Assignment 3

Arthur Junges Schmidt

28 July 2020

Problem 1

a)

The file ps3-1.csv contains a data set with 34 features (x1, x2, . . . x34) and 1 target variable (Y). Estimate a classifier model using Support Vector Machine and Random Forest algorithm in R, respectively, with the first 14628 rows of the data. Optimize your model so that a False Positive Rate is less than 10% for Y = 0 (actual Y = 1 cases falsely classified as Y = 0). Particularly, you are required to review relevant literature, use the k-fold cross-validation method to train the Random Forest model. Use grid search to find hyper-parameter setting: the best number of trees and features, maximum leaf nodes. Assess importance of each feature based on two criteria: Mean Decrease Accuracy and Mean Decrease Gini. Compare two estimation methods with confusion matrices

```r
Crude_Data <- read.csv("ps3-1.csv");
Crude_Data <- Crude_Data[,-1];
Crude_Data$Y <- factor(Crude_Data$Y, levels = c(0,1));
Training_Data <- Crude_Data[1:14628, ];
Test_Data <- Crude_Data[-(1:14628), ];
```

SVM

Firstly, a cluster with 3 processors is created. The library randomForest is loaded for the future runs of the random forest models.

Radial SVM were tried but didn't present better accuracy values while taking much longer to finish. For the linear SVM models, only the cost parameter is tuned. For cost values greater than 10, the models take an enormous amount of time to run. Because of this, the cost values tested are 10^{-4}, 0.001, 0.01, 0.1, 1, 10.

```r
###### Cluster creation for usage of 4 processor cores

cluster <- makeCluster(3, outfile = "cluster_log.txt")
clusterEvalQ(cluster, library(randomForest))
```

```
## [[1]]
## [1] "randomForest" "stats"        "graphics"     "grDevices"    "utils"
## [6] "datasets"     "methods"      "base"
##
## [[2]]
## [1] "randomForest" "stats"        "graphics"     "grDevices"    "utils"
## [6] "datasets"     "methods"      "base"
##
## [[3]]
## [1] "randomForest" "stats"        "graphics"     "grDevices"    "utils"
## [6] "datasets"     "methods"      "base"
```

```r
registerDoParallel(cluster)


SVM_tunegrid <- expand.grid(C = 10^(-4:1))
SVM_trControl <-  trainControl("cv", number = 10,
                               verboseIter = TRUE)

Start_Time1 <- Sys.time()

SVM_Fit <- train(Y ~ ., data = Training_Data,
                 method = 'svmLinear', tuneGrid = SVM_tunegrid,
                 preProcess = c("center", "scale"),
                 trControl = SVM_trControl
                 )
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting C = 0.01 on full training set
```

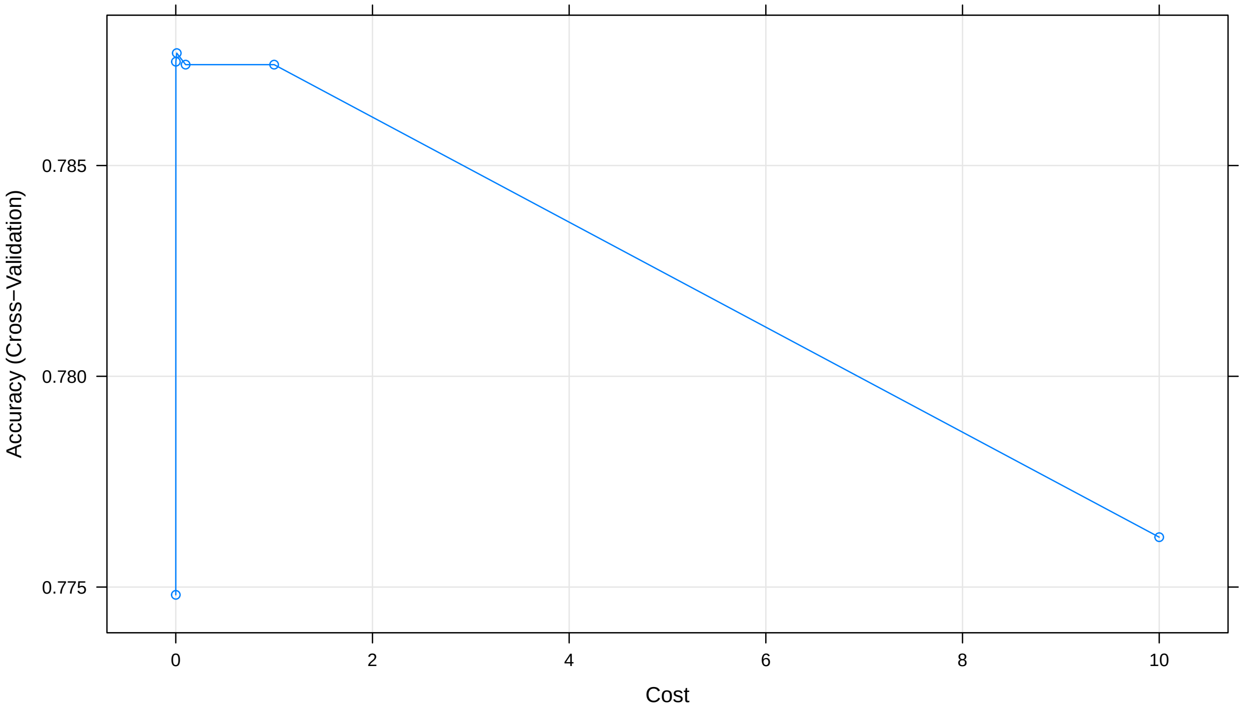
```r
End_time1 <- Sys.time()
End_time1 - Start_Time1
```

```
## Time difference of 33.62468 mins
```

```r
beep(sound = 3)
SVM_Fit
```

```
## Support Vector Machines with Linear Kernel
##
## 14628 samples
##    34 predictor
##     2 classes: '0', '1'
##
## Pre-processing: centered (34), scaled (34)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 13164, 13164, 13165, 13166, 13165, 13166, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   1e-04  0.7748156  0.0001958057
##   1e-03  0.7874621  0.1428249398
##   1e-02  0.7876667  0.1584670780
##   1e-01  0.7873933  0.1586325282
##   1e+00  0.7873933  0.1588643911
##   1e+01  0.7761834  0.0142333764
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

```r
plot(SVM_Fit)
```




```r
# stopCluster(cluster)
```

```r
confusionMatrix(SVM_Fit)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    0    1
##          0  3.1  1.8
##          1 19.4 75.6
##
##  Accuracy (average) : 0.7877
```

```r
confusionMatrix(SVM_Fit, scale = FALSE, norm = "none")
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##           Reference
## Prediction     0     1
##          0   456   269
##          1  2837 11066
##
##  Accuracy (average) : 0.7877
```

The False positive rate for the SVM model is:

$$\frac{274}{274 + 11061} = 0.0232558$$

Random Forest

According to Friedman, Hastie, and Tibshirani (2001) and James et al. (2013), the usual number of predictors candidates m from the full set of predictors p is:

$$m \approx \sqrt{p}$$

So with our data, $m \approx \sqrt{(}35) \approx 6$.

Very conveniently, the Caret package in R enables the user to modify the 'train' function with more options for parameter tuning. Originally, the only parameter accepted for tuning is 'mtry'. However, with the code below, the 'ntre' (number of trees) and 'maxnode' (maximum number of leaves) are added as tunable parameter. Additionally, these parameters are easily compared with the plot function after the training process. The time to train the random forest model with a wide grid of paramenters is very long. After many trials, these parameters in the code below showed the best results. If a wider range of values would have been used, the training process would take longer than 2 hours.

Cross validation is used, with the method "cv" on the traincontrol parameters. 10 folds of the data were used.

```
customRF <- list(type = "Classification",
                 library = "randomForest",
                 loop = NULL)

customRF$parameters <- data.frame(parameter = c("mtry", "ntree", "maxnodes"),
                                  class = rep("numeric", 3),
                                  label = c("mtry", "ntree", "maxnodes"))

customRF$grid <- function(x, y, len = NULL, search = "grid") {}

customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs) {
  randomForest(x, y,
               mtry = param$mtry,
               ntree=param$ntree,
               maxnodes = param$maxnodes)
}
# customRF$varImp = randomForest::importance()


#Predict label
customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata)

#Predict prob
customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata, type = "prob")

customRF$sort <- function(x) x[order(x[,1]),]
customRF$levels <- function(x) x$classes
###### Cluster creation for usage of 4 processor cores

# cluster <- makeCluster(4, outfile = "cluster_log.txt")

# registerDoParallel(cluster)

######
control <- trainControl(method="cv",
                        number=10,
                        allowParallel = TRUE,
                        verboseIter = TRUE
                        )

tunegrid <- expand.grid(.mtry=c(6, 12, 18),.ntree=c(100, 200, 500, 1000),
                        .maxnodes= seq(40, 80, by = 10))

set.seed(123)
Start_Time <- Sys.time()
```

```r
RF_fit <- train(Y ~ ., data=Training_Data,
                method=customRF,
                metric="Accuracy",
                tuneGrid=tunegrid,
                trControl=control,
                preProcess = c("center", "scale"))
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 18, ntree = 200, maxnodes = 70 on full training set
```

```r
RF_fit
```

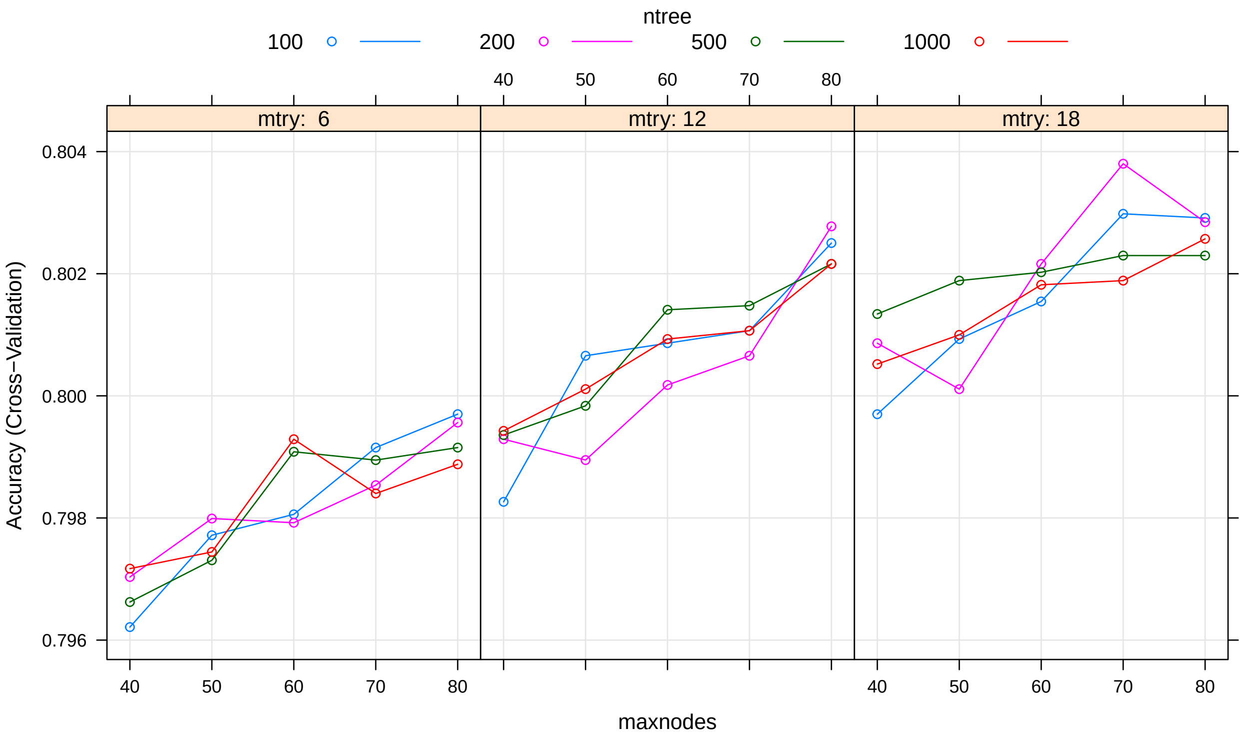
```
## 14628 samples
##    34 predictor
##     2 classes: '0', '1'
##
## Pre-processing: centered (34), scaled (34)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 13164, 13166, 13165, 13166, 13165, 13165, ...
## Resampling results across tuning parameters:
##
##   mtry  ntree  maxnodes  Accuracy   Kappa
##    6     100   40        0.7962132  0.2111804
##    6     100   50        0.7977171  0.2227248
##    6     100   60        0.7980593  0.2286118
##    6     100   70        0.7991532  0.2301188
##    6     100   80        0.7997000  0.2368729
##    6     200   40        0.7970333  0.2130106
##    6     200   50        0.7979906  0.2213336
##    6     200   60        0.7979220  0.2251047
##    6     200   70        0.7985373  0.2304688
##    6     200   80        0.7995627  0.2367970
##    6     500   40        0.7966234  0.2105034
##    6     500   50        0.7973072  0.2176746
##    6     500   60        0.7990841  0.2300480
##    6     500   70        0.7989474  0.2317502
##    6     500   80        0.7991527  0.2334957
##    6    1000   40        0.7971704  0.2119310
##    6    1000   50        0.7974434  0.2177843
##    6    1000   60        0.7992896  0.2316868
##    6    1000   70        0.7984006  0.2280635
##    6    1000   80        0.7988790  0.2315812
##   12     100   40        0.7982640  0.2333386
##   12     100   50        0.8006567  0.2488602
##   12     100   60        0.8008620  0.2512937
##   12     100   70        0.8010670  0.2554192
##   12     100   80        0.8025026  0.2603824
##   12     200   40        0.7992899  0.2395242
##   12     200   50        0.7989483  0.2422626
##   12     200   60        0.8001782  0.2482287
##   12     200   70        0.8006562  0.2518950
##   12     200   80        0.8027760  0.2627972
##   12     500   40        0.7993581  0.2394375
##   12     500   50        0.7998367  0.2424730
##   12     500   60        0.8014094  0.2532998
##   12     500   70        0.8014774  0.2569183
##   12     500   80        0.8021609  0.2601519
##   12    1000   40        0.7994265  0.2411453
##   12    1000   50        0.8001103  0.2445972
##   12    1000   60        0.8009303  0.2523560
##   12    1000   70        0.8010670  0.2528631
##   12    1000   80        0.8021608  0.2595027
##   18     100   40        0.7996993  0.2518039
```

```
##   18     100   50         0.8009303  0.2554983
##   18     100   60         0.8015455  0.2621105
##   18     100   70         0.8029811  0.2708361
##   18     100   80         0.8029133  0.2683676
##   18     200   40         0.8008619  0.2528957
##   18     200   50         0.8001098  0.2520912
##   18     200   60         0.8021606  0.2626078
##   18     200   70         0.8038011  0.2742724
##   18     200   80         0.8028444  0.2701257
##   18     500   40         0.8013400  0.2567492
##   18     500   50         0.8018871  0.2589970
##   18     500   60         0.8020239  0.2635747
##   18     500   70         0.8022977  0.2641732
##   18     500   80         0.8022974  0.2663044
##   18    1000   40         0.8005200  0.2525610
##   18    1000   50         0.8009987  0.2540426
##   18    1000   60         0.8018187  0.2623193
##   18    1000   70         0.8018873  0.2653432
##   18    1000   80         0.8025711  0.2676318
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 18, ntree = 200 and maxnodes
##  = 70.
```

```r
beepr::beep(sound = 3)
End_time <- Sys.time()
End_time - Start_Time
```

```
## Time difference of 45.41493 mins
```

```r
stopCluster(cluster)
plot(RF_fit)
```



It can be seen that after a tree size of 500, there's no real improvement in the model. On the contrary, the model with a $ntree = 1000$ has a lower accuracy.

For the False Negative Rate, the confusion matrix is needed. The confusion matrix is made from the best model found with the combination of the parameters. The results shown below represent the average percentage across the 10 folds.

```r
confusionMatrix.train(RF_fit, scale = FALSE)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    0    1
##          0  5.6  2.7
##          1 16.9 74.8
##
##  Accuracy (average) : 0.8038
```

For the absolute number of predictions on the confusion matrix:

```r
confusionMatrix(RF_fit, scale = FALSE, norm = "none")
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are un-normalized aggregated counts)
##
##           Reference
## Prediction     0     1
##          0   814   391
##          1  2479 10944
##
##  Accuracy (average) : 0.8038
```

For the False Negative Rate:

$$\frac{352}{352 + 10983} = 0.0344949$$

which is lower than 10%.

A problem on this custom model approach is that the function 'importance()' which returns the mean decrease accuracy and mean decrease Gini values for the variables only returns the latter. To fix this, the best model created from all the parameters is then performed on the standard 'randomForest' function, which will then return both values needed. The best tree found has the following parameters:

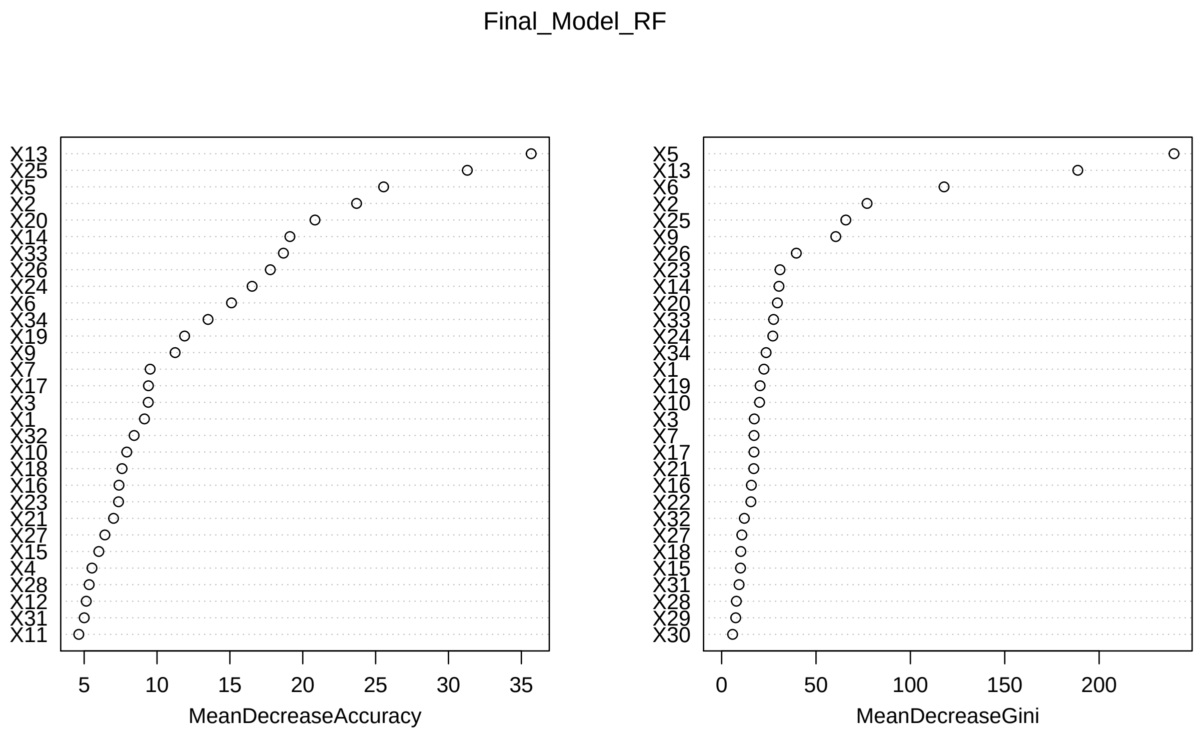
- mtry: 18
- ntree: 200
- maxnodes: 70

```r
Final_Model_RF <- randomForest::randomForest(Y ~ ., data = Training_Data,
                              importance = TRUE,
                              maxnodes = RF_fit$bestTune$maxnodes,
                              ntree = RF_fit$bestTune$ntree,
                              mtry = RF_fit$bestTune$mtry,
                              )

randomForest::importance(Final_Model_RF)
```

```
##              0         1 MeanDecreaseAccuracy MeanDecreaseGini
## X1  -3.1350077  8.684343             9.138065        22.398738
## X2  18.1803233 15.536954            23.693336        77.047578
## X3  -5.7730298  9.083533             9.398316        17.292237
## X4  -4.1526965  5.600003             5.535307         4.635864
## X5  17.0274271 18.193796            25.547996       239.685103
## X6   8.5005561 11.370602            15.109900       117.850555
## X7   8.2153407  4.806137             9.528824        17.153917
## X8   0.2614249  3.015978             3.313908         1.167246
## X9   4.4523950  8.724671            11.240888        60.460703
## X10 -1.9729567  7.520187             7.924722        20.092064
```

```
## X11   0.7159633  3.564119            4.633832        5.001636
## X12   4.6100149  2.297454            5.143486        2.824433
## X13  31.9027671 14.268899           35.674560      188.701261
## X14  14.6478751 13.705871           19.118585       30.368115
## X15  -2.0982609  5.565842            6.006956       10.000815
## X16  -6.3956444  8.454785            7.390562       15.742378
## X17  -4.0621721  8.904377            9.409999       17.127181
## X18  -5.5634895  8.370537            7.600518       10.159125
## X19  -6.2994073 11.267261           11.892713       20.377816
## X20 -19.9184915 22.081579           20.839146       29.537608
## X21  -2.5306942  6.641170            7.016895       17.006306
## X22  -6.8638469  6.037021            4.559389       15.494716
## X23  -0.1207487  6.890545            7.359970       30.904763
## X24   2.6830868 14.312232           16.523914       27.091645
## X25  14.7745846 25.864365           31.292830       65.790560
## X26   5.3442021 15.478463           17.773310       39.560023
## X27  -6.3485691  7.410440            6.422632       10.676986
## X28  -1.4514960  4.977894            5.344819        7.804195
## X29  -5.2105448  4.568787            3.199981        7.445018
## X30  -2.4896511  3.969252            3.572444        5.809545
## X31  -7.2262525  6.279071            5.005495        9.238386
## X32  -6.1637622  9.165715            8.431119       12.028401
## X33   4.4281556 13.748674           18.674398       27.501807
## X34  -8.0084093 13.119581           13.499581       23.556274
```

```
varImpPlot(Final_Model_RF)
```

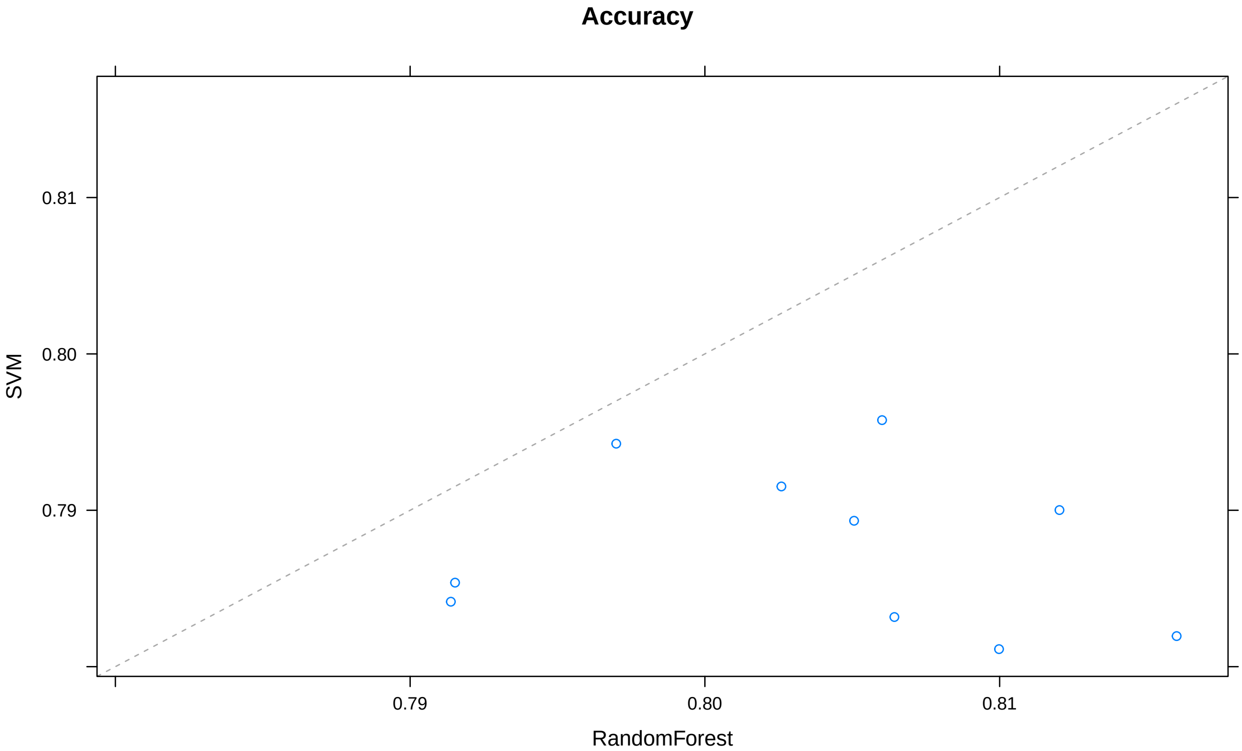



```
Both_models <- resamples(list(SVM = SVM_Fit,
                              RandomForest = RF_fit))

summary(Both_models)
```

```
##
## Call:
## summary.resamples(object = Both_models)
##
```

```
## Models: SVM, RandomForest
## Number of resamples: 10
##
## Accuracy
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM         0.7811218 0.7834186 0.7873511 0.7876667 0.7911466 0.7957650    0
## RandomForest 0.7913817 0.7983933 0.8055362 0.8038011 0.8090920 0.8160055    0
##
## Kappa
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## SVM         0.1241445 0.1449338 0.1579005 0.1584671 0.1643562 0.2001500    0
## RandomForest 0.2034334 0.2646406 0.2820759 0.2742724 0.2925561 0.3208643    0
```

```r
xyplot(Both_models)
```



b)

Use two estimated models to predict the last 200 rows of the data and compare prediction with the observed target value (Y).

```r
Predictions <- predict(list(SVM = SVM_Fit,
                            RandomForest = RF_fit),
                       newdata = Test_Data[, -35])

confusionMatrix(data = Predictions$SVM, reference = Test_Data[, 35])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0   6   1
##          1  44 149
##
##                Accuracy : 0.775
```

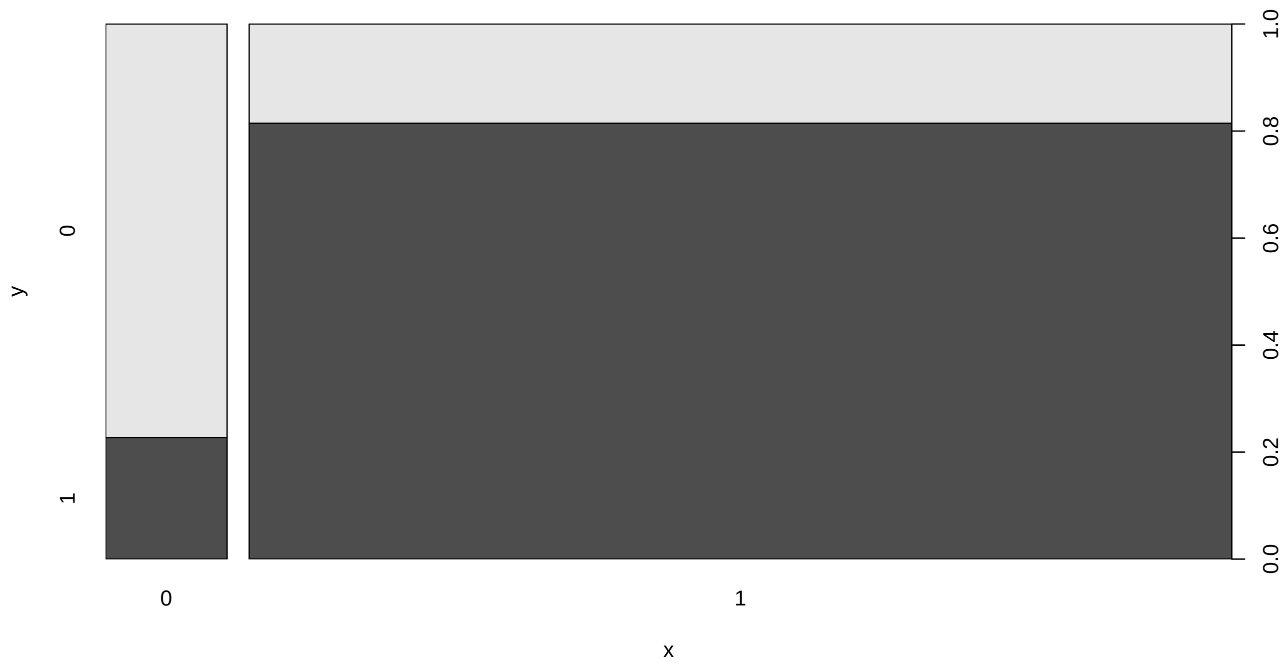
```
##                 95% CI : (0.7108, 0.8309)
##    No Information Rate : 0.75
##    P-Value [Acc > NIR] : 0.2332
##
##                  Kappa : 0.1589
##
## Mcnemar's Test P-Value : 3.825e-10
##
##            Sensitivity : 0.1200
##            Specificity : 0.9933
##         Pos Pred Value : 0.8571
##         Neg Pred Value : 0.7720
##             Prevalence : 0.2500
##         Detection Rate : 0.0300
##   Detection Prevalence : 0.0350
##      Balanced Accuracy : 0.5567
##
##       'Positive' Class : 0
##
```

```r
confusionMatrix(data = Predictions$RandomForest, reference = Test_Data[, 35])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  17   5
##          1  33 145
##
##                Accuracy : 0.81
##                  95% CI : (0.7487, 0.8619)
##    No Information Rate : 0.75
##    P-Value [Acc > NIR] : 0.02758
##
##                  Kappa : 0.377
##
## Mcnemar's Test P-Value : 1.187e-05
##
##            Sensitivity : 0.3400
##            Specificity : 0.9667
##         Pos Pred Value : 0.7727
##         Neg Pred Value : 0.8146
##             Prevalence : 0.2500
##         Detection Rate : 0.0850
##   Detection Prevalence : 0.1100
##      Balanced Accuracy : 0.6533
##
##       'Positive' Class : 0
##
```

```r
plot(Predictions$SVM, Test_Data[, 35])
```


```
plot(Predictions$RandomForest, Test_Data[, 35])
```



Problem 2

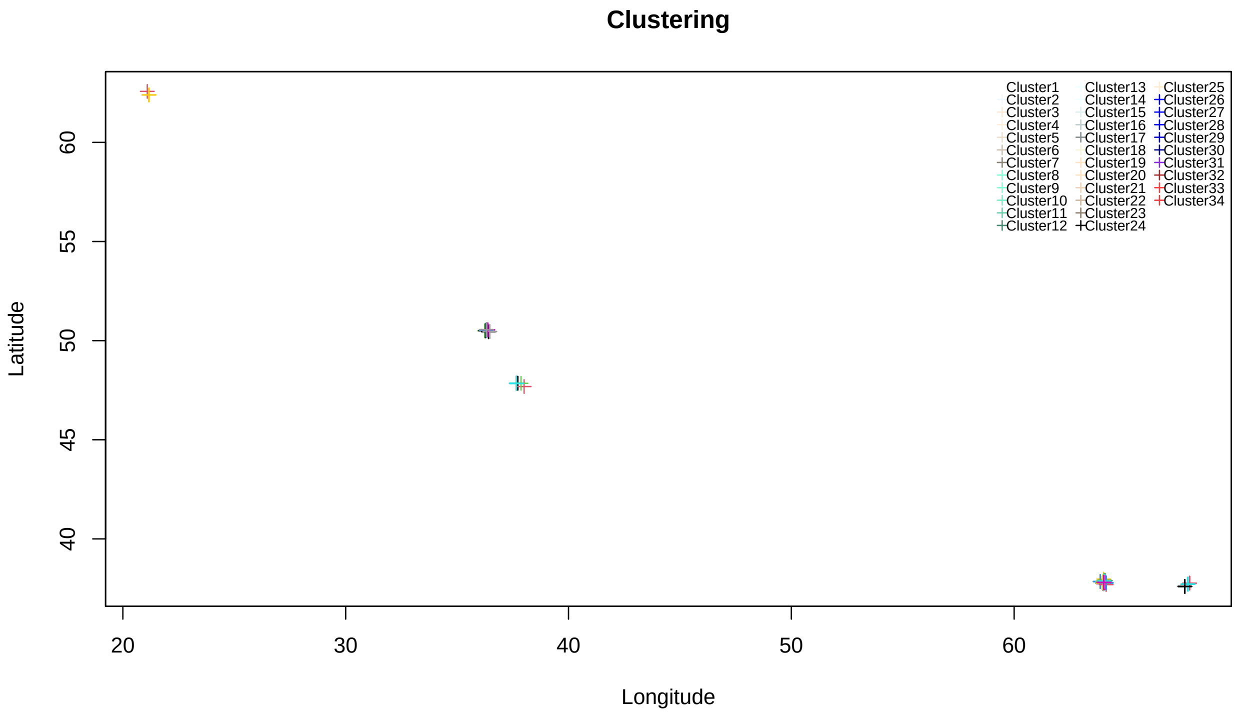
The file ps3-2.csv contains a data set with the coordinates in degrees (longitude/latitude) of the start and end points of the trips. Each row represents a trip. Use a clustering method in R to divide the start and end points into clusters respectively. The criterion used for clustering is that the maximum distance between the points in each cluster is less than 0.03. Treat a cluster of start points as an origin for a trip, and a cluster of end points as a destination for a trip. Construct an O-D matrix to indicate the number of trips between origins and destinations. Your report must include description of the approach used for clustering, the code, and the results.

```
EX2_Crude_Data <- read_csv("ps3-2.csv")

Start_Coord <- EX2_Crude_Data[, 1:2]
End_Coord <- EX2_Crude_Data[, 3:4]
```

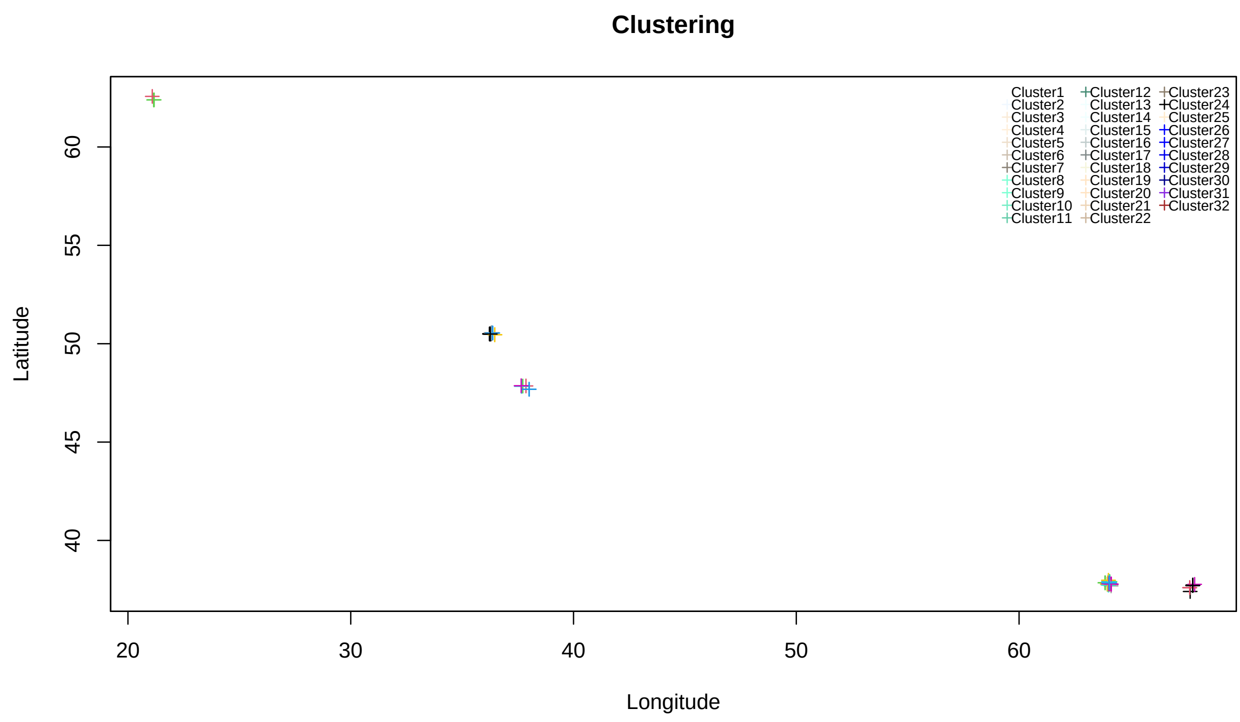
A distance of 1° converted to kilometer is equivalent to around 111.3 km. Consequently, the equivalent distance of 0.03° is ≈ 3.339, or ≈ 3339. The distGeo function calculates the Geodesic distances in meters.

The approach used to cluster the data involves calculating the geodesic distance between the coordinates and then applying a clustering function, cutting off the trees smaller than 3340 meters. This results that the points within the clusters are within that distance.

```
Start_Coord_Dist <- distm(cbind(Start_Coord$`Longitude - Start (deg)`,
                               Start_Coord$`Latitude - Start (deg)`),
                             fun = distGeo)

Start_Coord_Cluster <- hclust(as.dist(Start_Coord_Dist), method = "complete")
Start_Coord$Cluster <- cutree(Start_Coord_Cluster, h = 3340)

plot(x = Start_Coord$`Longitude - Start (deg)`,
     y = Start_Coord$`Latitude - Start (deg)`,
     col=factor(Start_Coord$Cluster), pch = 3,
        box(col="black"),
        main = "Clustering",
        xlab = "Longitude",
        ylab = "Latitude")
          legend("topright", legend=paste("Cluster", unique(Start_Coord$Cluster), sep=""),
                     col=grDevices::colors()[unique(Start_Coord$Cluster)],
                 pch=3, bg="white", bty = "n",
                 ncol = 3, cex = 0.7, y.intersp=0.7,x.intersp=0.3)
```




```
End_Coord_Dist <- distm(cbind(End_Coord$`Longitude - End (deg)`,
                              End_Coord$`Latitude - End (deg)`),
                        fun = distGeo)

End_Coord_Cluster <- hclust(as.dist(End_Coord_Dist), method = "complete")
End_Coord$Cluster <- cutree(End_Coord_Cluster, h = 3340)

plot(x = End_Coord$`Longitude - End (deg)`,
     y = End_Coord$`Latitude - End (deg)`,
     col=factor(End_Coord$Cluster), pch = 3,
        box(col="black"),
        main = "Clustering",
        xlab = "Longitude",
        ylab = "Latitude")
          legend("topright", legend=paste("Cluster", unique(End_Coord$Cluster), sep=""),
                      col=grDevices::colors()[unique(End_Coord$Cluster)],
                 pch=3, bg="white", bty = "n",
                 ncol = 3, cex = 0.7, y.intersp=0.7,x.intersp=0.3)
```




```r
Start_End_Movements <- data.frame(Origin = Start_Coord$Cluster, Destination = End_Coord$Cluster)
Start_End_Movements <- table(Start_End_Movements$Origin,
                             Start_End_Movements$Destination,
                             dnn = c("Origin", "Destination"))

print(Start_End_Movements)
```

```
##        Destination
## Origin  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
##      1  1  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##      2  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##      3  0  0  2  1  9  3  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
##      4  0  0  0 12 51  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##      5  0  0  6 23  9 80 37  0  0  3  0  5  0  4  0  0  0  0  0  0  0  0  4  0  0  0  0  0  1  0  1  0
##      6  0  0  0 23 51 16 34  0  0  1  0  3  0  1  0  0  0  0  0  0  4  0  5  1  0  0  0  0  0  0  0  0
##      7  0  0  0  7 22 32 13  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  2  0  0  0  1  0  1  0  0  0
##      8  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##      9  0  0  1  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     10  0  0  0  1  4  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  1  0
##     11  0  0  0  0  1  1  3  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     12  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
##     13  0  1  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     14  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0
##     15  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     16  1  0  0  0  0  0  0  7  0  0  0  0  0  0  2  0  5  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
##     17  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0
##     18  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     19  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     20  0  0  0  0  4  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     21  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0
##     22  0  0  0  1  2  3  2  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     23  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
##     24  0  0  0  1  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
##     25  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
##     26  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
##     27  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
##     28  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0
##     29  0  0  0  0  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     30  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
##     31  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     32  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     33  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##     34  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Problem 3

```r
Ex3_Crude_Data <- read_csv("ps3-1.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

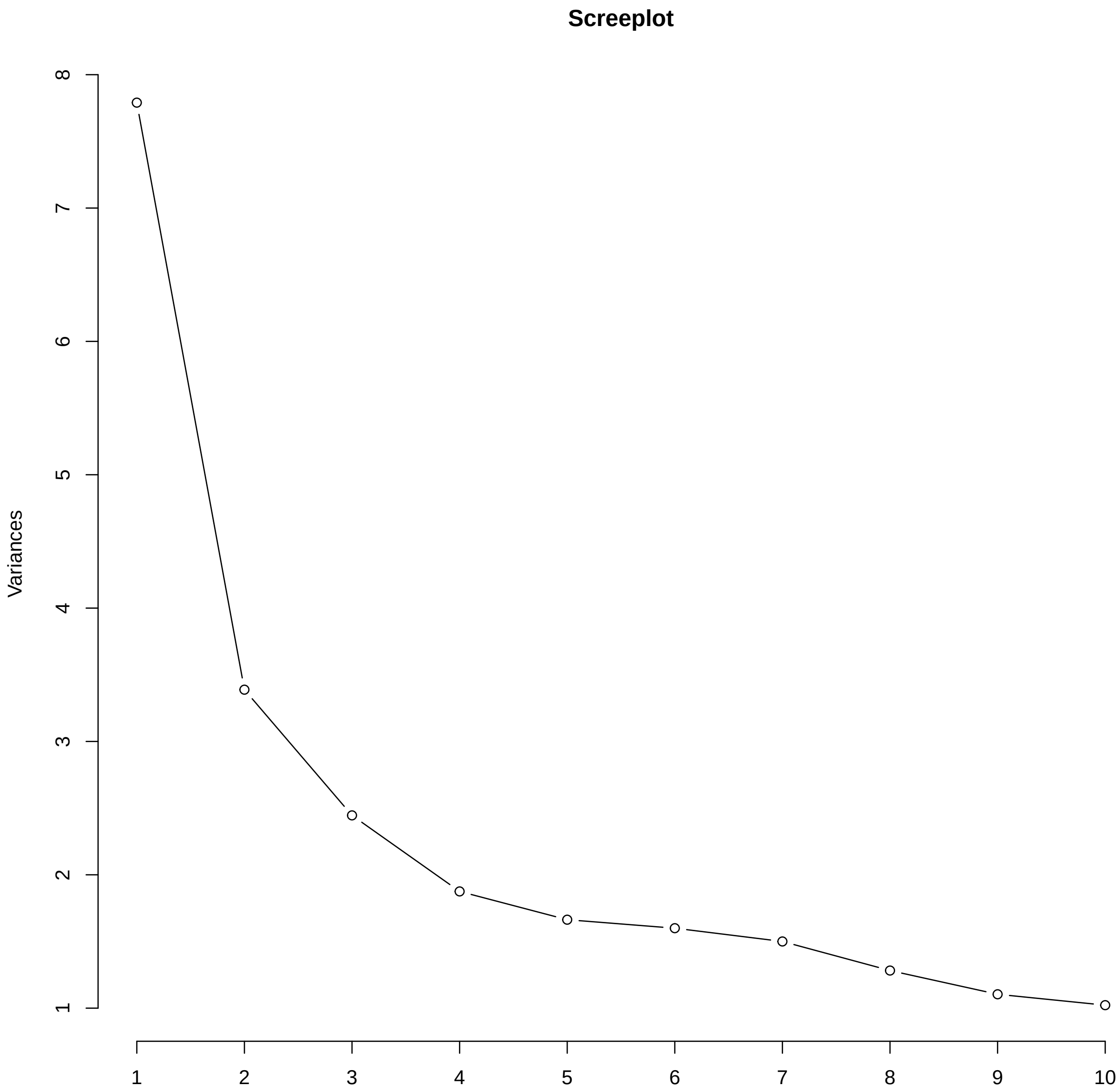
```r
Ex3_Crude_Data <- Ex3_Crude_Data[, -c(1, 36)] #Deleting the first and Y collumns
```

The PCA is performed with the prcomp function. The data is centered (making the mean of each variable zero) and scaled, so the variables also have an unitary variance. On summary of the PCA, it can be seen the cumulative percentage of the variable regarding the addition of components.

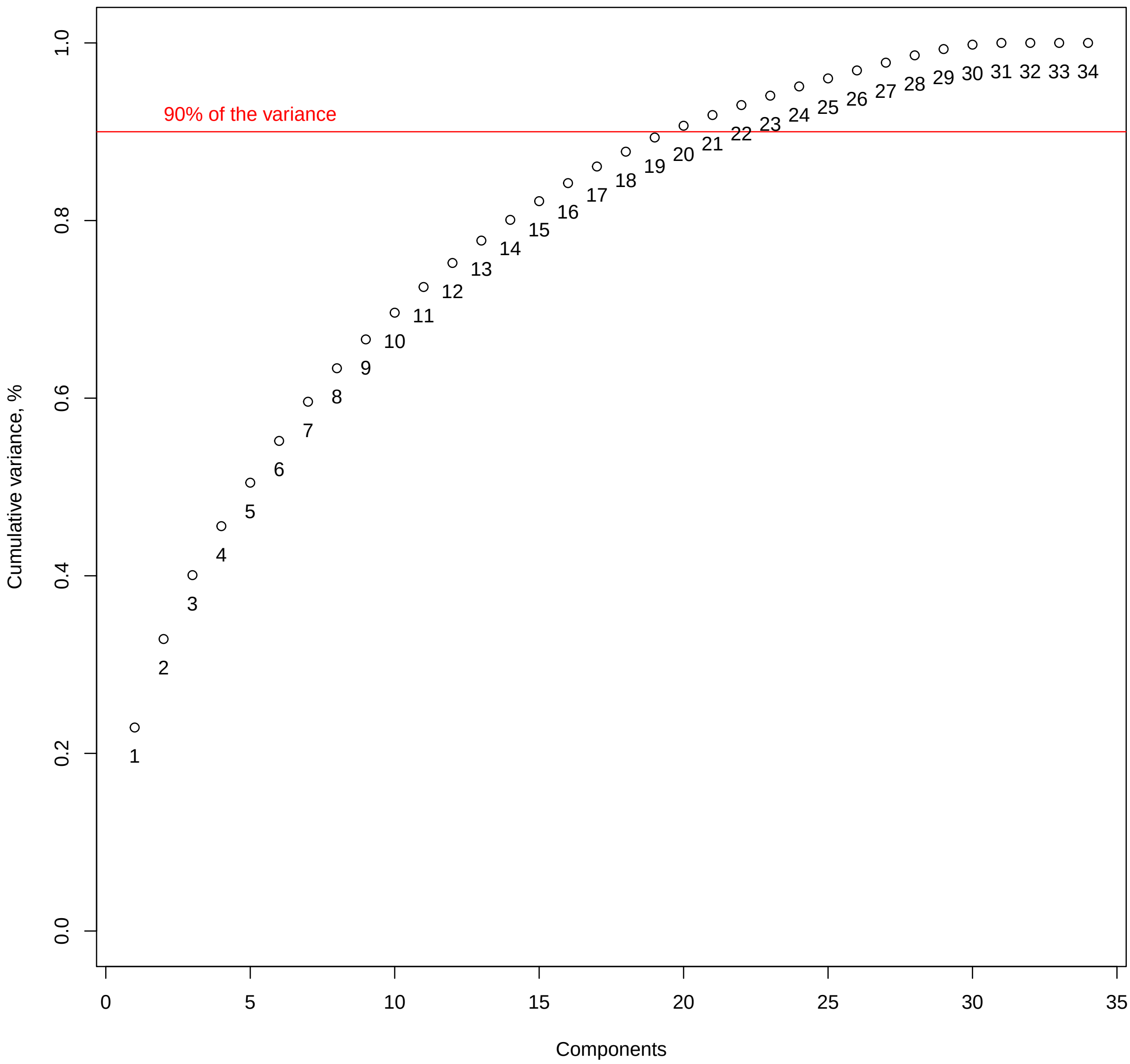
```r
Ex3_PCA <- prcomp(x = Ex3_Crude_Data, center = TRUE, scale. = TRUE)
summary(Ex3_PCA)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.7911 1.84073 1.56388 1.36944 1.28970 1.26477 1.22481
## Proportion of Variance 0.2291 0.09966 0.07193 0.05516 0.04892 0.04705 0.04412
## Cumulative Proportion  0.2291 0.32879 0.40072 0.45588 0.50480 0.55185 0.59597
##                            PC8     PC9    PC10    PC11    PC12   PC13    PC14
## Standard deviation     1.13228 1.05077 1.01114 0.99200 0.95909 0.9257 0.89089
## Proportion of Variance 0.03771 0.03247 0.03007 0.02894 0.02705 0.0252 0.02334
## Cumulative Proportion  0.63368 0.66615 0.69622 0.72517 0.75222 0.7774 0.80077
##                           PC15    PC16    PC17   PC18    PC19    PC20    PC21
## Standard deviation     0.84594 0.83170 0.79695 0.7536 0.73660 0.67116 0.64028
## Proportion of Variance 0.02105 0.02035 0.01868 0.0167 0.01596 0.01325 0.01206
## Cumulative Proportion  0.82181 0.84216 0.86084 0.8775 0.89350 0.90675 0.91880
##                           PC22    PC23   PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.61795 0.59932 0.5947 0.55405 0.55279 0.54587 0.52969
## Proportion of Variance 0.01123 0.01056 0.0104 0.00903 0.00899 0.00876 0.00825
## Cumulative Proportion  0.93004 0.94060 0.9510 0.96003 0.96902 0.97778 0.98604
##                           PC29    PC30    PC31     PC32      PC33      PC34
## Standard deviation     0.48965 0.41069 0.25759 0.003086 2.308e-05 3.686e-15
## Proportion of Variance 0.00705 0.00496 0.00195 0.000000 0.000e+00 0.000e+00
## Cumulative Proportion  0.99309 0.99805 1.00000 1.000000 1.000e+00 1.000e+00
```

```r
plot(Ex3_PCA, type = "l", main = "Screeplot")
```




```
plot(cumsum(Ex3_PCA$sdev^2 / sum(Ex3_PCA$sdev^2)), ylim=0:1,
     ylab = "Cumulative variance, %", xlab = "Components")
abline(h = 0.9, col = "red")
text(y = cumsum(Ex3_PCA$sdev^2 / sum(Ex3_PCA$sdev^2)),x= 1:34, labels = 1:34,
     pos = 1, offset = 1)
text(y = .92, x = 5, "90% of the variance", col = "red")
```



Looking at the second plot, it can be seen that the 90% threshold of the variance is achieved with the 21st principal component. On the other hand, for a more *ad hoc* approach, James et al. (2013) describes the elbow method. According to the authors, "This is done by eyeballing the scree plot, and looking for a point at which the proportion of variance explained by each subsequent principal component drops off". This would be achieved after the 5th or 6th principal component.

References

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. 10. Springer series in statistics New York.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.