library(ggplot2)  
library(caret)  
library(rpart)  
library(rpart.plot)  
library(tree)  
library(e1071)  
library(kernlab)   
library(neuralnet)  
library(NeuralNetTools)  
library(ROCR)  
library(pROC)

G.data<-read.csv("/Users/GERMAN\_DATA.csv",  
 header=TRUE)  
str(G.data)

## 'data.frame': 1000 obs. of 21 variables:  
## $ CHK\_ACCT\_ST : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...  
## $ DUR : int 6 48 12 42 24 36 24 36 12 30 ...  
## $ CRED\_HIST : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...  
## $ PURPOSE : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...  
## $ CRED\_AMT : int 1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...  
## $ SAV\_ACCT\_BOND : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...  
## $ EMPLYMT\_ST : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...  
## $ INST\_RT\_PER\_DISP\_INCM: int 4 2 2 2 3 2 3 2 2 4 ...  
## $ PERS\_ST\_SEX : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...  
## $ COAPP\_GURNTR : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...  
## $ DUR\_RES : int 4 2 3 4 4 4 4 2 4 2 ...  
## $ PROPERTY : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...  
## $ AGE : int 67 22 49 45 53 35 53 35 61 28 ...  
## $ OTHR\_INSTL : Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ HOUS\_ST : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...  
## $ NUM\_CRED : int 2 1 1 1 2 1 1 1 1 2 ...  
## $ JOB : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...  
## $ NUM\_PEOP\_LIABL : int 1 1 2 2 2 2 1 1 1 1 ...  
## $ PHONE : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...  
## $ FRGN\_WORKR : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...  
## $ Y : int 1 2 1 1 2 1 1 1 1 2 ...

summary(G.data)

## CHK\_ACCT\_ST DUR CRED\_HIST PURPOSE CRED\_AMT   
## A11:274 Min. : 4.0 A30: 40 A43 :280 Min. : 250   
## A12:269 1st Qu.:12.0 A31: 49 A40 :234 1st Qu.: 1366   
## A13: 63 Median :18.0 A32:530 A42 :181 Median : 2320   
## A14:394 Mean :20.9 A33: 88 A41 :103 Mean : 3271   
## 3rd Qu.:24.0 A34:293 A49 : 97 3rd Qu.: 3972   
## Max. :72.0 A46 : 50 Max. :18424   
## (Other): 55   
## SAV\_ACCT\_BOND EMPLYMT\_ST INST\_RT\_PER\_DISP\_INCM PERS\_ST\_SEX COAPP\_GURNTR  
## A61:603 A71: 62 Min. :1.000 A91: 50 A101:907   
## A62:103 A72:172 1st Qu.:2.000 A92:310 A102: 41   
## A63: 63 A73:339 Median :3.000 A93:548 A103: 52   
## A64: 48 A74:174 Mean :2.973 A94: 92   
## A65:183 A75:253 3rd Qu.:4.000   
## Max. :4.000   
##   
## DUR\_RES PROPERTY AGE OTHR\_INSTL HOUS\_ST   
## Min. :1.000 A121:282 Min. :19.00 A141:139 A151:179   
## 1st Qu.:2.000 A122:232 1st Qu.:27.00 A142: 47 A152:713   
## Median :3.000 A123:332 Median :33.00 A143:814 A153:108   
## Mean :2.845 A124:154 Mean :35.55   
## 3rd Qu.:4.000 3rd Qu.:42.00   
## Max. :4.000 Max. :75.00   
##   
## NUM\_CRED JOB NUM\_PEOP\_LIABL PHONE FRGN\_WORKR Y   
## Min. :1.000 A171: 22 Min. :1.000 A191:596 A201:963 Min. :1.0   
## 1st Qu.:1.000 A172:200 1st Qu.:1.000 A192:404 A202: 37 1st Qu.:1.0   
## Median :1.000 A173:630 Median :1.000 Median :1.0   
## Mean :1.407 A174:148 Mean :1.155 Mean :1.3   
## 3rd Qu.:2.000 3rd Qu.:1.000 3rd Qu.:2.0   
## Max. :4.000 Max. :2.000 Max. :2.0   
##

G.data$DUR<-as.numeric(G.data$DUR)  
G.data$CRED\_AMT<-as.numeric(G.data$CRED\_AMT)  
G.data$INST\_RT\_PER\_DISP\_INCM<-as.numeric(G.data$INST\_RT\_PER\_DISP\_INCM)  
G.data$DUR\_RES<-as.numeric(G.data$DUR\_RES)  
G.data$AGE<-as.numeric(G.data$AGE)  
G.data$NUM\_CRED<-as.numeric(G.data$NUM\_CRED)  
G.data$NUM\_PEOP\_LIABL<-as.numeric(G.data$NUM\_PEOP\_LIABL)  
G.data$Y<-as.factor(G.data$Y)

sapply(G.data, function(x) sum(is.na(x)))

## CHK\_ACCT\_ST DUR CRED\_HIST   
## 0 0 0   
## PURPOSE CRED\_AMT SAV\_ACCT\_BOND   
## 0 0 0   
## EMPLYMT\_ST INST\_RT\_PER\_DISP\_INCM PERS\_ST\_SEX   
## 0 0 0   
## COAPP\_GURNTR DUR\_RES PROPERTY   
## 0 0 0   
## AGE OTHR\_INSTL HOUS\_ST   
## 0 0 0   
## NUM\_CRED JOB NUM\_PEOP\_LIABL   
## 0 0 0   
## PHONE FRGN\_WORKR Y   
## 0 0 0

# split data into training and test sets  
set.seed(800)   
index <- 1:nrow(G.data)  
test\_set\_index <- sample(index, trunc(length(index)/3))  
test\_set <- G.data[test\_set\_index,]  
train\_set <- G.data[-test\_set\_index,]  
train\_set1 <- G.data[-test\_set\_index,]  
  
# determine the max/min from the training set  
d\_max <- sapply(train\_set[,c(2,5,8,11,13,16,18)], max)  
d\_min <- sapply(train\_set[,c(2,5,8,11,13,16,18)], min)  
  
# normalize the data to [0,1] use rescale function of scales package  
  
# function for rescale the columns based on the training set max/min   
rescale <- function(dat, d\_min, d\_max) {  
 c <- ncol(dat)  
 for (i in 1:c) {  
 dat[,i] <- sapply(dat[,i], function(x) (x - d\_min[i])/(d\_max[i] - d\_min[i]))  
 }  
 return (dat)  
}  
  
# normalize the training/testing set  
train\_set[,c(2,5,8,11,13,16,18)] <- rescale(train\_set[,c(2,5,8,11,13,16,18)], d\_min, d\_max)  
test\_set[,c(2,5,8,11,13,16,18)] <- rescale(test\_set[,c(2,5,8,11,13,16,18)], d\_min, d\_max)

# Logistic Regression Model  
log\_fit1 <- train(Y~., data=train\_set, method="glm", family="binomial")  
  
summary(log\_fit1)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1721 -0.6954 -0.3621 0.7162 2.6803   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.54252 1.28725 0.421 0.673423   
## CHK\_ACCT\_STA12 -0.56660 0.27273 -2.077 0.037756 \*   
## CHK\_ACCT\_STA13 -1.18573 0.47393 -2.502 0.012352 \*   
## CHK\_ACCT\_STA14 -1.87698 0.29384 -6.388 1.68e-10 \*\*\*  
## DUR 1.55806 0.77860 2.001 0.045381 \*   
## CRED\_HISTA31 0.15717 0.68800 0.228 0.819298   
## CRED\_HISTA32 -0.54026 0.54328 -0.994 0.320010   
## CRED\_HISTA33 -0.62625 0.57995 -1.080 0.280220   
## CRED\_HISTA34 -1.22520 0.53807 -2.277 0.022785 \*   
## PURPOSEA41 -1.64968 0.47894 -3.444 0.000572 \*\*\*  
## PURPOSEA410 -0.68814 0.81065 -0.849 0.395951   
## PURPOSEA42 -1.00982 0.33602 -3.005 0.002654 \*\*   
## PURPOSEA43 -0.89454 0.30958 -2.890 0.003858 \*\*   
## PURPOSEA44 -0.24180 0.76841 -0.315 0.753008   
## PURPOSEA45 -0.17377 0.67317 -0.258 0.796301   
## PURPOSEA46 0.27010 0.49799 0.542 0.587554   
## PURPOSEA48 -2.32235 1.33729 -1.737 0.082457 .   
## PURPOSEA49 -0.58741 0.40182 -1.462 0.143771   
## CRED\_AMT 1.76120 0.96808 1.819 0.068869 .   
## SAV\_ACCT\_BONDA62 -0.64374 0.36922 -1.744 0.081244 .   
## SAV\_ACCT\_BONDA63 -1.11515 0.56701 -1.967 0.049217 \*   
## SAV\_ACCT\_BONDA64 -1.32455 0.58407 -2.268 0.023342 \*   
## SAV\_ACCT\_BONDA65 -1.17164 0.32713 -3.582 0.000342 \*\*\*  
## EMPLYMT\_STA72 -0.18383 0.54978 -0.334 0.738096   
## EMPLYMT\_STA73 -0.29918 0.53181 -0.563 0.573731   
## EMPLYMT\_STA74 -0.56412 0.57329 -0.984 0.325116   
## EMPLYMT\_STA75 0.06347 0.53331 0.119 0.905265   
## INST\_RT\_PER\_DISP\_INCM 0.53868 0.33081 1.628 0.103441   
## PERS\_ST\_SEXA92 -0.42360 0.47689 -0.888 0.374402   
## PERS\_ST\_SEXA93 -0.83222 0.47485 -1.753 0.079667 .   
## PERS\_ST\_SEXA94 0.10197 0.56205 0.181 0.856030   
## COAPP\_GURNTRA102 0.01857 0.51516 0.036 0.971252   
## COAPP\_GURNTRA103 -0.86363 0.49838 -1.733 0.083120 .   
## DUR\_RES 0.13028 0.32519 0.401 0.688704   
## PROPERTYA122 0.48987 0.31961 1.533 0.125341   
## PROPERTYA123 0.39319 0.29648 1.326 0.184780   
## PROPERTYA124 1.20818 0.56079 2.154 0.031205 \*   
## AGE -1.55415 0.63953 -2.430 0.015093 \*   
## OTHR\_INSTLA142 0.54688 0.52356 1.045 0.296239   
## OTHR\_INSTLA143 -0.25168 0.29864 -0.843 0.399380   
## HOUS\_STA152 -0.30417 0.29647 -1.026 0.304903   
## HOUS\_STA153 -0.83819 0.62275 -1.346 0.178323   
## NUM\_CRED 0.30724 0.80317 0.383 0.702063   
## JOBA172 1.14655 0.95647 1.199 0.230636   
## JOBA173 1.18793 0.93054 1.277 0.201742   
## JOBA174 1.46394 0.92334 1.585 0.112854   
## NUM\_PEOP\_LIABL 0.45206 0.31389 1.440 0.149810   
## PHONEA192 -0.31518 0.24771 -1.272 0.203239   
## FRGN\_WORKRA202 -1.00380 0.69100 -1.453 0.146315   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 813.02 on 666 degrees of freedom  
## Residual deviance: 592.12 on 618 degrees of freedom  
## AIC: 690.12  
##   
## Number of Fisher Scoring iterations: 5

# Prediction  
pred<-predict(log\_fit1, newdata=test\_set)  
  
confusionMatrix(pred,test\_set$Y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 203 54  
## 2 29 47  
##   
## Accuracy : 0.7508   
## 95% CI : (0.7007, 0.7963)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.01719   
##   
## Kappa : 0.3659   
##   
## Mcnemar's Test P-Value : 0.00843   
##   
## Sensitivity : 0.8750   
## Specificity : 0.4653   
## Pos Pred Value : 0.7899   
## Neg Pred Value : 0.6184   
## Prevalence : 0.6967   
## Detection Rate : 0.6096   
## Detection Prevalence : 0.7718   
## Balanced Accuracy : 0.6702   
##   
## 'Positive' Class : 1   
##

# 10-Fold Cross Validation:  
ctrl <- trainControl(method = "repeatedcv", number = 10, savePredictions = TRUE)  
  
log\_fit2 <- train(Y~.,data=train\_set, method="glm", family="binomial",  
 trControl = ctrl, tuneLength = 5)  
  
summary(log\_fit2)

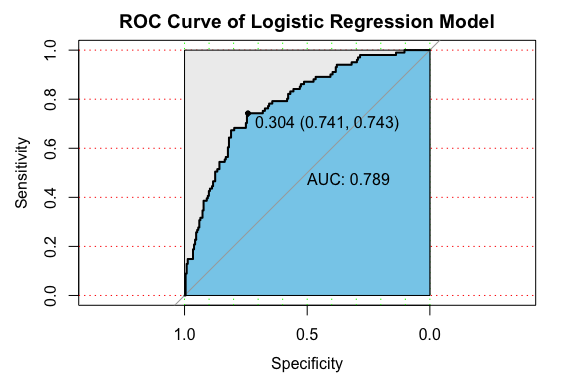
##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1721 -0.6954 -0.3621 0.7162 2.6803   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 0.54252 1.28725 0.421 0.673423   
## CHK\_ACCT\_STA12 -0.56660 0.27273 -2.077 0.037756 \*   
## CHK\_ACCT\_STA13 -1.18573 0.47393 -2.502 0.012352 \*   
## CHK\_ACCT\_STA14 -1.87698 0.29384 -6.388 1.68e-10 \*\*\*  
## DUR 1.55806 0.77860 2.001 0.045381 \*   
## CRED\_HISTA31 0.15717 0.68800 0.228 0.819298   
## CRED\_HISTA32 -0.54026 0.54328 -0.994 0.320010   
## CRED\_HISTA33 -0.62625 0.57995 -1.080 0.280220   
## CRED\_HISTA34 -1.22520 0.53807 -2.277 0.022785 \*   
## PURPOSEA41 -1.64968 0.47894 -3.444 0.000572 \*\*\*  
## PURPOSEA410 -0.68814 0.81065 -0.849 0.395951   
## PURPOSEA42 -1.00982 0.33602 -3.005 0.002654 \*\*   
## PURPOSEA43 -0.89454 0.30958 -2.890 0.003858 \*\*   
## PURPOSEA44 -0.24180 0.76841 -0.315 0.753008   
## PURPOSEA45 -0.17377 0.67317 -0.258 0.796301   
## PURPOSEA46 0.27010 0.49799 0.542 0.587554   
## PURPOSEA48 -2.32235 1.33729 -1.737 0.082457 .   
## PURPOSEA49 -0.58741 0.40182 -1.462 0.143771   
## CRED\_AMT 1.76120 0.96808 1.819 0.068869 .   
## SAV\_ACCT\_BONDA62 -0.64374 0.36922 -1.744 0.081244 .   
## SAV\_ACCT\_BONDA63 -1.11515 0.56701 -1.967 0.049217 \*   
## SAV\_ACCT\_BONDA64 -1.32455 0.58407 -2.268 0.023342 \*   
## SAV\_ACCT\_BONDA65 -1.17164 0.32713 -3.582 0.000342 \*\*\*  
## EMPLYMT\_STA72 -0.18383 0.54978 -0.334 0.738096   
## EMPLYMT\_STA73 -0.29918 0.53181 -0.563 0.573731   
## EMPLYMT\_STA74 -0.56412 0.57329 -0.984 0.325116   
## EMPLYMT\_STA75 0.06347 0.53331 0.119 0.905265   
## INST\_RT\_PER\_DISP\_INCM 0.53868 0.33081 1.628 0.103441   
## PERS\_ST\_SEXA92 -0.42360 0.47689 -0.888 0.374402   
## PERS\_ST\_SEXA93 -0.83222 0.47485 -1.753 0.079667 .   
## PERS\_ST\_SEXA94 0.10197 0.56205 0.181 0.856030   
## COAPP\_GURNTRA102 0.01857 0.51516 0.036 0.971252   
## COAPP\_GURNTRA103 -0.86363 0.49838 -1.733 0.083120 .   
## DUR\_RES 0.13028 0.32519 0.401 0.688704   
## PROPERTYA122 0.48987 0.31961 1.533 0.125341   
## PROPERTYA123 0.39319 0.29648 1.326 0.184780   
## PROPERTYA124 1.20818 0.56079 2.154 0.031205 \*   
## AGE -1.55415 0.63953 -2.430 0.015093 \*   
## OTHR\_INSTLA142 0.54688 0.52356 1.045 0.296239   
## OTHR\_INSTLA143 -0.25168 0.29864 -0.843 0.399380   
## HOUS\_STA152 -0.30417 0.29647 -1.026 0.304903   
## HOUS\_STA153 -0.83819 0.62275 -1.346 0.178323   
## NUM\_CRED 0.30724 0.80317 0.383 0.702063   
## JOBA172 1.14655 0.95647 1.199 0.230636   
## JOBA173 1.18793 0.93054 1.277 0.201742   
## JOBA174 1.46394 0.92334 1.585 0.112854   
## NUM\_PEOP\_LIABL 0.45206 0.31389 1.440 0.149810   
## PHONEA192 -0.31518 0.24771 -1.272 0.203239   
## FRGN\_WORKRA202 -1.00380 0.69100 -1.453 0.146315   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 813.02 on 666 degrees of freedom  
## Residual deviance: 592.12 on 618 degrees of freedom  
## AIC: 690.12  
##   
## Number of Fisher Scoring iterations: 5

# Prediction  
pred <- predict(log\_fit2, newdata=test\_set)  
confusionMatrix(data=pred, test\_set$Y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 203 54  
## 2 29 47  
##   
## Accuracy : 0.7508   
## 95% CI : (0.7007, 0.7963)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.01719   
##   
## Kappa : 0.3659   
##   
## Mcnemar's Test P-Value : 0.00843   
##   
## Sensitivity : 0.8750   
## Specificity : 0.4653   
## Pos Pred Value : 0.7899   
## Neg Pred Value : 0.6184   
## Prevalence : 0.6967   
## Detection Rate : 0.6096   
## Detection Prevalence : 0.7718   
## Balanced Accuracy : 0.6702   
##   
## 'Positive' Class : 1   
##

t1<-table(data=pred, test\_set$Y)  
log\_acc<-(t1[1,1]+t1[2,2])/(t1[1,1]+t1[2,2]+t1[1,2]+t1[2,1])  
log\_spec<-t1[2,2]/(t1[1,2]+t1[2,2])

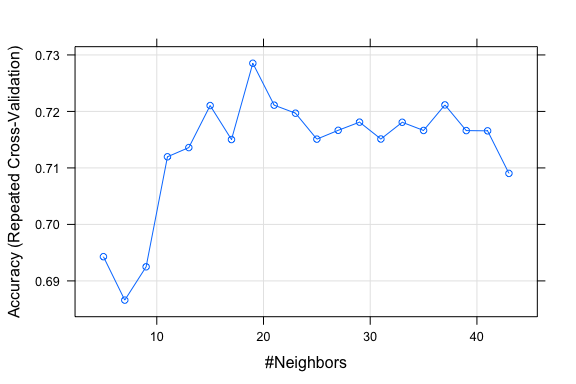
# Logistic ROC curve  
log.predd <- predict(log\_fit2, type='prob',test\_set, probability = TRUE)  
  
modelroc <- roc(test\_set$Y,log.predd[,2])  
plot(modelroc, type="S",print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),  
 grid.col=c("green", "red"), max.auc.polygon=TRUE,  
 auc.polygon.col="skyblue", print.thres=TRUE, main="ROC Curve of Logistic Regression Model")



# kNN  
ctrl <- trainControl(method = "repeatedcv", number = 10, savePredictions = TRUE)  
  
knn\_fit <- train(Y ~ ., data = train\_set, method = "knn",   
 trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)  
  
# Output of kNN fit  
knn\_fit

## k-Nearest Neighbors   
##   
## 667 samples  
## 20 predictor  
## 2 classes: '1', '2'   
##   
## Pre-processing: centered (48), scaled (48)   
## Resampling: Cross-Validated (10 fold, repeated 1 times)   
## Summary of sample sizes: 601, 600, 600, 600, 600, 600, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.6942887 0.1876046  
## 7 0.6865971 0.1494744  
## 9 0.6924980 0.1543284  
## 11 0.7119702 0.1976489  
## 13 0.7135998 0.1806212  
## 15 0.7210399 0.1942151  
## 17 0.7150238 0.1781192  
## 19 0.7285259 0.2134942  
## 21 0.7211084 0.1985490  
## 23 0.7196851 0.1940685  
## 25 0.7150931 0.1623568  
## 27 0.7166541 0.1432209  
## 29 0.7181015 0.1519827  
## 31 0.7151164 0.1303891  
## 33 0.7180781 0.1392873  
## 35 0.7166315 0.1224959  
## 37 0.7211544 0.1413605  
## 39 0.7166082 0.1155436  
## 41 0.7165623 0.1113073  
## 43 0.7090311 0.0833919  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 19.

# Plotting yields Number of Neighbours Vs accuracy (based on repeated cross validation)  
plot(knn\_fit)

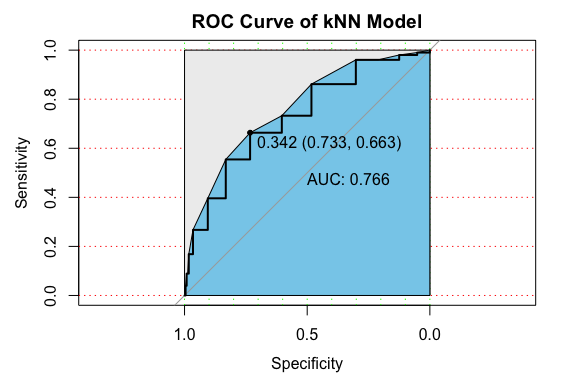


knnPredict <- predict(knn\_fit,newdata = test\_set )  
#Get the confusion matrix to see accuracy value and other parameter values  
confusionMatrix(knnPredict, test\_set$Y )

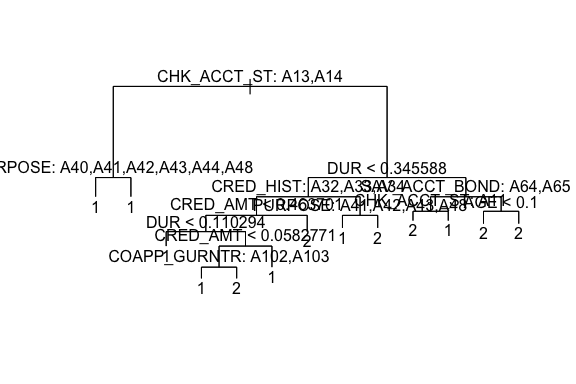
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 224 74  
## 2 8 27  
##   
## Accuracy : 0.7538   
## 95% CI : (0.7038, 0.7991)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.01256   
##   
## Kappa : 0.2855   
##   
## Mcnemar's Test P-Value : 7.071e-13   
##   
## Sensitivity : 0.9655   
## Specificity : 0.2673   
## Pos Pred Value : 0.7517   
## Neg Pred Value : 0.7714   
## Prevalence : 0.6967   
## Detection Rate : 0.6727   
## Detection Prevalence : 0.8949   
## Balanced Accuracy : 0.6164   
##   
## 'Positive' Class : 1   
##

t1<-table(knnPredict, test\_set$Y)  
knn\_acc<-(t1[1,1]+t1[2,2])/(t1[1,1]+t1[2,2]+t1[1,2]+t1[2,1])  
knn\_spec<-t1[2,2]/(t1[1,2]+t1[2,2])

# kNN ROC curve  
knn.predd <- predict(knn\_fit, type='prob',test\_set, probability = TRUE)  
  
modelroc <- roc(test\_set$Y,knn.predd[,2])  
plot(modelroc, type="S",print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),  
 grid.col=c("green", "red"), max.auc.polygon=TRUE,  
 auc.polygon.col="skyblue", print.thres=TRUE, main="ROC Curve of kNN Model")



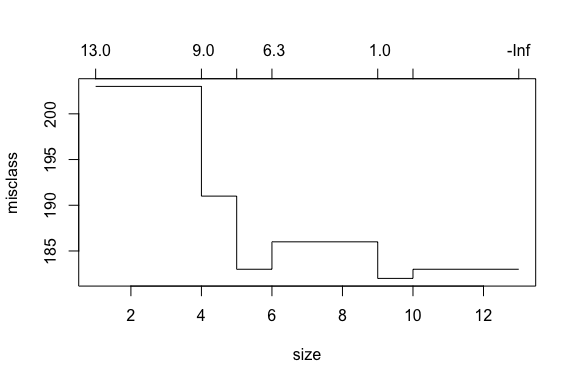
# Decision Tree   
  
trees <- tree(Y~., train\_set)  
plot(trees)  
text(trees, pretty=0)



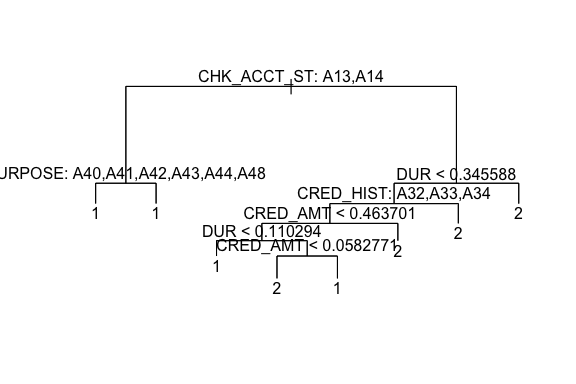
# Prediction and confusion matrix  
treesPredict <- predict(trees,newdata = test\_set , type="class")  
confusionMatrix(treesPredict, test\_set$Y )

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 203 58  
## 2 29 43  
##   
## Accuracy : 0.7387   
## 95% CI : (0.6881, 0.7851)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.052290   
##   
## Kappa : 0.3273   
##   
## Mcnemar's Test P-Value : 0.002683   
##   
## Sensitivity : 0.8750   
## Specificity : 0.4257   
## Pos Pred Value : 0.7778   
## Neg Pred Value : 0.5972   
## Prevalence : 0.6967   
## Detection Rate : 0.6096   
## Detection Prevalence : 0.7838   
## Balanced Accuracy : 0.6504   
##   
## 'Positive' Class : 1   
##

# Cross validation and plot the tree  
cv.trees <- cv.tree(trees, FUN = prune.misclass)  
plot(cv.trees)



prune.trees <- prune.tree(trees, best=6)  
plot(prune.trees)  
text(prune.trees, pretty=0)

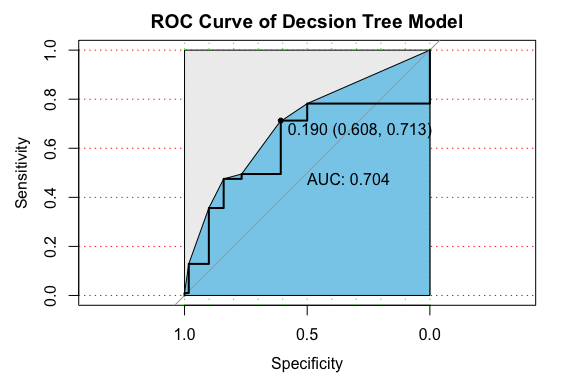


prune.treesPredict <- predict(prune.trees,newdata = test\_set , type="class")  
confusionMatrix(prune.treesPredict, test\_set$Y )

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 195 53  
## 2 37 48  
##   
## Accuracy : 0.7297   
## 95% CI : (0.6786, 0.7767)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.1045   
##   
## Kappa : 0.3305   
##   
## Mcnemar's Test P-Value : 0.1138   
##   
## Sensitivity : 0.8405   
## Specificity : 0.4752   
## Pos Pred Value : 0.7863   
## Neg Pred Value : 0.5647   
## Prevalence : 0.6967   
## Detection Rate : 0.5856   
## Detection Prevalence : 0.7447   
## Balanced Accuracy : 0.6579   
##   
## 'Positive' Class : 1   
##

t1<-table(prune.treesPredict, test\_set$Y)  
dt\_acc<-(t1[1,1]+t1[2,2])/(t1[1,1]+t1[2,2]+t1[1,2]+t1[2,1])  
dt\_spec<-t1[2,2]/(t1[1,2]+t1[2,2])

# Decision Tree ROC curve  
dt.predd <- predict(prune.trees,newdata = test\_set)  
  
modelroc <- roc(test\_set$Y,dt.predd[,2])  
plot(modelroc, type="S",print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),  
 grid.col=c("green", "red"), max.auc.polygon=TRUE,  
 auc.polygon.col="skyblue", print.thres=TRUE, main="ROC Curve of Decsion Tree Model")



# Naive Bayes  
ctrl <- trainControl(method = "repeatedcv", number = 10, savePredictions = TRUE)  
nb\_fit = train(train\_set[,1:20],train\_set[,21],'nb',  
 trControl=ctrl)  
nb\_fit

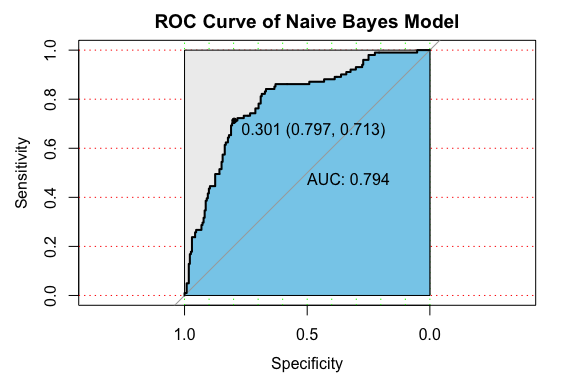
## Naive Bayes   
##   
## 667 samples  
## 20 predictor  
## 2 classes: '1', '2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 1 times)   
## Summary of sample sizes: 600, 601, 600, 600, 600, 601, ...   
## Resampling results across tuning parameters:  
##   
## usekernel Accuracy Kappa   
## FALSE 0.7513116 0.3792598  
## TRUE 0.7316373 0.2645873  
##   
## Tuning parameter 'fL' was held constant at a value of 0  
## Tuning  
## parameter 'adjust' was held constant at a value of 1  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were fL = 0, usekernel = FALSE and adjust  
## = 1.

nbPredict <- predict(nb\_fit$finalModel,newdata = test\_set[,1:20] )$class  
#Get the confusion matrix to see accuracy value and other parameter values  
confusionMatrix(nbPredict, test\_set$Y )

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 203 55  
## 2 29 46  
##   
## Accuracy : 0.7477   
## 95% CI : (0.6975, 0.7935)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.023181   
##   
## Kappa : 0.3563   
##   
## Mcnemar's Test P-Value : 0.006377   
##   
## Sensitivity : 0.8750   
## Specificity : 0.4554   
## Pos Pred Value : 0.7868   
## Neg Pred Value : 0.6133   
## Prevalence : 0.6967   
## Detection Rate : 0.6096   
## Detection Prevalence : 0.7748   
## Balanced Accuracy : 0.6652   
##   
## 'Positive' Class : 1   
##

t1<-table(nbPredict, test\_set$Y)  
nb\_acc<-(t1[1,1]+t1[2,2])/(t1[1,1]+t1[2,2]+t1[1,2]+t1[2,1])  
nb\_spec<-t1[2,2]/(t1[1,2]+t1[2,2])

# Naive Bayes ROC curve  
nb.predd <- predict(nb\_fit$finalModel, type='prob',test\_set, probability = TRUE)  
  
modelroc <- roc(test\_set$Y,nb.predd$posterior[,2])  
plot(modelroc, type="S",print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),  
 grid.col=c("green", "red"), max.auc.polygon=TRUE,  
 auc.polygon.col="skyblue", print.thres=TRUE, main="ROC Curve of Naive Bayes Model")



# Support Vector Machine  
  
tc <- tune.control(cross = 10)  
tune.out <- tune(svm, Y~.,  
 data = train\_set, kernel = "radial",  
 ranges = list(cost = 10^(-1:2),  
 gamma = c(0.25,0.5,1,2,5)),  
 tunecontrol = tc)  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 1 0.25  
##   
## - best performance: 0.2638851   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.1 0.25 0.2983265 0.04829376  
## 2 1.0 0.25 0.2638851 0.03877373  
## 3 10.0 0.25 0.2743329 0.03580819  
## 4 100.0 0.25 0.2743329 0.03580819  
## 5 0.1 0.50 0.2983265 0.04829376  
## 6 1.0 0.50 0.2968114 0.04941090  
## 7 10.0 0.50 0.2922886 0.04187982  
## 8 100.0 0.50 0.2922886 0.04187982  
## 9 0.1 1.00 0.2983265 0.04829376  
## 10 1.0 1.00 0.2983265 0.04829376  
## 11 10.0 1.00 0.2953189 0.04805805  
## 12 100.0 1.00 0.2953189 0.04805805  
## 13 0.1 2.00 0.2983265 0.04829376  
## 14 1.0 2.00 0.2983265 0.04829376  
## 15 10.0 2.00 0.2983265 0.04829376  
## 16 100.0 2.00 0.2983265 0.04829376  
## 17 0.1 5.00 0.2983265 0.04829376  
## 18 1.0 5.00 0.2983265 0.04829376  
## 19 10.0 5.00 0.2983265 0.04829376  
## 20 100.0 5.00 0.2983265 0.04829376

print(tune.out) # best parameters: cost=1, gamma=0.25

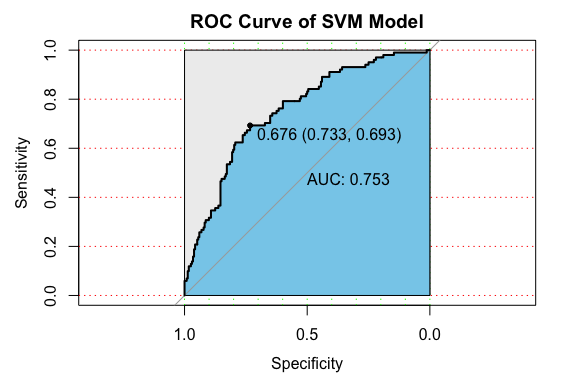
##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 1 0.25  
##   
## - best performance: 0.2638851

svm.prediction = predict(tune.out$best.model,newdata=test\_set,type='class')  
confusionMatrix(svm.prediction,as.factor(test\_set$Y))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 228 89  
## 2 4 12  
##   
## Accuracy : 0.7207   
## 95% CI : (0.6692, 0.7683)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.1861   
##   
## Kappa : 0.1332   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.9828   
## Specificity : 0.1188   
## Pos Pred Value : 0.7192   
## Neg Pred Value : 0.7500   
## Prevalence : 0.6967   
## Detection Rate : 0.6847   
## Detection Prevalence : 0.9520   
## Balanced Accuracy : 0.5508   
##   
## 'Positive' Class : 1   
##

t1<-table(svm.prediction,as.factor(test\_set$Y))  
svm\_acc<-(t1[1,1]+t1[2,2])/(t1[1,1]+t1[2,2]+t1[1,2]+t1[2,1])  
svm\_spec<-t1[2,2]/(t1[1,2]+t1[2,2])

# SVM ROC curve  
svm\_fit2 <- svm(Y~., data =train\_set, cost=1, gamma=0.25, probability = TRUE)  
  
svm.predd <- predict(svm\_fit2, type='prob',test\_set, probability = TRUE)  
  
modelroc <- roc(test\_set$Y,attr(svm.predd, "probabilities")[,2])  
plot(modelroc, type="S",print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),  
 grid.col=c("green", "red"), max.auc.polygon=TRUE,  
 auc.polygon.col="skyblue", print.thres=TRUE, main="ROC Curve of SVM Model")



# Neural Network  
ctrl <- trainControl(method = "repeatedcv", number = 10, savePredictions = TRUE)  
  
nn\_fit <- train(Y~., data = train\_set,   
 method = 'nnet', preProcess = c('center', 'scale'), trControl = ctrl,  
 tuneGrid=expand.grid(size=c(10), decay=c(0.1)))

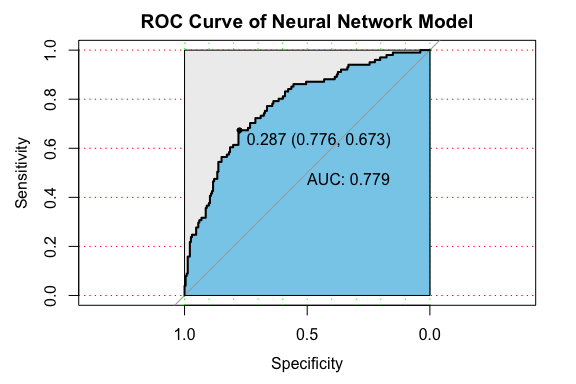
## # weights: 501  
## initial value 378.551862   
## iter 10 value 229.345916  
## iter 20 value 140.148831  
## iter 30 value 102.800617  
## iter 40 value 84.044160  
## iter 50 value 75.353764  
## iter 60 value 70.603753  
## iter 70 value 68.100085  
## iter 80 value 65.951893  
## iter 90 value 64.374881  
## iter 100 value 63.469051  
## final value 63.469051   
## stopped after 100 iterations  
## # weights: 501  
## initial value 420.305694   
## iter 10 value 220.711299  
## iter 20 value 136.128495  
## iter 30 value 97.915459  
## iter 40 value 82.491493  
## iter 50 value 75.510341  
## iter 60 value 71.897257  
## iter 70 value 69.858317  
## iter 80 value 67.928207  
## iter 90 value 65.434115  
## iter 100 value 64.457521  
## final value 64.457521   
## stopped after 100 iterations  
## # weights: 501  
## initial value 552.360593   
## iter 10 value 220.909124  
## iter 20 value 131.060228  
## iter 30 value 95.513050  
## iter 40 value 81.269554  
## iter 50 value 75.141856  
## iter 60 value 71.792300  
## iter 70 value 69.183298  
## iter 80 value 66.450365  
## iter 90 value 65.187997  
## iter 100 value 64.297136  
## final value 64.297136   
## stopped after 100 iterations  
## # weights: 501  
## initial value 402.502539   
## iter 10 value 231.395112  
## iter 20 value 140.503706  
## iter 30 value 107.517453  
## iter 40 value 89.699414  
## iter 50 value 80.142133  
## iter 60 value 74.823809  
## iter 70 value 71.839871  
## iter 80 value 68.396420  
## iter 90 value 65.405487  
## iter 100 value 63.357552  
## final value 63.357552   
## stopped after 100 iterations  
## # weights: 501  
## initial value 405.172051   
## iter 10 value 220.868294  
## iter 20 value 134.407466  
## iter 30 value 97.560551  
## iter 40 value 83.137364  
## iter 50 value 77.157712  
## iter 60 value 73.817408  
## iter 70 value 71.311964  
## iter 80 value 69.376511  
## iter 90 value 68.485206  
## iter 100 value 67.480413  
## final value 67.480413   
## stopped after 100 iterations  
## # weights: 501  
## initial value 407.348070   
## iter 10 value 252.482362  
## iter 20 value 152.334322  
## iter 30 value 117.264633  
## iter 40 value 101.258846  
## iter 50 value 89.565782  
## iter 60 value 79.640363  
## iter 70 value 73.608504  
## iter 80 value 68.623819  
## iter 90 value 65.813268  
## iter 100 value 63.472290  
## final value 63.472290   
## stopped after 100 iterations  
## # weights: 501  
## initial value 423.486007   
## iter 10 value 217.319835  
## iter 20 value 136.057195  
## iter 30 value 106.315123  
## iter 40 value 91.581530  
## iter 50 value 82.924591  
## iter 60 value 78.376898  
## iter 70 value 74.405124  
## iter 80 value 71.121894  
## iter 90 value 69.045569  
## iter 100 value 67.352874  
## final value 67.352874   
## stopped after 100 iterations  
## # weights: 501  
## initial value 548.970825   
## iter 10 value 240.274156  
## iter 20 value 152.699492  
## iter 30 value 116.226528  
## iter 40 value 100.112875  
## iter 50 value 90.408266  
## iter 60 value 85.523246  
## iter 70 value 80.677037  
## iter 80 value 77.988379  
## iter 90 value 75.460147  
## iter 100 value 73.248058  
## final value 73.248058   
## stopped after 100 iterations  
## # weights: 501  
## initial value 408.709785   
## iter 10 value 223.489350  
## iter 20 value 137.293482  
## iter 30 value 99.448567  
## iter 40 value 82.276306  
## iter 50 value 73.986803  
## iter 60 value 71.077440  
## iter 70 value 68.775037  
## iter 80 value 67.107895  
## iter 90 value 65.340468  
## iter 100 value 64.557698  
## final value 64.557698   
## stopped after 100 iterations  
## # weights: 501  
## initial value 525.060046   
## iter 10 value 235.861842  
## iter 20 value 139.450955  
## iter 30 value 93.920575  
## iter 40 value 77.324171  
## iter 50 value 70.690617  
## iter 60 value 66.814938  
## iter 70 value 64.728801  
## iter 80 value 63.246874  
## iter 90 value 62.454986  
## iter 100 value 61.497749  
## final value 61.497749   
## stopped after 100 iterations  
## # weights: 501  
## initial value 488.565700   
## iter 10 value 251.615467  
## iter 20 value 164.827677  
## iter 30 value 129.921785  
## iter 40 value 108.266790  
## iter 50 value 95.254008  
## iter 60 value 87.155702  
## iter 70 value 82.276184  
## iter 80 value 78.902746  
## iter 90 value 76.602232  
## iter 100 value 74.727299  
## final value 74.727299   
## stopped after 100 iterations

nnPredict <- predict(nn\_fit,newdata = test\_set )  
#Get the confusion matrix to see accuracy value and other parameter values  
confusionMatrix(nnPredict, test\_set$Y )

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2  
## 1 197 46  
## 2 35 55  
##   
## Accuracy : 0.7568   
## 95% CI : (0.707, 0.8019)  
## No Information Rate : 0.6967   
## P-Value [Acc > NIR] : 0.009043   
##   
## Kappa : 0.4062   
##   
## Mcnemar's Test P-Value : 0.266521   
##   
## Sensitivity : 0.8491   
## Specificity : 0.5446   
## Pos Pred Value : 0.8107   
## Neg Pred Value : 0.6111   
## Prevalence : 0.6967   
## Detection Rate : 0.5916   
## Detection Prevalence : 0.7297   
## Balanced Accuracy : 0.6968   
##   
## 'Positive' Class : 1   
##

t1<-table(nnPredict, test\_set$Y)  
nn\_acc<-(t1[1,1]+t1[2,2])/(t1[1,1]+t1[2,2]+t1[1,2]+t1[2,1])  
nn\_spec<-t1[2,2]/(t1[1,2]+t1[2,2])

# Neural Network ROC curve  
nn.predd <- predict(nn\_fit, type='prob',test\_set, probability = TRUE)  
  
modelroc <- roc(test\_set$Y,nn.predd[,2])  
plot(modelroc, type="S",print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),  
 grid.col=c("green", "red"), max.auc.polygon=TRUE,  
 auc.polygon.col="skyblue", print.thres=TRUE, main="ROC Curve of Neural Network Model")



# Comparison and Conclusions  
cat("\n")

cat(" Logistic regression model's accuracy: ", log\_acc,  
 "; Specificity: ",log\_spec ,"\n",  
 "kNN model's accuracy: ", knn\_acc,  
 "; Specificity: ",knn\_spec ,"\n",  
 "Decision tree model's accuracy: ", dt\_acc,  
 "; Specificity: ",dt\_spec ,"\n",  
 "Naive Bayes model's accuracy: ", nb\_acc,  
 "; Specificity: ",nb\_spec ,"\n",  
 "Support vector machine model's accuracy: ", svm\_acc,  
 "; Specificity: ",svm\_spec ,"\n",  
 "Neural Network model's accuracy: ", nn\_acc,  
 "; Specificity: ",nn\_spec ,"\n")

## Logistic regression model's accuracy: 0.7507508 ; Specificity: 0.4653465   
## kNN model's accuracy: 0.7537538 ; Specificity: 0.2673267   
## Decision tree model's accuracy: 0.7297297 ; Specificity: 0.4752475   
## Naive Bayes model's accuracy: 0.7477477 ; Specificity: 0.4554455   
## Support vector machine model's accuracy: 0.7207207 ; Specificity: 0.1188119   
## Neural Network model's accuracy: 0.7567568 ; Specificity: 0.5445545