataset is 70%and while it is68.4% on the 'test' datatset.Accuracy is only part of the story what we see is better here is that this model classifies 'bad' customersbetter better for both train and test dataset than decision tree (60+% for both train and test for naive compared 36.84% train, 23.68% test for decision tree).

2.5

```
set.seed(12345)
credit_tree <- tree(good_bad~., data=train, split = c("deviance"))
credit_naive_model = naiveBayes(good_bad ~., data=train)
# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=7)
# predicting class, getting probability
predict_prune_test_prob <- predict(credit_tree_sniped, newdata = test)
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")
# data mugging
probability_data_naive <- as.data.frame(cbind(predict_naive_test_prob,
                                              as.character(test$good_bad), "naivebayes"))
probability_data_tree <- as.data.frame(cbind(predict_prune_test_prob,
                                             as.character(test$good_bad), "tree"))
probability_data_combined <- rbind(probability_data_tree, probability_data_naive)
colnames(probability_data_combined) <- c("prob_bad", "prob_good",
                                         "actual_test_class", "model")
# final dataset
probability_data_combined$prob_good <- as.numeric(as.character(probability_data_combined$prob_good))
# changing the threshold and printing the probability
tree_list <- NULL
naive_list <- NULL
final <- NULL
for(threshold in seq(from = 0.05, to = 0.95, by = 0.05)){
 probability_data_combined$predicted_class <- ifelse(probability_data_combined$prob_good > threshold, "

  df2 <- probability_data_combined[,c("model", "actual_test_class", "predicted_class")]
  df2$threshold <- threshold
  df2$match <- ifelse(df2$actual_test_class == df2$predicted_class, 1, 0)
```

```
  final <- rbind(df2, final)
}
# Creating the FRP and TRP for each model and threshold
final$temp <- 1
final_summary <- final %>%
group_by(model, threshold) %>%
summarise(total_positive = sum(temp[actual_test_class == "good"]),
          total_negative = sum(temp[actual_test_class == "bad"]),
          correct_positive = sum(temp[actual_test_class == "good" & predicted_class == "good"]),
          false_positive = sum(temp[actual_test_class == "bad" & predicted_class == "good"])) %>%
    mutate(TPR = correct_positive/total_positive, FPR = false_positive/total_negative) %>%
  select(model, threshold, TPR, FPR)
ggplot(data = final_summary, aes(x = FPR, y=TPR)) + geom_line(aes(colour = model)) +
  geom_abline(intercept = 0, slope = 1) +
  ggtitle("ROC curve for the Naive Bayes vs. Tree Model")
```

2.6

```
set.seed(12345)
credit_naive_model = naiveBayes(good_bad ~., data=train)
# predicting class, getting probability
predict_naive_train_prob <- predict(credit_naive_model, newdata=train, type = "raw")
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")
train <- cbind(predict_naive_train_prob, train)
test <- cbind(predict_naive_test_prob, test)
# class based on the loss matrix
train$predicted_class <- ifelse(train$good > 10*train$bad, "good", "bad")
test$predicted_class <- ifelse(test$good > 10*test$bad, "good", "bad")
# confusion matrix
conf_naive_train <- table(train$good_bad, train$predicted_class)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
conf_naive_test <- table(test$good_bad, test$predicted_class)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
```

Analysis: We see a major drop in accuracy for both train and test cases for using the new loss matrix, theaccuracy for train is 45.4% and for test cases its 49.2%, this was 70% and 68.4% respectively.The biggest difference is in recognizing the bad customers, where the accuracy is 93.20% and 93.42% for thetrain and test cases respectively. This implies our model identifies 'bad' customers with a very high accuracywhile it suffers to detect 'good' customers (accuracy of 25% and 29%). This is actually good for businessbecause its many times more important to identify the bad customers instead of good customer(minimizerisk).The change in the accuracy is due to loss matrix where we have ensured that a customer is considered 'good'customer only if the probability of them being a 'good'customer is 10 times greater than them being a 'bad'customer.

Assignment 3. Uncertainty estimation

Uncertainty estimation

ASS3-method

3.1

```
rm(list=ls())
set.seed(12345)
state_data <- read.csv2("state.csv")
state_data <- state_data %>% arrange(MET)
ggplot(data = state_data, aes(x=MET, y = EX)) +
geom_point() +
geom_smooth(method = 'loess') +
ggtitle("Plot of MET vs. EX")
```

Analysis: As evident from the graph the best model, linear regression will not be a good fit, even the trend is non linear. Piece wise linear model(spline) might be a good one, thus regression per group/cluster will be a good approach.

3.2

```
set.seed(12345)
state_tree_regression <- tree(data = state_data, EX~MET,
control = tree.control(nobs=NROW(state_data),
minsize = 8))
state_cv_tree <- cv.tree(state_tree_regression, FUN = prune.tree)
temp <- cbind(size = state_cv_tree$size, deviance = state_cv_tree$dev) %>% as.data.frame()
ggplot(data = temp, aes(x = size, y = deviance)) +
geom_point() +theme_bw() +
ggtitle("Deviance vs. number of leaves")
# The best size is either 3 or 4
# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, best = 3)
plot(state_cv_tree_purned, main="Pruned Tree for the given dataset")
text(state_cv_tree_purned)
compare_data <- predict(state_cv_tree_purned, newdata = state_data)
compare_data <- cbind(compare_data, state_data$EX, state_data$MET)
compare_data <- as.data.frame(compare_data)
colnames(compare_data) <- c("predicted_value", "EX", "MET")
compare_data$residual <- compare_data$EX - compare_data$predicted_value
# plots
ggplot(data=compare_data, aes(x = MET, y = EX)) +
geom_point(aes(x = MET,y=EX)) +
geom_line(aes(x = MET, y=predicted_value), colour="blue") +
ggtitle("EX value along with Predicted Value")
ggplot(compare_data, aes(x = EX, y = predicted_value)) +
geom_point() +
```

```
ggtitle("Plot of Actual vs. Predicted Value")
ggplot(compare_data, aes(x = predicted_value, y = residual)) +
geom_point() + geom_abline(slope=0, intercept=0) +
ggtitle("Plot of Predicted Value vs. Residual")
# Residuals
ggplot(data = compare_data, aes(residual)) +
geom_histogram(col="black",aes(fill=..count..),bins = 30) +
scale_fill_gradient("Count", low = "grey", high = "black") +
theme_bw() +
ggtitle("Histogram of residuals") +
labs(x="Residual", y="Count")
```

Analysis: The predicted vs. Actual provides us the insight that for lower values of variable, our model is over predicting(actual value ~200) while the predicted value is ~250. While for larger values (~400) our model is underpredicting (~330). At around the mean value (~300) the predicted values are both under and over predictedthus no bias. Thus for values that are away from the mean our model is baised towards over/under predictedwhile values close to mean our model is not biased. From the plot of Predicted vs. Residual values we can see that error appears random, neither is largebias/concentration of the error towards any value expect at lower/higher values(more points on one side ofthe line)From the histogram we can see that the histogram has higher values on the left of zero, and a longer tailon the right. Thus from the above three points we can see that there is scope of improvement in the model especially in the extreme values of the predicted values.

3.3

```
set.seed(12345)
# computing bootstrap samples
bootstrap <- function(data, indices){
data <- state_data[indices,]
model <- tree(data = data,
EX~MET,
control = tree.control(nobs=NROW(data),
minsize = 8))
model_purned <- prune.tree(model, best = 3)
final_fit_boot <- predict(model_purned, newdata = state_data)
return(final_fit_boot)
}
e <- envelope(res, level = 0.95)
state_tree_regression <- tree(data = state_data, EX~MET,
control = tree.control(nobs=NROW(state_data),
minsize = 8))
# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, best = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)
data_for_ci <- cbind(upper_bound = e$point[1,],
lower_bound = e$point[2,],
EX = state_data$EX,
MET = state_data$MET,
predicted_value = predict_for_ci) %>% as.data.frame()
#plot cofidence bands
ggplot(data=data_for_ci, aes(x = MET, y = EX)) +
geom_point(aes(x = MET,y=EX)) +
geom_line(aes(x = MET, y=predicted_value), colour="blue") +
```

```
geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
ggtitle("EX value along with 95% Confidence band")
res <- boot(state_data, bootstrap, R=1000) #make bootstrap
```

Analysis: The confidence bands certainly appear to be bumpy and not smooth, the confidence bands are bumpy becausethe predicted values shows large flucations. From the width of the confidence band we can assume that ourmodel is not a good one. Ideally we want the confidence band to be narrow thus a wider band suggests weneed further tuning to the model

3.4

```
set.seed(12345)
mle=prune.tree(state_tree_regression, best = 3)
#data2 = state_data
rng=function(data, mle) {
data1=data.frame(EX=data$EX, MET=data$MET)
n=length(data$EX)
pred <- predict(mle, newdata = data1)
residual <- data1$EX - pred
#generate new Price
data1$EX=rnorm(n, pred, sd(residual))
return(data1)
}
# computing bootstrap samples
conf.fun <- function(data){
model <- tree(data = data,
EX~MET,
control = tree.control(nobs=NROW(data),
minsize = 8))
model_purned <- prune.tree(model, best = 3)
final_fit_boot <- predict(model_purned, newdata = state_data)
return(final_fit_boot)
}
# computing bootstrap samples
pred.fun <- function(data){
model <- tree(data = data,
EX~MET,
control = tree.control(nobs=NROW(data),
minsize = 8))
model_purned <- prune.tree(model, best = 3)
final_fit_boot <- predict(model_purned, newdata = state_data)
final_fit <- rnorm(n = length(final_fit_boot), mean = final_fit_boot, sd=sd(residuals(mle)))
return(final_fit)
}
conf_para = boot(state_data, statistic=conf.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")
pred_para = boot(state_data, statistic=pred.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")
e1 <- envelope(conf_para, level = 0.95)
e2 <- envelope(pred_para, level = 0.95)

# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, best = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)
```

```
data_for_ci_para <- cbind(upper_bound = e1$point[1,],
lower_bound = e1$point[2,],
upper_bound_pred = e2$point[1,],
lower_bound_pred = e2$point[2,],
EX = state_data$EX,
MET = state_data$MET,
predicted_value = predict_for_ci) %>% as.data.frame()
ggplot(data=data_for_ci_para, aes(x = MET, y = EX)) +
geom_point(aes(x = MET,y=EX)) +
geom_line(aes(x = MET, y=predicted_value), colour="blue") +
geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
geom_ribbon(aes(x = MET, ymin=lower_bound_pred, ymax=upper_bound_pred), alpha = 0.3) +
ggtitle("EX value along with 95% Confidence(dark grey) and Prediction band")
```

Analysis: From the plot we see that the width of the 'confidence band'(darker grey) is narrower compared to nonparametric.We also notice that most of the data is at in the 'confidence band', thus I believe that ourregression model does make sense.We can defintely see that most of the data is under the 'prediction band' more than 95%, thus we canreasonably expect that less than 5% of the data is outside of the 'prediction band'. Based on our assumptionof data is normally distributed it does makes sense for prediction band to contain 95% of the data.

3.5

```
set.seed(12345)
# Residuals
geom_histogram(col="black",aes(fill=..count.., y = ..density..),bins = 30) +
scale_fill_gradient("Count", low = "grey", high = "black") +
geom_density(colour = "red") +
theme_bw() +
ggtitle("Histogram of residuals")
```

Analysis:Judging from the density plot we can say that the density resembles that of a 'Beta Distribution' thus a known distribution is fit better using parametric bootstrap.

Assignment 4. Principal components

Question 1

```
library(fastICA)
library(lattice)
library(plotly)
```

```
nirspectra <- read.csv2("NIRspectra.csv")
```

```
ncolum <- ncol(nirspectra)

NIRdata=scale(nirspectra[,-c(ncolum)])

nir_pca=prcomp(NIRdata, center = T)
```

```r
lambda=nir_pca$sdev^2
#proportion of variation
sprintf("%2.3f",lambda[c(1,2,3,4)]/sum(lambda)*100)

# plot percentage of variance explained for each principal component
screeplot(nir_pca, main = "Scree plot", xlab = "Principal Components")
```

```r
NIRdata2 <- cbind(NIRdata, nir_pca$x[,1:2])

NIRdata2 <- as.data.frame(NIRdata2)

#scores
 ggplot(data = NIRdata2, aes(x=PC1, y=PC2) )+
  geom_point()+
  ggtitle("Score plot - PC2 vs PC1")
```

pc-trace plots

2. Make trace plots of the loadings of the components selected in step 1. Is there any principle component that is explained by mainly a few original features?

```r
#the rotation component gives the loadings
U=nir_pca$rotation

#nrow(U)
#sorted.loadings <- U[order(U[, 1]), 1]

myXlab  <- "Variables"
myYlab <- "Variable Loadings"

plot(U[,1], main="Loadings Plot for PC1", xlab=myXlab, ylab=myYlab, cex=1.5, col="red")
plot(U[,2], main="Loadings Plot for PC2", xlab=myXlab, ylab=myYlab, cex=1.5, col="red")
```

From the trace plots above, it is clearly visible that the Principal Component 2 has few features explining it. Among 127 features which explains a diesel fuel, only around 20 features are correlated to PC2 as shown in the Loadings plot for PC2. PC1 is correlated to a higher number of features.

Question 3a

```r
set.seed(12345)
ica<-fastICA(NIRdata, n.comp=2)
W_p <- ica$K %*% ica$W

xvar <- "Features"
yvar <- "Correlation"
plot(W_p[,1], main="Traceplot", cex = 1.5, col = "red", xlab = xvar, ylab = yvar)

plot(W_p[,2],main="Traceplot", cex = 1.5, col = "red", xlab = xvar, ylab = yvar)
```

Question 3b

```
NIRdata3 <- cbind(ica$S[,1:2],NIRdata)

NIRdata3 <- as.data.frame(NIRdata3)
#scores
ggplot(data = NIRdata3, aes(x=V1, y=V2))+geom_point()+
     ggtitle("Score plot")
```

my code PCA

```
 nirspectra<-read.csv2("NIRSpectra.csv",header = TRUE)
```

4.1-PCA

```
#plot on viscosity
data_NIR=nirspectra
data_NIR$Viscosity=c()
res=prcomp(data_NIR)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
```

```
#rotation
U_rotation=res$rotation
head(U_rotation)
#plot on rotation
plot(res$x[,1], res$x[,2])
```

4ass-2method-an

4.1

```
rm(list=ls())
set.seed(12345)

NIR_data <- read.csv2("NIRSpectra.csv")
pca_data = select(NIR_data,-c(Viscosity))
pca_result = prcomp(pca_data)
contribution <- summary(pca_result)$importance
knitr::kable(contribution[,1:5],
caption = "Contribution of PCA axis towards varience explaination")
eigenvalues = pca_result$sdev^2
```

```r
# plotting proportion of variation for principal components
plotData = as.data.frame(cbind(pc = 1:3,
variationProportion = eigenvalues[1:3]/sum(eigenvalues),
cummulative = cumsum(eigenvalues[1:3]/sum(eigenvalues))))
ggplot(data = plotData) +
geom_col(aes(x = pc, y = variationProportion), width = 0.3, fill = "grey70") +
geom_line(data = plotData,
aes(x = pc, y = cummulative)) +
geom_text(aes(x = pc, y = cummulative, label = round(cummulative, 3)), size = 4,
position = "identity", vjust = 1.5) +
theme_bw() +
ggtitle("Proportion of variation for principal components")
# pca components and the viscocity
pca_result_data = cbind(first_component = pca_result$x[,1],
second_component = pca_result$x[,2]) %>% as.data.frame()
# plotting the data variation and the viscocity
ggplot(data = pca_result_data, aes(x = first_component, y = second_component)) +
geom_point() + ggtitle("Score Plot of PCA components")
# showing the score of PCA component
factoextra::fviz_pca_var(pca_result,
col.var = "contrib", # Color by contributions to the PC
gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
repel = TRUE # Avoid text overlapping
)
```

Analysis: From the plot of PCA component vs. Viscoity, we can see that 2 components are needed (second component of PCA is needed), this is due to the fact the most of the data is vertically spread, removing this dimension would make it impossible to differenitate the types of diesel. The minimum number of components that account for 99% of the total varience is 2, mainly PC1 and PC2. From the plot we can also see that there are evidently some diesel that are outliers (diesel with first_component> 0.5 and second component~4).

4.2

```r
set.seed(12345)
# creating extra columns
aload <- abs(pca_result$rotation[,1:2])
components <- sweep(aload, 2, colSums(aload), "/")
components <- as.data.frame(components)
components$feature_name <- rownames(components)
components$feature_index <- 1:nrow(components)
ggplot(data = components, aes(x = feature_index, y = PC1)) +
geom_point() +
ggtitle("Traceplot of feature index vs. PC1")
ggplot(data = components, aes(x = feature_index, y = PC2)) +
geom_point() +
ggtitle("Traceplot of feature index vs. PC2")
knitr::kable(components[1:10,],
caption = "Contribution of Features towards the Principle Components")
```

Analysis: While for PC1, the loadings of the variables do not differ too much (maximum loading ~ 0.11, minimum loading ~ 0.075), for PC the loadings differ extremely. This principal component is explained mainly by the variables with a high index (on the right of the plot). Here, the maximum loading (~ 0.35)

is much higher than for most of the other variables. From these two plots is evident that there are no few features(<5) which form the core essence of the PCA components, thus the PCA components is made up of many (20+) features atleast and they cannot be limited

4.3

```
set.seed(12345)
# X -> pre-processed data matrix
# K -> pre-whitening matrix that projects data onto the first n.compprincipal components.
# W -> estimated un-mixing matrix (see definition in details)
# A -> estimated mixing matrix
# S -> estimated source matrix
X <- as.matrix(pca_data)
ICA_extraction <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200,
tol = 0.0001, verbose = TRUE)
W_prime <- ICA_extraction$K %*% ICA_extraction$W
#trace plots
plot(W_prime[,1], main = "Trace plot of W` Matrix")
#trace plots
plot(W_prime[,2], main = "Trace plot of W` Matrix")

# pca components and the viscocity
ICA_result_data = cbind(first_component = ICA_extraction$S[,1],
second_component = ICA_extraction$S[,2],
Viscosity = NIR_data$Viscosity) %>% as.data.frame()
# plotting the data variation
ggplot(data = ICA_result_data, aes(x = first_component, y = second_component)) +
geom_point() + ggtitle("Score Plot for ICA components")
```

Analysis:Comparing with the trace plots from step 2, we see that ICA trace plots are almost a (flipped using horizontalaxis) flipped image of the PCA trace plots.The W'(multipled matrix) represents the loadings of the ICA components, which are very similar to the PCA loadings.Comparing the Score plots here we find that the plot is again a flipped version of the score plot from PCA(flipped along vertical axis).