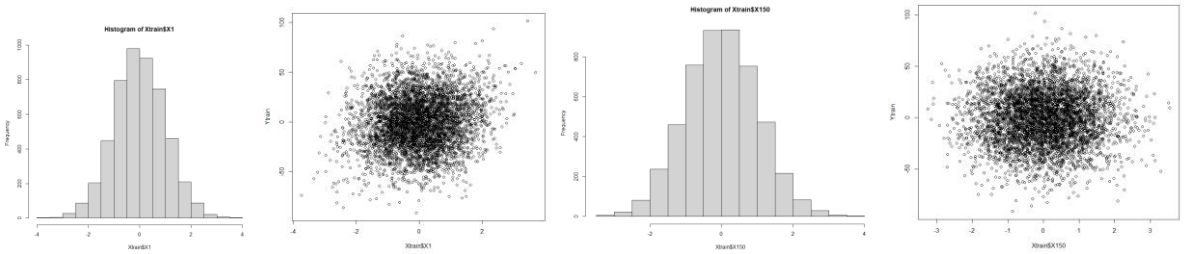
STAT 656 – Applied Analytics – Homework 6

Overview of Techniques

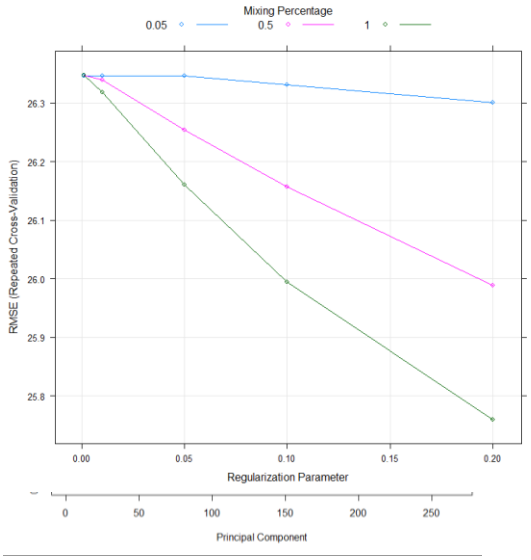
The dataset contains 5000 & 1000 observations in both training and testing with 265 features. All the required packages are initialized at once and the percentage of missing values in all the features is calculated. Looking at the distribution of few features, it looks like almost all of them have the similar structure.



The highest percentage of missing values was found to 0.26% considering both training and testing. Only one feature, X265 is found to be categorical among 265 features with three different levels and is converted to dummies. Missing values are imputed using KnnImpute & Mode_Impute for continuous variables & categorical variables respectively. Using Pearson correlation with a cutoff of 0.8, highly correlated are tried to be identified, but there are no such variables. After that, we check for skewness with a value of greater than 2, but surprisingly we could not find one.

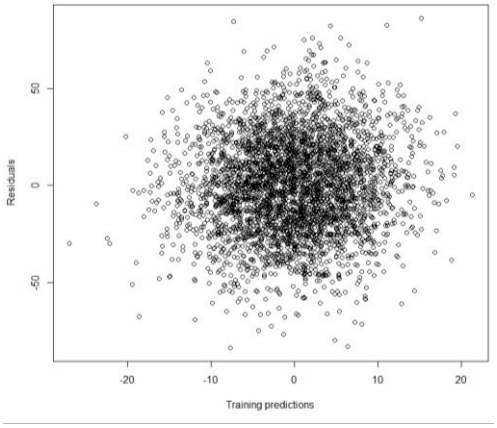
Here, I use PCA for dimension reduction, but looking at the scree plot, we could not find the elbow to find the number of principal components, nor the proportion of variance is explained properly. So, I dropped the idea of implementing PCA and move forward directly with building the model.

So, I am splitting the data into training and validation using the given training data where 75% is given for training and the remaining 25% for validation. I have selected 5 different methods like ElasticNet Regression, MARS, SVM, Decision Tree and Boosting. All these models have repeated cross-validation with 2 repeats and 10-fold cv.



For the **ElasticNet**, hyperparameters, alpha has a set of values from {0.001, ...,0 .2} and lambda from {0.05, …,1} we run the glmnet. From the plot, the best alpha and lambda values are found to be 1 and 0.2.

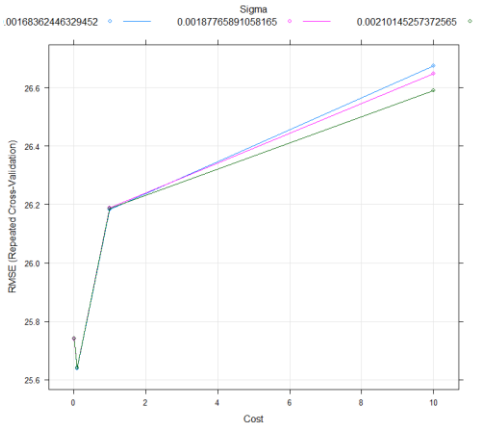
Refitting the elasticNet with the best parameters and looking at the coefficients of the glmnet, X1 and X33 are some of the strongly associated features with the supervisor.



The residual plot looks completely fine with no extreme observations.

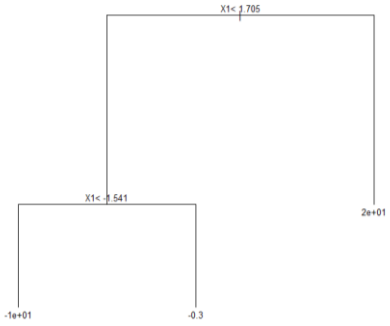
Next, the **MARS** model is used with hyperparameters degree(1 and 2) and nprune, the number of terms to retain between 15 and 20.

To calculate, the sigma for S**upport Vector Machines**, first we calculate the pairwise distance using Euclidean and then we get the final value from looking at the quantiles at 0.9, 0.5, and 0.1. The Cost parameter is set within the range of 0.01 and 10. Out of all the models, SVM has huge time complexity.

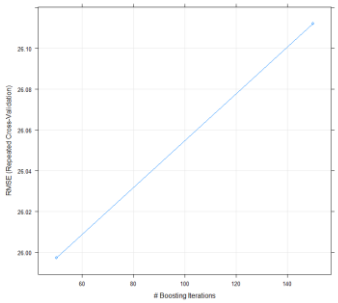


Sigma value of 0.001683624 and C value of 0.1 are selected as the best tuning parameters.

Decision Tree has a complexity parameter between 0.001 and 10 and after plotting the final model from the tree, we see that it gives more importance to the X1 feature and has only that feature in the tree.



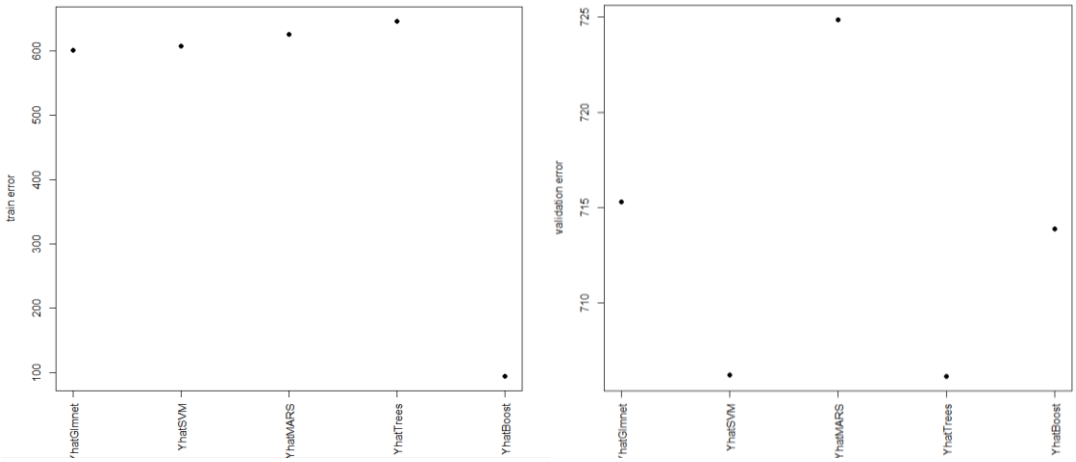
The **Boosting** model has many hyperparameters like nrounds, the number of trees with a value of 50 and 150, the maximum depth of a tree is set to 50 and the subsampling value is set to 0.5. Using method = "xgbTree" we fit the boosting model.



nrounds = 50 has the lowest value and the final model for boosting has the following parameters.

nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
50	50	0.05	0	1	0	0.5

Using the gain from the boosting model, we find that some of the important features are X1, X2, X6,… Finally, we come to the **prediction** part for all the five models using the training and validation data. All models except Boosting have the squared error above 600 on the training data and looking at the squared error on the validation data, it looks like Support Vector Machine and Trees have the lowest error rate.



Comparing the error rate between training and validation data, the boosting model tends to highly overfit the training data (Low Bias and High Variance). The Final model is chosen as the Support Vector Machine to get the predictions of the supervisor from the testing data. This model would give better performance on the unknown data compared to the other models.

Appendix:

```
yourName = 'naveenVeluchamy'#fill in your name, no spaces, leave quotes
load('Ytrain.Rdata')
load('Xtrain.Rdata')
load('Xtest.Rdata')
packs = c('dplyr','ggplot2','AppliedPredictiveModeling', 'caret','corrplot','doParallel',
      'glmnet','earth','kernlab','xgboost','ranger','rpart','missMethods')
lapply(packs,require,character.only=TRUE)
missingPercentXtrain = colMeans(is.na.data.frame(Xtrain))
missingPercentXtest = colMeans(is.na.data.frame(Xtest))
# The highest percentage of missing values was only 0.26% in a feature comparing the train and test
table(sapply(Xtrain, class))
#table(sapply(Xtest, class))
sapply(Xtrain,function(x){ length(unique(x)) })
unique(Xtrain$X265)
# X265 is categorical with 3 factors (a, b, and c). It also considers NA as a factor initially
plot(Xtrain$X1,Ytrain)
hist(Xtrain$X1)
plot(Xtrain$X150,Ytrain)
plot(Xtrain$X209,Ytrain)
#Imputation for training
XimputeKNNTrain = predict(preProcess(Xtrain, method=c("knnImpute")), Xtrain)
XfinalTrain = impute_mode(XimputeKNNTrain, type = "columnwise")
anyNA(XfinalTrain)
dummyModelTrain  = dummyVars(~ ., data = XfinalTrain, fullRank = FALSE)
XtrainFinal = predict(dummyModelTrain,XfinalTrain)
head(XtrainFinal)
#Imputation for testing
XimputeKNNTest = predict(preProcess(Xtest, method=c("knnImpute")), Xtest)
```

```
XfinalTest = impute_mode(XimputeKNNTest, type = "columnwise")
anyNA(XfinalTest)
dummyModelTest  = dummyVars(~ ., data = XfinalTest, fullRank = FALSE)
XtestFinal = predict(dummyModelTest,XfinalTest)
head(XtestFinal)
#Correlation plot
corrplot(cor(XtrainFinal), tl.cex = 0.5)
correlationmatrix<-cor(XtrainFinal, method = "pearson", use = "pairwise.complete.obs")
highlyCorrelated <- findCorrelation(correlationmatrix, cutoff=0.8, verbose=TRUE)
print(highlyCorrelated)
(highCorr = findCorrelation(cor(XtrainFinal), .8, verbose=TRUE, names = TRUE))
#It looks like no features are correlated
skewnessVec = XtrainFinal %>%
  sapply(., e1071::skewness)
names(XtrainFinal)[abs(skewnessVec)> 2]
# No features has skewness greater than 2
#principal component analysis
prin_comp <- prcomp(XtrainFinal, scale. = T)
names(prin_comp)
std_dev <- prin_comp$sdev
#compute variance
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)
plot(prop_varex, xlab = "Principal Component",
    ylab = "Proportion of Variance Explained",
    type = "b")
# PCA doesnt have a huge impact
set.seed(1)
inTrain        = createDataPartition(Ytrain, p = 3/4, list = FALSE)
```

```
YtrainFinal     = Ytrain[inTrain]
Yvalid          = Ytrain[-inTrain]
Xtrainsplit     = XtrainFinal[inTrain,]
Xvalid          = XtrainFinal[-inTrain,]
cl = makeCluster(4)
registerDoParallel(cl)
set.seed(1)
trControl = trainControl(method = "repeatedcv", repeats = 2, number = 10)
lassoGrid = expand.grid(lambda = c(0.001, .001, .01, .05, .1,.2),
                        alpha = c(0.05, 0.5,1))
elasticOut = train(x = Xtrainsplit, y = YtrainFinal,
                   method = "glmnet",
                   tuneGrid = lassoGrid,
                   trControl = trControl)
plot(elasticOut)
elasticOut$bestTune
# 1 and 0.2
glmnetOut       = glmnet(x = Xtrainsplit, y = YtrainFinal, alpha = elasticOut$bestTune$alpha)
betaHatGlmnet   = coef(glmnetOut, s = elasticOut$bestTune$lambda)
YhatTrainGlmnet = predict(glmnetOut, Xtrainsplit,  s = elasticOut$bestTune$lambda)
residuals = YtrainFinal - YhatTrainGlmnet
plot(YhatTrainGlmnet, residuals,
    xlab = 'Training predictions', ylab = 'Residuals')
points(YhatTrainGlmnet[c(extreme1,extreme2)],
      residuals[c(extreme1,extreme2)],
      col = 'red', pch=16)
# Residuals are looking good and it seems like there are no extreme observations
tuneGrid = expand.grid(degree = 1:2,
                       nprune = 15:20)
```

```
marsOut = train(x = Xtrainsplit, y = YtrainFinal,
                method = "earth",
                tuneGrid = tuneGrid,
                trControl = trControl)
#SVM
pairWiseDist = dist(scale(Xtrainsplit), method = 'euclidean')**2
sigmaRange = quantile(pairWiseDist, c(0.9,0.5,0.1))
hist(pairWiseDist)
abline(v = sigmaRange[1])
abline(v = sigmaRange[2])
abline(v = sigmaRange[3])
set.seed(1)
tuneGrid = expand.grid(C = c(.01,.1,1,10),
                       sigma = 1/sigmaRange)
svmOut = train(x = Xtrainsplit, y = YtrainFinal,
               method = "svmRadial",
               tuneGrid = tuneGrid,
               preProc = c("center", "scale"),
               trControl = trControl)
plot(svmOut)
#Prune Tree
tuneGrid = expand.grid(cp = c(0.001, 0.01, 0.1,1, 10))
rpartOut = train(x = Xtrainsplit, y = YtrainFinal,
                 method = "rpart",
                 tuneGrid = tuneGrid,
                 trControl = trControl)
plot(rpartOut$finalModel,margin= rep(.1,4))
text(rpartOut$finalModel, cex = 0.8, digits = 1)
#Boosting
```

```
set.seed(1)
tuneGrid = data.frame('nrounds'=c(50,150),
                      'max_depth' = 50,
                      'eta' = .05,
                      'gamma' = 0,
                      'colsample_bytree' = 1,
                      'min_child_weight' = 0,
                      'subsample' = .5)
boostOut   = train(x = Xtrainsplit, y = YtrainFinal,
                   method = "xgbTree", verbose = 0,
                   tuneGrid = tuneGrid,
                   trControl = trControl)
plot(boostOut)
boostOut$finalModel
boostImportance0 = xgb.importance(model = boostOut$finalModel)
boostImportance0
#performance
resultsTrain = data.frame(
  YhatGlmnet = c(predict(glmnetOut, Xtrainsplit,  s = elasticOut$bestTune$lambda)),
  YhatSVM    = predict(svmOut, Xtrainsplit),
  YhatMARS   = c(predict(marsOut, Xtrainsplit)),
  YhatTrees  = predict(rpartOut, Xtrainsplit),
  YhatBoost  = predict(boostOut, Xtrainsplit))
plot(1:length(resultsTrain), sapply(resultsTrain,function(Yhat){mean((Yhat - YtrainFinal)^2)}),
     xlab='',xaxt='n', pch = 16, ylab='train error')
axis(1,labels = names(resultsTrain),at =1:ncol(resultsTrain),las = 3)
resultsValid = data.frame(
  YhatGlmnet = c(predict(glmnetOut, Xvalid,  s = elasticOut$bestTune$lambda)),
  YhatSVM    = predict(svmOut, Xvalid),
```

```
  YhatMARS   = c(predict(marsOut, Xvalid)),
  YhatTrees  = predict(rpartOut, Xvalid),
  YhatBoost  = predict(boostOut, Xvalid))
plot(1:length(resultsValid), sapply(resultsValid,function(Yhat){mean((Yhat - Yvalid)^2)}),
     xlab='',xaxt='n', pch = 16, ylab='validation error')
axis(1,labels = names(resultsValid),at =1:ncol(resultsValid),las = 3)
Yhatsvmtest  = predict(svmOut, XtestFinal)
### get preds:
Yhat = data.frame('Yhat' = Yhatsvmtest)
#write.table
if(yourName == 'firstLast'){
  print('fill in your name!')
}else{
  fName = paste(c(yourName,'Predictions.txt'),collapse='')
  write.table(Yhat,file=fName,row.names=FALSE,col.names=FALSE)
}
```