machine learning(732A99) lab2

Anubhav Dikshit(anudi287)

10 December 2018

Contents

Loading The Libraries

Assignment 2

2.1 Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.

```r
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*0.4))
train=credit_data[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=credit_data[id2,]

id3=setdiff(id1,id2)
test=credit_data[id3,]
```

2.2 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.

```r
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)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = c("deviance"))
## Variables actually used in tree construction:
##  [1] "duration" "history"  "marital"  "existcr"  "amount"   "purpose"
##  [7] "savings"  "resident" "age"      "other"
## Number of terminal nodes:  22
## Residual mean deviance:  0.7423 = 277.6 / 374
## Misclassification error rate: 0.1869 = 74 / 396
```

```r
summary(credit_tree_gini)
```

```
##
## Classification tree:
## tree(formula = good_bad ~ ., data = train, split = c("gini"))
## Variables actually used in tree construction:
##  [1] "foreign"  "coapp"    "depends"  "telephon" "existcr"  "savings"
##  [7] "history"  "property" "amount"   "marital"  "duration" "resident"
## [13] "job"      "installp" "purpose"  "employed" "housing"
## Number of terminal nodes:  53
## Residual mean deviance:  0.9468 = 324.7 / 343
## Misclassification error rate: 0.2247 = 89 / 396
```

```
# 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)
```

```
## Confusion Matrix and Statistics
##
##              P2redicted Test
## Actual Test bad good
##        bad   49   43
##        good  56  152
##
##                Accuracy : 0.67
##                  95% CI : (0.6136, 0.723)
##     No Information Rate : 0.65
##     P-Value [Acc > NIR] : 0.2539
##
##                   Kappa : 0.2534
##  Mcnemar's Test P-Value : 0.2278
##
##             Sensitivity : 0.4667
##             Specificity : 0.7795
##          Pos Pred Value : 0.5326
##          Neg Pred Value : 0.7308
##              Prevalence : 0.3500
##          Detection Rate : 0.1633
##    Detection Prevalence : 0.3067
##       Balanced Accuracy : 0.6231
##
##        'Positive' Class : bad
##
```

```
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)
```

```
## Confusion Matrix and Statistics
##
##              Predicted Test
## Actual Test bad good
##        bad   25   67
##        good  42  166
##
##                Accuracy : 0.6367
##                  95% CI : (0.5794, 0.6912)
##     No Information Rate : 0.7767
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.0755
##  Mcnemar's Test P-Value : 0.02152
##
##             Sensitivity : 0.37313
##             Specificity : 0.71245
##          Pos Pred Value : 0.27174
##          Neg Pred Value : 0.79808
##              Prevalence : 0.22333
##          Detection Rate : 0.08333
##    Detection Prevalence : 0.30667
##       Balanced Accuracy : 0.54279
##
##        'Positive' Class : bad
##
```

Analysis: On the Training dataset model with 'deviance' had a misclassfication rate of 18.7% while the model with 'gini' split had the misclassification rate of 22.8%.

For the test dataset we see that the model with 'deviance' type of split has a accuracy of 67% or misclassifiaction rate of 33%, we see that to predict 'good' the accuracy is 73.08% but for predicting bad its just 53.26%. 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 62.7% or misclassifiaction rate of 37.3%, we also see that to predict 'good' the accuracy is 78.8% but for predicting bad its just 26%. Thus our model is heavily baised towards predicting cases as 'good' even more than the model which uses 'deviance' to split variable.

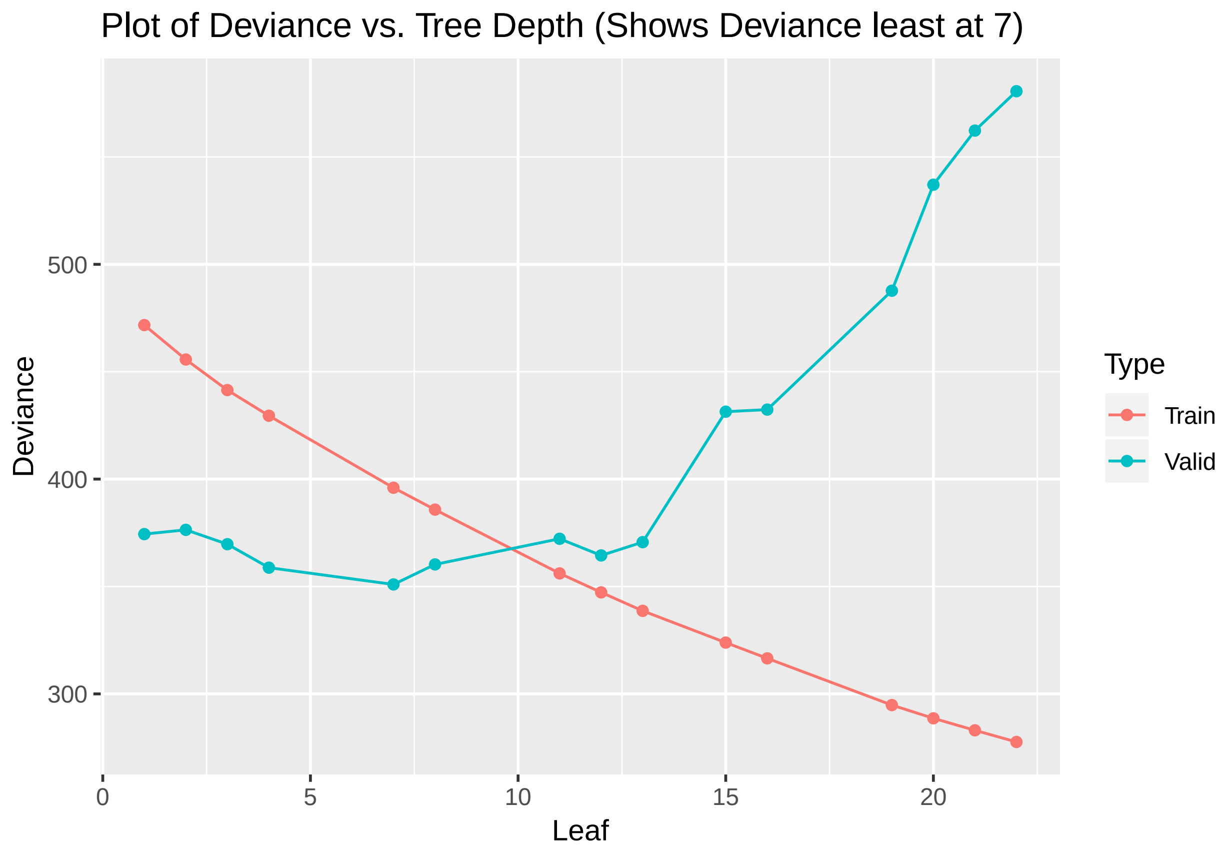
Both our models would lead to many bad loan applicants to be given loans which is never a good thing, however among the model the one using 'deviance' mode for split is better by 27%.

Thus we will select model using 'deviance' for further model building.

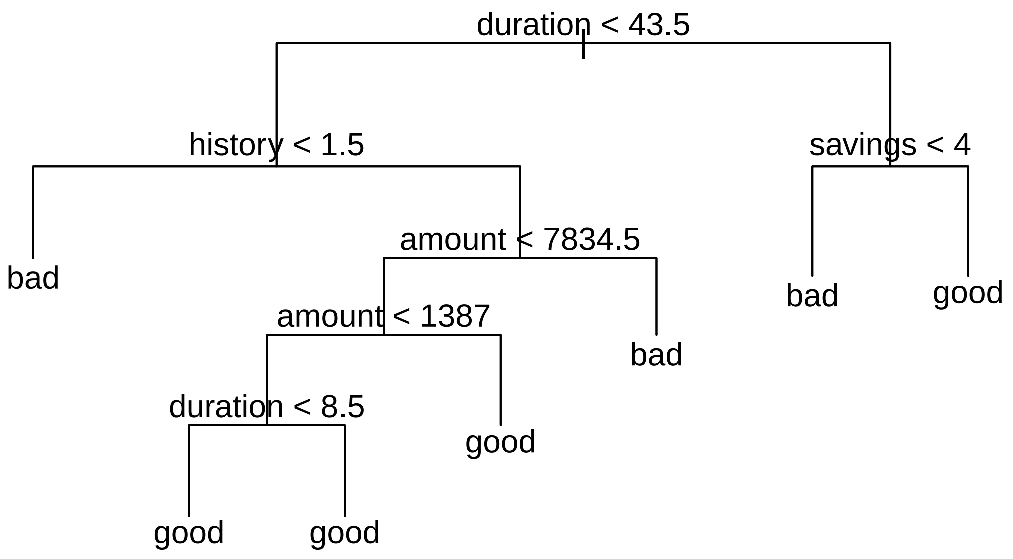
3. Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by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

```r
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 7)")
```



```
# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=7)

plot(credit_tree_sniped)
text(credit_tree_sniped)
```



```r
# 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)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test bad good
##        bad   32   60
##        good  19  189
##
##                Accuracy : 0.7367
##                  95% CI : (0.683, 0.7856)
##     No Information Rate : 0.83
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2929
##  Mcnemar's Test P-Value : 0.000006784
##
##             Sensitivity : 0.6275
##             Specificity : 0.7590
##          Pos Pred Value : 0.3478
##          Neg Pred Value : 0.9087
##              Prevalence : 0.1700
##          Detection Rate : 0.1067
##    Detection Prevalence : 0.3067
##       Balanced Accuracy : 0.6932
##
##        'Positive' Class : bad
##
```

Analysis: Choosing optimal depth tree we get that '7' as the best depth. The variables used in the best tree are- duration, history, savings, amount.

From the tree structure we can see that the following variables are best to split on, 'duration' < 43.5 then 'savings' < 4, 'history' < 1.5 and finally 'amount'.

The accuracy on the model trained on 'train' dataset is (77% and misclassification 23%) and on the 'test' dataset accuracy is 73.67%, thus the misclassification rate is 26.33%. We see that model predicts 'good' applicants very well (accuracy of 90%) while it classifies 'bad' applicant way badly (accuracy is 34.78%).

Thus this model would be very bad for the business and would likely to run the business bankrupt.

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.

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

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   bad  good
## 0.285 0.715
##
## Conditional probabilities:
##       duration
## Y          [,1]     [,2]
##   bad  24.85965 14.95946
##   good 19.13287 11.11076
##
##       history
## Y          [,1]     [,2]
##   bad  2.263158 1.113494
##   good 2.744755 1.033419
##
##       purpose
## Y          [,1]     [,2]
##   bad  2.883929 2.859300
##   good 2.718310 2.409833
```

```
##
##       amount
## Y           [,1]     [,2]
##   bad  3661.202 3504.882
##   good 2995.524 2431.313
##
##       savings
## Y           [,1]     [,2]
##   bad  1.736842 1.343965
##   good 2.314685 1.664572
##
##       employed
## Y           [,1]     [,2]
##   bad  3.236842 1.250151
##   good 3.500000 1.147843
##
##       installp
## Y           [,1]      [,2]
##   bad  3.175439 0.9798466
##   good 2.898601 1.1271131
##
##       marital
## Y           [,1]      [,2]
##   bad  2.596491 0.7252622
##   good 2.716783 0.6650668
##
##       coapp
## Y           [,1]      [,2]
##   bad  1.105263 0.4072011
##   good 1.171329 0.5383695
##
##       resident
## Y           [,1]     [,2]
##   bad  2.824561 1.074611
##   good 2.807692 1.121439
##
##       property
## Y           [,1]     [,2]
##   bad  2.456140 1.073744
##   good 2.300699 1.069743
##
##       age
## Y           [,1]     [,2]
##   bad  33.91228 10.97589
##   good 36.51399 11.21050
##
##       other
## Y           [,1]      [,2]
##   bad  2.500000 0.8227947
##   good 2.751748 0.6362558
##
##       housing
## Y           [,1]      [,2]
##   bad  1.938596 0.5991895
##   good 1.965035 0.5014028
##
##       existcr
## Y           [,1]      [,2]
##   bad  1.368421 0.5988656
##   good 1.458042 0.5893693
##
##       job
## Y           [,1]      [,2]
##   bad  2.824561 0.7194592
##   good 2.849650 0.6503305
##
##       depends
## Y           [,1]      [,2]
##   bad  1.140351 0.3488843
##   good 1.164336 0.3712295
##
##       telephon
## Y           [,1]      [,2]
##   bad  1.315789 0.4668818
##   good 1.388112 0.4881745
##
##       foreign
## Y           [,1]      [,2]
##   bad  1.017544 0.1318659
##   good 1.045455 0.2086640
```

```
#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)
```

```
## Confusion Matrix and Statistics
##
##               Predicted Train
## Actual Train bad good
##         bad   62   52
##         good  55  231
##
##                Accuracy : 0.7325
##                  95% CI : (0.6863, 0.7753)
##     No Information Rate : 0.7075
##     P-Value [Acc > NIR] : 0.1480
##
##                   Kappa : 0.3488
##  Mcnemar's Test P-Value : 0.8467
##
##             Sensitivity : 0.5299
##             Specificity : 0.8163
##          Pos Pred Value : 0.5439
##          Neg Pred Value : 0.8077
##              Prevalence : 0.2925
##          Detection Rate : 0.1550
##    Detection Prevalence : 0.2850
##       Balanced Accuracy : 0.6731
##
##        'Positive' Class : bad
##
```

```
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)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Test
## Actual Test bad good
##        bad   50   42
##        good  45  163
##
##                Accuracy : 0.71
##                  95% CI : (0.6551, 0.7607)
##     No Information Rate : 0.6833
##     P-Value [Acc > NIR] : 0.1762
##
##                   Kappa : 0.3242
##  Mcnemar's Test P-Value : 0.8302
##
##             Sensitivity : 0.5263
##             Specificity : 0.7951
##          Pos Pred Value : 0.5435
##          Neg Pred Value : 0.7837
##              Prevalence : 0.3167
##          Detection Rate : 0.1667
##    Detection Prevalence : 0.3067
##       Balanced Accuracy : 0.6607
##
##        'Positive' Class : bad
##
```

Analysis:

For the train dataset using NaiveBayes method we get accuracy 73% or misclassification of 27%, here we also notice that the accuracy of class 'bad' is 54% while for class 'good' is 80%, thus the model is more balanced in predicting, thus its still baised in predict one class over the other.

For the test dataset using NaiveBayes method we get accuracy 71% or misclassification of 29%, here we also notice that the accuracy of class 'bad' is 54% while for class 'good' is 78%, thus the model is almost the same compared to train.

Compared to step3, we see that for the 'train' dataset the optimal tree has accuracy of 77% while it is 73% on the 'test' dataset. For the NaiveBayes model, accuracy on the 'train' dataset is 73% and while it is 71% on the 'test' datatset.

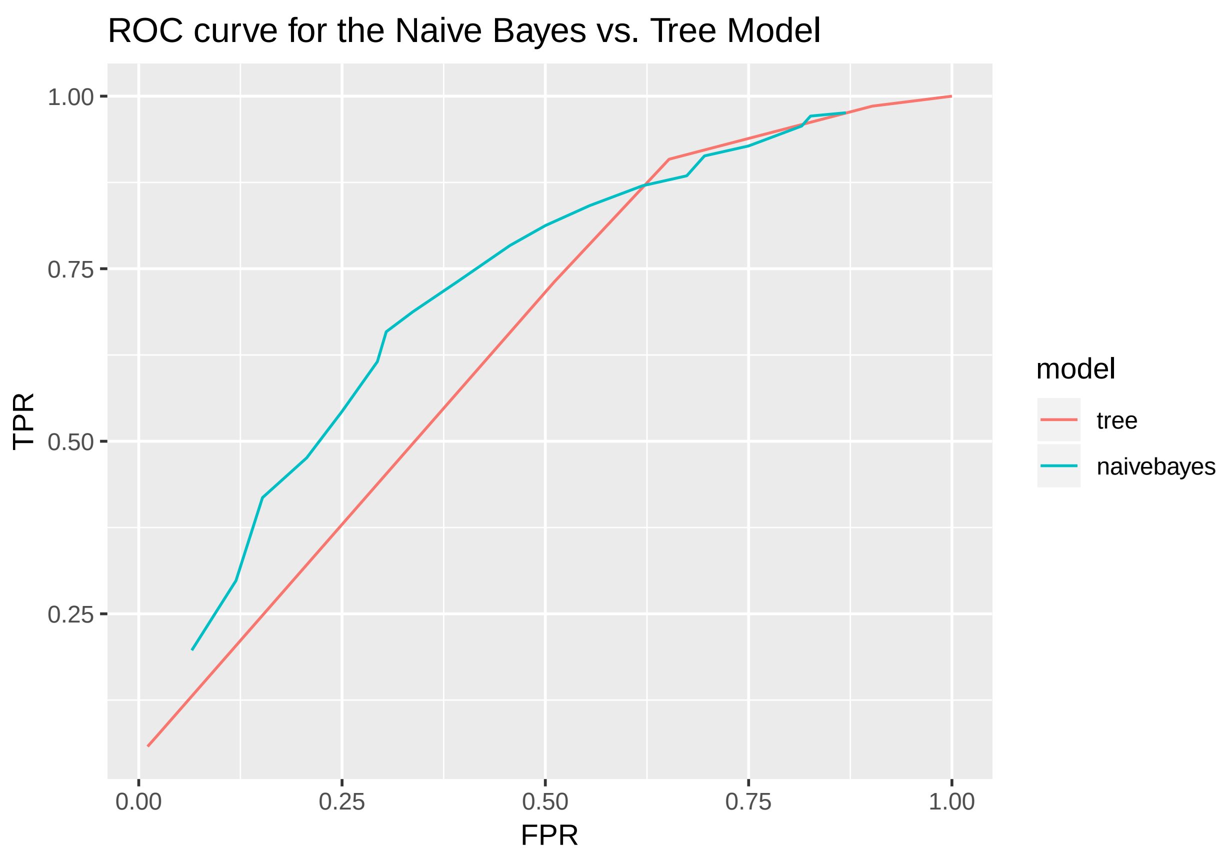
Accuracy is only part of the story what we see is better here is that this model classifies 'bad' customers better better for both train and test dataset than decision tree (54% for both train and test for naive compared 38% train, 34% test for decision tree).

Thus the model is better to be used for the business than the one in the step3, the risk of providing loans to bad applicant is lesser than the previous model but its still not good enough!

5. Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle: where prob(Y|'good')=A, where A=0.05,0.10,. . . ..0.95.

Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?

```
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$good_bad), "na
probability_data_tree <- as.data.frame(cbind(predict_prune_test_prob, as.character(test$good_bad), "tre
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)) + ggtitle("ROC curve
```



Analysis: We find that 'naivebayes' model is better than 'tree' model for across varying threshold values.

6. Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix (good loss 1, bad loss 10) 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.

```
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")
```

```r
caret::confusionMatrix(conf_naive_train)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Train
## Actual Train bad good
##         bad  105    9
##         good 202   84
##
##                Accuracy : 0.4725
##                  95% CI : (0.4227, 0.5227)
##     No Information Rate : 0.7675
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1423
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.3420
##             Specificity : 0.9032
##          Pos Pred Value : 0.9211
##          Neg Pred Value : 0.2937
##              Prevalence : 0.7675
##          Detection Rate : 0.2625
##    Detection Prevalence : 0.2850
##       Balanced Accuracy : 0.6226
##
##        'Positive' Class : bad
##
```

```r
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)
```

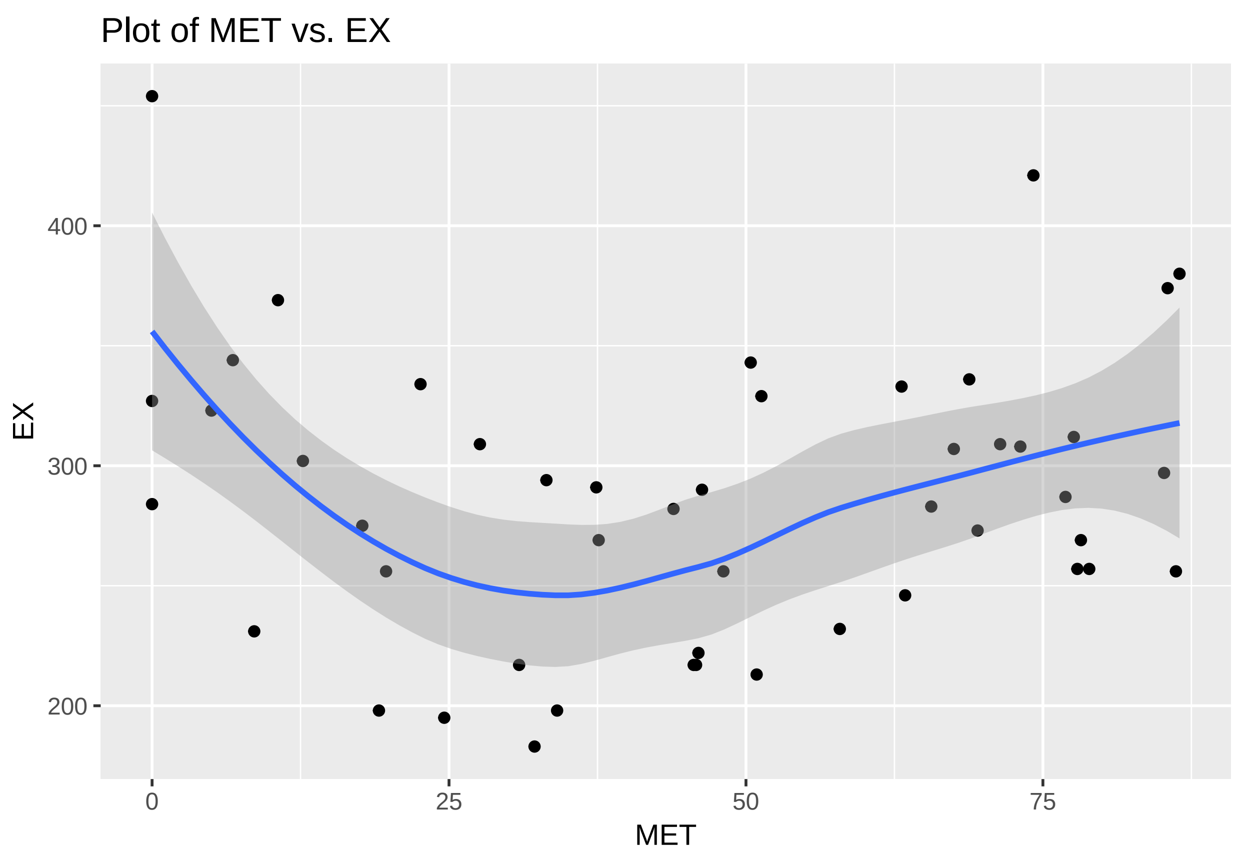
```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test bad good
##        bad   81   11
##        good 147   61
##
##                Accuracy : 0.4733
##                  95% CI : (0.4157, 0.5315)
##     No Information Rate : 0.76
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.123
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.3553
##             Specificity : 0.8472
##          Pos Pred Value : 0.8804
##          Neg Pred Value : 0.2933
##              Prevalence : 0.7600
##          Detection Rate : 0.2700
##    Detection Prevalence : 0.3067
##       Balanced Accuracy : 0.6012
##
##        'Positive' Class : bad
##
```

Assignment 3

1. Reorder your data with respect to the increase of MET and plot EX versus MET. Discuss what kind of model can be appropriate here. Use the reordered data in steps 2-5.

```r
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() +
  ggtitle("Plot of MET vs. EX")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Analysis:

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
if (!require("pacman")) install.packages("pacman")
pacman::p_load(xlsx, ggplot2, MASS, tidyr, dplyr, reshape2, gridExtra,
               tree, caret, e1071, pROC)

set.seed(12345)
options("jtools-digits" = 2, scipen = 999)
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*0.4))
train=credit_data[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
```

```r
valid=credit_data[id2,]

id3=setdiff(id1,id2)
test=credit_data[id3,]
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)
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 7)")

# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=7)

plot(credit_tree_sniped)
text(credit_tree_sniped)

# misclassification rate for best pruned tree
```

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

#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)
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$good_bad), "na
probability_data_tree <- as.data.frame(cbind(predict_prune_test_prob, as.character(test$good_bad), "tre
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)) + ggtitle("ROC curve


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)


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() +
  ggtitle("Plot of MET vs. EX")
```