Code

This is the data prepration code. We use this to make the basic and modified training and test sets. The datasets were used for model building, the code for which is described in this document

```r
library(caret)

# This line will need to be changed to the appropriate directory"
bank.full <- read.csv("~/Documents/STOR_565/Group project/565project/data/bank-additional-full.csv", sep

# Remove duration column
bank.full <- bank.full[bank.full$default != 'yes',-11]

# Create pdays
pdays <- bank.full$pdays

#Create height categories of A = 0-7, B=7-14, C=14-998, D=998+
cutpdays <- cut(pdays, breaks=c(-1,7,14,998,999), labels=c("A", "B", "C", "D"), right=TRUE)
bank.full$cutpdays <- cutpdays

# Split data into training and test set
set.seed(565)
train.index <- createDataPartition(bank.full$y, p = .75, list = FALSE)
train <- bank.full[ train.index,]
test  <- bank.full[-train.index,]

# Check to make sure class proportions have been maintained
mean(bank.full$y == 'yes')
mean(train$y == 'yes')
mean(test$y == 'yes')


Principal_Component <- function(dataframe){
  "Subset of raw data with only the economic features"
  dataframe_features_subset <- data.frame("emp.var.rate"=dataframe$emp.var.rate,"cons.price.idx"=datafra

  "Create Principal Components of the economic features"
  dataframe_subset_princomp <- prcomp(dataframe_features_subset,center=TRUE,scale.=TRUE)

  "Create a new dataframe by removing the original economic features and replacing them with the princip
  dataframe_new <- dataframe[, !(colnames(dataframe) %in% c("emp.var.rate","cons.price.idx","cons.conf.i
  dataframe_new$economic1 <- dataframe_subset_princomp$x[,1]
  dataframe_new$economic2 <- dataframe_subset_princomp$x[,2]

  "Return the new dataframe"
  return (dataframe_new)
}

train.pca <- Principal_Component(train)
test.pca <- Principal_Component(test)

# Output basic train/test
write.csv(train[,!(colnames(train) %in% c("cutpdays"))], "~/Documents/STOR_565/Group project/565project/
write.csv(test[,!(colnames(test) %in% c("cutpdays"))], "~/Documents/STOR_565/Group project/565project/da

# Output changed train/test
write.csv(train.pca[,!(colnames(train.pca) %in% c("pdays"))], "~/Documents/STOR_565/Group project/565pr
write.csv(test.pca[,!(colnames(test.pca) %in% c("pdays"))], "~/Documents/STOR_565/Group project/565proj
```

The function below was used to generate artificial sample points using Synthetic Minority oversampling Procedure(SMOTE) in R. Before using SMOTE, we arranged the dataframe in R by making the first column the response variable and the other columns to be features. The function takes a numerical dataframe, and an integer k denoting the number of nearest neighbors to be used to impute data.

```
syn_data_borderline <- function(dataframe,k){
    library(smotefamily)
    set.seed(1005)
    data <- ADAS(dataframe[,2:ncol(dataframe)],dataframe[,1],K=k)
    newdf <- data.frame("y"=data$data$class)
    newdf <- cbind(newdf,data$data[,1:length(data$data)-1])
    return (newdf)
}
```

The function below was used to do 5-fold cross validation for random forests. The cross validation procedure was used to tune the 'mtry' argument and arrive at the optimal number of predictors to be used in each tree. The function takes a numerical dataframe; and tree size ,which is the number of predictors to be used in each tree for the random forest. The model generates synthetic datapoints using the function above for every sub-training set

```
cross_validation_trees <- function(trainingdata,treesize){
  library(caret)
  library(randomForest)
  library(pROC)
  #Divide the data into 5 folds
  data_folds <- createFolds(trainingdata$y,k=5,list=FALSE)

  #Create different training sets
  data_train_fold1<- trainingdata[data_folds!=5,]
  data_train_fold2<- trainingdata[data_folds!=1,]
  data_train_fold3<- trainingdata[data_folds!=2,]
  data_train_fold4<- trainingdata[data_folds!=3,]
  data_train_fold5<- trainingdata[data_folds!=4,]

  #Create different test sets
  test_data_fold1 <- trainingdata[data_folds==5,]
  test_data_fold2 <- trainingdata[data_folds==1,]
  test_data_fold3 <- trainingdata[data_folds==2,]
  test_data_fold4 <- trainingdata[data_folds==3,]
  test_data_fold5 <- trainingdata[data_folds==4,]

  #Create a vector to store number of correctly classified points at every step.
  result_list <- c(rep(0,5))
  area_under_curve <- c(rep(0,5))
  trainlist <- c(rep(0,5))
  testlist <- c(rep(0,5))
  #Create list to store each training and test fold. The lists will be used inside a for loop to build
  trainlist <- list(data_train_fold1,data_train_fold2,data_train_fold3,data_train_fold4,data_train_fold
  testlist <- list(test_data_fold1,test_data_fold2,test_data_fold3,test_data_fold4,test_data_fold5)
  for (i in 1:5){
    trainlist[[i]] <- syn_data_borderline(trainlist[[i]],5)
    randomforest_model <- randomForest(factor(y)~.,data=trainlist[[i]],mtry=treesize,importance=TRUE)
    prediction <- predict(randomforest_model,newdata=testlist[[i]],type='response')
    table <- table(prediction,testlist[[i]]$y)
    result_list[i] <- (table[1]+table[4])
    rf.roc <- roc(testlist[[i]]$y,ifelse(prediction==1,1,0))
    area_under_curve[i] <- rf.roc$auc[1]

  }
  return (list(result_list,randomforest_model,area_under_curve))
}
```

The function below was used to generate principal compnents for the economic features in our dataset. The economic features were correlated to the response, but also highly correlated to one another. We used PCA to generate two derived economic features which are representative of all the economic features in our dataset.

```
Principal_Component <- function(dataframe){
  "Subset of raw data with only the economic features"
  dataframe_features_subset <- data.frame("emp.var.rate"=dataframe$emp.var.rate,"cons.price.idx"=datafra

  "Create Principal Components of the economic features"
  dataframe_subset_princomp <- prcomp(dataframe_features_subset,center=TRUE,scale.=TRUE)

  "Create a new dataframe by removing the original economic features and replacing them with the princip
  dataframe_new <- dataframe[, !(colnames(dataframe) %in% c("emp.var.rate","cons.price.idx","cons.conf.
  dataframe_new$PC1 <- dataframe_subset_princomp$x[,1]
  dataframe_new$PC2 <- dataframe_subset_princomp$x[,2]
  dataframe_new$PC3 <- dataframe_subset_princomp$x[,3]
  dataframe_new$PC4 <- dataframe_subset_princomp$x[,4]
  dataframe_new$PC5 <- dataframe_subset_princomp$x[,5]

  "Return the new dataframe"
  return (dataframe_new)
}
```

The functions below were used to clean the basic and modified dataset. Although the data was used for cleaning, they were essential for running random forests. The data had to be prepared to be able to pass it into the random forest function.

```
augmented_dataframe <- function(dataframe){
    library(mlr)

    'Initilize the new dataframe'
    bank_aug <- data.frame(y=ifelse(dataframe$y=="yes",1,0))
    bank_aug$age <- dataframe$age

    'Create the dummy variables'
    job_dummy <- createDummyFeatures(dataframe$job)
    marital_dummy <- createDummyFeatures(dataframe$marital)
    education_dummy <- createDummyFeatures(dataframe$education)
    default_dummy <- createDummyFeatures(dataframe$default)
    housing_dummy <- createDummyFeatures(dataframe$housing)
    loan_dummy <- createDummyFeatures(dataframe$loan)
    contact_dummy <- createDummyFeatures(dataframe$contact)
    month_dummy <- createDummyFeatures(dataframe$month)
    day_of_week_dummy <- createDummyFeatures(dataframe$day_of_week)
    poutcome_dummy <- createDummyFeatures(dataframe$poutcome)

    'Adding dummy columns to the main dataframe'
    bank_aug <- cbind(bank_aug,job_dummy[,1:length(job_dummy)-1])
    bank_aug <- cbind(bank_aug,marital_dummy[,1:length(marital_dummy)-1])
    bank_aug <- cbind(bank_aug,education_dummy[,1:length(education_dummy)-1])
    bank_aug <- cbind(bank_aug,default_dummy[,1:length(default_dummy)])
    bank_aug <- cbind(bank_aug,housing_dummy[,1:length(housing_dummy)-1])
    bank_aug <- cbind(bank_aug,loan_dummy[,1:length(loan_dummy)-1])
    bank_aug <- cbind(bank_aug,contact_dummy[,1:length(contact_dummy)-1])
    bank_aug <- cbind(bank_aug,month_dummy[,1:length(month_dummy)-1])
    bank_aug <- cbind(bank_aug,day_of_week_dummy[,1:length(day_of_week_dummy)-1])
    bank_aug <- cbind(bank_aug,dataframe$campaign)
    bank_aug <- cbind(bank_aug,dataframe$pdays)
    bank_aug <- cbind(bank_aug,dataframe$previous)
    bank_aug <- cbind(bank_aug,poutcome_dummy[,1:length(poutcome_dummy)-1])
    bank_aug <- cbind(bank_aug,dataframe$emp.var.rate)
    bank_aug <- cbind(bank_aug,dataframe$cons.price.idx)
    bank_aug <- cbind(bank_aug,dataframe$cons.conf.idx)
    bank_aug <- cbind(bank_aug,dataframe$euribor3m)
    bank_aug <- cbind(bank_aug,dataframe$nr.employed)

   names(bank_aug)[4] <- "blue.collar"
   names(bank_aug)[9] <- "self.employed"

   names(bank_aug)[24] <- "default_no"

   names(bank_aug)[26] <- "housing_no"
   names(bank_aug)[27] <- "housing_unknown"
   names(bank_aug)[28] <- "loan_no"
   names(bank_aug)[29] <- "loan_unknown"
   names(bank_aug)[30] <- "contact_cellular"
   names(bank_aug)[44] <- "campaign"
   names(bank_aug)[45] <- "pdays"
   names(bank_aug)[46] <- "previous"

   names(bank_aug)[49] <- "emp.var.rate"
   names(bank_aug)[50] <- "cons.price.idx"
   names(bank_aug)[51] <- "cons.conf.idx"
   names(bank_aug)[52] <- "euribor3m"
   names(bank_aug)[53] <- "nr.employed"


   return (bank_aug)
}

#Dummy variable creation for combolog dataset
augmented_dataframe_combodata <- function(dataframe){
  library(mlr)

  'Initilize the new dataframe'
```

```r
  bank_aug <- data.frame(y=ifelse(dataframe$y=="yes",1,0))
  bank_aug$age <- dataframe$age
  'Create the dummy variables'
  job_dummy <- createDummyFeatures(dataframe$job)
  marital_dummy <- createDummyFeatures(dataframe$marital)
  education_dummy <- createDummyFeatures(dataframe$education)
  default_dummy <- createDummyFeatures(dataframe$default)
  housing_dummy <- createDummyFeatures(dataframe$housing)
  loan_dummy <- createDummyFeatures(dataframe$loan)
  contact_dummy <- createDummyFeatures(dataframe$contact)
  month_dummy <- createDummyFeatures(dataframe$month)
  day_of_week_dummy <- createDummyFeatures(dataframe$day_of_week)
  poutcome_dummy <- createDummyFeatures(dataframe$poutcome)
  pdays_dummy <- createDummyFeatures(dataframe$cutpdays)
  'Adding dummy columns to the main dataframe'
  bank_aug <- cbind(bank_aug,job_dummy[,1:length(job_dummy)-1])
  bank_aug <- cbind(bank_aug,marital_dummy[,1:length(marital_dummy)-1])
  bank_aug <- cbind(bank_aug,education_dummy[,1:length(education_dummy)-1])
  bank_aug <- cbind(bank_aug,default_dummy[,1:length(default_dummy)])
  bank_aug <- cbind(bank_aug,housing_dummy[,1:length(housing_dummy)-1])
  bank_aug <- cbind(bank_aug,loan_dummy[,1:length(loan_dummy)-1])
  bank_aug <- cbind(bank_aug,contact_dummy[,1:length(contact_dummy)-1])
  bank_aug <- cbind(bank_aug,month_dummy[,1:length(month_dummy)-1])
  bank_aug <- cbind(bank_aug,day_of_week_dummy[,1:length(day_of_week_dummy)-1])
  bank_aug <- cbind(bank_aug,dataframe$campaign)
  bank_aug <- cbind(bank_aug,pdays_dummy[,1:length(pdays_dummy)-1])
  bank_aug <- cbind(bank_aug,dataframe$previous)
  bank_aug <- cbind(bank_aug,poutcome_dummy[,1:length(poutcome_dummy)-1])
  bank_aug <- cbind(bank_aug,dataframe$economic1)
  bank_aug <- cbind(bank_aug,dataframe$economic2)


  names(bank_aug)[4] <- "blue.collar"
  names(bank_aug)[9] <- "self.employed"
  names(bank_aug)[30] <- "contact_cellular"
  names(bank_aug)[44] <- "campaign"
  names(bank_aug)[48] <- "previous"
  names(bank_aug)[51] <- "economic1"
  names(bank_aug)[52] <- "economic2"

  return (bank_aug)
}

Principal_Component <- function(dataframe){
  "Subset of raw data with only the economic features"
  dataframe_features_subset <- data.frame("emp.var.rate"=dataframe$emp.var.rate,"cons.price.idx"=datafra

  "Create Principal Components of the economic features"
  dataframe_subset_princomp <- prcomp(dataframe_features_subset,center=TRUE,scale.=TRUE)

  "Create a new dataframe by removing the original economic features and replacing them with the princip
  dataframe_new <- dataframe[, !(colnames(dataframe) %in% c("emp.var.rate","cons.price.idx","cons.conf.
  dataframe_new$PC1 <- dataframe_subset_princomp$x[,1]
  dataframe_new$PC2 <- dataframe_subset_princomp$x[,2]
  dataframe_new$PC3 <- dataframe_subset_princomp$x[,3]
  dataframe_new$PC4 <- dataframe_subset_princomp$x[,4]
  dataframe_new$PC5 <- dataframe_subset_princomp$x[,5]

  "Return the new dataframe"
  return (dataframe_new)
}


syn_data_borderline <- function(dataframe,k){
    library(smotefamily)
    set.seed(1005)
    data <- ADAS(dataframe[,2:ncol(dataframe)],dataframe[,1],K=k)
    newdf <- data.frame("y"=data$data$class)
    newdf <- cbind(newdf,data$data[,1:length(data$data)-1])
    return (newdf)
}
```

The code below is used to run the random forest model. Then, we use the cross validation function described above to find the optimal tuning parameter for random forests by doing a grid search. With the optimal tuning parameter, we fit the model on the entire training set and predict on the test set.

```
train <- read.csv("~/Documents/565Project/Data/basic_train.csv")
test <- read.csv("~/Documents/565Project/Data/basic_test.csv")
train <- augmented_dataframe(train)
#Grid search and cv on basic train and test set
#Cross Validation TO IDENTIFY OPTIMAL TREESIZE
treesize = c(4,5,6,7)
validation <- c(rep(0,length(treesize)))
auc_tree <- list(rep(0,5))

for (i in 1:4){
  results <- cross_validation_trees(train,treesize[i])
  validation[i] <- sum(results[[1]])/nrow(train)
  auc_tree[[i]] <- results[[3]]
}
print (validation)
print (auc_tree)
write.csv(auc_tree,file="basicmodelauc.csv")
write.csv(validation, file = "basicmodelvalidation.csv")

#USING CV RESULTS TO TRAIN MODEL ON ENTIRE TRAINING SET AND PREDICT ON TEST SET

train <- syn_data_borderline(train,5)

test <- augmented_dataframe(test)

modeldf <- rbind(train,test)
#######Fit the Model
model <- randomForest(factor(y)~.,data=modeldf[1:nrow(train),],mtry=5,importance=TRUE)
library(pROC)
prediction_basic <- predict(model,test,type='response')
pred_table_basic <- table(prediction_basic,test$y)
```

```
rf.roc_bestmodel_basic <- roc(test$y,ifelse(prediction_basic==1,1,0))
area_under_curve_bestmodel_basic <- rf.roc_bestmodel_basic
save(rf.roc_bestmodel_basic,file="~/Documents/565Project/evaluation/comparison.R")
```

The code below is used to run a random forest model after feature engineering on the training set. We changed the ‘p-days’ variable by modifying it to be ordinal with 5 different levels. We also replaced the economic features with 2 derived economic variables using PCA. The augmented_dataframe_combodata is a function used to data cleaning. The rest of the process is the same as above.

```
modified_train <- read.csv("~/Documents/565Project/Data/modified_train.csv")
modified_test <- read.csv("~/Documents/565Project/Data/modified_test.csv")
modified_train <- augmented_dataframe_combodata(modified_train)
names(modified_train)[25] <- "default_no"
names(modified_train)[26] <- "default_unknown"
names(modified_train)[27] <- "housing_no"
names(modified_train)[28] <- "housing_unknown"

#USING MODIFIED TRAINING DATA TO PICK OPTIMAL TREE SIZE WITH CROSS VALIDATION
treesize = c(4,5,6,7)
validation_complexmodel <- c(rep(0,length(treesize)))
auc_tree_complexmodel <- list(rep(0,5))

z=0
for (i in 1:4){
  results <- cross_validation_trees(modified_train,treesize[i])
  validation_complexmodel[i] <- sum(results[[1]])/nrow(modified_train)
  auc_tree_complexmodel[[i]] <- results[[3]]

}
print (validation_complexmodel)
print (auc_tree_complexmodel)

write.csv(validation_complexmodel, file = "complexmodelvalidation.csv")

#USING CV TO BUILD A RF MODEL ON ENTIRE TRAINING SET AND PREDICT ON TEST SET
modified_train <- syn_data_borderline(modified_train,5)
modified_test <- augmented_dataframe_combodata(modified_test)

names(modified_test)[25] <- "default_no"
names(modified_test)[26] <- "default_unknown"
names(modified_test)[27] <- "housing_no"
names(modified_test)[28] <- "housing_unknown"

dataframe <-rbind(modified_train,modified_test)
modified_model <- randomForest(factor(y)~.,data=dataframe[1:55093,],mtry=5,importance=TRUE)
prediction_modified <- predict(modified_model,modified_test,type='response')
pred_table_modified <- table(prediction_modified,modified_test$y)
rf.roc_bestmodel_modified <- roc(modified_test$y,ifelse(prediction_modified==1,1,0))
area_under_curve_bestmodel_modified <- rf.roc_bestmodel_modified
save(rf.roc_bestmodel_modified,file="~/Documents/565Project/evaluation/comparison.R")
```

Logistic Regression with LASSO:

```
library(glmnet)
library(pROC)
```

```r
bank.train.mod <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\modified_train.csv")
bank.test.mod <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\modified_test.csv")
bank.train <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\basic_train.csv")
bank.test <- read.csv("C:\\Users\\tbian\\Documents\\GitHub\\565project\\data\\basic_test.csv")
```

Step2. do the lASSO logistic model based on the basic training dataset:

```r
set.seed(1005)
x.matrix.b<-model.matrix(~.,bank.train[,-20])[,-1]
x.test.b<-model.matrix(~.,bank.test[,-20])[,-1]
y.test.b=bank.test$y
foldid=sample(1:4,size=length(bank.train$y),replace=TRUE)
bank.lasso.b<-cv.glmnet(x.matrix.b,bank.train$y, family="binomial", type.measure="class", alpha=1)
plot(bank.lasso.b)
min(bank.lasso.b$cvm)
```

Step 3. Fit the model with tuning lamda and generate test error and ROC Curve.

```r
set.seed(1005)
fit.b<-glmnet(x.matrix.b,bank.train$y, family="binomial", alpha=1,lambda = bank.lasso.b$lambda.1se)
logistic.predict.b<-predict (fit.b, newx = x.test.b , type="response")
log.pre.b<-ifelse(logistic.predict.b<0.5,0,1)
y.test.b<-ifelse(y.test.b=='no',0,1)
table(y.test.b, log.pre.b)
1-mean(y.test.b==log.pre.b)  #####[1] 0.1012044
log.roc.b <- roc(y.test.b, as.numeric(logistic.predict.b))
plot(log.roc.b)
log.roc.b$auc
```

Now, we will do the lasso logistic regression on the modified train dataset and test dataset:

```r
set.seed(1005)
x.matrix.m<-model.matrix(~.,bank.train.mod[,-14])[,-1]
x.test.m<-model.matrix(~.,bank.test.mod[,-14])[,-1]
y.test.m=bank.test.mod$y
foldid.m=sample(1:4,size=length(bank.train.mod$y),replace=TRUE)
bank.lasso.m<-cv.glmnet(x.matrix.m,bank.train.mod$y, family="binomial", type.measure="class", alpha=1)
plot(bank.lasso.m)
min(bank.lasso.m$cvm)
```

Check the test error and generate ROC curve:

```r
set.seed(1005)
fit.m<-glmnet(x.matrix.m,bank.train.mod$y, family="binomial", alpha=1,lambda = bank.lasso.m$lambda.1se)
logistic.predict.m<-predict (fit.m, newx = x.test.m , type="response")
log.pre.m<-ifelse(logistic.predict.m<0.5,0,1)
y.test.m<-ifelse(y.test.m=="no",0,1)
table(y.test.m, log.pre.m)
1-mean(y.test.m==log.pre.m)   #####[1] 0.1007187
log.roc.m <- roc(y.test.m, as.numeric(logistic.predict.m))
plot(log.roc.m)
log.roc.m$auc
```

Load required packages.

```r
library("e1071")
library(pROC)
library(DMwR)
```

Read in training datasets, basic version and modified version with feature engineering.

```
bank.train <- read.csv("~/Documents/STOR_565/Group project/565project/data/basic_train.csv")
mod.train <- read.csv("~/Documents/STOR_565/Group project/565project/data/modified_train.csv")
```

Split training set into training and validation sets.

```
set.seed(5)
sample.index <- sample(1:nrow(bank.train),floor(0.3*nrow(bank.train)),replace=FALSE)
train.sample <- bank.train[sample.index,]
valid.sample <- bank.train[-sample.index,]
```

Perform cross-validation on training set to find optimal hyperparameters for linear, radial and polynommial kernels

```
grid <- c(0.1,0.2,0.3)
set.seed(565)
start.time <- Sys.time()
#tune.out <- tune(svm, y ~ ., data = train.sample,
#                             kernel = "linear", ranges = list(cost = grid),
#                             tunecontrol = tune.control(cross = 5))
#tune.out <- tune(svm,y ~ ., data = train.sample,kernel="radial",
#                        ranges=list(cost=c(0.1,1),gamma=c(0.5,3) ), tunecontrol = tune.control(cross =
tune.out <- tune(svm,y ~ ., data = train.sample,kernel="polynomial",
                            ranges=list(cost=c(0.1,1),degree=c(1,2,3),tunecontrol = tune.control(cross =
stop.time <- Sys.time()
stop.time - start.time
plot(tune.out)
summary(tune.out)
```

Using hyperparameters from above, fit models on full training set. Linear kernel give just as good prediction accuracy as more complex kernels so it is chosen.

```
svm.linear <- svm(y ~ ., data = train.sample, kernel = "linear", cost = 0.1, probability = TRUE)
svm.radial <- svm(y ~ ., data = train.sample, kernel = "radial", cost = 0.1, probability = TRUE)
svm.poly <- svm(y ~ ., data = train.sample, kernel = "polynomial", cost = 0.1, degree = 2, probability =
svm.predicts <- predict(svm.poly, newdata = test.sample, probability = TRUE)
svm.err <- mean(svm.predicts != test.sample$y)
```

Create function to model svm with different datasets and with and without SMOTE.

```
svm.model <- function(full_data, smote=FALSE){
  set.seed(5)
  sample.index <- sample(1:nrow(full_data),floor(0.3*nrow(bank.train)),replace=FALSE)
  train.sample <- full_data[sample.index,]
  test.sample <- full_data[-sample.index,]

  if (smote) {
    set.seed(10)
    train.sample <- SMOTE(y ~ ., train.sample)
  }

  svm.linear <- svm(y ~ ., data = train.sample, kernel = "linear", cost = 0.1, probability = TRUE)
  svm.predicts <- predict(svm.linear, newdata = test.sample, probability = TRUE)
  (svm.err <- mean(svm.predicts != test.sample$y))
  svm.roc <- roc(test.sample$y, attr(svm.predicts, "probabilities")[,1])
  plot(svm.roc)
  svm.roc$auc

  results <- list("model"=svm.linear, "predictions"=svm.predicts, "error"=svm.err, "roc"=svm.roc)
  return(results)
}
```

Fit svm model with basic/modified and SMOTE/no SMOTE. Model with basic data and SMOTE has best performance.

```
basic_nosmote <- svm.model(bank.train)
mod_nosmote <- svm.model(mod.train)

basic_smote <- svm.model(bank.train, smote=TRUE)
mod_smote <- svm.model(mod.train, smote=TRUE)

best.svm <- basic_smote
save(best.svm, file = "~/Documents/STOR_565/Group project/565project/modeling/best.svm.R")
```

Decision Tree creation process. NOTE: This set of code doesn't use the SMOTE algorithm beforehand. Step 1: Fitting a tree on the training data, with 'y' as the response variable. Predicting the response on the test data. We use this to calculate the test error of the tree.

```
#modified_train.csv is the training dataset
#modified_test.csv is the test dataset
library(tree)
bank.train <- tree(y ~., data=modified_train) #creating decision tree
summary(bank.train)
plot(bank.train)
text(bank.train, pretty=0)
bank.pred <- predict(bank.train, modified_test, type="class")
table(bank.pred, modified_test$y)
```

Step 2: Pruning the tree that was made on the training data. Predicting the response on the test data. We use this to calculate the test error of the pruned tree.

```
prune.bank <- prune.misclass(bank.train, best=2)
plot(prune.bank)
text(prune.bank, pretty=0)
prune.pred <- predict(prune.bank, modified_test, type="class")
table(prune.pred, modified_test$y)
```

Step 3: Determining the optimal tree size.

```
cv.bank <- cv.tree(bank.train, FUN = prune.misclass)
plot(cv.bank$size, cv.bank$dev, type="b", xlab="Tree size", ylab="Deviance")
```

Step 4: Performing boosting on the training set with 1000 trees for a range of lambda values.

```
library(gbm)
powers <- seq(-5, -0.2, by=0.1)
lambdas <- factor(10^powers)
bank.train.err <- rep(NA, length(lambdas))
train.matrix <- model.matrix(~ ., data=modified_train)[,-1]
for (i in 1:length(lambdas)) {
  bank.boost.train <- gbm(y ~ ., data = modified_train, distribution = "gaussian", n.trees = 1000, shri
  bank.pred.train <- predict(bank.boost.train, modified_train, n.trees = 1000)
  diff <- as.numeric(bank.pred.train) - as.numeric(modified_train$y)
  bank.train.err[i] <- mean((diff)^2)
}
plot(lambdas, bank.train.err, type="b", xlab="Shrinkage Values", ylab="Training MSE")
```

Step 5: Performing boosting on the test set with 1000 trees for a range of lambda values. Attempting to find the lambda that produces the minimum error.

```
bank.test.err <- rep(NA, length(lambdas))
for (i in 1:length(lambdas)) {
  bank.boost.test <- gbm(y ~ ., data=modified_train, distribution = "gaussian", n.trees = 1000,
                     shrinkage = lambdas[i])
  yhat <- predict(bank.boost.test, modified_test, n.trees = 1000)
  diff2 <- as.numeric(yhat) - as.numeric(modified_test$y)
  bank.test.err[i] <- mean((diff2)^2)
}
plot(lambdas, bank.test.err, type="b", xlab="Shrinkage Values", ylab="Test MSE")
min(bank.test.err)
lambdas[which.min(bank.test.err)]
```

Step 6: Trying to determine which variable is the most important.

```
library(gbm)
library(randomForest)
boost.bank <- gbm(y ~ ., data = modified_train, distribution = "gaussian", n.trees=1000,
                  shrinkage = lambdas[which.min(bank.test.err)])
summary(boost.bank)
```

Step 7: Using bagging. Determining area under the curve. Determining which variable is the most important in the bagging model. Predicting on test data and test error.

```
library(randomForest)
library(pROC)
bank.bag <- randomForest(y ~ ., data = modified_train, mtry = 10, ntree = 500)
yhat.bag <- predict(bank.bag, newdata = modified_test, type= "response")
yhat.bag[1:5]
mean((as.numeric(yhat.bag) - as.numeric(modified_test$y))^2)
pred_table <- table(yhat.bag, modified_test$y)
bag.roc_basic.yes <- roc(modified_test$y, ifelse(yhat.bag=="yes",1,0))
plot(bag.roc_basic.yes)
bag.roc_basic.yes$auc
varImpPlot(bank.bag)
```

Step 8: Creating two more bagging models. Taking a random sample of predictions with 20 predictors, and using the square root and dividing by 3 to choose as split candidates. Using these bagging models to predict on the test data, calculate the test error, and finding the area under the curve.

```
set.seed(1)
bank.bag1 <- randomForest(y ~ ., data=modified_train, mtry = sqrt(20), ntree=500)
bank.bag2 <- randomForest(y ~ ., data=modified_train, mtry = 20/3, ntree= 500)
varImpPlot(bank.bag1)
varImpPlot(bank.bag2)
library(randomForest)
library(pROC)
```

```
yhat.bag1 <- predict(bank.bag1, newdata = modified_test) #prediction
mean((as.numeric(yhat.bag1) - as.numeric(modified_test$y))^2)
pred_table1 <- table(yhat.bag1, modified_test$y)
bag1.roc_basic <- roc(modified_test$y, ifelse(yhat.bag1=="yes",1,0))
auc_basic1 <- bag1.roc_basic
plot(bag1.roc_basic)
bag1.roc_basic$auc
yhat.bag2 <- predict(bank.bag2, newdata = modified_test) #prediction
mean((as.numeric(yhat.bag2) - as.numeric(modified_test$y))^2)
pred_table2 <- table(yhat.bag2, modified_test$y)
bag2.roc_basic <- roc(modified_test$y, ifelse(yhat.bag2=="yes",1,0))
auc_basic2 <- bag2.roc_basic
plot(bag2.roc_basic)
bag2.roc_basic$auc
```

Boosting Function to test training and validation error at different shrinkage values

```
library(gbm)
boost <- function(train,test,trees,maxshrinkage){
    train_error <- c(rep(1,30))
    test_error <- c(rep(1,30))
    grid=c(seq(0.01,maxshrinkage,0.01))
    j=0
    for (i in grid){
        j=j+1
        boost.tree.model <- gbm(y~.,data=train,distribution="bernoulli",n.trees=trees,interaction.depth
        prediction_train <- predict(boost.tree.model,train,n.trees=trees,type='response')
        prediction_class <- ifelse(prediction_train>0.5,1,0)
        table <- table(prediction_class,train$y)
        train_error[j]= (mean(prediction_class!=train$y))

        prediction <- predict(boost.tree.model,test,n.trees=trees,type='response')
        prediction_class <- ifelse(prediction>0.5,1,0)

        table <- table(prediction_class,test$y)
        test_error[j] = (mean(prediction_class!=test$y))

    }
    return(list(train_error,test_error))

}
```

Fit the boosting model. The smote function was giving an error for the modified dataset. Hence we chose to not use smote on the modified dataset. We have three different models below. We fit a boosting model on the basic dataset with and without SMOTE. We also fit a boosting model on the modified dataset without SMOTE.

```
#install.packages("ParamHelpers")
#Boosting on Basic data sets
library(splines)
library(parallel)
library(survival)
library(lattice)
library(gbm)
setwd("C:/Users/sunhwa/Documents/Spring2018/STOR565ML/group project/565project")
```

```
source("565_proj_func.r")
bank.train <- read.csv("basic_train.csv")
bank.train.aug<-augmented_dataframe(bank.train)
bank.test<-read.csv("basic_test.csv")
bank.test.aug<-augmented_dataframe(bank.test)
set.seed(1)
idx = sample(1:nrow(bank.train.aug), size=0.75*nrow(bank.train.aug))

########################  without smote
# Split data
subtrain<-bank.train.aug[idx,]
#subtain <- syn_data_borderline(subtrain,5)
validate<-bank.train.aug[-idx,]
newdf <- rbind(subtrain,validate,bank.test.aug)
bank.boot<-boost(newdf[1:nrow(subtrain),],newdf[nrow(subtrain)+1:nrow(subtrain)+nrow(validate),],1000,0
model <- bank.boot[[3]]
#bank_train <- syn_data_borderline(bank.train.aug,5)### f
boost.bank = gbm(y~.,data=bank.train.aug, distribution= "bernoulli", n.trees=100,
                 shrinkage=0.01, verbose =F, cv.folds = 5)
prediction<-predict(model,newdf[30890:nrow(newdf),],n.trees=1000,type='response')
prediction_class <- ifelse(prediction>0.5,1,0)

table <- table(prediction_class,bank.test.aug$y)
test_error = (mean(prediction_class!=bank.test.aug$y))
library(pROC)
roc_obj <- roc(prediction_class, bank.test.aug$y)
auc(roc_obj)
plot(roc_obj)

####################### with smote

# Split data
subtrain<-bank.train.aug[idx,]
subtrain.sm <- syn_data_borderline(subtrain,5)
validate<-bank.train.aug[-idx,]
newdf <- rbind(subtrain.sm,validate,bank.test.aug)
bank.boot.sm<-boost(newdf[1:nrow(subtrain.sm),],newdf[nrow(subtrain.sm)+1:nrow(subtrain.sm)+nrow(valida
model.sm <- bank.boot.sm[[3]]
bank_train <- syn_data_borderline(bank.train.aug,5)### f
boost.bank.sm = gbm(y~.,data=bank.train.aug, distribution= "bernoulli", n.trees=1000,
                    shrinkage=0.01, verbose =F, cv.folds = 5)
prediction<-predict(boost.bank.sm,bank.test.aug,n.trees=1000,type='response')
prediction_class <- ifelse(prediction >0.5,1,0)
table <- table(bank.test.aug$y,prediction_class)
test_error = (mean(prediction_class!=bank.test.aug$y))
library(pROC)
roc_obj2 <- roc(prediction_class, bank.test.aug$y)
auc(roc_obj2)
plot(roc_obj2)


################### Boosting on modified data################################
library(gbm)
```

```
setwd("C:/Users/sunhwa/Documents/Spring2018/STOR565ML/group project/565project")

source("565_proj_func.R")
#PREPROCESSING MODIFIED TRAINING DATA
modified_test <-read.csv("modified_test.csv")
modified_train<-read.csv("modified_test.csv")

modified_test <- augmented_dataframe_combodata(modified_test)
names(modified_test)[25] <- "default_no"
names(modified_test)[26] <- "default_unknown"
names(modified_test)[27] <- "housing_no"
names(modified_test)[28] <- "housing_unknown"

modified_train <- augmented_dataframe_combodata(modified_train)
names(modified_train)[25] <- "default_no"
names(modified_train)[26] <- "default_unknown"
names(modified_train)[27] <- "housing_no"
names(modified_train)[28] <- "housing_unknown"

idx2 = sample(1:nrow(modified_train), size=0.75*nrow(modified_train))
#Split data
subtrain.mod<-modified_train[idx2,]
validate.mod<-modified_train[-idx2,]
modified.boot<-boost(subtrain.mod,validate.mod,1000,0.01)

boost.modified = gbm(y~.,data=modified_train, distribution= "bernoulli", n.trees=1000,
                     shrinkage=0.01, verbose =F, cv.folds = 5)

model <- modified.boot[[3]]
save(modified.boot,file="model.smoter")
###### to predict ###########
prediction<-predict(model,modified_test,n.trees=1000,type='response')
prediction_class <- ifelse(prediction >0.5,1,0)
save(prediction,file="prediction_mod.smotecsv")
table <- table(prediction_class,modified_test$y)
test_error = (mean(prediction_class!=modified_test$y))

#### AUC graph#################
#install.packages("pROC")
#### without smote
library(pROC)
roc_obj <- roc(prediction_class, modified_test$y)
auc(roc_obj)
plot(roc_obj)
```