Credit Card Report 4

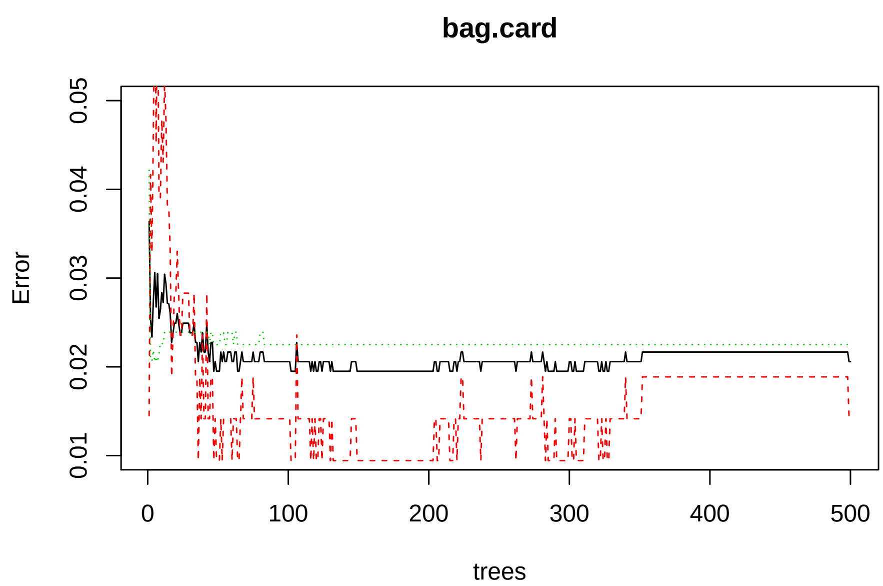
Fourth Report

4a. Splitting dataset

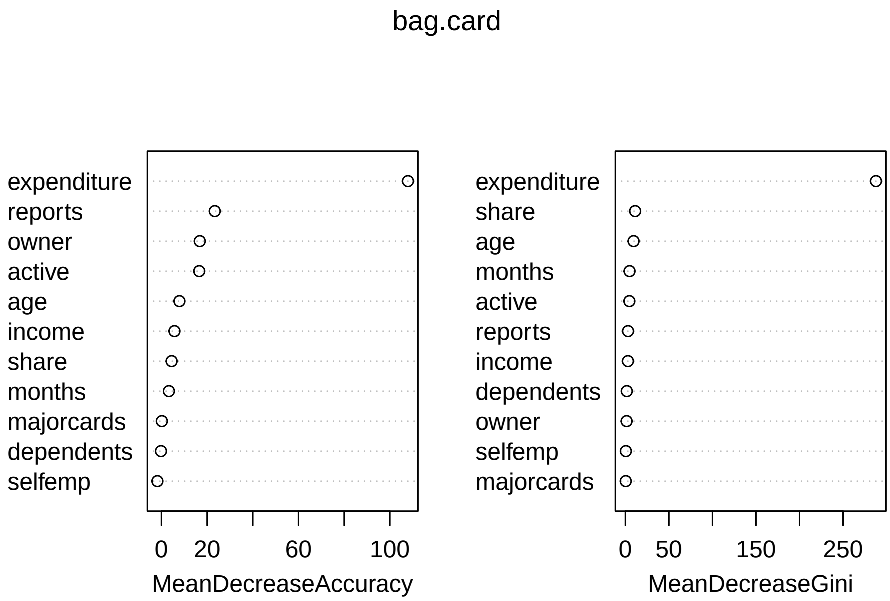
```r
library(AER)
library(dplyr)
data("CreditCard")
CreditCard = data.frame(CreditCard)
library(randomForest)
set.seed(123)
card_train = CreditCard %>%
  sample_frac(.70)

card_test = CreditCard %>%
  setdiff(card_train)
```

4b. Bagging

```r
set.seed(123)
bag.card = randomForest(card ~., data = card_train,
                        mtry=ncol(card_train) - 1,
                        importance=TRUE)
plot(bag.card, ylim=c(0.01, .05))
```



```r
varImpPlot(bag.card)
```



This plot displays the out of bag error rate as a function of number of trees. It tells us the misclassification rate of the overall training data which is the line in black. The red line indicates the misclassification rate for the yes'. The green line tells us the misclassification rate for the no's. The x-axis is the trees and the y-axis is the error rate. Our argument mtry tells us that all 11 predictors are going to be considered for each split of the tree. Also, We see that the error is decreasing as we keep splitting the trees which is correct.

```r
set.seed(123)
yhat.bag = predict(bag.card, newdata = card_test)
table(yhat.bag, card_test$card)
```

```
##
## yhat.bag  no yes
##      no   80   4
##      yes   4 308
```

```r
CM = table(yhat.bag, card_test$card)
baggingperf = (sum(diag(CM)))/sum(CM)
baggingperf
```

```
## [1] 0.979798
```

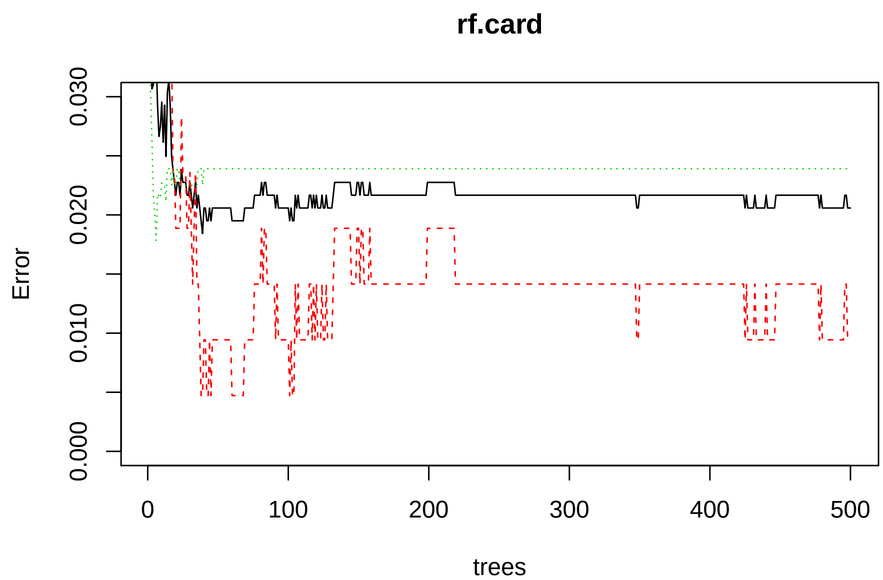
Bagging regression predicted our testing data to 97.98% accuracy.

4c. Random Forest

```r
set.seed(123)
rf.card = randomForest(card~.,
                         data = card_train,
                         mtry = 5,
                         importance = TRUE,
                         do.trace = 100)
```

```
## ntree      OOB      1      2
##   100:   2.06%  0.94%  2.39%
##   200:   2.28%  1.89%  2.39%
##   300:   2.17%  1.42%  2.39%
##   400:   2.17%  1.42%  2.39%
##   500:   2.06%  0.94%  2.39%
```

```r
plot(rf.card, ylim = c(0, 0.03))
```




```r
yhat.rf = predict(rf.card, newdata = card_test)
table(yhat.rf, card_test$card)
```

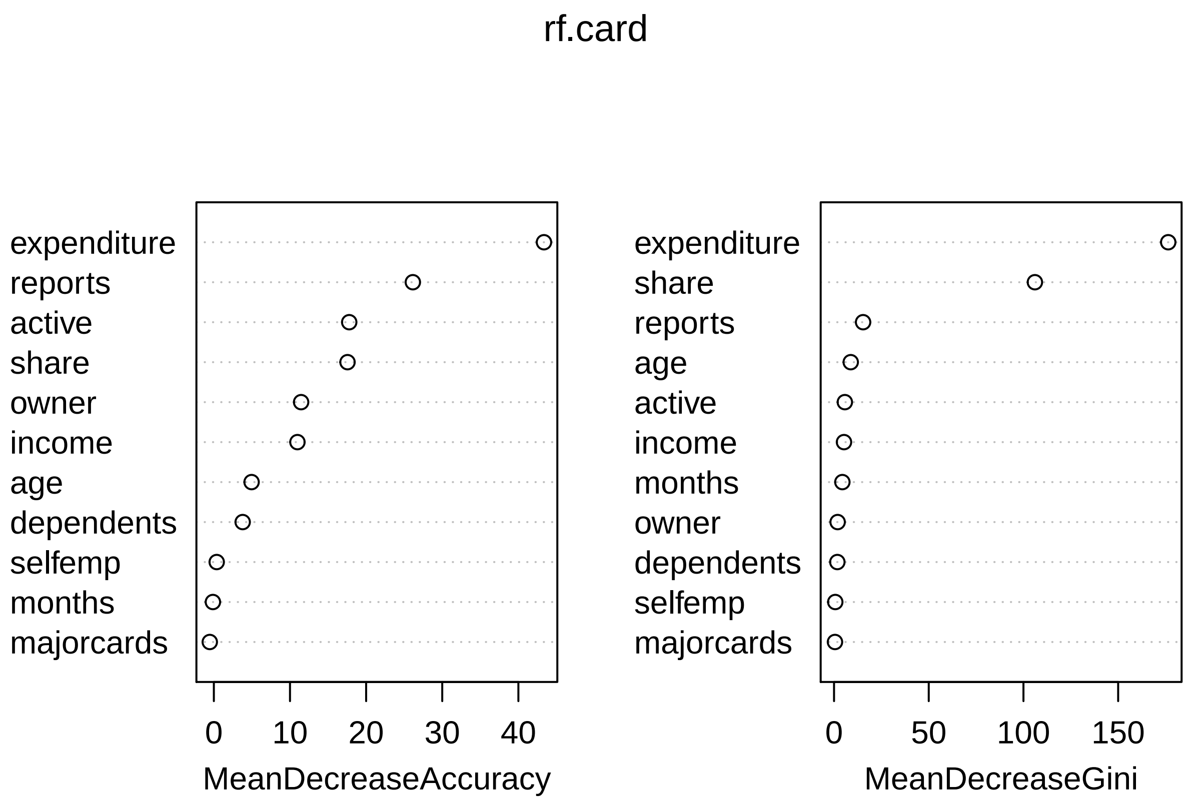
```
##
## yhat.rf  no yes
##      no   81   4
##      yes   3 308
```

```r
CM = table(yhat.rf, card_test$card)
rfperf = (sum(diag(CM)))/sum(CM)
rfperf
```

```
## [1] 0.9823232
```

This graph shows us the error of our random forest as we increase the number of trees. The green and red lines are errors of application accepted and application denied.

```r
varImpPlot(rf.card)
```



The hyperparameters for random forest are: mtry: which is the number of variables used at each split, ntree: which is the total number of trees, nodesize: which is the number of observations that we want in the terminal nodes (closely related to the depth of each tree). Also, looking at the plot, as we can see, this plot displays the out of random forest error rate as a function of number of trees. It tells us the misclassification rate of the overall training data which is the line in black. The red line indicates the misclassification rate for the yes'. The green line tells us the misclassification rate for the no's. The x-axis is the trees and the y-axis is the error rate. Our argument mtry tells us that all 11 predictors are going to be considered for each split of the tree. Also, We see that the error is decreasing as we keep splitting the trees which is correct.

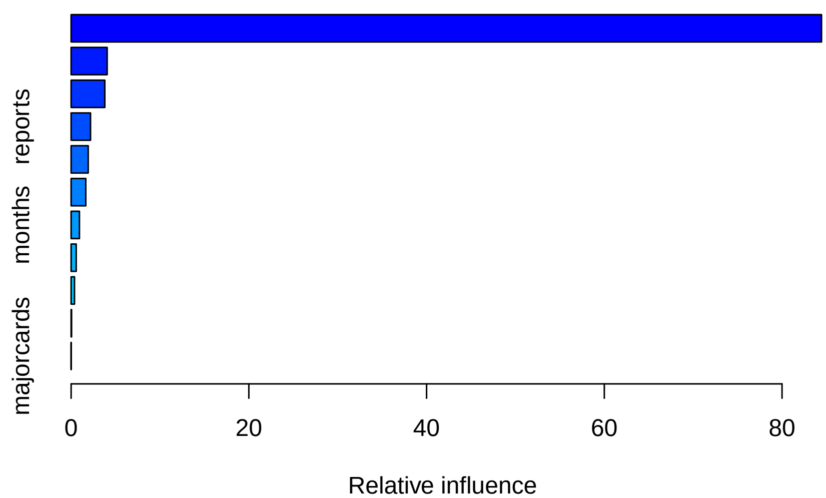
4d. Boosting

```
library(gbm)
library(randomForest)
data("CreditCard")
ls(CreditCard)
```

```
##  [1] "active"      "age"         "card"        "dependents"  "expenditure"
##  [6] "income"      "majorcards"  "months"      "owner"       "reports"
## [11] "selfemp"     "share"
```

```
CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)

set.seed(123)
card_train = CreditCard %>%
  sample_frac(.70)
```

```r
card_test = CreditCard %>%
  setdiff(card_train)

set.seed(123)
boost.card = gbm(card~.,
                   data = card_train,
                   distribution = "bernoulli",
                   n.trees = 500,
                   interaction.depth = 4)

summary(boost.card)
```




```
##                     var      rel.inf
## expenditure expenditure 84.442363473
## share             share  4.057486424
## age                 age  3.793389031
## reports         reports  2.185358785
## active           active  1.923705119
## income           income  1.649576005
## months           months  0.937511760
## dependents   dependents  0.572464156
## owner             owner  0.380654947
## selfemp         selfemp  0.050880616
## majorcards   majorcards  0.006609683
```

```r
yhat.boost = predict(boost.card,
                         newdata = card_test,
                         n.trees = 5000)

boost_pred = predict(boost.card, card_test, n.trees=500, type = "response")
y_pred = ifelse(boost_pred>0.2,1,0)
```

```r
boost_matrix = table(card_test$card, y_pred)
boostingperf = (sum(diag(boost_matrix)))/sum(boost_matrix)
boostingperf
```

```
## [1] 0.9722222
```

```r
boost.card
```

```
## gbm(formula = card ~ ., distribution = "bernoulli", data = card_train,
##     n.trees = 500, interaction.depth = 4)
## A gradient boosted model with bernoulli loss function.
## 500 iterations were performed.
## There were 11 predictors of which 11 had non-zero influence.
```

The relative influence of Boosting shows us that the most important variables in determine the status of an application is mainly expenditure, but then share and age.

```r
boost_matrix
```

```
##    y_pred
##       0   1
##   0  76   8
##   1   3 309
```

```r
boostingperf
```

```
## [1] 0.9722222
```

Boosting has a classification rate of 97.2%.

4e. XGBoost

```r
library(xgboost)
data("CreditCard")
CreditCard = data.frame(CreditCard)

CreditCard$card <- ifelse(CreditCard$card=="yes", 1, 0)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)

card_train = CreditCard %>%
  sample_frac(.70)

card_test = CreditCard %>%
  setdiff(card_train)

Y_train <- as.matrix(card_train[,"card"])
X_train <- as.matrix(card_train[!names(card_train) %in% c("card")])
dtrain <- xgb.DMatrix(data = X_train, label = Y_train)

X_test <- as.matrix(card_test[!names(card_train) %in% c("card")])
```

```r
set.seed(123)
set.seed(2)
card.xgb = xgboost(data=dtrain,
                     max_depth=2,
                     eta = 0.1,
                     nrounds=40,
                     lambda=0,
                     print_every_n = 10,
                     objective="binary:logistic")
```

```
## [1]  train-error:0.011918
## [11] train-error:0.011918
## [21] train-error:0.011918
## [31] train-error:0.011918
## [40] train-error:0.011918
```

```r
set.seed(123)
yhat.xgb <- predict(card.xgb,X_test)
y_pred = ifelse(yhat.xgb>0.2,1,0)
xgboost_matrix = table(y_pred, card_test$card)
xgboostperf = (sum(diag(xgboost_matrix)))/sum(xgboost_matrix)
xgboost_matrix
```

```
##
## y_pred   0   1
##      0  86  10
##      1   1 299
```

```r
xgboostperf
```

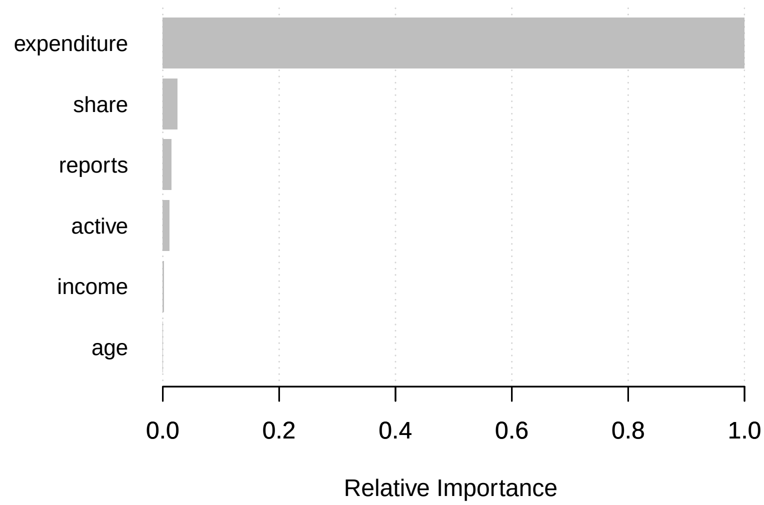
```
## [1] 0.9722222
```

XGBoost had the same performance as Boosting.

```r
importance <- xgb.importance(colnames(X_train),model=card.xgb)
importance
```

```
##        Feature         Gain      Cover  Frequency
## 1: expenditure 9.486217e-01 0.56324246 0.30275229
## 2:       share 2.423926e-02 0.03679621 0.06422018
## 3:     reports 1.417048e-02 0.18102553 0.26605505
## 4:      active 1.077693e-02 0.04322104 0.13761468
## 5:      income 2.191567e-03 0.03445663 0.10091743
## 6:         age 3.022896e-08 0.14125814 0.12844037
```

```r
xgb.plot.importance(importance, rel_to_first=TRUE, xlab="Relative Importance")
```



XGBoost also found expenditure, share, and reports as the most important variables.

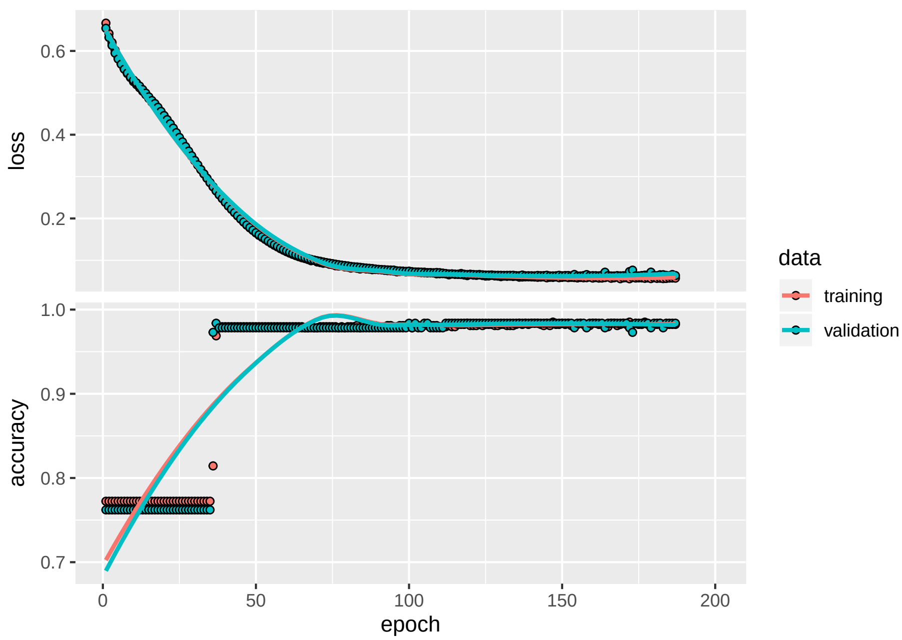
4e. Neural Net

Setting up the data for the Neural Net

```
library(keras)
library(ISLR)
library(dplyr)
library(AER)
library(tensorflow)
data("CreditCard")
CreditCard = data.frame(CreditCard)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
```

```
CreditCard$card_yes <- ifelse(CreditCard$card == "yes",1,0)
set.seed(123)
card_train = CreditCard %>%
  sample_frac(.7)

card_test = CreditCard %>%
  setdiff(card_train)

train_labels <- to_categorical(card_train[,"card_yes"],2)
train_data <- as.matrix(card_train[!names(card_train) %in% c("card","card_yes")])
test_data <- as.matrix(card_test[!names(card_train) %in% c("card","card_yes")])
test_labels <- to_categorical(card_test[,"card_yes"])

model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "softmax",
              input_shape = dim(train_data)[2]) %>%
```

```
  layer_dense(units = 64, activation = "softmax") %>%
  layer_dense(units = 2, activation= "softmax")

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 20)

epochs=200
history_class <- model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = 0.2,
  callbacks = list(early_stop)
)
plot(history_class)
```



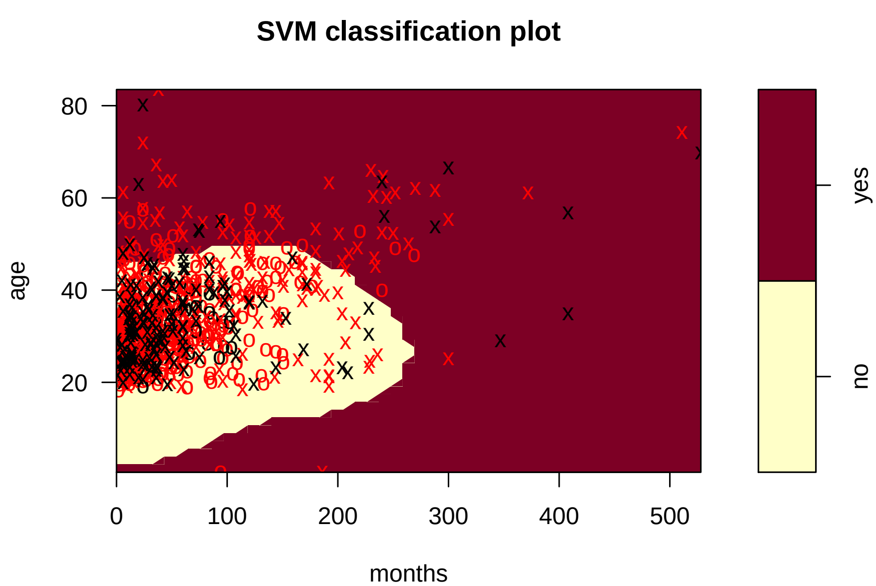

```
test_predictions <- model %>% predict(test_data)
test_class <- model %>% predict_classes(test_data)
error_table = table(test_labels[,2], test_class)
neuralnetperf = (sum(diag(error_table)))/sum(error_table)
neuralnetperf
```

```
## [1] 0.9873737
```

The Neural Net had a classification rate of 98.7%. The graph also shows us the accuracy and loss of our training and validation sets as our epochs increase. Our epochs is the number of back propogations we want to do for the model and we set it to stop after it doesnt change much in a certain number of iterations. As the epoch starts to approach 75, it does not change too much.

4g. Support Vector Machines

```
data("CreditCard")
library(dplyr)
library(ggplot2)
library(e1071)
set.seed(123)
CreditCard$card = as.factor(CreditCard$card)
CreditCard$owner <- ifelse(CreditCard$owner=="yes", 1, 0)
CreditCard$selfemp <- ifelse(CreditCard$selfemp=="yes", 1, 0)
training_set = CreditCard %>%
  sample_frac(.7)

testing_set = CreditCard %>%
  setdiff(training_set)

dat = data.frame(x = X_train, y = as.factor(Y_train))

CreditCard$card = factor(CreditCard$card, levels = c(0, 1))

set.seed(123)
tune.out = tune(svm, card~., data = training_set, kernel = "radial",
                ranges = list(cost = c(0.1,1,10,100,1000), gamma = c(0.5,1,2,3,4)))
bestmod = tune.out$best.model
plot(bestmod, training_set, age~months)
```




```
svm_table = table(true = testing_set$card, pred = predict(tune.out$best.model,
                                                   newdata = testing_set))
svmperf = (sum(diag(svm_table)))/sum(svm_table)
svmperf
```

```
## [1] 0.8737374
```

The SVM only had an accuracy rate of 87.37 percent. Also with the graph above, it is observed that an SVM can still be applied to data that is not linearly separable. It makes the assumption that similar data points will be close together on a graph. We can see that to an extent re low values and low values of age would typically get rejected for a credit card.

{**5. Model Comparisons**}

```r
mlperformance<-matrix(c(baggingperf,
                        rfperf, boostingperf, xgboostperf, neuralnetperf, svmperf),
                        ncol=1,byrow=TRUE)
rownames(mlperformance)<-c("Bagging","Random Forest",
                           "Boosting", "XGBoost", "Neural Network",
                           "Support Vector Machine")
colnames(mlperformance)<-c("Performance")
mlperformance
```

```
##                        Performance
## Bagging                  0.9797980
## Random Forest            0.9823232
## Boosting                 0.9722222
## XGBoost                  0.9722222
## Neural Network           0.9873737
## Support Vector Machine   0.8737374
```