DataMining_BackPain_Project

Hai Long, Le- 18200524

April 15, 2019

IMPORT PACKAGES AND DATA.

```
library(caret)
library(corrplot)
library(randomForest)
library(adabag)
library(ROCR)
library(tidyverse)
```

```
load('D:/UCD/Data Mining/Project/backpain (1).RData')
```

```
library(doParallel)
cl <- makeCluster(2)
registerDoParallel(cl)
```

Machine Learning is not just a simple step of applying the well-known algorithms to the Dataset to get the prediction accuracy. More than that, in order to get the best out of, Machine Learning requires multiple other steps such as: Feature Selection, Pre-Processing, Algorithm Evaluation, and Parameters Tuning. In my project, I will apply these steps to Backpain Dataset to get the best performance and make the Machine Learning even much more powerful. The main objective of this project is to build the Predictive Model for Binary Classification of Backpain Dataset to optimize the clinical outcomes. Using different of methods to train, evaluate, and discuss to find out the most effective method. Building the powerful Machine Learning Classifier is important but it is not the only factor of this project. I will also apply the knowledges I have learnt throughout this module to get other interesting and dynamic features to understand more about how the algorithms work such as the Variable Importance, Resampling techniques, Bias-Variance TradeOff, Model Comparison and Selection, etc. I will focus on 3 Classification methods we spent quite time in this module to train my model which are: Ensemble, Bagging, and Kernel. Moreover, I will perform the Tuning Parameters to improve the performance baseline models in order to have the best prediction on the unseen data.

1. INTRODUCTION

The Dataset “Backpain” we use for this project have 380 observations, 31 Predictors with Target is in binary class (Nociceptive/ Neuropathic). This Dataset is the collection of both categorical and numerical predictors. We are required to build the Binary Classifier to optimize the clinical outcomes by predicting either that person has Nociceptive or Neuropathic.

The main objective of this project is to apply the knowledges I have learnt in this Module to build the Machine Learning and get the most out of it, and evaluate using multiple performance metrics.

First of all, I will split my data into 3 parts: Train Set, Validation Set, and Test Set to allow how much of the data my Machine Learning model are allowed to see. After that, I will break my projects into 6 small tasks:

Task 1: Exploratory-Data-Analysis. This task will have me understand more about my data using Descriptive analysis. Task 2: Transforming Data. At this step I will transform my data to more standardized structure to help the machine learning model easily uncover my data *Task 3: Apply Algorithm to Baseline Models and Evaluate. At this step, I will apply 3 different methods(Ensemble, Bagging, Kernel) to train my baseline model and then evaluate them using Validation set.* Task 4: Improve Accuracy: I will apply some Feature Selection methods and PreProcessing to the Baseline models and evaluate Validation set to check for improvements. *Task 5: Tuning Parameters: I will select 2 best model from Baseline models to tune the parameters to improve the results.* Task 6: Finalize Model: I will select the “best” model to predict for unseen data in Test set and present the classification results.

Along with the small tasks above, I will answers some interesting questions relate to my machine learning model for this dataset. I would like to know both Bias and Variance of each Baseline models on the Train Set. Also, I believe that I would be helpful if I can know the prediction of my Baseline models on the Validation Set (underfitting, well-fitting, or overfitting). As I will apply the Pre-Processing to my models, I am interested in how beneficial the Pre-Processing steps will contribute to the overall accuracy of Validation Test. In this step, I will make the decision based on the Variable Importance given by Random Forest, and try to standardize and reduce data dimensions the data by Principle Components. Furthermore, I will try to figure out the best parameters of top 1 baseline models to improve the accuracy for the Test Set. The dataset is balanced data so I will access the performance of my models using Accuracy, Specificity, Sensitivity, and ROC curve.

2. DATA SPLITTING.

```
set.seed(123)
inTrain <- createDataPartition(y = dat$PainDiagnosis, p = 0.90, list= FALSE)
TrainandValidationSet <- dat[inTrain,]
TestSet <- dat[-inTrain,]
```

```
set.seed(123)
inTrain2 <- createDataPartition(y = TrainandValidationSet$PainDiagnosis, p = 0.80, lis
t= FALSE)
TrainSet <- TrainandValidationSet[inTrain2,]
ValidationSet <- TrainandValidationSet[-inTrain2,]
```

In this project, I divided my dataset into 3 different sets which are Train Set, Validation Set, and Test Set. Train Set is the set of data to train/ fit the models. The models can see the whole data to learn from it. Validation Set will be used to evaluate the model fit based on the Train Set. Validation Set will provide unbiased evaluation on the models. Using Validation Set, I will access the ability of model fit whether

overfitting or underfitting. The Test Set is to evaluate the final model fit on the unseen data to confirm the prediction power. I randomly selected 10% of my data as Test Set, 20% of remaining data as Validation Set, and the rest is Train Set.

3. EDA (EXPLORATORY DATA ANALYSIS)

```
glimpse(TrainSet)
```

```
## Observations: 275
## Variables: 32
## $ PainDiagnosis   <fct> Nociceptive, Neuropathic, Nociceptive, Neuropa...
## $ Age             <dbl> 34, 53, 45, 41, 32, 49, 54, 42, 29, 54, 25, 47...
## $ Gender          <fct> Female, Female, Female, Male, Female, Male, Ma...
## $ DurationCurrent <fct> 7-12 months, > 1 year, > 1 year, 4-6 weeks, 7-...
## $ PainLocation    <fct> Back, Back + Uni Leg, Back, Uni BK, Back, Back...
## $ SurityRating    <dbl> 9, 7, 9, 9, 9, 6, 9, 9, 6, 9, 9, 8, 8, 9, 10, ...
## $ RMDQ            <dbl> 15, 9, 10, 21, 14, 5, 1, 12, 12, 14, 14, 6, 14...
## $ vNRS            <dbl> 8.5, 3.0, 4.5, 8.0, 4.0, 1.0, 5.0, 5.5, 5.0, 5...
## $ SF36PCS         <dbl> 29.6, 41.9, 33.1, 21.2, 19.4, 28.4, 43.7, 42.4...
## $ SF36MCS         <dbl> 46.0, 35.3, 28.4, 27.2, 53.9, 34.6, 51.2, 44.5...
## $ PF              <dbl> 27.6, 42.3, 23.4, 17.0, 29.7, 44.4, 48.6, 46.5...
## $ BP              <dbl> 29.2, 37.2, 33.0, 19.9, 29.2, 19.9, 37.2, 24.9...
## $ GH              <dbl> 43.4, 43.4, 32.9, 36.2, 28.1, 28.1, 45.8, 59.1...
## $ VT              <dbl> 33.4, 39.6, 42.7, 20.9, 27.1, 33.4, 49.0, 52.1...
## $ MH              <dbl> 41.6, 38.7, 24.7, 30.3, 55.6, 33.1, 50.0, 38.7...
## $ HADSAnx         <dbl> 13, 7, 11, 13, 9, 11, 3, 5, 13, 6, 10, 11, 14,...
## $ HADSDep         <dbl> 5, 4, 16, 16, 4, 5, 3, 2, 8, 7, 4, 8, 12, 2, 1...
## $ Criterion2      <fct> Present, Present, Present, Present, Present, P...
## $ Criterion4      <fct> Absent, Absent, Absent, Absent, Absent, Absent...
## $ Criterion6      <fct> Absent, Absent, Absent, Absent, Absent, Absent...
## $ Criterion7      <fct> Absent, Present, Absent, Present, Absent, Abse...
## $ Criterion8      <fct> Present, Absent, Present, Absent, Present, Pre...
## $ Criterion9      <fct> Absent, Present, Absent, Present, Absent, Abse...
## $ Criterion10     <fct> Absent, Absent, Absent, Absent, Absent, Absent...
## $ Criterion13     <fct> Absent, Absent, Absent, Absent, Absent, Absent...
## $ Criterion19     <fct> Absent, Present, Present, Present, Absent, Pre...
## $ Criterion20     <fct> Present, Present, Present, Present, Present, P...
## $ Criterion26     <fct> Absent, Absent, Absent, Present, Absent, Absen...
## $ Criterion28     <fct> Present, Present, Present, Present, Present, P...
## $ Criterion32     <fct> Present, Absent, Present, Absent, Present, Pre...
## $ Criterion33     <fct> Absent, Absent, Absent, Absent, Absent, Absent...
## $ Criterion36     <fct> Absent, Absent, Absent, Absent, Absent, Absent...
```

Performing the Exploratory Data Analysis would help us to understand more about the dataset we are working with. There are 2 types of Data which are Numeric and Categorical in the dataset. We want to look into the set of Numerical and set of Categorical separately to see if the datatype of all variables make sense. In other words, I maybe miss-typed if the categorical data has too many levels.

```
Num_Var <- unlist(lapply(TrainSet, is.numeric))
Cat_Var <-  unlist(lapply(TrainSet, is.factor))
```

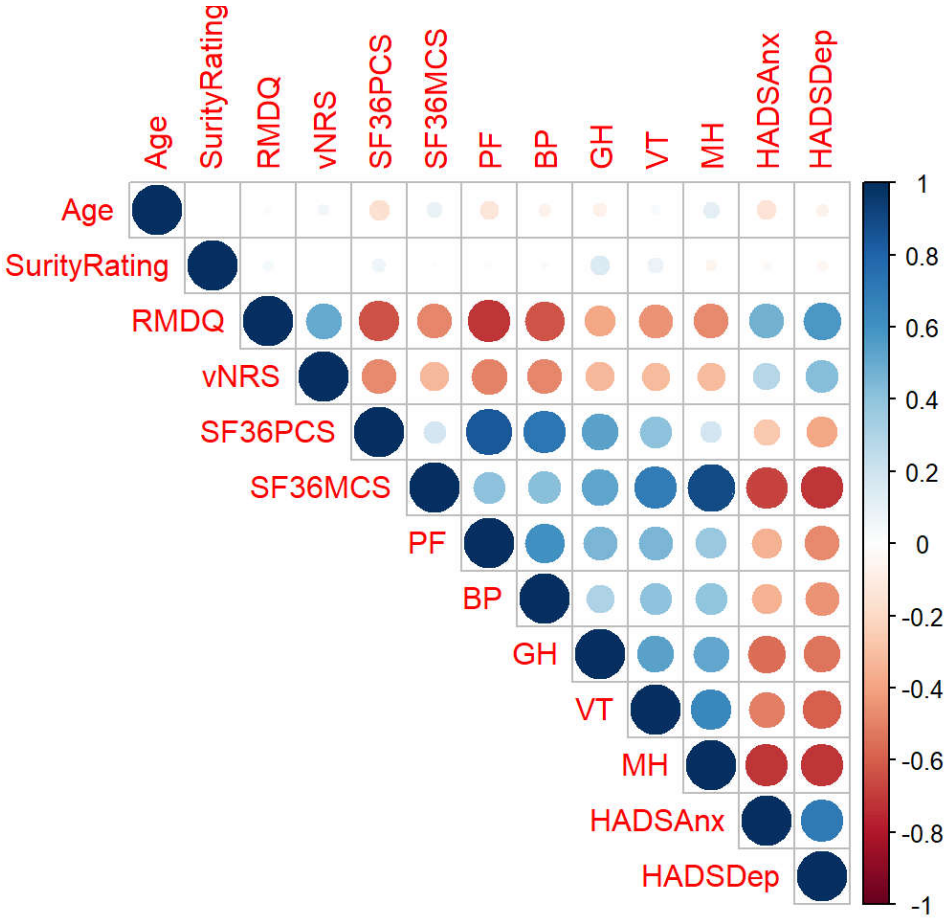
- The chunk of code above is to separate the numerical variables and categorical variables into 2 subsets.

```
str(TrainSet[,Cat_Var])
```

```
## 'data.frame':    275 obs. of  19 variables:
##  $ PainDiagnosis  : Factor w/ 2 levels "Nociceptive",..: 1 2 1 2 1 1 1 2 2 1 ...
##  $ Gender         : Factor w/ 2 levels "Female","Male": 1 1 1 2 1 2 2 2 1 1 ...
##  $ DurationCurrent: Factor w/ 6 levels "0-3 weeks","4-6 weeks",..: 5 6 6 2 3 6 4 1
4 3 ...
##  $ PainLocation   : Factor w/ 6 levels "Back","Back+Thigh",..: 1 5 1 3 1 1 1 3 2 1
...
##  $ Criterion2     : Factor w/ 2 levels "Absent","Present": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Criterion4     : Factor w/ 2 levels "Absent","Present": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Criterion6     : Factor w/ 2 levels "Absent","Present": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Criterion7     : Factor w/ 2 levels "Absent","Present": 1 2 1 2 1 1 1 2 1 1 ...
##  $ Criterion8     : Factor w/ 2 levels "Absent","Present": 2 1 2 1 2 2 2 1 1 2 ...
##  $ Criterion9     : Factor w/ 2 levels "Absent","Present": 1 2 1 2 1 1 1 2 2 1 ...
##  $ Criterion10    : Factor w/ 2 levels "Absent","Present": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Criterion13    : Factor w/ 2 levels "Absent","Present": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Criterion19    : Factor w/ 2 levels "Absent","Present": 1 2 2 2 1 2 2 2 1 1 ...
##  $ Criterion20    : Factor w/ 2 levels "Absent","Present": 2 2 2 2 2 2 2 1 2 2 ...
##  $ Criterion26    : Factor w/ 2 levels "Absent","Present": 1 1 1 2 1 1 1 1 1 1 ...
##  $ Criterion28    : Factor w/ 2 levels "Absent","Present": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Criterion32    : Factor w/ 2 levels "Absent","Present": 2 1 2 1 2 2 2 1 1 2 ...
##  $ Criterion33    : Factor w/ 2 levels "Absent","Present": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Criterion36    : Factor w/ 2 levels "Absent","Present": 1 1 1 1 1 1 1 1 1 1 ...
```

Correlation Plot of Numeric Variable

```
corrplot(cor(TrainSet[,Num_Var]), method="circle", type= "upper")
```



Some of numeric variables have very strong correlation (Pearson type), this is a strong indication of using PCA to pre-process data in later steps.

```
prop.table(table(TrainSet$PainDiagnosis))
```

```
##
## Nociceptive Neuropathic
##   0.5454545   0.4545455
```

The ratio of 2 levels of response variable (Nociceptive and Neuropathic) is quite balanced in Train Data. We do not need to worry about Imbalanced Data in this Dataset.

3. Feature Selection

3.1 NZV and ZV

```
Near_Zero_Variance = nearZeroVar(TrainSet, saveMetrics = T)
Near_Zero_Variance[Near_Zero_Variance[, "zeroVar"] > 0, ]
```

```
## [1] freqRatio     percentUnique zeroVar       nzv
## <0 rows> (or 0-length row.names)
```

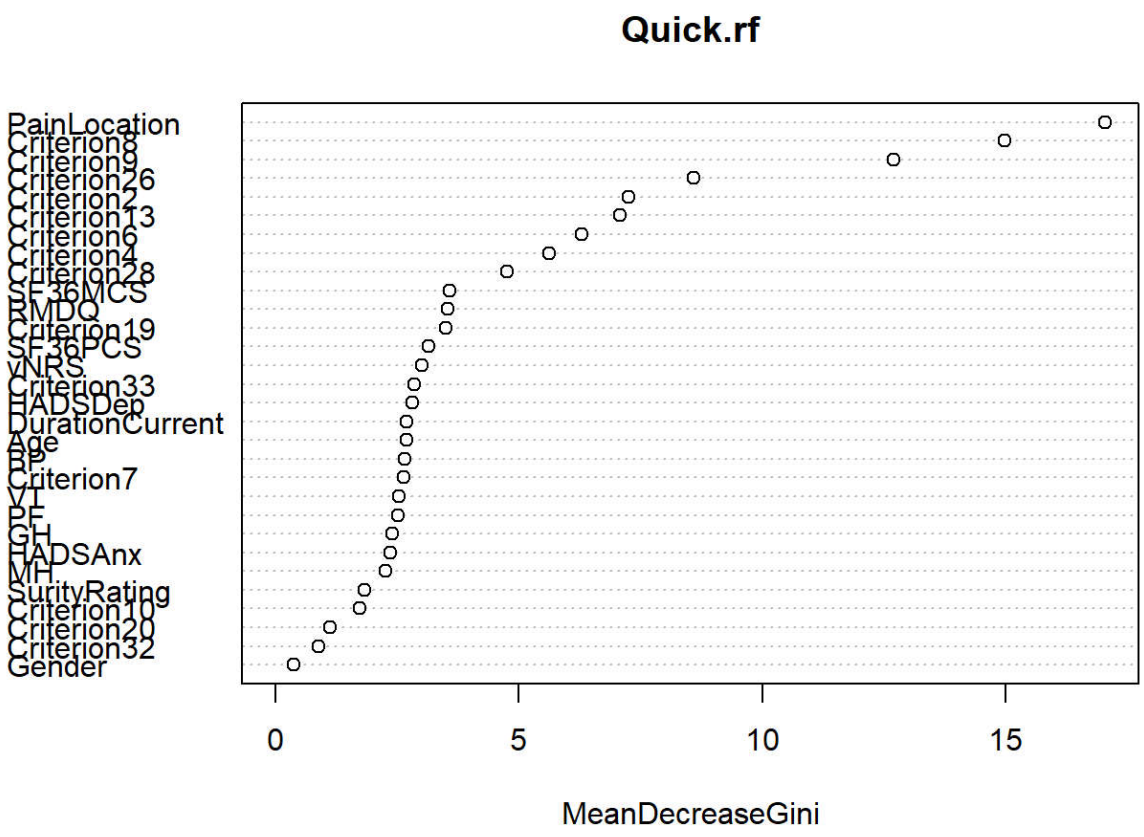
```
Near_Zero_Variance[Near_Zero_Variance[, "nzv" ] > 0, ]
```

```
## [1] freqRatio     percentUnique zeroVar       nzv
## <0 rows> (or 0-length row.names)
```

3.2 Variable Selection using Variable Importance

Variable Selection is an important step of model building. The Variable Selection is expected to improve the prediction accuracy by eliminating the unnecessary variables to remove the noise and improve computation efficiency. Building the “quick random forest” with 500 trees and accessing the Variable Importance, I will remove few least contributing Variables based on Mean Decrease Gini. The Predictors with low Mean Decrease Gini will not appear in many splits in tree-based models.

```
set.seed(123)
Quick.rf <- randomForest(PainDiagnosis ~ ., data=TrainSet, ntree=1000)
varImpPlot(Quick.rf,type=2)
```



The Variables have low Mean Decrease Gini have direct proportion to their participation in growing trees. We can remove some least important variables such as Gender, Criterion32, Criterion20, Criterion10

4. Model Building.

4.1 Baseline Experiment with methods.

In my Baselines models, I have used 4 different methods which are Random Forest, Bagging, Boosting, and SVM. After that, I will access the Bias Error and Variance Error of these baseline models.

```
set.seed(123)
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
```

“Repeated Cross-Validation” is a powerfulresampling technique to estimate the accuracy of the model. In my project, with 10-fold and each fold will be repeated the estimation for 3 times. In other words, it will divide my data into 10 folds, 9 folds will be use to train the model and then test the error rate with 1 fold left, and it will be repeated for 3 times. The error rate of estimations will be averaged out to have the expected error of the whole model. Using this Repeated CV not only gives the better estimation of model performance but also reduce the Bias to improve the Prediction Accuracy.

Ensemble Method (Random Forest)

```
set.seed(123)
fit.rf <- train(PainDiagnosis~., data=TrainSet, method="rf", metric=metric, trControl=
trainControl)
```

Random Forest: is a powerful classification method using tree-based model with bootstrapping to make it more stable and reliable. Random Forest will take the bootstrap of the data, build the classification tree at each split using only the random subset of variables and then test with OutOf-Bag samples.

Bagging (adabag)

```
set.seed(123)
fit.bagging <- train(PainDiagnosis~., data=TrainSet, method="treebag", metric=metric,
trControl=trainControl)
```

Bagging: is also known as Bootstrap Aggregation. Bagging works quite similar to Random Forest. The main difference between these 2 methods is Bagging does not create the subset of variable at random. Theoretically, Random Forest is expected to be more powerful than Bagging.

SVM

```
set.seed(123)
fit.svmRadial <- train(PainDiagnosis~., data=TrainSet, method="svmPoly", metric=metri
c, trControl=trainControl)
```

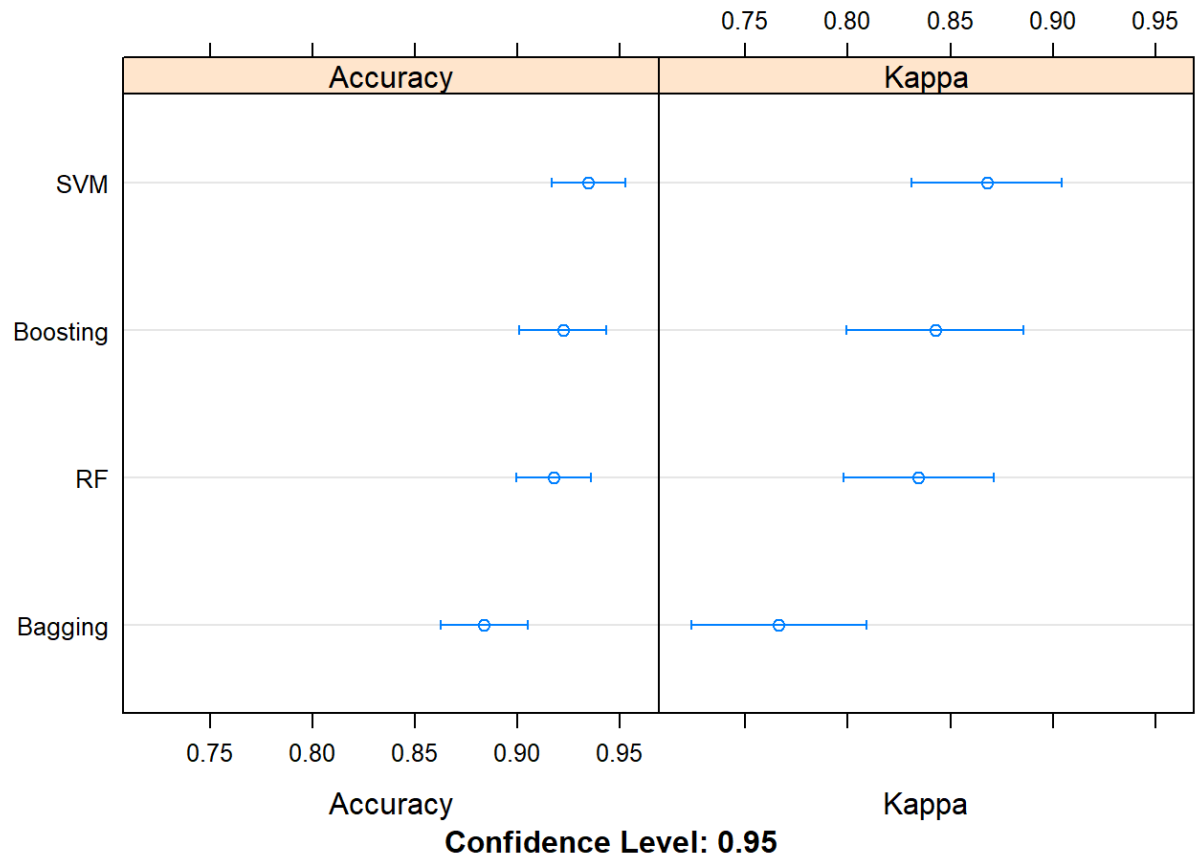
SVM: is the method classifies data using hyperplane. SVM is solved using the Constrained Optimization problem by finding the Maximum Margin Hyperplane, and applying Cost function. For Non-Linearity, It transform the data to high-dimension to separate the linear plane easier

Boosting

```
fit.boosting <- train(PainDiagnosis~., data=TrainSet, method="adaboost", metric=metri
c, trControl=trainControl)
```

Boosting: is another ensemble technique to create a strong classifier from many weak learner. Weak learners are prepared on the training data using Weighting. At each iteration, the misclassified observations will be upweighted. After that, Boosting classifies the Testing data using the calculated weighting

```
results <- resamples(list(RF = fit.rf, Bagging = fit.bagging, SVM= fit.svmRadial, Boos
ting= fit.boosting))
dotplot(results)
```



Accessing the Fit of Baseline Models with Validation Test

```
validation.predict.rf <- predict(fit.rf, ValidationSet)
ConfMat.RF <- confusionMatrix(validation.predict.rf, ValidationSet$PainDiagnosis)
ConfMat.RF$overall['Accuracy']
```

```
##  Accuracy
## 0.9411765
```

```
validation.predict.boosting <- predict(fit.boosting, ValidationSet)
ConfMat.Boosting <- confusionMatrix(validation.predict.boosting, ValidationSet$PainDia
gnosis)
ConfMat.Boosting$overall['Accuracy']
```

```
##  Accuracy
## 0.9411765
```

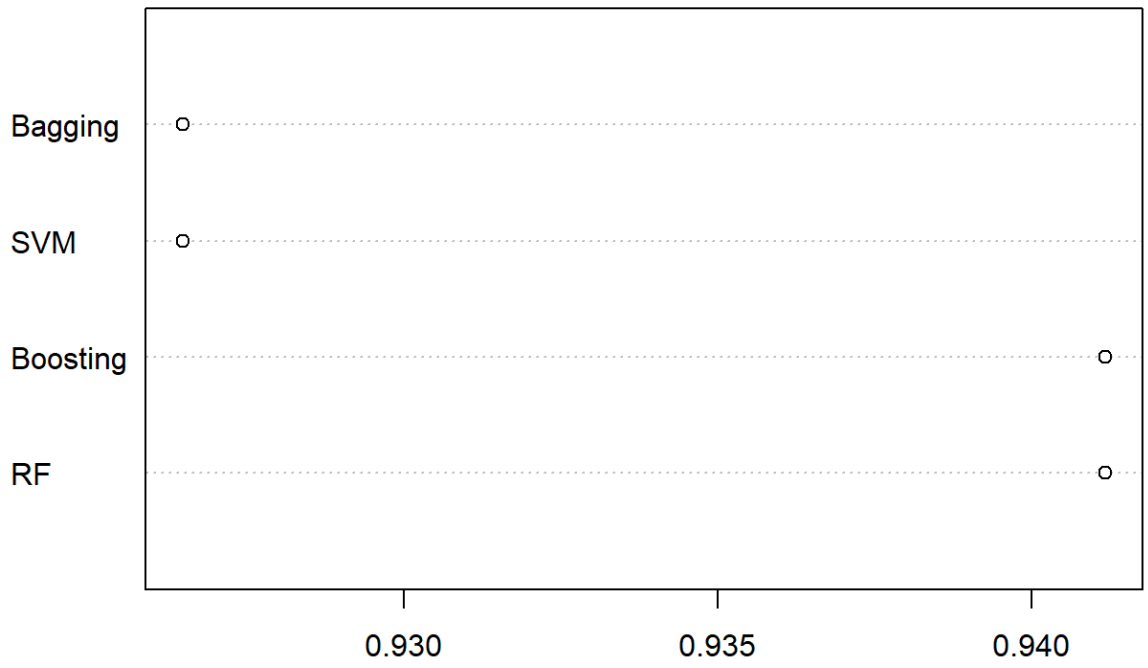
```
validation.predict.svm <- predict(fit.svmRadial, ValidationSet)
ConfMat.svm <- confusionMatrix(validation.predict.svm, ValidationSet$PainDiagnosis)
ConfMat.svm$overall['Accuracy']
```

```
##  Accuracy
## 0.9264706
```

```
validation.predict.bagging <- predict(fit.bagging, ValidationSet)
ConfMat.bagging <- confusionMatrix(validation.predict.bagging, ValidationSet$PainDiagn
osis)
ConfMat.bagging$overall['Accuracy']
```

```
##  Accuracy
## 0.9264706
```

```
Validation_Result <- c(ConfMat.RF$overall['Accuracy'] , ConfMat.Boosting$overall['Accu
racy'],
                       ConfMat.svm$overall['Accuracy'] , ConfMat.bagging$overall['Accu
racy'])
Validation_Method <- c("RF", "Boosting", "SVM", "Bagging")
dotchart(Validation_Result,labels=Validation_Method)
```



In my Baselines models, I have used 4 different methods which are Random Forest, Bagging, Boosting, and SVM. Looking at the Accuracy metrics for Training Set, SVM performs the best among methods, Boosting and Random Forest have quite similar accuracy. However, Random Forest seems to have the lowest variance.

The Accuracy for all 4 Baseline Models significantly changes when we test with the Validation Test. Boosting and Random Forest have the same Accuracy and outperform SVM and Bagging. Comparing the Accuracy of Training Data and Validation Data, SVM and Bagging overfitting, Boosting is underfitting, and Random Forest is well fit. At this stage, I think Random Forest is chosen model because it gives the most consistent accuracy among 4 models and it is the “best” model in term of Bias-Variance Trade-Off.

4.2 Improve Accuracy with Variable Selection (VarImp), Standardized Data, and PCA.

```
Trainset_Select <- TrainSet
```

From the Variable Selection step above using VarImp method, I could find out some unimportant variables. In other word, I will not contribute any information to to tree-based model in model building. Moreover, I will also eliminate some unwanted noise in building model.

```
Trainset_Select$Gender <- NULL
Trainset_Select$Criterion32 <- NULL
Trainset_Select$Criterion20 <- NULL
Trainset_Select$SurityRating <- NULL
```

```
preProcess2 <- preProcess(Trainset_Select[,-1], method= c("center", "scale", "pca"))
Trainset_Select[,-1] <- predict(preProcess2, Trainset_Select[,-1])
```

Standardize Data will make the attributes have Mean of 0, Variance of 1. In other words, Standardized data will treat all the variables in our data equally. It also supports PCA.

PCA is a multivariate technique to deal with highly correlated variable, and to perform the dimensionality reduction with minimal loss of the information. PCA will extract the important information of the whole dataset and will express with fewer new variables. After performing the EDA of the dataset using Pearson Correlation for Continuous Variables, I have figured out that there are some highly correlated variables so It may useful to apply PCA to data.

preProcess Summary

```
preProcess2
```

```
## Created from 275 samples and 27 variables
##
## Pre-processing:
##   - centered (12)
##   - ignored (15)
##   - principal component signal extraction (12)
##   - scaled (12)
##
## PCA needed 9 components to capture 95 percent of the variance
```

The Variables have low Mean Decrease Gini have direct proportion to their participation in growing trees. We can remove some least important variables such as Gender, Criterion32, Criterion20, Criterion10 from VarImp. Furtheremore, by applying these 2 techniques to my data, I successfully reduced 7 columns in Dataset but still can expect almost full information.

```
set.seed(123)
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
```

Random Forest

```
set.seed(123)
fit.rf.select <- train(PainDiagnosis~., data=Trainset_Select, method="rf", metric=metr
ic, trControl=trainControl)
```

Bagging (treebag)

```
set.seed(123)
fit.bagging.select <- train(PainDiagnosis~., data=Trainset_Select, method="treebag", m
etric=metric, trControl=trainControl)
```

SVM

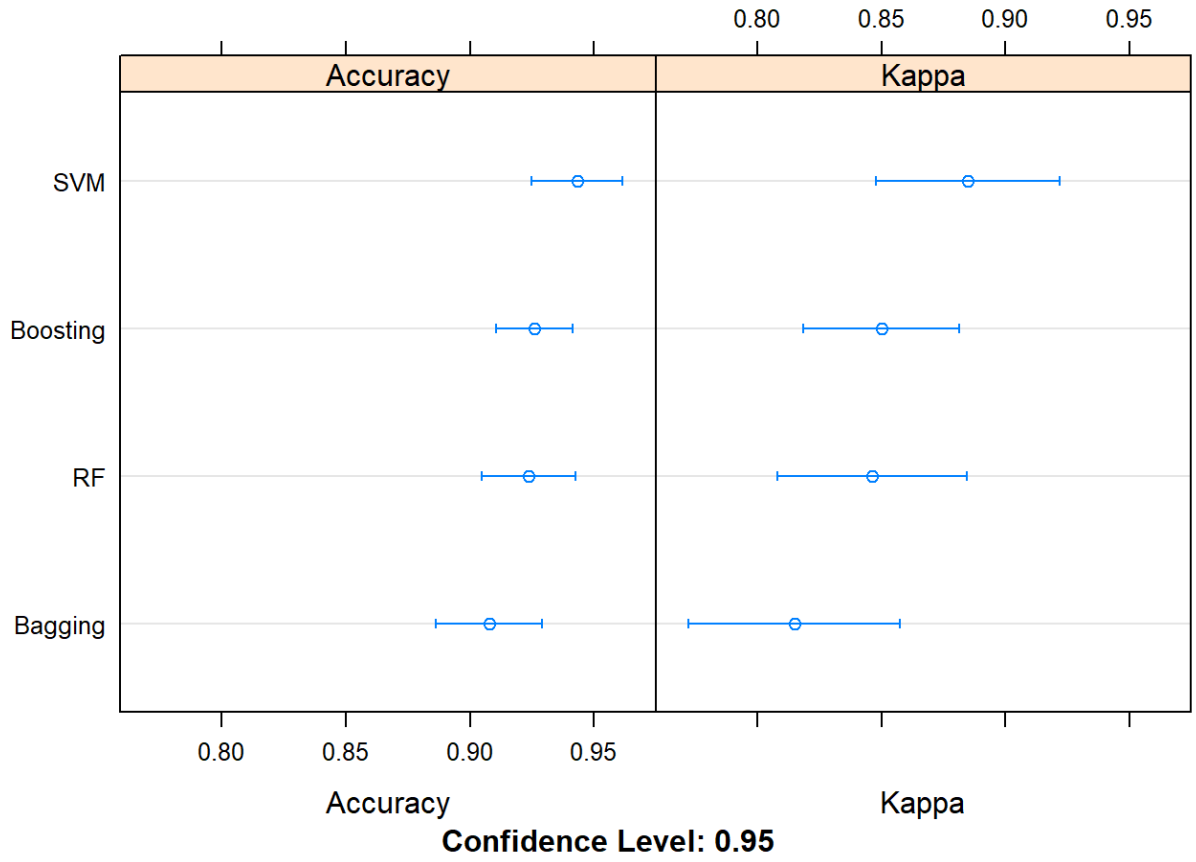
```
set.seed(123)
fit.svmRadial.select <- train(PainDiagnosis~., data=Trainset_Select, method="svmPoly",
metric=metric,trControl=trainControl)
```

Boosting

```
fit.boosting.select <- train(PainDiagnosis~., data=Trainset_Select, method="adaboost",
metric=metric, trControl=trainControl)
```

Result of Traning Set

```
results.select <- resamples(list(RF = fit.rf.select, Bagging = fit.bagging.select, SVM
= fit.svmRadial.select, Boosting= fit.boosting.select))
dotplot(results.select)
```



Test Pre-Process with Validation Test

```
ValidationSet.Select <- ValidationSet
```

```
ValidationSet.Select$Gender <- NULL
ValidationSet.Select$Criterion32 <- NULL
ValidationSet.Select$Criterion20 <- NULL
ValidationSet.Select$SurityRating <- NULL
```

```
ValidationSet.Select[,-1] <- predict(preProcess2, ValidationSet.Select[,-1])
```

```
validation.predict.rf.select <- predict(fit.rf.select, ValidationSet.Select)
ConfMat.RF.select <- confusionMatrix(validation.predict.rf.select, ValidationSet.Selec
t$PainDiagnosis)
ConfMat.RF.select$overall['Accuracy']
```

```
##  Accuracy
## 0.9264706
```

```
validation.predict.boosting.select <- predict(fit.boosting.select, ValidationSet.Selec
t)
ConfMat.Boosting.select <- confusionMatrix(validation.predict.boosting.select, Validat
ionSet.Select$PainDiagnosis)
ConfMat.Boosting.select$overall['Accuracy']
```

```
##  Accuracy
## 0.9264706
```

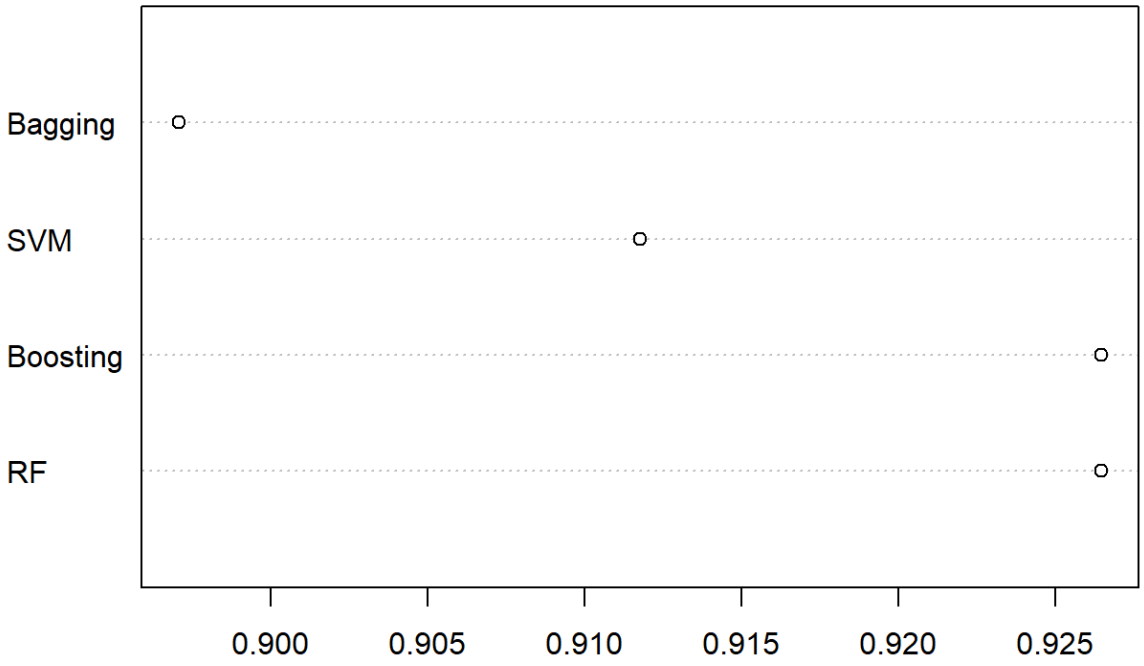
```
validation.predict.svm.select <- predict(fit.svmRadial.select, ValidationSet.Select)
ConfMat.svm.select <- confusionMatrix(validation.predict.svm.select, ValidationSet.Sel
ect$PainDiagnosis)
ConfMat.svm.select$overall['Accuracy']
```

```
##  Accuracy
## 0.9117647
```

```
validation.predict.bagging.select <- predict(fit.bagging.select, ValidationSet.Select)
ConfMat.bagging.select <- confusionMatrix(validation.predict.bagging.select, Validatio
nSet.Select$PainDiagnosis)
ConfMat.bagging.select$overall['Accuracy']
```

```
##  Accuracy
## 0.8970588
```

```
Validation_Result.select <- c(ConfMat.RF.select$overall['Accuracy'] , ConfMat.Boostin
g.select$overall['Accuracy'],
                      ConfMat.svm.select$overall['Accuracy'] , ConfMat.bagging.select
$overall['Accuracy'])
Validation_Method.select <- c("RF", "Boosting", "SVM", "Bagging")
dotchart(Validation_Result.select,labels=Validation_Method.select)
```



The Accuracy of models after Pre-Processing and Variable Selection on training data are quite identical to Baseline Models, except the accuracy of Bagging is slightly improve. Similarly, The Accuracy of models after Pre-Processing and Variable Selection on Validation data is quite similar. However, The SVM is well fits and no longer overfitting as in Baseline Models. Random Forest still be the most consistent model in both Training and Validation data, in both Baseline and after transformation. I decided to apply Pre-Processing and Variable Selection to Random Forest, to perform Tuning Parameters to improve the prediction ability of the model. This is also known as my Final Model. After Tuning Parameter, I would use my Final Model to predict for Test Data. Moreover, I would use the ROC metrics to tune parameters instead of Accuracy. In the next step, I will apply Tuning Parameter technique to the Pre-Processing and Variable Selection to Random Forest.

5. Tuning Parameters.

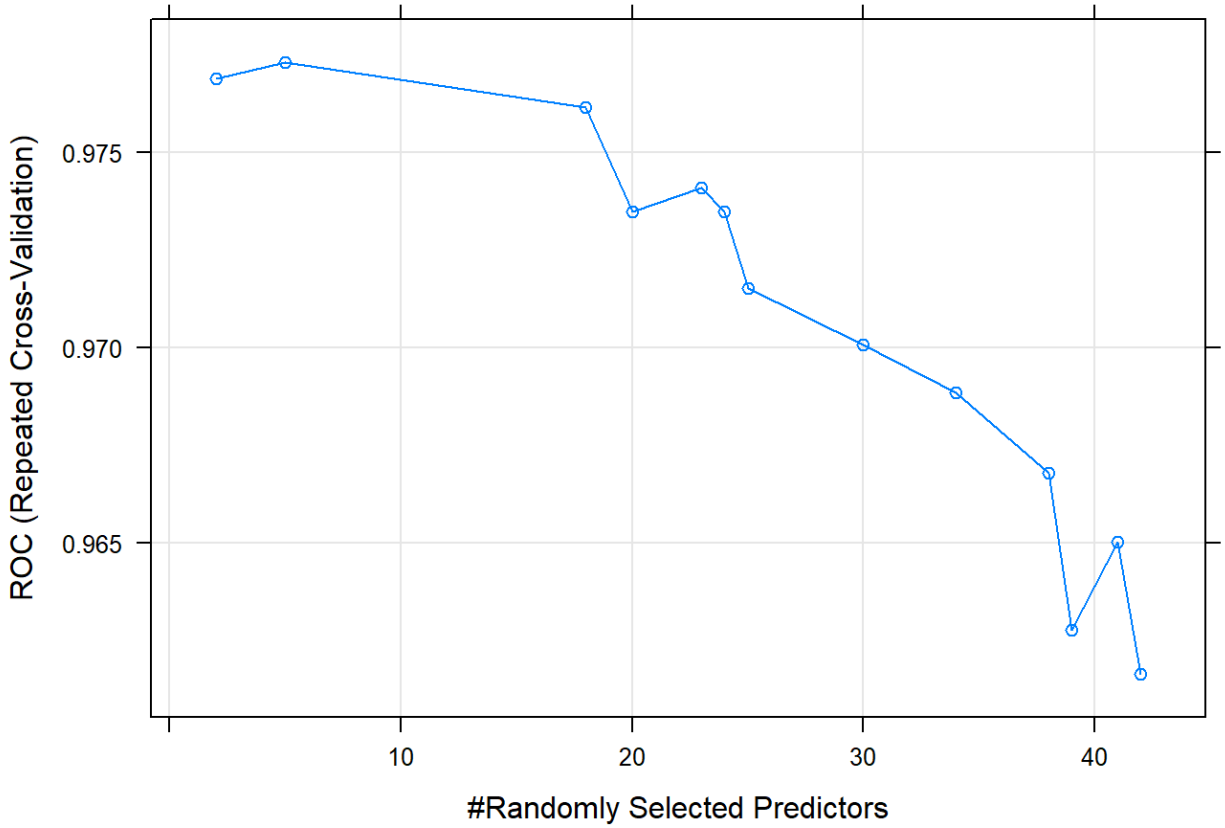
From the Baseline Models, we would choose the “best” model in term of Bias-Variance Tradeoff to tune the parameters. Tuning Parameters is to find the optimal parameters for algorithm to make the powerful machine learning perform even better. Each machine learning will have unique parameters to tune. The algorithms are parameterized are expected to perform better.

```
# Random Search
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3, search="rando
m",summaryFunction = twoClassSummary,classProbs = TRUE)
set.seed(123)
metric <- "ROC"
mtry <- sqrt(ncol(Trainset_Select[,-1]))
rfRandom <- train(PainDiagnosis~., data=Trainset_Select, method="rf", metric=metric, t
uneLength=15, trControl=trainControl)
```

```
rfRandom
```

```
## Random Forest
##
## 275 samples
##  27 predictor
##   2 classes: 'Nociceptive', 'Neuropathic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 248, 248, 248, 247, 247, 247, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.9768946  0.9711111  0.8512821
##    5    0.9773219  0.9600000  0.9072650
##   18    0.9761610  0.9400000  0.9149573
##   20    0.9734900  0.9400000  0.9149573
##   23    0.9741026  0.9355556  0.9149573
##   24    0.9734900  0.9311111  0.9123932
##   25    0.9715171  0.9244444  0.9123932
##   30    0.9700712  0.9133333  0.9123932
##   34    0.9688533  0.9177778  0.9098291
##   38    0.9668020  0.9000000  0.9098291
##   39    0.9627707  0.9000000  0.9072650
##   41    0.9650142  0.8977778  0.9096154
##   42    0.9616311  0.9088889  0.9070513
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

```
plot(rfRandom)
```




```
print(rfRandom)
```

```
## Random Forest
##
## 275 samples
##  27 predictor
##   2 classes: 'Nociceptive', 'Neuropathic'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 248, 248, 248, 247, 247, 247, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.9768946  0.9711111  0.8512821
##    5    0.9773219  0.9600000  0.9072650
##   18    0.9761610  0.9400000  0.9149573
##   20    0.9734900  0.9400000  0.9149573
##   23    0.9741026  0.9355556  0.9149573
##   24    0.9734900  0.9311111  0.9123932
##   25    0.9715171  0.9244444  0.9123932
##   30    0.9700712  0.9133333  0.9123932
##   34    0.9688533  0.9177778  0.9098291
##   38    0.9668020  0.9000000  0.9098291
##   39    0.9627707  0.9000000  0.9072650
##   41    0.9650142  0.8977778  0.9096154
##   42    0.9616311  0.9088889  0.9070513
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

For Random Forest model, I would tune the “mtry” which is the random selected Predictors. In others words, my tuning parameter is the number of selected predictor at each split. From the result of tuning parameter for Random Forest, the best “mtry” for my model on Training Data is at 5 because it has the highest ROC of 97.73% for Training Data

6. Test the Final model with Test Data

```
TestSet$Gender <- NULL
TestSet$Criterion32 <- NULL
TestSet$Criterion20 <- NULL
TestSet$SurityRating <- NULL
```

- Eliminate the unimportant variables found from Variable Selection steps above.

```
TestSet[,-1] <- predict(preProcess2, TestSet[,-1])
```

- PreProcess the Data similar to previous step.

```
TestSet.Predict <- predict(rfRandom, TestSet)
Final <- confusionMatrix(TestSet.Predict, TestSet$PainDiagnosis)
Final
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nociceptive Neuropathic
##   Nociceptive          19           1
##   Neuropathic           1          16
##
##                Accuracy : 0.9459
##                  95% CI : (0.8181, 0.9934)
##     No Information Rate : 0.5405
##     P-Value [Acc > NIR] : 6.688e-08
##
##                   Kappa : 0.8912
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9500
##             Specificity : 0.9412
##          Pos Pred Value : 0.9500
##          Neg Pred Value : 0.9412
##              Prevalence : 0.5405
##          Detection Rate : 0.5135
##    Detection Prevalence : 0.5405
##       Balanced Accuracy : 0.9456
##
##        'Positive' Class : Nociceptive
##
```

- Predict the Test Set using final model and accessing the accuracy metric.

Finally, I applied my “parameterized final model” to predict for Test Data and accessing the Performance Metrics by Confusion Matrix and ROC curve. Looking at the Confusion Matrix, Accuracy is 94.59%, which means my Final Model correctly classified 35 observation out of 37 observations in Test Data without looking at Test Data in advanced. We have balanced Responses (Nociceptive, Neuropathic) so the Balanced Accuracy is quite high as expected which is 94.56%. The Sensitivity is 95.00%, which means 19 out 20 observations were correctly classified as Nociceptive. Similarly, The Specificity is 94.12%, which means 16 out of 17 observations from Neuropathic were correctly classified.

We have balanced class data, besides the Accuracy, I will access other performance metrics such as Balanced Accuracy, Sensitivity and Specificity. Because of Binary Classification, I would pay more attention on ROC Curve to have a deeper look into Sensitivity and Specificity.

Convert Predicted Values of Test Set to Probablity for ROC curve.

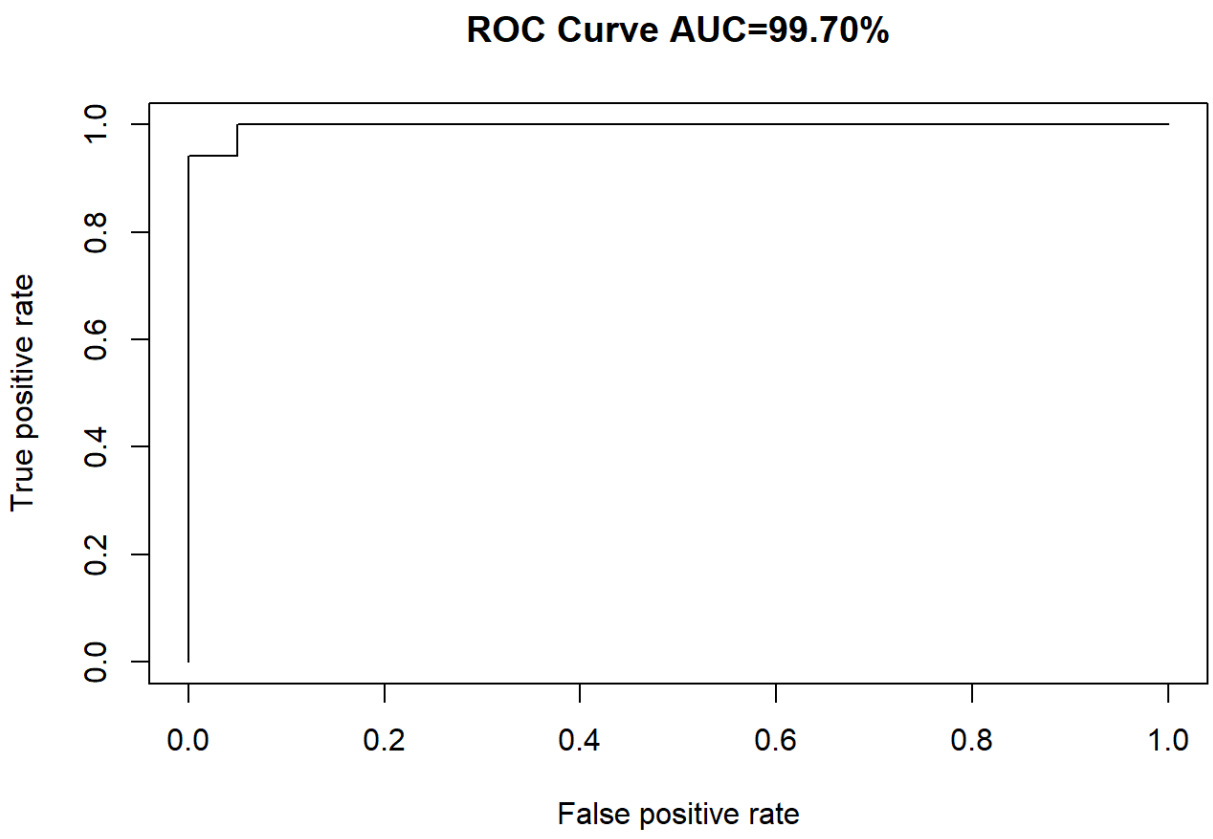
```
Pred_prob <-  predict(rfRandom, TestSet, type = "prob")
Pred_prob_Neuro <-  as.data.frame(Pred_prob)
Pred_prob_Neuro <- Pred_prob_Neuro[,2]
```

```
ROC_Actual = ifelse(TestSet$PainDiagnosis == "Nociceptive",0,1)
```

```
library(ROCR)
predobj<-prediction(as.numeric(Pred_prob_Neuro), as.numeric(ROC_Actual))
```

```
perf <- performance(predobj,"tpr","fpr")
```

```
plot(perf, main = "ROC Curve AUC=99.70%")
```




```
auc2 <- performance(predobj, measure = "auc")
auc2@y.values[[1]]
```

```
## [1] 0.9970588
```

```
cutoffs <- data.frame(cut=perf@alpha.values[[1]], fpr=perf@x.values[[1]],
                      tpr=perf@y.values[[1]])
head(subset(cutoffs, tpr >= 0.9))
```

```
##      cut  fpr       tpr
## 17 0.538 0.00 0.9411765
## 18 0.518 0.05 0.9411765
## 19 0.486 0.05 1.0000000
## 20 0.450 0.10 1.0000000
## 21 0.294 0.15 1.0000000
## 22 0.274 0.20 1.0000000
```

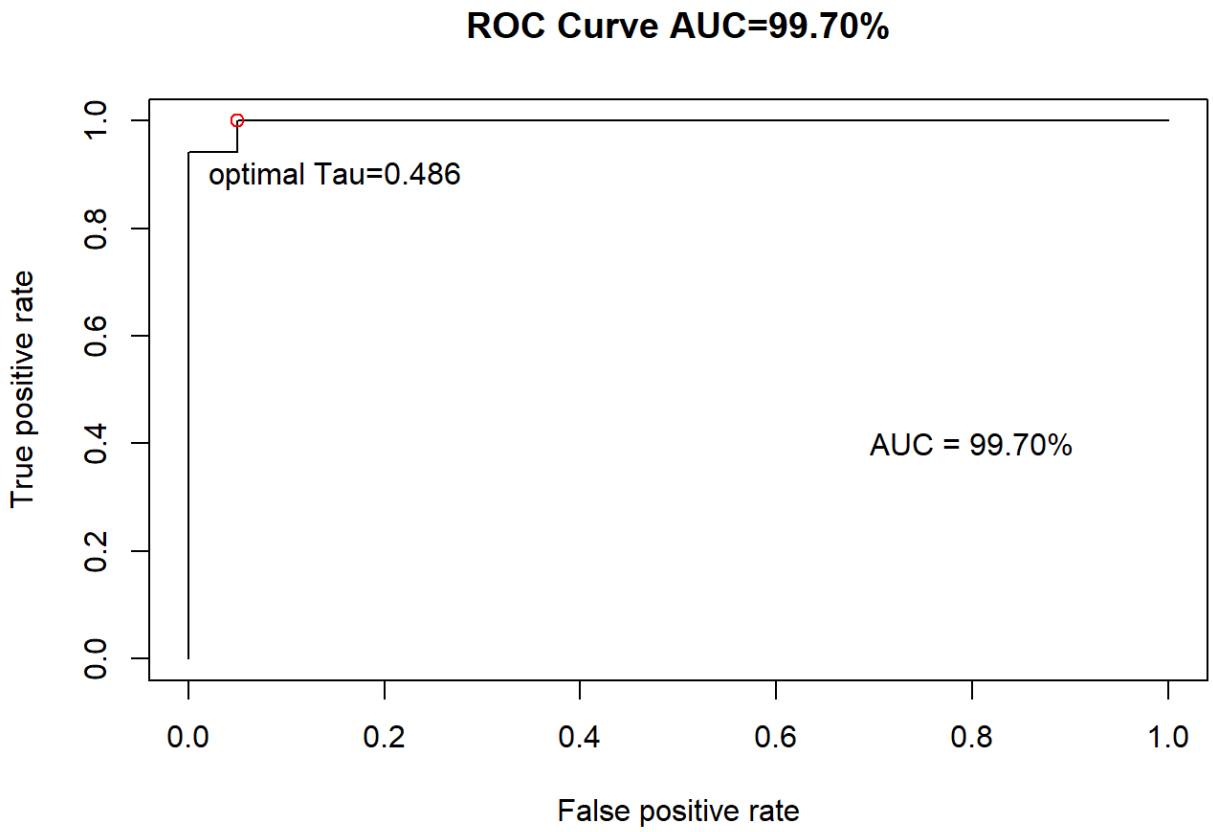
```
# Maximize Sum of Specificity and Sensitivity find optimal Threshold.
cutoffs$Sum <- cutoffs$tpr - cutoffs$fpr
head(subset(cutoffs, Sum >= 0.95))
```

```
##      cut  fpr tpr  Sum
## 19 0.486 0.05   1 0.95
```

```
perf2 <- performance(predobj, "sens","spec")
t.ind <- which.max(unlist(perf2@x.values) + unlist(perf2@y.values))
tau   <- unlist(perf2@alpha.values)[t.ind]
tau
```

```
## [1] 0.486
```

```
plot(perf, main = "ROC Curve AUC=99.70%")
points(0.05, y =1, type = "p", col="red")
text(0.8, 0.4, "AUC = 99.70%")
text(0.15, 0.9, "optimal Tau=0.486")
```



The ROC is a more in-depth look into the Sensitivity and Specificity. Particularly, the ROC of Binary Classification problem is the trade-off between Sensitivity and Specificity. In order to measure ROC, we need to have the Predicted value as Probability. The ROC Curve for my “Parameterized Final Model” on Test Data is very close to 45-degree line, which means the AUC is extremely close to one. In fact, the Area Under Curve (AUC) for my ROC curve is 99.70%. Moreover, the optimal threshold for my ROC curve is at tau=0.486. At this threshold, the sum of Sensitivity and Specificity is maximized (Sensitivity = 100%, Specificity is 95%). In other words, at tau=0.486 my model is able to distinguish between Nociceptive and Neuropathic at its best.

Comparing to limited classification abilities of Logistic Regression (~85% on both Training and Validation Data), Machine Learning algorithms are totally a better approach for this Binary Classification problem on this clinical dataset. There are some disadvantages with Machine Learning algorithms such as interpretability and computation efficiency but we clearly can optimize these 2 disadvantages. We can understand which variables are important in growing the trees in Tree-Based models by accessing the Variable Importance calculated by Mean Decrease Gini. This idea also support Variable Selection step in my project. Combining with PrincipleComponent Analysis, I could successfully reduce 7 variables in my data to improve the computation efficiency. Using Baseline Models to compare on both Train and Validation data, I decided to choose Random Forest (with Variable Selection, Pre-Processing) as my “best” model in term of Bias-Variance error. Furthermore, to improve my Final Model, I could tune parameters and figured out that with “mtry”=5 (random selected Predictors at each split) is the “best” parameter for my “Parameterized Final Model”. In fact, my “Parameterized Final Model” performs really well on unseen data of Test Set. My model could correctly classified a whole Test Data with Accuracy of

94.59%, which is expected and quite similar to Accuracy on both Train and Validation data. For binary classification problem, Sensitivity, Specificity and trade-off of SensitivitySpecificity (ROC curve) are critical performance metrics. ROC curve and its Area-Under-Curve (AUC) would provide how well my model can distinguish between Nociceptive and Neuropathic. My model has AUC of 99.70%, which is extreme strong ability to distinguish between 2 classes in this problem. Even more, at the threshold of tau=0.486, where sum of Sensitivity and Specificity is maximized, my “parameterized final model” can distinguish between Nociceptive and Neuropathic at its best with (Sensitivity = 100%, Specificity is 95%).