Assignment 3 - STAT30270 – Statistical Machine Learning

K.Saketh Sai Nigam (22201204)

2023-04-28

Data: DeepSolar database

The data is a subset of the DeepSolar database, a solar installation database for the United States, built by extracting information from satellite images. Photovoltaic panel installations are identified from over one billion image tiles covering all urban areas as well as locations in the US by means of an advanced machine learning framework. Each image tile records the amount of solar panel systems (in terms of panel surface and number of solar panels) and is complemented with features describing social, economic, environmental, geographical, and meteorological aspects, which have been collected from readily available data sources (NASA Surface Meteorology and Solar Energy and the American Community Survey). As such, the database can be employed to relate key environmental, weather and socioeconomic factors with the adoption of solar photovoltaic energy production.

More information about this database is at the link:

http://web.stanford.edu/group/deepsolar/home (http://web.stanford.edu/group/deepsolar/home)

The dataset data in the file data_hw3_deepsolar.RData contains a subset of the DeepSolar database. Each row of the dataset is a "tile" of interest, that is an area related to a detected solar power system. A solar power system consists of a set of solar panels on top of a building, or at a single location such as a solar farm, or a similar system to convert solar irradiance to energy. For each system, a collection of features record social, economic, housing, geographical, and meteorological aspects of the tile (area) in which the system has been detected. Information about these features are reported below:

- **solar_system_coverage:** *a binary indicator variable indicating the coverage of solar power systems in a tile. low for low-to-medium coverage, high for high coverage (more than 10 systems).*
- **average_household_income:** *average annual houeshold income (US dollars).*
- **employ_rate:** *employment rate.*
- **population_density:** *population density in a tile (mile2).*
- **housing_unit_count:** *total number of housing units.*
- **housing_unit_median_value:** *median housing unit value (US dollars).*
- **occupancy_vacant_rate:** *ratio of vacant housing units.*
- **heating_fuel_gas_rate:** *ratio of house units using gas as heating fuel.*
- **heating_fuel_electricity_rate:** *ratio of house units using electricity as heating fuel.*
- **heating_fuel_oil_rate:** *ratio of house units using oil as heating fuel.*
- **land_area:** *total land area (mile2).*
- **water_area:** *total water area (mile2).*
- **air_temperature:** *air temperature (Celsius).*
- **earth_temperature:** *earth temperature (Celsius).*
- **daily_solar_radiation:** *daily solar radiation (kWh/m2).*

Task: Predict solar power system coverage

The target variable is solar_system_coverage. This variable is a binary variable indicating the coverage of solar power systems in a given tile. The variable takes outcome low if the tile has a low-to-medium number of solar power systems, while it takes outcome high if the tile has a larger number of solar power systems (more than 10).

Detection and estimation of the coverage level of a tile is an expensive process. In fact, labeling of the tiles in relation to their solar power system coverage involves processing and analysis of satellite image data, and implementation of complex and computationally intensive advanced machine learning methods. Researchers have interest in building a reliable and scalable classifier, which, using the available data on environmental, weather, and socioeconomic measure- ments, can predict the classification of new tiles associated with their solar power system coverage. Predictions from this "simpler" classifier could subsequently be used to aid the tile labeling process, making it less computationally expensive in some instances.

The task is to build a supervised learning model to predict if a tile can be considered as containing low-to-medium or high solar power system coverage, using the available social, economic, housing, geographical, and meteorological features.

==

1. Implement at least 3 different supervised learning methods to predict if a tile has high (high) solar power system coverage or not, on the basis of the input features. Employ an appropriate framework to compare and tune the different methods considered, evaluating and discussing their relative merits. (70 marks)

```r
library(rpart)
library(caret)
library(glmnet)
library(randomForest)
library(ROCR)
```

developing decision trees is done with the "rpart" package, and developing predictive models and pre-processing data is done with the "caret" package's capabilities. The regularized linear model fitting program "glmnet" and the random forest model building program "randomForest" are both employed.Last but not least, the "ROCR" software calculates several measures like accuracy, precision, recall, and area under the ROC curve *to assess the effectiveness of classification models.*

```
# load the data
load("/Users/saketh/Desktop/MS SEM 2/Statistical ML/Assignment 3/data_hw3_deepsolar.RData")
DeepSolarDataFrame = data

# set high = 1 and low = 0
DeepSolarDataFrame$solar_system_coverage <- ifelse(DeepSolarDataFrame[,1] == "high", 1, 0)
# convert the solar_system_coverage (target) to factor
DeepSolarDataFrame$solar_system_coverage = factor(DeepSolarDataFrame$solar_system_coverage)
```

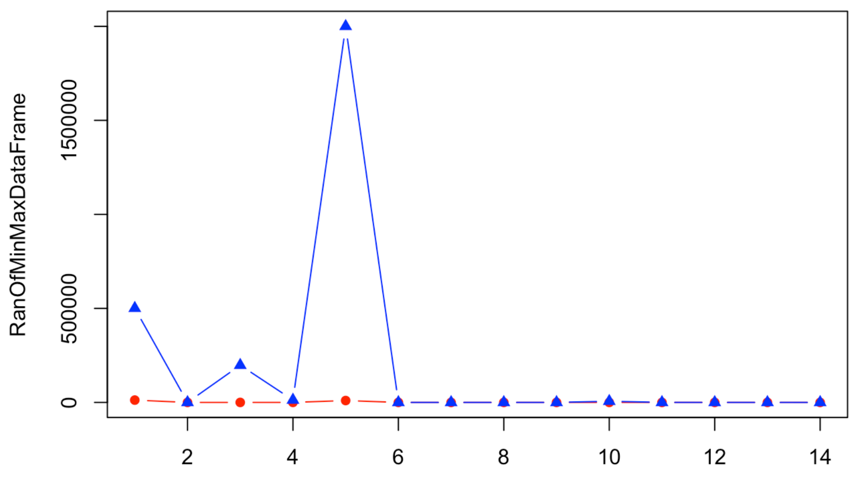
Getting data ready for training and testing

- *The "DeepSolarDataFrame" has 15 characteristics and 10926 observations.*
- *The data is divided as 80 percent of the DeepSolarDataFrame observations are training data and 20% of the DeepSolarDataFrame observations are testing data.*

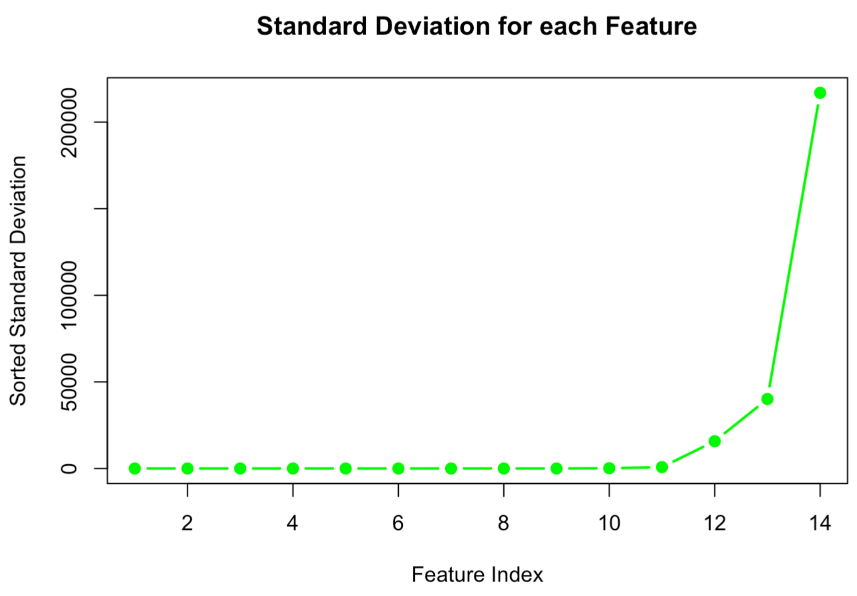
```
# set seed for sampling
set.seed(22201204)

Nrows = nrow(DeepSolarDataFrame)
# Generate a random sample of row indices for the test set
TestIndex = sample((1:Nrows), round(Nrows*0.2))
# Return all the indexes between 1 and "Nrows" that are not in test
TrainIndex <- setdiff(1:Nrows, TestIndex)
TrainData <- DeepSolarDataFrame[TrainIndex,]
NrowsTrainData <- nrow(TrainData)
TestData = DeepSolarDataFrame[TestIndex,]
TrainOut = as.numeric(as.character(TrainData[,1]))
TestOut = as.numeric(as.character(TestData[,1]))
```

- *We decide to standardize the input characteristics to have a mean of 0 and a variance of 1 after "checking the degree of variability across the RanOfMinMax of variation of the features".*

```
# returns min and max for each variables
RanOfMinMax <- apply( DeepSolarDataFrame[,-1], 2, range )
RanOfMinMaxDataFrame = data.frame(t(RanOfMinMax))
colnames(RanOfMinMaxDataFrame) = c("min","max")
#plot the RanOfMinMax(max, min) for each feature
matplot(RanOfMinMaxDataFrame, type = "b", lty = 1, col = c("red", "blue"), pch = c(16, 17))
```



```
# returns the standard deviation for each feature
StanDevFeat <- apply(TrainData[,-1], 2, sd)
#plot the standard deviation in ascending order
plot(sort(StanDevFeat), pch = 19,
     xlab = "Feature Index",
     ylab = "Sorted Standard Deviation",
     main = "Standard Deviation for each Feature",
     col = "green", type = "b", lwd = 2)
```



- *In the first block of code, the lowest and maximum values are calculated for each feature in the dataset, a data frame is made with the findings, and a matplot function is used to plot the ranges for each feature. The minimum and maximum values for each feature are shown on the plot as red and blue lines, respectively. Any features with a limited or wide range of values can be found using this graphic.*
- *The second block of code calculates the standard deviation for each feature in the dataset, arranges the standard deviations in descending order, and displays the results as a scatterplot with a green line summing the points. The features with high or low variability in their values can be found using this figure.*

Regulate the training and test data

- *To address the range issue, set the mean to 0 and the variance to 1. We normalize it using scale to avoid bias because different RanOfMinMax for features leads to biased models.*

```
# scale the training data
TrainScaleData <- scale(TrainData[,-1])
TrainData[,-1] = TrainScaleData
TestData[,-1] = scale(TestData[,-1],
                        center = attr(TrainScaleData, "scaled:center"),
                        scale = attr(TrainScaleData, "scaled:scale"))
```

Metric mthods for comparing and analyzing

```r
ClassAccuracy <- function(y, yhat) {
  TableValue <- table(y, yhat)
  return(sum(diag(TableValue))/sum(TableValue))
}


F1Score = function(y_true, y_pred){
  # Compute confusion matrix
  ConfusionMatrixOfData <- table(y_true, y_pred)

  # Calculate PrecisionValue, RecallValue, and F1 score
  PrecisionValue <- ConfusionMatrixOfData[2,2] / sum(ConfusionMatrixOfData[,2])
  RecallValue <- ConfusionMatrixOfData[2,2] / sum(ConfusionMatrixOfData[2,])
  F1Score <- 2 * PrecisionValue * RecallValue / (PrecisionValue + RecallValue)

  # Print results
  return(F1Score)
}

PrecisionValue = function(y_true, y_pred){
  # Compute confusion matrix
  ConfusionMatrixOfData <- table(y_true, y_pred)

  # Calculate PrecisionValue
  PreValue <- ConfusionMatrixOfData[2,2] / sum(ConfusionMatrixOfData[,2])
  return(PreValue)
}

RecallValue = function(y_true, y_pred){
  # Compute confusion matrix
  ConfusionMatrixOfData <- table(y_true, y_pred)

  # Calculate RecallValue
  RecalValue <- ConfusionMatrixOfData[2,2] / sum(ConfusionMatrixOfData[2,])
  return(RecalValue)
}

SpecificityValue <- function(y_true, y_pred) {
  ConfusionMatrixOfData <- table(y_true, y_pred)
  SpecValue <- ConfusionMatrixOfData[1,1] / sum(ConfusionMatrixOfData[1,])
  return(SpecValue)
}
```

Framework for evaluating and fine-tuning the various supervised learning techniques

There are 6 Classification Trees

```r
ClassificationTrees <- c("ClassificationTree1", "ClassificationTree2", "LogisticReggresion1", "LogisticReggresion
2","RandomForest1","RandomForest2")

# No of k-FoldsValue
K = 5

# No of iterations
R = 2

OutputVector <- vector("list", R)
OutputF1Score <- vector("list", R)
OutputRecall <- vector("list", R)
StartOfTime <- Sys.time()
for ( r in 1:R ) {
  AccuracyVal <- matrix(NA, K, 6) # accuracy of the ClassificationTrees in the K FoldsValue
  F1ScoreVal <- matrix(NA, K, 6) # f1-score of the ClassificationTrees in the K FoldsValue
  recValue <- matrix(NA, K, 6) # RecallValue of the ClassificationTrees in the K FoldsValue

  FoldsValue <- rep( 1:K, ceiling(NrowsTrainData/K) )
  FoldsValue <- sample(FoldsValue) # random permute
  FoldsValue <- FoldsValue[1:NrowsTrainData]

  for ( k in 1:K ) {
    TrainFold <- which(FoldsValue != k)
    ValidationVal <- setdiff(1:NrowsTrainData, TrainFold)

    #fit ClassificationTrees on the training data
    # classification tree
    FitclassificationTree1 = rpart(solar_system_coverage ~ .,
                     data = TrainData,
                     subset = TrainFold,
                     control = list(cp = 0.0005))
    FitclassificationTree2 = rpart(solar_system_coverage ~ .,
                       data = TrainData,
                       subset = TrainFold,
                       control = list(cp = 0.005))

    # logistic regression
    FitLogisticRegression1 <- glmnet(TrainData[TrainFold,-1],
                           TrainOut[TrainFold],
                           family = "binomial",
                           alpha = 1,
                           lambda = 0.001)
    FitLogisticRegression2 <- glmnet(TrainData[TrainFold,-1],
                           TrainOut[TrainFold],
                           family = "binomial",
                           alpha = 1,
                           lambda = 0.0001)

    # random forest
    FitRandomForest1 <- randomForest(solar_system_coverage ~ .,
                                data = TrainData,
                                subset = TrainFold,
                                ntree = 200,
                                mtry = 2)
    FitRandomForest2 <- randomForest(solar_system_coverage ~ .,
                                data = TrainData,
                                subset = TrainFold,
                                ntree = 200,
                                mtry = sqrt(ncol(TrainData)))

    # predict the classification of the ValidationVal data observations in the dropped fold
    # classification tree
    PredictedClassificationTree1 = predict(FitclassificationTree1, type = "class", newdata = TrainData[Validation
Val,])
    AccuracyVal[k,1] = ClassAccuracy(PredictedClassificationTree1, TrainData$solar_system_coverage[ValidationVal]
)
    F1ScoreVal[k,1] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedClassificationTree1)
    recValue[k,1] = RecallValue(TrainData$solar_system_coverage[ValidationVal], PredictedClassificationTree1)


    PredictedClassificationTree2 <- predict(FitclassificationTree2, type = "class", newdata = TrainData[Validatio
nVal,])
    AccuracyVal[k,2] <- ClassAccuracy(PredictedClassificationTree2, TrainData$solar_system_coverage[ValidationVal
])
    F1ScoreVal[k,2] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedClassificationTree2)
    recValue[k,2] = RecallValue(TrainData$solar_system_coverage[ValidationVal], PredictedClassificationTree2)

    ##logistic regression -

    PredictedLoggesticRegression1 <- predict(FitLogisticRegression1, type = "response", newx = as.matrix(TrainDat
a[ValidationVal,-1]))
    PredictedLoggesticRegression1 <- ifelse(PredictedLoggesticRegression1 > 0.5, 1, 0)
    AccuracyVal[k,3] <- ClassAccuracy(PredictedLoggesticRegression1, TrainData$solar_system_coverage[ValidationVa
l])
    F1ScoreVal[k,3] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedLoggesticRegression1)
    recValue[k,3] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedLoggesticRegression1)

    PredictedLoggesticRegression2 <- predict(FitLogisticRegression2, type = "response", newx = as.matrix(TrainDat
a[ValidationVal,-1]))
    PredictedLoggesticRegression2 <- ifelse(PredictedLoggesticRegression2 > 0.5, 1, 0)
    AccuracyVal[k,4] <- ClassAccuracy(PredictedLoggesticRegression2, TrainData$solar_system_coverage[ValidationVa
l])
    F1ScoreVal[k,4] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedLoggesticRegression2)
    recValue[k,4] = RecallValue(TrainData$solar_system_coverage[ValidationVal], PredictedLoggesticRegression2)

    ##random forest
    PredictedRandomForest1 <- predict(FitRandomForest1, type = "class", newdata = TrainData[ValidationVal,])
    AccuracyVal[k,5] <- ClassAccuracy(PredictedRandomForest1, TrainData$solar_system_coverage[ValidationVal])
    F1ScoreVal[k,5] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedRandomForest1)
    recValue[k,5] = RecallValue(TrainData$solar_system_coverage[ValidationVal], PredictedRandomForest1)

    PredictedRandomForest2 = predict(FitRandomForest2, type = "class", newdata = TrainData[ValidationVal,])
    AccuracyVal[k,6] = ClassAccuracy(PredictedRandomForest2, TrainData$solar_system_coverage[ValidationVal])
    F1ScoreVal[k,6] = F1Score(TrainData$solar_system_coverage[ValidationVal], PredictedRandomForest2)
    recValue[k,6] = RecallValue(TrainData$solar_system_coverage[ValidationVal], PredictedRandomForest2)
  }
  OutputVector[[r]] = AccuracyVal
  OutputF1Score[[r]] = F1ScoreVal
  OutputRecall[[r]] = recValue
}
EndOfTime <- Sys.time()
TotalTime <- as.numeric(difftime(EndOfTime, StartOfTime, units = "secs"))
print(paste("Total time in seconds:", TotalTime))
```

```
## [1] "Total time in seconds: 35.4066340923309"
```

CAUSE FOR NOT COMPARING SVM

- *In comparison to SVM, logistic regression, classification trees, and random forests are computationally less expensive. For large data sets, SVM requires parameter optimization and takes longer to converge to the best result. As the program runs numerous ClassificationTrees with R iterations, avoiding SVM will help conserve compute resources.*

JUSTIFICATION FOR K=5 AND R=2

- *R (Iterations) equals 2. For each classifier, there are 15 models. These 15 models are averaged to produce a generalized model for the classifier, which prevents overfitting or biased models.*
- *k = 5. For the same rationale as earlier, the data is divided into 5 FoldsValue or sets at random. Four of the sets are used for training, while the fifth set is utilized for validation.*
- *Increases in K and R will need more time and processing resources, thus these values were chosen to maintain them.*

COMPARING THE ACCURACY, RecallValue AND F-1 SCORE Metrics for ValidationVal Data

```
cat("ClassificationTrees :", ClassificationTrees)
```

```
## ClassificationTrees : ClassificationTree1 ClassificationTree2 LogisticReggresion1 LogisticReggresion2 RandomFo
rest1 RandomForest2
```

```
AvgVal <- t(sapply(OutputVector, colMeans))
MeanAccuracyVal <- colMeans(AvgVal) # estimated mean accuracy
cat("Average Accuracy : ", MeanAccuracyVal)
```

```
## Average Accuracy :  0.9366206 0.9390226 0.9288983 0.9287262 0.9480038 0.9492052
```

```
AvgF1Val <- t( sapply(OutputF1Score, colMeans))
MeanF1Val <- colMeans(AvgF1Val) # estimated mean F-1 score
cat("Average F1Score : ", MeanF1Val)
```

```
## Average F1Score :  0.658043 0.6579225 0.5604351 0.5695875 0.6873051 0.7029084
```

```
AvgRecallVal = t( sapply(OutputRecall, colMeans))
MeanRecallVal <- colMeans(AvgRecallVal) # estimated mean RecallValue
cat("Average RecallValue : ", MeanRecallVal)
```

```
## Average RecallValue :  0.6014834 0.5780908 0.5604351 0.4649 0.5626108 0.5918372
```

- *Almost all of the model's accuracy is above 92%, which appears to be fairly high and indicates that the model correctly predicts results 94% of the time. However, this is due to "imbalance classification". The most accurate model is the random forest with the mtry = sqrt(ncol(TrainData) model.*
- *In terms of accuracy and F1Score, random forest with mtry = sqrt(ncol(TrainData) is the best model; in terms of RecallValue, it is second only to classification tree, and the difference is extremely slight. As a result, random forest outperforms all ClassificationTrees.*

SUBSEQUENT MERITS AND DEMERITS

Classification trees

- *It has several advantages over logistic regression, including:It is the simplest to comprehend and display.It works best with high-dimensional data and nonlinear relationships. It only uses categorical response variables*

Logistic Regression

- *Simple and quick, assuming a linear model The logistic regression has the lowest predictive performance measure among the three supervised models in the program, showing that there is some complex link. If there are non-linear relationships between x and y, then it will not perform well.*

Random Forest

- *When compared to logistic and classification trees, this model performed better because it can handle complex relationships and non-linear decision boundaries. However, it can become slow for large data sets and is challenging to interpret because the algorithm uses a lot of classification trees. It tends to be superior to classification trees because it employs a greater variety of classification trees, which leads to more varied data and a more generalized model.*

==

2. Select the best model a predicting if a tile has high solar power system coverage from the available numerical features data. (10 marks)

Best supervised classification model

Imbalance classification

- *0 represens LOW value*
- *1 represens HIGH value*

```
cat("The Tables of the TrainOut:-","\n")
```

```
## The Tables of the TrainOut:-
```

```
table(TrainOut)
```

```
## TrainOut
##    0    1
## 7850  891
```

```
cat("\n")
```

```
cat("The Tables of the TestOut:-","\n")
```

```
## The Tables of the TestOut:-
```

```
table(TestOut)
```

```
## TestOut
##    0    1
## 1986  199
```

mtry = (the number of features to take into account when creating decision trees at each split)
mtrees = (how many trees will be grown in the forest).

CONCLUSION :

- *Random forest is the best model among those employed in this program to determine whether a tile has a high solar power system coverage with an F1Score of roughly 70% and a RecallValue of 59%.*
- *With a RecallValue of 0.59, the Random Forest model predicts the high solar power system correctly 59% of the time. Since this is not a high value, other techniques can be used to enhance the model.*

==

3. Use appropriately some test data in order to evaluate the generalized predictive performance of the best selected classifier. Provide a discussion about the ability of the selected model at detecting correctly high and low coverage. (20 marks)

BEST SELECTED CLASSIFIER : “Random forest”

```
# fit the random forest model
FittingRandomForest <- randomForest(solar_system_coverage ~ .,
                       data = TrainData,
                       ntree = 200,
                       mtry = sqrt(ncol(TrainData)))

# predict the values for test data using the training data model
PredictedBestRandomForest <- predict(FittingRandomForest, type = "class", newdata = TestData)
TableValues <- table(TestData$solar_system_coverage, PredictedBestRandomForest)
TableValues
```

```
##    PredictedBestRandomForest
##        0    1
##   0 1944   42
##   1   84  115
```

```
#metrics
CAvalue = ClassAccuracy(TestData$solar_system_coverage, PredictedBestRandomForest)
cat("Accuracy for test data: ", CAvalue)
```

```
## Accuracy for test data:  0.9423341
```

```
SCvalue = F1Score(TestData$solar_system_coverage, PredictedBestRandomForest)
cat("F1 score for test data: ", SCvalue)
```

```
## F1 score for test data:  0.6460674
```

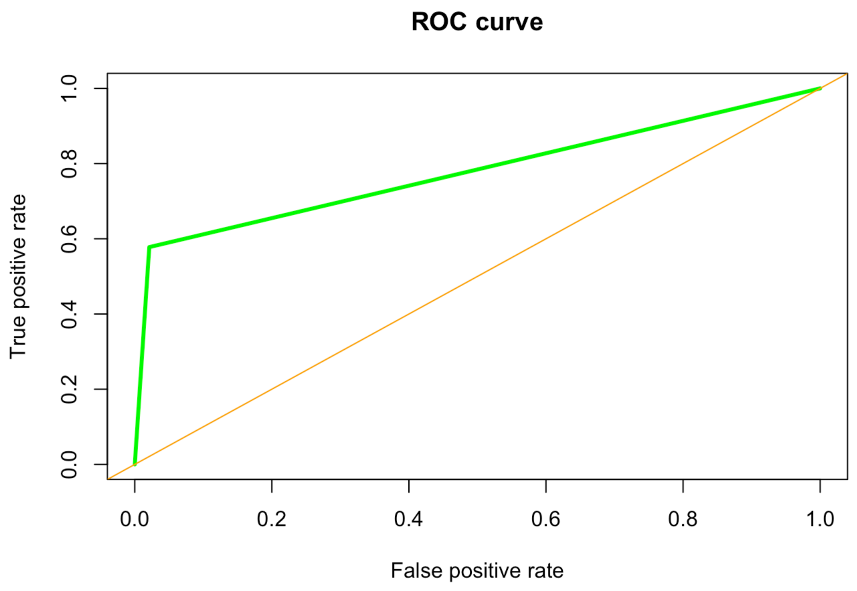
```
Recvalue = RecallValue(TestData$solar_system_coverage, PredictedBestRandomForest)
cat("RecallValue for test data: ", Recvalue)
```

```
## RecallValue for test data:  0.5778894
```

```
SPvalue = SpecificityValue(TestData$solar_system_coverage, PredictedBestRandomForest)
cat("SpecificityValue for test data: ", SPvalue)
```

```
## SpecificityValue for test data:  0.978852
```

```
PredictedBestRandomForest <- factor(PredictedBestRandomForest, levels = levels(TestData$solar_system_coverage))
ActualTestData  <- factor(TestData$solar_system_coverage)
ObjectPrediction <- prediction(as.numeric(PredictedBestRandomForest), as.numeric(ActualTestData ))
ObjectPerformance <- performance(ObjectPrediction, "tpr", "fpr")
plot(ObjectPerformance, main="ROC curve", col="green", lwd=3, ylim=c(0,1), xlim=c(0,1), type="l")
abline(0, 1, lty=1, col="orange")
```



EVALUATION OF PERFORMANCE USING ROC curve and F1 score

The true positive rate (TPR) against false positive rate (FPR) at various threshold levels is shown by the ROC curve. When the class distribution is balanced, it shows the classifier's performance at all potential threshold values and is useful. Here, it says that the true positive rate (TPR) and false positive rate (FPR) are classified moderately, with room for improvement.

The PrecisionValue and RecallValue of the classifier are combined to provide the F1 score, which offers a general assessment of the model's accuracy. When the distribution of classes is imbalanced, it is useful. Since our data is not balanced, f1 is therefore more useful than ROC curve.