Analytics using Python

Learning outcomes

1. You will learn Python , a useful language

2. Use programming for problem solving

Great Lakes Institute of Management

A guide to learn python for analytics

P. V. Subramanian

**A workbook on Analytics using Python**

**P. V. SUBRAMANIAN**

**Visiting faculty – Great Lakes Institute of Management**

*Copyright @ 2019 Great Lakes Institute of Management : All contents in this book, including (but not limited to) its contents are proprietary to Great Lakes Institute of Management, Chennai. Any effort at re-using, modifying and circulating both through Digital or Physical means, without the express written permission of the author, will amount to infringement of copyrights, and will be liable for suitable action.*

**Chapter 9. Ensemble Techniques basics**

Table of Contents

[**1)** **ENSEMBLE MODELS** 4](#_Toc12218492)

[**2)** **BAGGING ALGORITHMS** 4](#_Toc12218493)

[**2.1.** **RANDOM FOREST** 5](#_Toc12218496)

[**2.1.1** **HOW RANDOM FOREST WORKS FOR CLASSIFICATION?** 5](#_Toc12218500)

[**2.1.2** **EXAMPLE FOR Random Forest for classification** 6](#_Toc12218501)

[***2.1.3*** **HYPERPARAMETER TUNING IN THIS EXAMPLE** 20](#_Toc12218502)

[**2.1.4** **VARIABLE IMPORTANCE** 27](#_Toc12218503)

[**3)** **BOOSTING ALGORITHMS** 28](#_Toc12218505)

[**3.1.** **XGBOOST** 29](#_Toc12218507)

[**3.1.1** **WHAT IS XGBOOST?** 29](#_Toc12218510)

[**3.1.2** **WHY XGBOOST IS POPULAR?** 29](#_Toc12218511)

[**3.1.3** **HOW XGBOOST WORKS?** 30](#_Toc12218512)

[**3.1.4** **EXAMPLE: PIMA INDIANS DIABETES DATA SET** 30](#_Toc12218513)

# **ENSEMBLE MODELS**

* **Ensembles can increase accuracy on your dataset.**

**Most popular ensemble methods**

# **BAGGING ALGORITHMS**

* **Bagging derives its name from Boot-Strap Algorithm with Aggregation.**
* **Bootstrap Aggregation or bagging involves taking multiple samples from your training dataset with replacement and training a model for each sample. The final output predictions is averaged across the predictions of all of the sub-models.**
* **Popular bagging models:**

**1. Bagged Decision Tree**

**2. Random Forest**

**3. Extra Trees**

* **We shall focus only on Random Forest.**



## **RANDOM FOREST**

* **Random Forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type (decision trees) resulting in a forest of trees which tend to improve prediction.**
* **The random forest algorithm can be used for both regression and classification tasks.**

**Reference:**

* **https://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/**
* **https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/**
* **https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/tutorial-random-forest-parameter-tuning-r/tutorial/**
* [**https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/**](https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/)
* [**https://blog.treasuredata.com/blog/2016/04/12/get-started-with-machine-learning-with-**](https://blog.treasuredata.com/blog/2016/04/12/get-started-with-machine-learning-with-)**random-forests-and-hivemall/**



### **HOW RANDOM FOREST WORKS FOR CLASSIFICATION?**

* **In case of random forest, the randomly created decision trees are the ensemble classifiers .**
* **Construct a decision tree for each sample from a given dataset and get a prediction result from each decision tree.**
* **Each decision tree is a single classifier which votes to one of the target classes.**
* **The prediction result with the most votes is selected as the final target prediction.**

### **EXAMPLE FOR Random Forest for classification**

[**https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-**](https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-)**583891c99652**

**Example 1: Implement a random forest model in python for classification problem using the Pima Indians diabetes data set.**

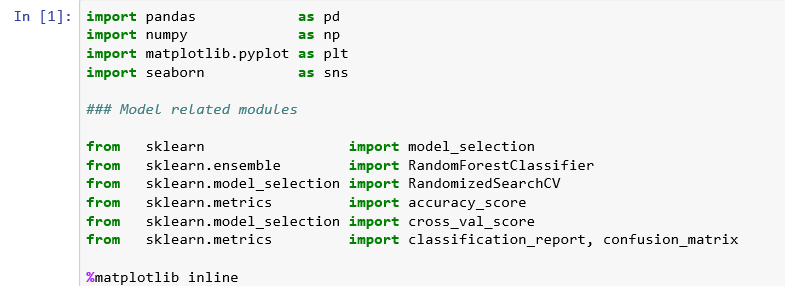
Dataset pima-indians-diabetes.data is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

The objective is to predict based on diagnostic measurements whether a patient has diabetes. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Source: Kaggle https://www.kaggle.com/uciml/pima-indians-diabetes-database

|  |  |  |
| --- | --- | --- |
| **SlNo** | **Variable** | **Description** |
| 1 | Pregnancies | Number of times pregnant |
| 2 | Glucose | Plasma glucose concentration - 2 hours in an oral glucose tolerance test |
| 3 | BloodPressure | Diastolic blood pressure (mm Hg) |
| 4 | SkinThickness | Triceps skin fold thickness (mm) |
| 5 | Insulin | 2-Hour serum insulin (mu U/ml) |
| 6 | BMI | Body mass index (weight in kg/(height in m)^2) |
| 7 | DiabetesPedigreeFunction | Diabetes pedigree function |
| 8 | Age | Age (years) |
| 9 | label | Class variable (0 or 1) |

**Step 1: Import required machine learning packages**

****

We use the following packages for random forest classifier implementation:

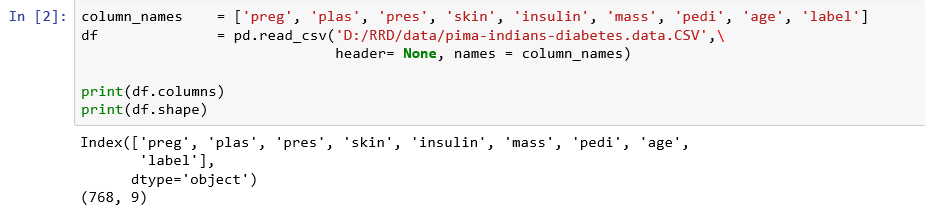
1. pandas to perform data manipulation tasks

2. numpy to perform numerical calculations

3. matplotlib and seaborn for visualization

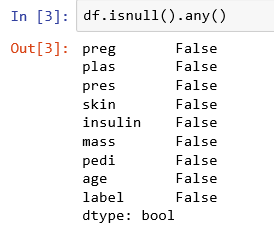
4. sklearn for performing machine learning algorithms

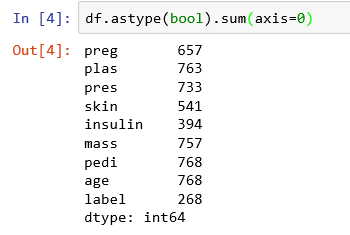
**Step 2: Load the data**

**Observations:**

* 1. There are 768 observations and nine variables including the target variable, label which indicates the presence of Diabetes.
  2. Let us check for null values or zeros in any of the columns.

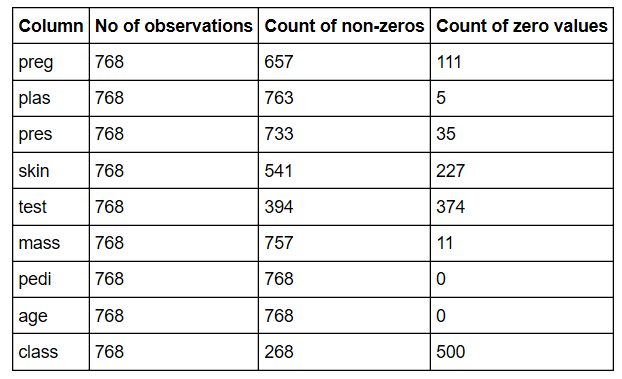
1. **Step 3: Pre-process the data – Check for null or missing values**





**Observations:**

1. Even though there are no null values, there are several zero values.
2. Zero values in the columns, plas, pres, skin and mass is not possible.
3. To get the value of zero counts, simply subtract these numbers from 768.

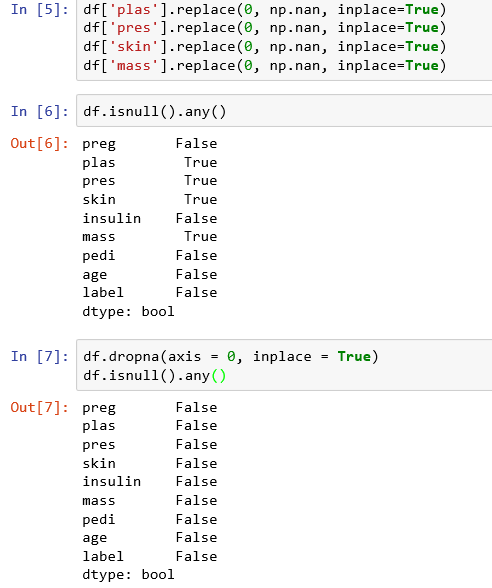
****

**Let us analyze each column.**

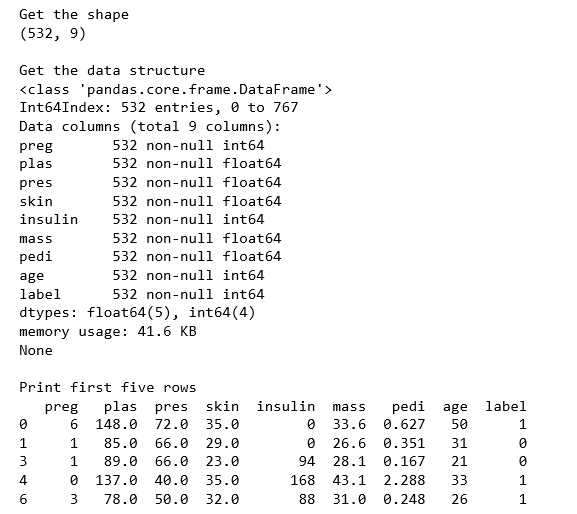
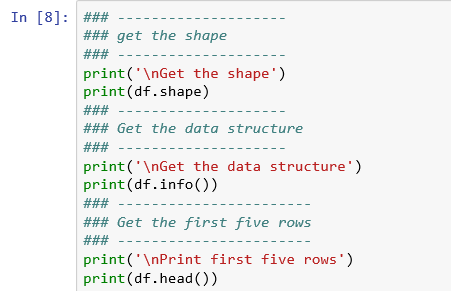
1. There are 111 zero values for the column, preg (Number of times pregnant). These zero values make sense since the patient was not pregnant at all at any point in time.
2. There are 5 zero values for the column, plas - which represents Plasma glucose concentration - 2 hours in an oral glucose tolerance test. No living person can have zero plasma. So we need to remove them or impute.
3. There are 35 zero values for the column, pres - which represents Diastolic blood pressure (mm Hg). No living person can have zero blood pressure. So, we need to remove them or impute.
4. There are 227 zero values for the column, skin - which represents Triceps skin fold thickness (mm)
5. No living person can have zero skin thickness. So, we need to remove them or impute.
6. There are 374 zero values for the column, test - which represents 2-Hour serum insulin (mu U/ml)
7. No living person can have zero serum insulin blood sugar. So we need to remove them or impute.
8. There are 11 zero values for the column, mass - which represents Body mass index (weight in kg/(height in m)^2) No living person can have zero BMI. So we need to remove them or impute.
9. There are 500 zero values for the column, class, which is a target variable. There are 268 diabetic people and 500 non-diabetic people. No need for any imputation.

**Convert the zeros in the columns, plas, pres, skin and mass to NA.**

**Remove NAs**

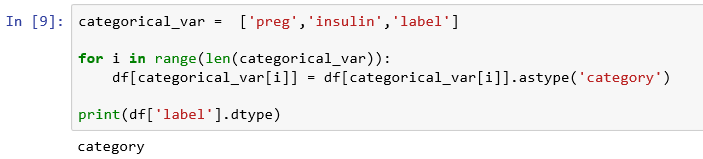


**Understand the data structure**

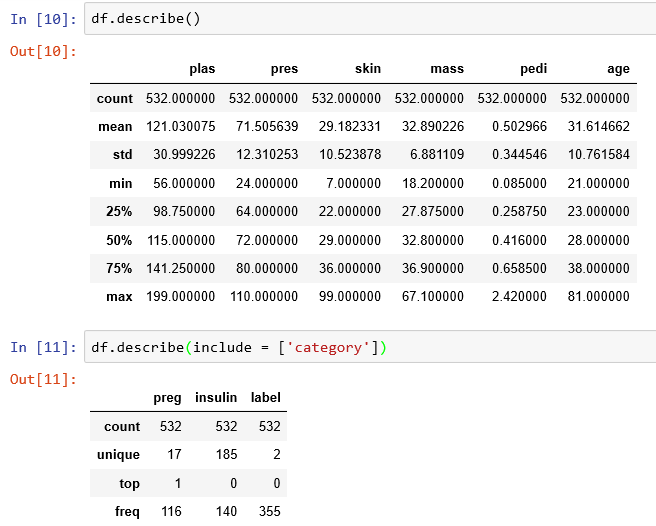
****

**Observations:**

1. Now we have only 532 observations and nine variables without missing values.
2. All variables are numeric. Only three variables ( pregnant, insulin and class) need to be converted to categorical variables.

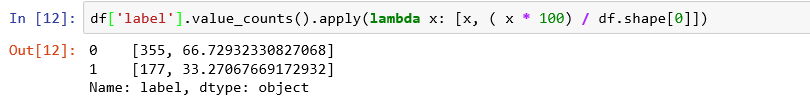
****

**Get descriptive statistics for the data.**

****

1. **Step 4: Check if the data set is balanced or not by checking the value counts of target**

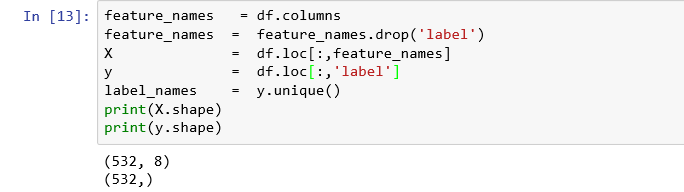
**variable.**

****

**Observations:**

Class distribution in the target variable is in the ratio, 67% : 33% and hence, data is not balanced.

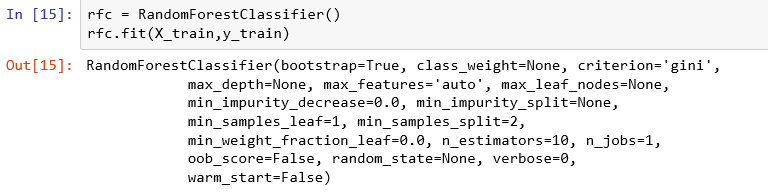
1. **Step 5: Create X, set of independent variables and y, dependent variable.**

****

**Step 6: Split the data into training and testing data in the ratio 70:30 and set the seed for reproducibility.**

****

**Step 7: Create a Random Forest model classifier and train the model using the training dataset**

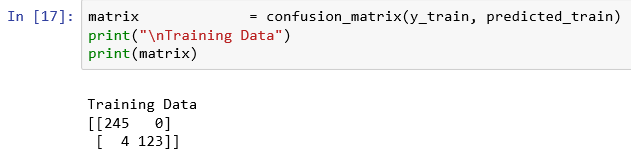


**Step 8: Predict the response for the training data set**

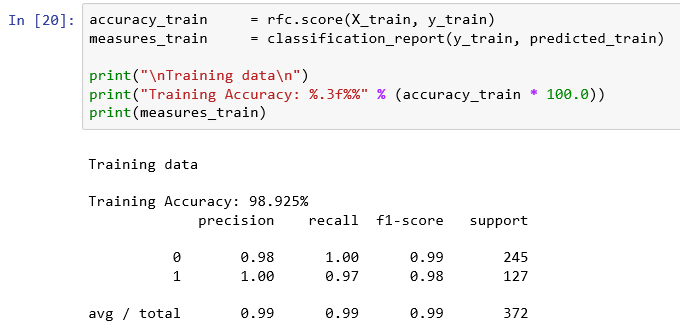
****

**Step 9: Evaluate the model performance for the training data set**

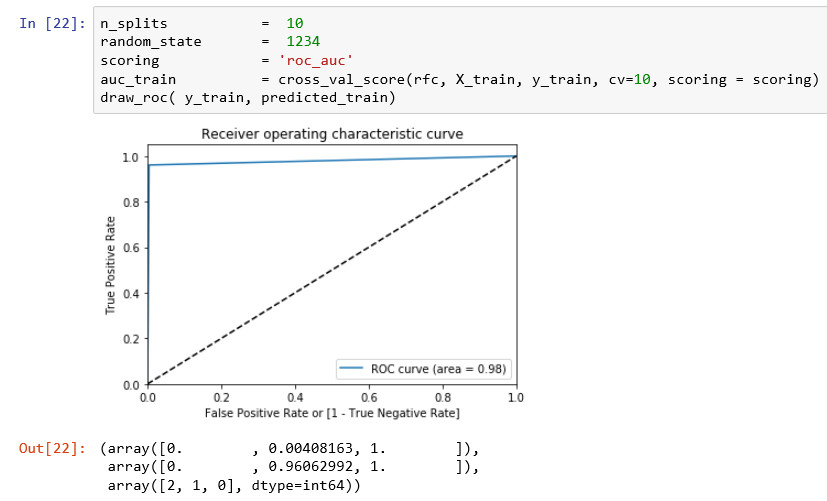
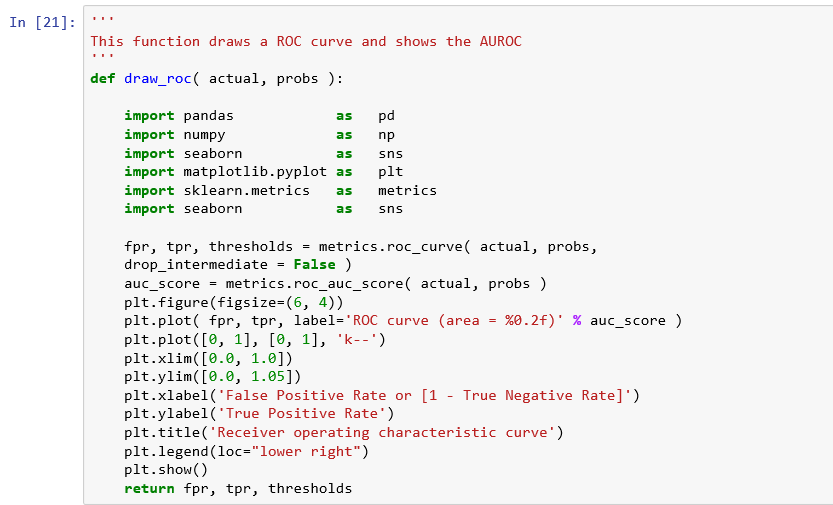
**Create confusion matrix**

****

****

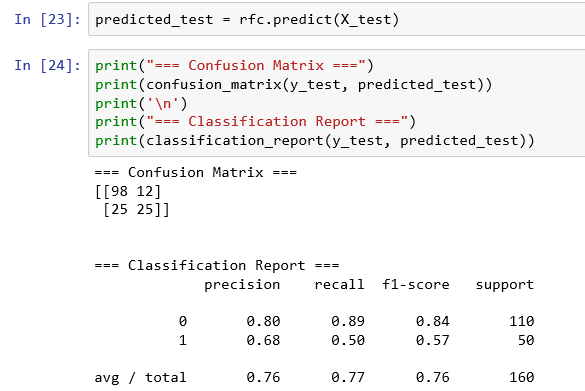
****

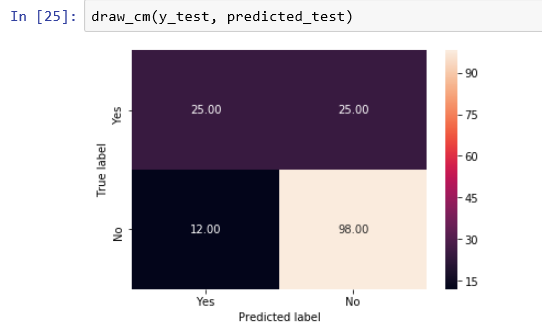
**Draw ROC curve and get AUROC measure for training data set**

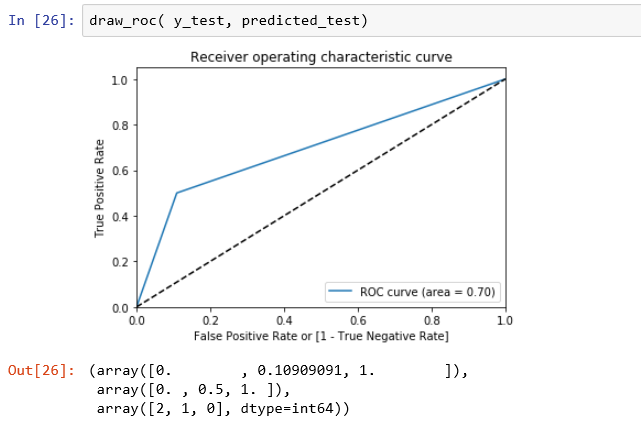
****

**We observe that the AUROC for the training data set is 0.98**

**Step 10: Evaluate the model performance for the test data set**

****

****

****

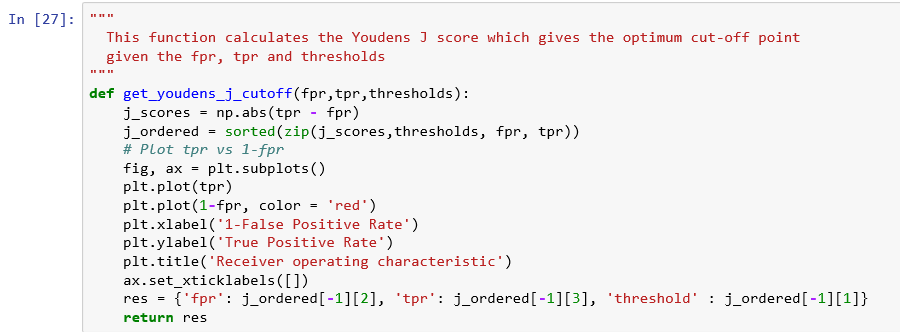
**We observe that the AUROC for the test data set is 0.70.**

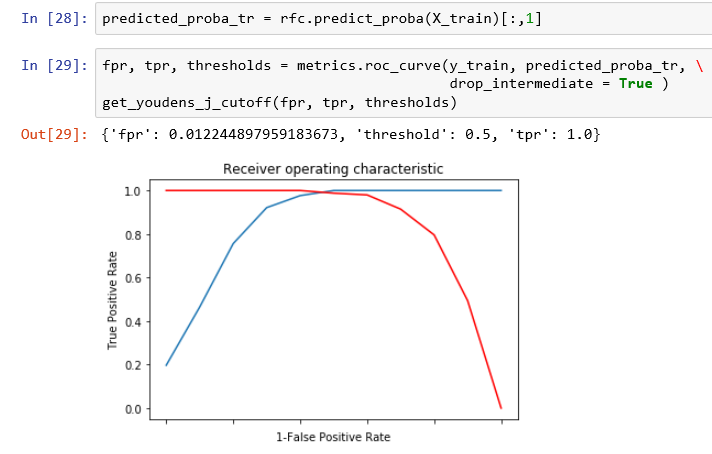
**Step 10:** Compare the model performance on both the training and test data to check if there is any overfit.

|  |  |  |
| --- | --- | --- |
| **Performance Measure** | **Training data** | **Test data** |
| F1 score for class 1 (Minority class) | 0.98 | 0.57 |
| AUROC | 0.98 | 0.70 |

Huge difference in the performance measures of the model for predicting the training and test data, indicates a bad quality of the predictive power of the model. We know that we have an unbalanced data set and we have used a standard cut-off of 50% for creating the confusion matrix. Standard cutoff of 50% may not be optimum. Let us check if this is true.

**Step 11: Find optimum cut-off (youden J score) and evaluate the model performance**

****



***New cutoff is 0.50. So, no reworking is required***

### **HYPERPARAMETER TUNING IN THIS EXAMPLE**

**What are hyperparameter?**

The parameters which define the machine learning model architecture are known as hyperparameters. You need to set the hyperparameters before training.

Hyperparameters attempt to address the following design questions:

a. What should be the maximum depth allowed for the decision tree?

b. What should be the minimum number of samples required at a leaf node in my decision tree?

c. How many trees should I include in my random forest?

d. Number of features to be considered by each tree before splitting a node

**How hyperparameter tuning is done?**

Hyperparameter tuning is done by trying different combinations of parameter values and evaluate the performance of the model.

Hyperparameters optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function ( for example, representing the cost of inaccuracy of predictions in classification problems). Cross validation is used to evaluate the generalization performance to avoid overfitting (a model that works well with the training data and performs poorly on testing data).

**Steps involved in hyperparameter tuning**

1. Define a model

2. Define the range of possible values for all hyperparameters

3. Define a method for sampling hyperparameter values

Some of the hyperparameter tuning methods

1. **Grid search.** Build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results.
2. **Random search**. Build a model for the values obtained from random sampling of the statistical distribution of each hyperparameter
3. **Bayesian optimization**. This belongs to a class of sequential model-based optimization (SMBO) algorithms that enables us to use the results of the previous iteration to improve the sampling method of the next experiment.

4. Define an evaluation criterion to judge the model performance

1. Criterion will measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Note: this parameter is tree-specific.
2. Gini impurity of a node is the probability that a randomly chosen sample in a node would be incorrectly labeled it it was labeled by the distribution of samples in the node.
3. Entropy gives measure of impurity in a node.
4. Information gain is the conditional expected value of the Kullback–Leibler divergence of the univariate probability distribution of one variable from the conditional distribution of this variable given the other one.
5. In mathematical statistics, the Kullback–Leibler divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution. Refer to https://en.wikipedia.org/wiki/Kullback–Leibler\_divergence

5. Define a cross validation method. We shall try 10-fold cross validation.

**Step 12: Tune the parameters to check if the model performance improves.**

What are hyperparameter? The parameters which define the machine learning model architecture are known as hyperparameters.

***Let us tune the following parameters in the model:***

* n\_estimators is the number of trees to be used in the forest
* max\_depth is the maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
* max\_features is the number of features to consider when looking for the best split:
* If int, then consider max\_features features at each split.
* If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
* If “sqrt”, then max\_features = sqrt(n\_features) (same as “auto”).
* If “log2”, then max\_features = log2(n\_features).
* If None, then max\_features = n\_features.
* max\_features is the number of features that are considered on a per-split level, rather than on the entire decision tree construction.
* Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.
* Criterion defines an evaluation criterion to judge the model performance. We are using “gini” for the Gini impurity and “entropy” for the information gain.

Refer:

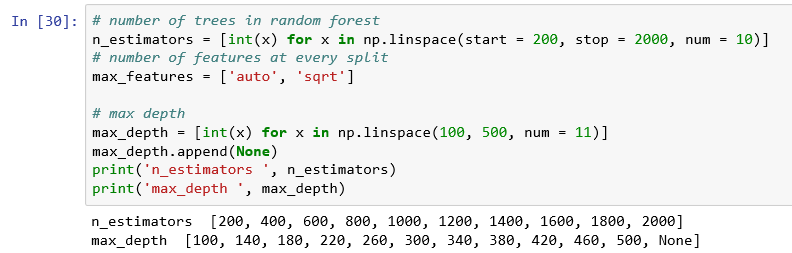
* [*https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
* [*https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74*](https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74)
* [*https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d*](https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d)
* [*https://stats.stackexchange.com/questions/7295/what-should-be-the-optimal-parameters-for-random-forest-classifier*](https://stats.stackexchange.com/questions/7295/what-should-be-the-optimal-parameters-for-random-forest-classifier)
* *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*
* Using linspace() function, we can generate evenly spaced numbers over the specified interval :

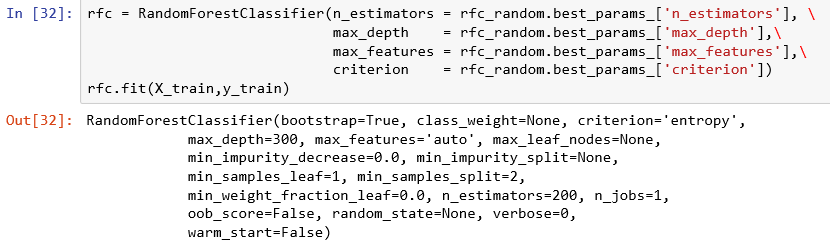
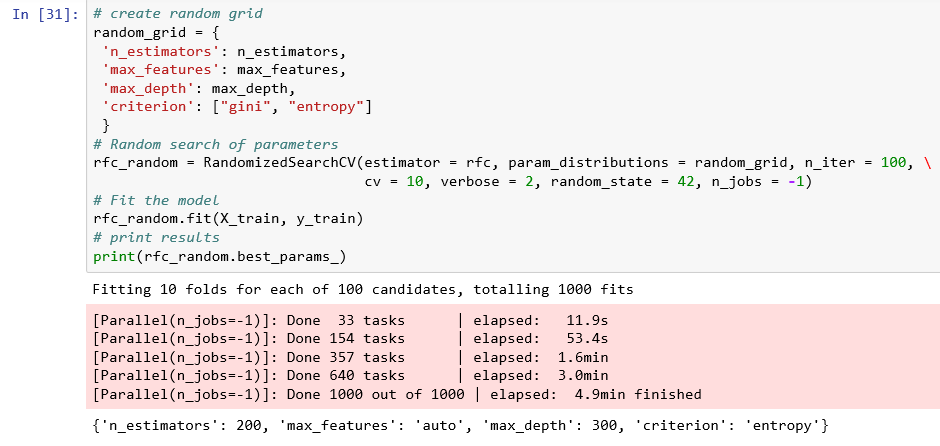
*a. n\_estimators: a range of 10 numbers from 200 to 2000*

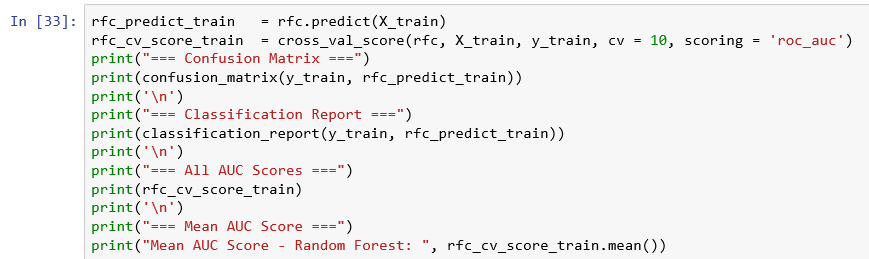
*b. max\_depth : a range of 11 numbers from 100 to 500*

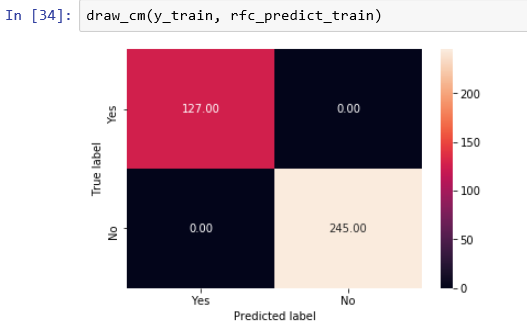
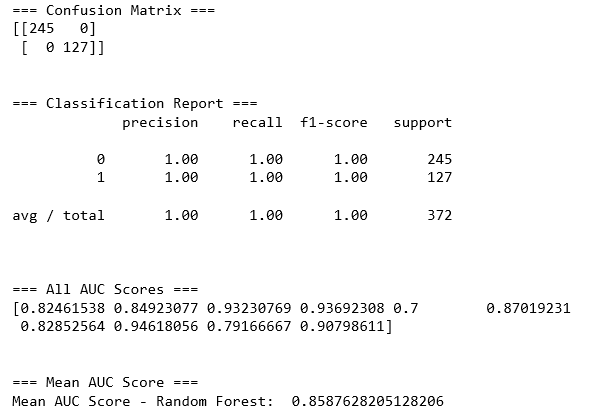
The best set of parameters as found by the grid search is given below:

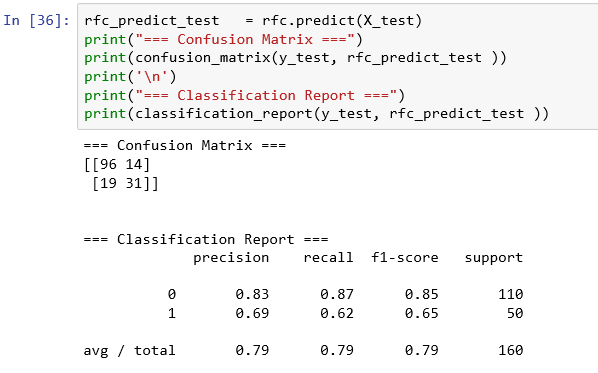
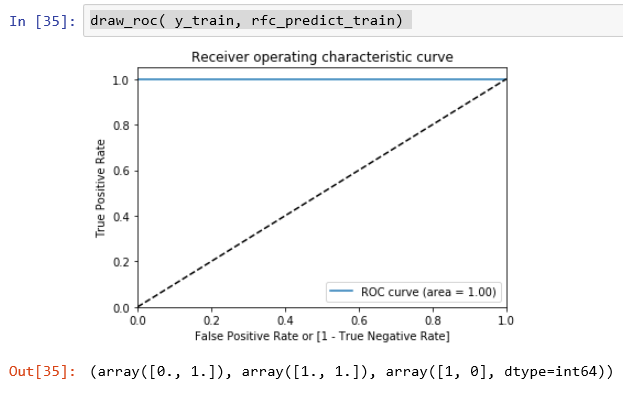
'n\_estimators': 1400, 'max\_features': 'auto', 'max\_depth': 140, 'criterion': 'entropy'

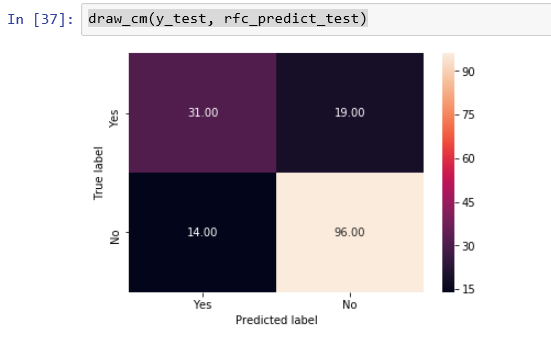


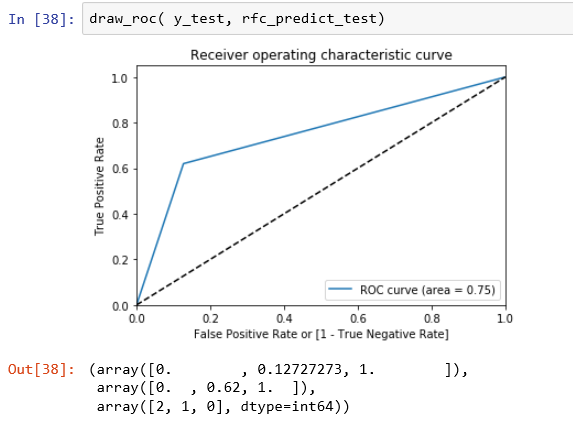






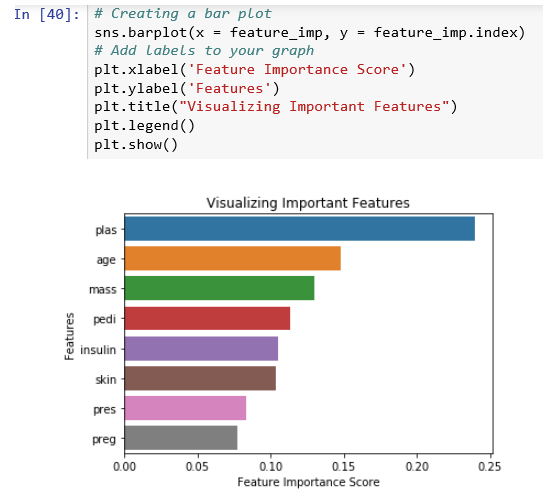
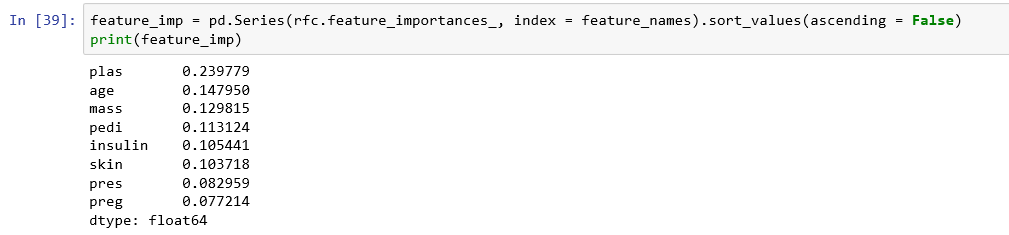






### **VARIABLE IMPORTANCE**

***Here, the importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance.***



Compare the model performance on both the training and test data to check if there is any overfit.

|  |  |  |
| --- | --- | --- |
| **Performance Measure** | **Training data** | **Test data** |
| F1 score for class 1 (Minority class) | 1 | 0.65 |
| AUROC | 1 | 0.75 |

Even though the performance measures have improved when we fitted the model with best values for the parameters, there is still a big difference in the performance measures of the model for predicting the training and test data, indicates a bad quality of the predictive power of the model.

Random Forest models are prone to overfitting.

According to the above variable importance plot, the top five variables,

* 1. plas (Plasma glucose concentration - 2 hours in an oral glucose tolerance test),
  2. age,
  3. mass (Body Mass Index) ,
  4. insulin (2-Hour serum insulin (mu U/ml) and
  5. pedi (Diabetes pedigree function)

are important in predicting the target variable, class (diabetic or not).



# **BOOSTING ALGORITHMS**

* **Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence. Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a ﬁnal output prediction.**
* **The two most common boosting ensemble machine learning algorithms are:**
  + **AdaBoost** works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns.
  + **In Stochastic Gradient Boosting**, at each iteration of the algorithm, a base learner should be fit on a subsample of the training set drawn at random without replacement.
  + **XGBoost** stands for eXtreme Gradient Boosting. It is a penalized gradient boosting method with a variety of base-learners. It creates a sparse and robust boosted regression model.
* **We will focus on XGBoost model.**
* **https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/**



## **XGBOOST**

<https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/>



### **WHAT IS XGBOOST?**

* XGBoost, short for eXtreme Gradient Boosting is a fast, portable, and distributed implementation of the gradient boosting (tree) algorithm. Gradient boosting is an ensemble (i.e. meta) machine learning algorithm that builds a strong model based on many weaker ones sequentially by using gradient descent. When this technique is used with decision trees, it is called gradient boosting trees - Yassine Alouini on Quora
* XGBoost is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.
* XGBoost was created by Tianqi Chen, PhD student, University of Washington.

### **WHY XGBOOST IS POPULAR?**

* **Regularization**: It uses regularization technique, which avoids overfitting in linear and tree-based models.
* **Parallel Computing:** It implements parallel processing and extremely fast as compared to other Gradient Boosting Models.
* **Built-in Cross Validation:** It enables the user to run a cross-validation at each iteration of the boosting process
* **Handles Missing Values:** It has an in-built routine to handle missing values
* **Flexibility:** It allows user to define custom optimization objectives and evaluation criteria.
* **Continue on Existing Model:** It allows user to start training an XGBoost model from its last iteration of previous run.
* **Tree Pruning:** It grows the tree upto max\_depth and then prune backward until the improvement in loss function is below a threshold.

### **HOW XGBOOST WORKS?**

* Boosting is an ensemble method that seeks to create a strong classifier model based on weak classifiers (a measure indicating how correlated are the learners to the actual target variable).
* Gradient boosting sequentially adds predictors and corrects previous models.
* This method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction.
* This is supported for both regression and classification problems.
* XGBoost, known as Regularized Boosting technique specifically, implements this algorithm for decision tree boosting with an additional custom regularization term in the objective function.

*Refer: https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html*

### **EXAMPLE: PIMA INDIANS DIABETES DATA SET**

**Example 1: Implement a random forest model in python for classification problem using the Pima Indians diabetes data set.**

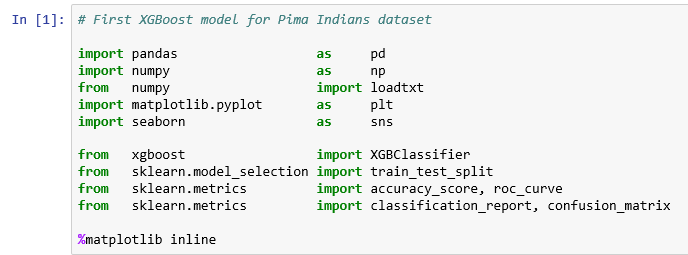
Dataset pima-indians-diabetes.data is originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

The objective is to predict based on diagnostic measurements whether a patient has diabetes. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

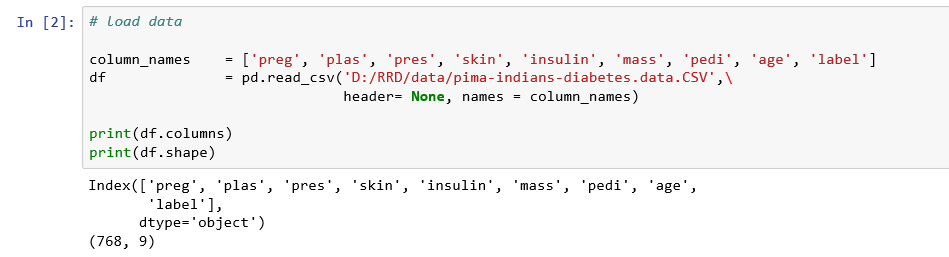
Source: Kaggle https://www.kaggle.com/uciml/pima-indians-diabetes-database

|  |  |  |
| --- | --- | --- |
| **SlNo** | **Variable** | **Description** |
| 1 | Pregnancies | Number of times pregnant |
| 2 | Glucose | Plasma glucose concentration - 2 hours in an oral glucose tolerance test |
| 3 | BloodPressure | Diastolic blood pressure (mm Hg) |
| 4 | SkinThickness | Triceps skin fold thickness (mm) |
| 5 | Insulin | 2-Hour serum insulin (mu U/ml) |
| 6 | BMI | Body mass index (weight in kg/(height in m)^2) |
| 7 | DiabetesPedigreeFunction | Diabetes pedigree function |
| 8 | Age | Age (years) |
| 9 | label | Class variable (0 or 1) |

**Step 1: Import required machine learning packages**



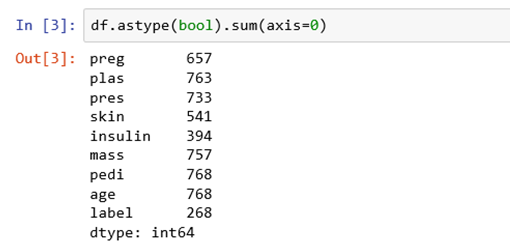
**Step 2: Load the data**



**Observations:**

* 1. There are 768 observations and nine variables including the target variable, label which indicates the presence of Diabetes.
  2. Let us check for null values or zeros in any of the columns.

1. **Step 3: Pre-process the data – Check for null or missing values**

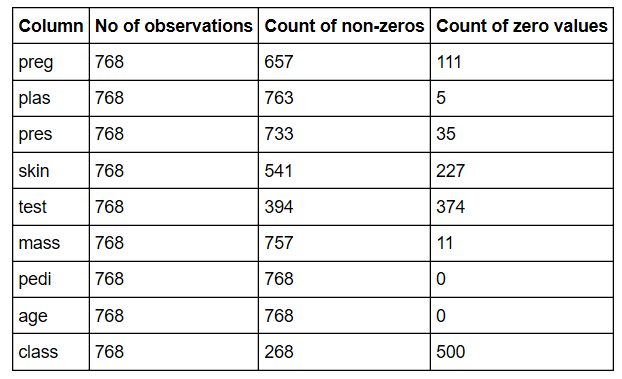


**Observations**:

a) Even though there are no null values, there are several zero values.

b) Zero values in the columns, plas, pres, skin and mass is not possible.

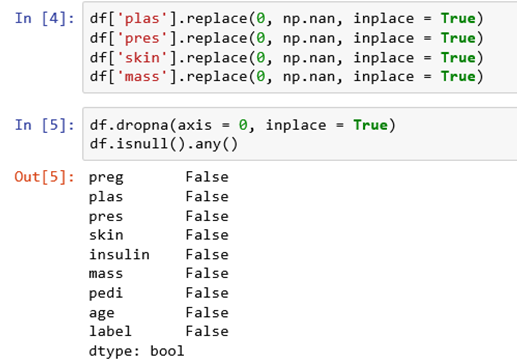
c) To get the value of zero counts, simply subtract these numbers from 768.

****

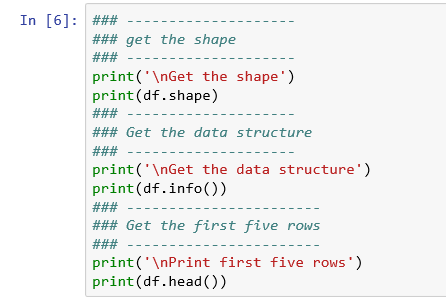
**Let us analyze each column.**

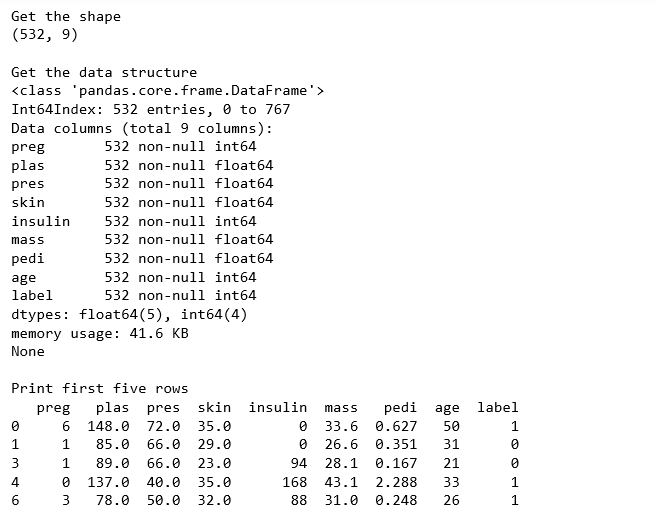
* 1. There are 111 zero values for the column, preg (Number of times pregnant). These zero values make sense since the patient was not pregnant at all at any point in time.
  2. There are 5 zero values for the column, plas - which represents Plasma glucose concentration - 2 hours in an oral glucose tolerance test. No living person can have zero plasma. So we need to remove them or impute.
  3. There are 35 zero values for the column, pres - which represents Diastolic blood pressure (mm Hg). No living person can have zero blood pressure. So, we need to remove them or impute.
  4. There are 227 zero values for the column, skin - which represents Triceps skin fold thickness (mm)
  5. No living person can have zero skin thickness. So, we need to remove them or impute.
  6. There are 374 zero values for the column, test - which represents 2-Hour serum insulin (mu U/ml)
  7. No living person can have zero serum insulin blood sugar. So we need to remove them or impute.
  8. There are 11 zero values for the column, mass - which represents Body mass index (weight in kg/(height in m)^2) No living person can have zero BMI. So we need to remove them or impute.
  9. There are 500 zero values for the column, class, which is a target variable. There are 268 diabetic people and 500 non-diabetic people. No need for any imputation**.**

**Convert the zeros in the columns, plas, pres, skin and mass to NA and remove NAs**

****

**Understand the data structure**

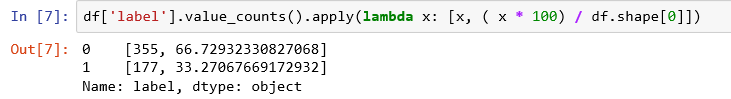
****

****

**Observations:**

1. Now we have only 532 observations and nine variables without missing values.
2. **Step 4: Check if the data set is balanced or not by checking the value counts of target**

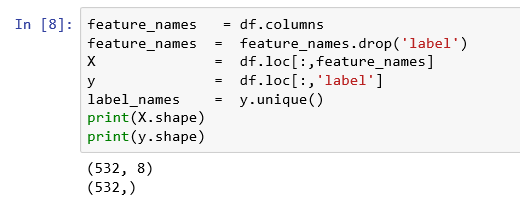
**variable.**



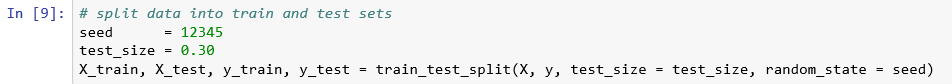
**Observations:**

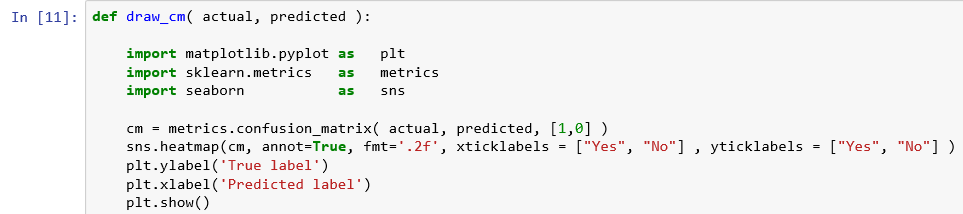
Class distribution in the target variable is in the ratio, 67% : 33% and hence, data is not balanced.

1. **Step 5: Create X, set of independent variables and y, dependent variable.**

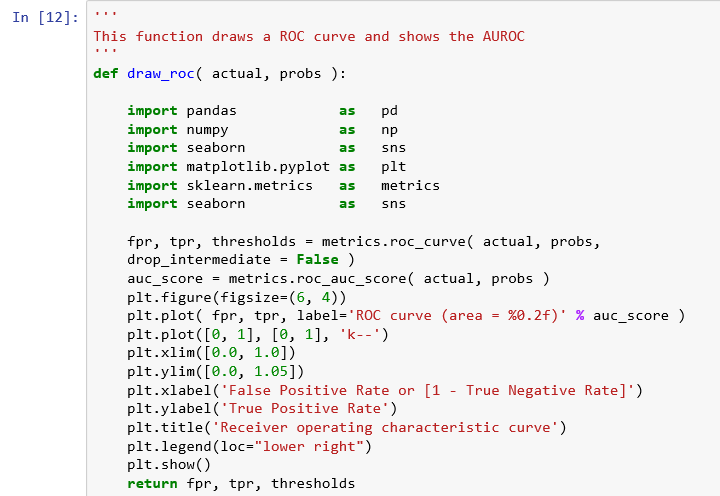
****

**Step 6: Split the data into training and testing data in the ratio 70:30 and set the seed for reproducibility.**

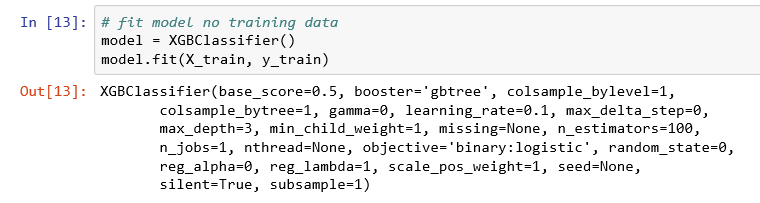
****

**Define useful function for creating a picture of Confusion Matrix:**

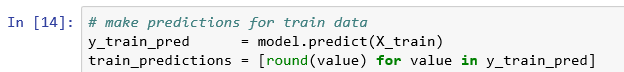
**Define useful function for creating a picture of Confusion Matrix:**

****

**Step 7: Create a XGBoost model classifier and train the model using the training dataset**

****

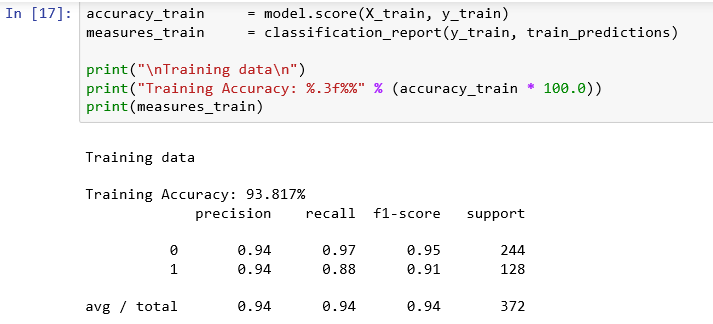
**Step 8: Predict the response for the training data set**

****

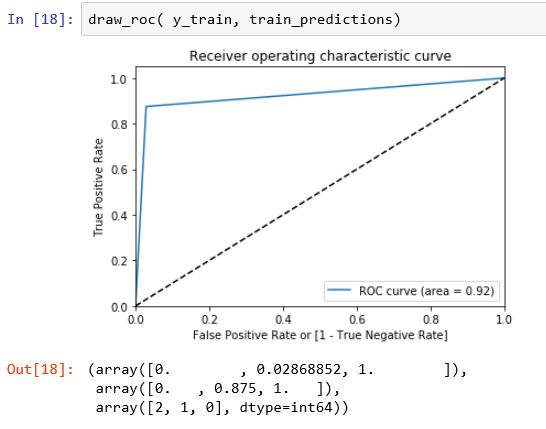
**Step 9: Evaluate the model performance for the training data set**

**Create confusion matrix**

****

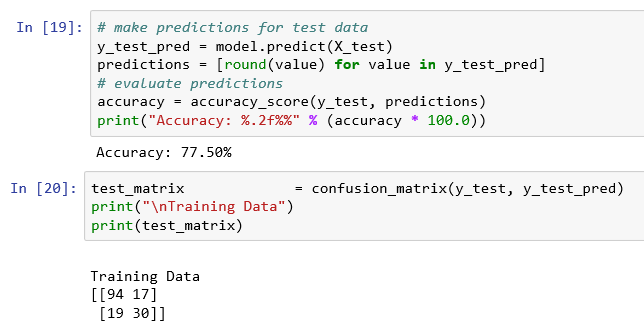
****

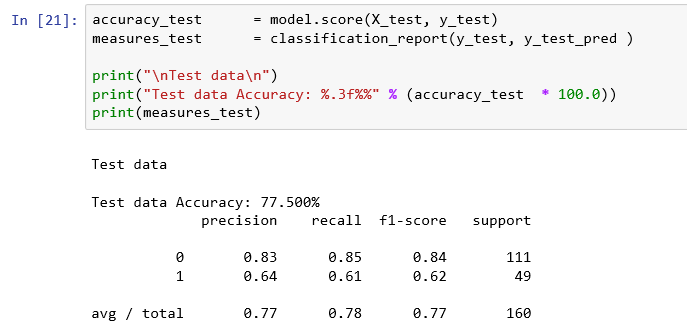
**Draw ROC curve and get AUROC measure for training data set**

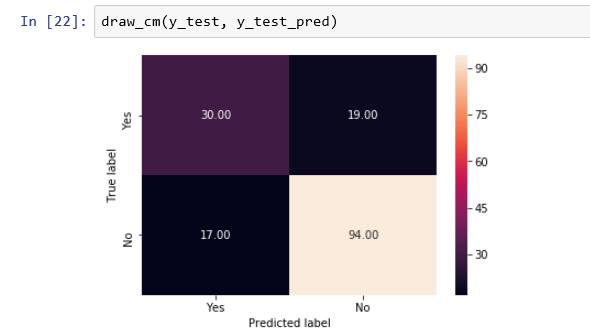
****

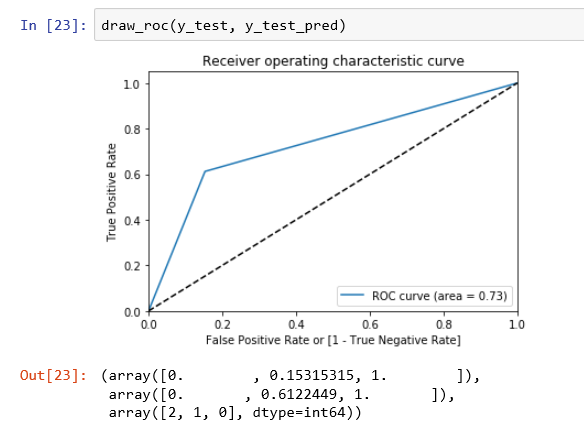
**We observe that the AUROC for the training data set is 0.92**

**Step 10: Evaluate the model performance for the test data set**

****

****

****

****

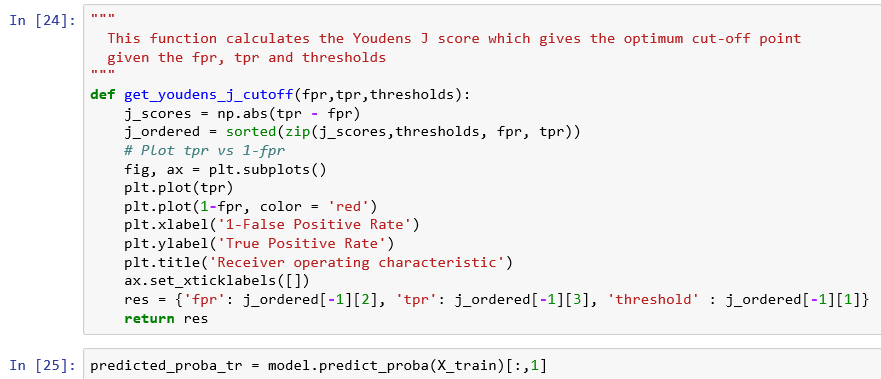
**We observe that the AUROC for the test data set is 0.73.**

**Step 10:** Compare the model performance on both the training and test data to check if there is any overfit.

|  |  |  |
| --- | --- | --- |
| **Performance Measure** | **Training data** | **Test data** |
| F1 score for class 1 (Minority class) | 0.91 | 0.62 |
| AUROC | 0.92 | 0.73 |

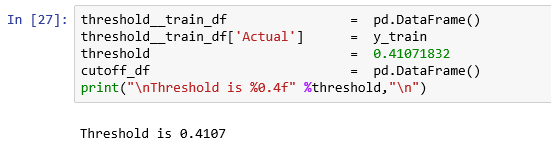
Huge difference in the performance measures of the model for predicting the training and test data, indicates a bad quality of the predictive power of the model. We know that we have an unbalanced data set and we have used a standard cut-off of 50% for creating the confusion matrix. Standard cutoff of 50% may not be optimum. Let us check if this is true.

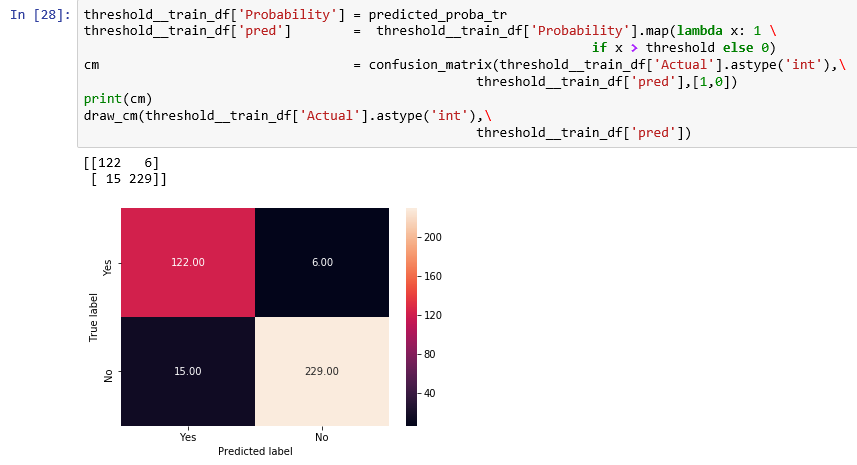
**Step 11: Find optimum cut-off (Youden J score) and evaluate the model performance**

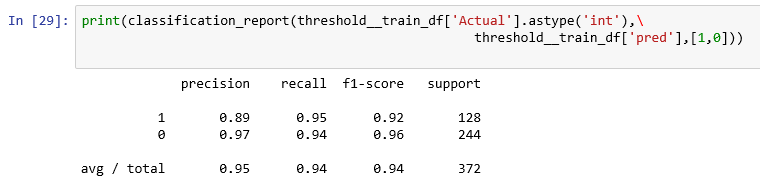
****

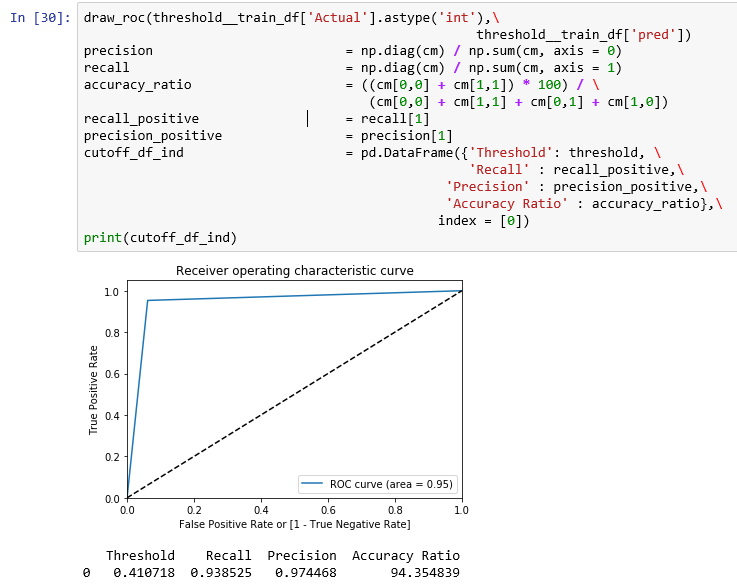


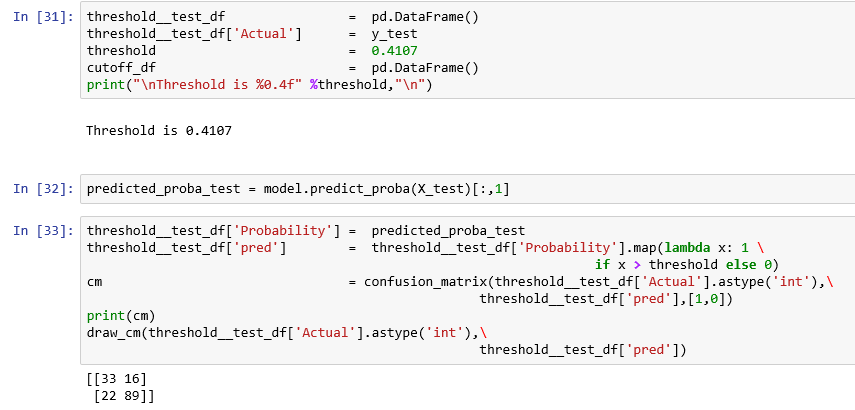
***New cutoff is 0.4107. So, reworking is required.***

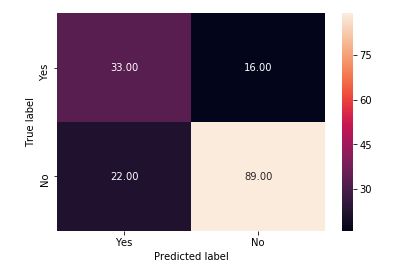
****

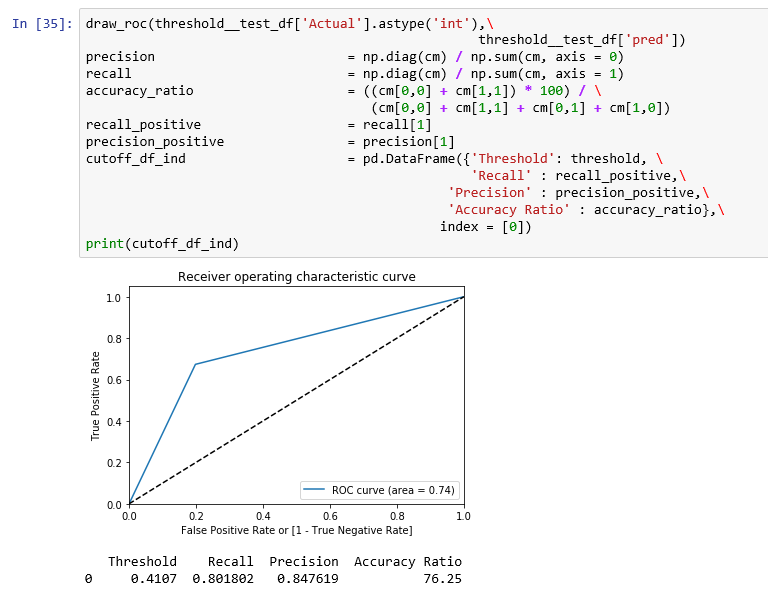
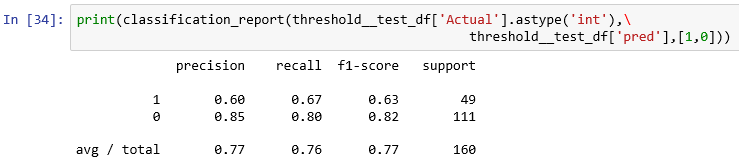
****

****

****

****

****

****

|  |  |  |
| --- | --- | --- |
| **Performance Measure** | **Training data** | **Test data** |
| F1 score for class 1 (Minority class) | 0.96 | 0.82 |
| AUROC | 0.95 | 0.74 |

**Observations:**

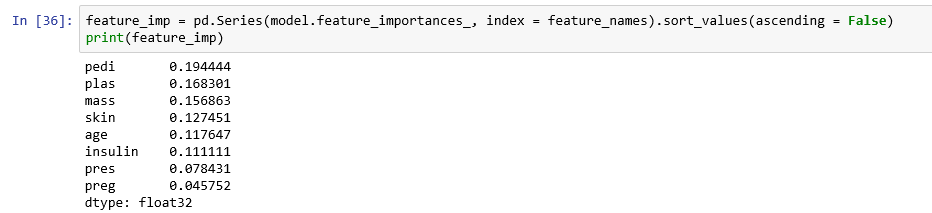
1. **When the optimum cut=off of 41.07% is applied, difference in the F1 score between training data and test data ( 96% and 82%) and is narrower than when standard cut-off of 50% is applied (91% and 62%).**

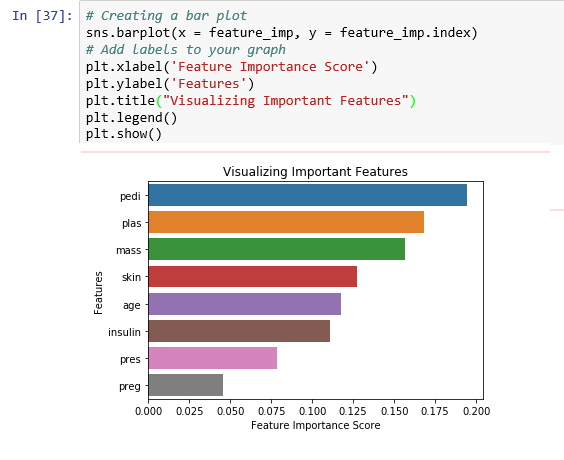
**Step 12: Find feature importance**

* Feature importance is calculated explicitly for each attribute in the dataset, enabling attributes to be ranked and compared to each other.
* Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for.
* The performance measure (such as the purity - Gini index) is used to select the split points or another more specific error function.
* The feature importance is then averaged across all the decision trees within the model.

Refer:

<https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/>

****

****

According to the above variable importance plot, the following top five variables:

* 1. pedi (Diabetes pedigree function)

1. plas (Plasma glucose concentration - 2 hours in an oral glucose tolerance test)
2. mass (Body Mass Index)
3. Skin (Triceps skin fold thickness (mm)
4. age

are important in predicting the target variable, class (diabetic or not).