RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Subject Code	CSE402A
Subject Name	Data Mining
Program/Course	B-Tech/CSE
Department	CSE
Faculty	FET

Name of the Student	Chaithanya Kumar A
Reg. No	16ETCS002026
Semester/Year	7th sem/4th Year
Subject Leader/s	prof .N. D. Gangadhar

Ramaiah University of Applied Sciences
University House, Gnanagangothri Campus, New BEL Road,
M S R Nagar, Bangalore, Karnataka, INDIA - 560 054.

Declaration Sheet			
Student Name	Chaithanya Kumar A		
Reg. No	16ETCS002026		
Program/Course	B.Tech/CSE	Semester/Year	7th sem / 4th Year
Subject Code	CSE402A		
Subject Title	Data Mining		
Subject Date		to	
Subject Leader	prof .N. D. Gangadhar		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Subject Leader and date	Signature of the Reviewer and date

Question No. 1

Solution to Question No. 1:

1.1 Introduction

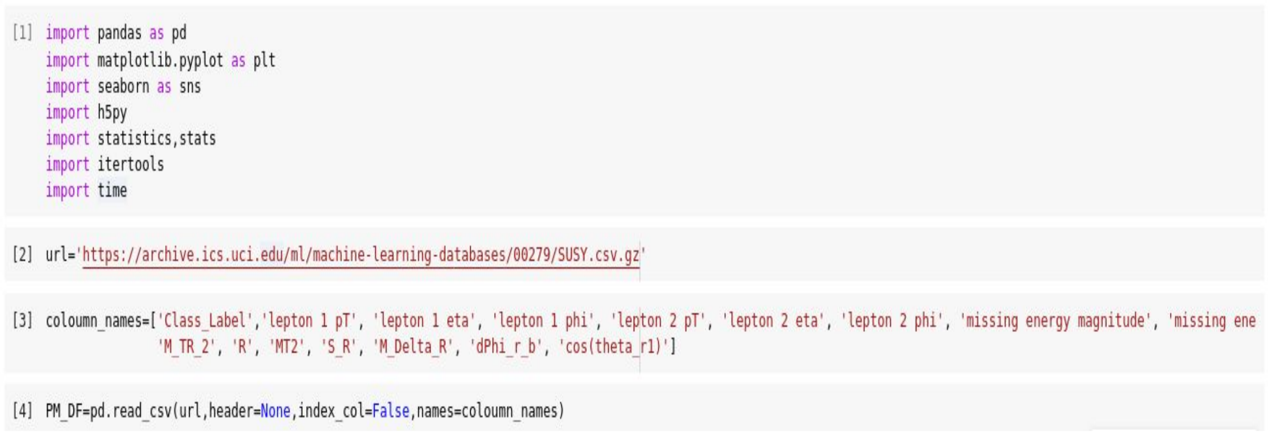
Collisions at high-energy particle colliders are a traditionally fruitful source of exotic particle discoveries. Finding these rare particles requires solving difficult signal-versus-background classification problems, hence machine-learning approaches are often used.

The primary tools of experimental high-energy physicists are modern accelerators, which collide protons and/or antiprotons to create exotic particles that occur only at extremely high-energy densities. Observing these particles and measuring their properties may yield critical insights about the very nature of matter. Such discoveries require powerful **statistical methods, and machine-learning tools** have a critical role. Given the limited quantity and expensive nature of the data, improvements in analytical tools directly boost particle discovery potential.

The high dimensionality of data, referred to as the feature space, makes it intractable to generate enough simulated collisions to describe the relative likelihood in the full feature space, and machine-learning tools are used for dimensionality reduction. Machine-learning classifiers such as Decision Trees and Ensembles of Decision Trees provide a powerful way to solve this learning problem.

This assignment focuses on solving the problem of classification of measurement data from particle accelerators to detect the presence of a fundamental particle, where the algorithm needs to classify the problem as either Made By particle or Background Noise.

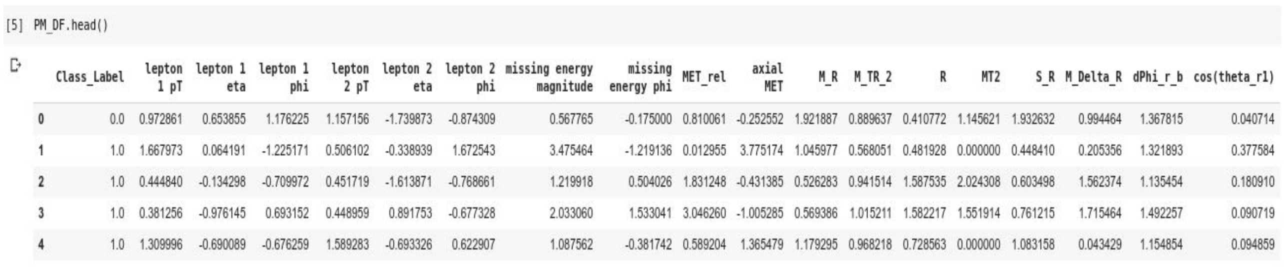
Importing the Dataset (SUSY) as a Pandas Dataframe from UCI Machine Learning Repository.



```
[1]  import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import h5py
     import statistics,stats
     import itertools
     import time

[2]  url='https://archive.ics.uci.edu/ml/machine-learning-databases/00279/SUSY.csv.gz'

[3]  coloumn_names=['Class_Label','lepton 1 pT', 'lepton 1 eta', 'lepton 1 phi', 'lepton 2 pT', 'lepton 2 eta', 'lepton 2 phi', 'missing energy magnitude', 'missing ene
                    'M_TR_2', 'R', 'MT2', 'S_R', 'M_Delta_R', 'dPhi_r_b', 'cos(theta_r1)']

[4]  PM_DF=pd.read_csv(url,header=None,index_col=False,names=coloumn_names)
```

The dataset required for the classification of measurement of data was imported from UCI ML Repo as the Pandas Dataframe. The dataset was imported as a CSV file trough URL by using the read_csv method from the Pandas package.

The first column in the dataset is the Class label which is either 1 (Background Signal) and 0 (Made By Particle) followed by 18 Features. The **first 8 features are kinematic properties** measured by the particle detectors in the accelerator. The **last ten features are functions of the first 8 features,** these are high-level features derived by physicists to help discriminate between the two classes.

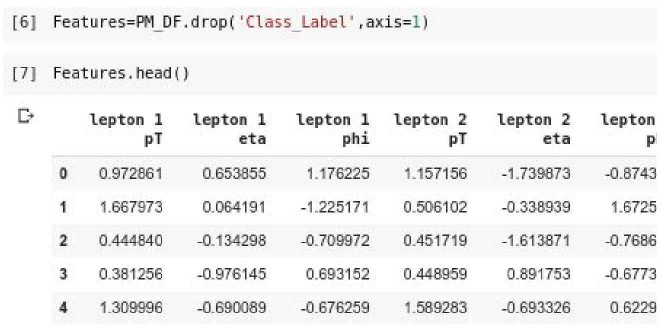


```
[5] PM_DF.head()
```

	Class_Label	lepton 1 pT	lepton 1 eta	lepton 1 phi	lepton 2 pT	lepton 2 eta	lepton 2 phi	missing energy magnitude	missing energy phi	MET_rel	axial MET	M_R	M_TR_2	R	MT2	S_R	M_Delta_R	dPhi_r_b	cos(theta_r1)
0	0.0	0.972861	0.653855	1.176225	1.157156	-1.739873	-0.874309	0.567765	-0.175000	0.810061	-0.252552	1.921887	0.889637	0.410772	1.145621	1.932632	0.994464	1.367815	0.040714
1	1.0	1.667973	0.064191	-1.225171	0.506102	-0.338939	1.672543	3.475464	-1.219136	0.012955	3.775174	1.045977	0.568051	0.481928	0.000000	0.448410	0.205356	1.321893	0.377584
2	1.0	0.444840	-0.134298	-0.709972	0.451719	-1.613871	-0.768661	1.219918	0.504026	1.831248	-0.431385	0.526283	0.941514	1.587535	2.024308	0.603498	1.562374	1.135454	0.180910
3	1.0	0.381256	-0.976145	0.693152	0.448959	0.891753	-0.677328	2.033060	1.533041	3.046260	-1.005285	0.569386	1.015211	1.582217	1.551914	0.761215	1.715464	1.492257	0.090719
4	1.0	1.309996	-0.690089	-0.676259	1.589283	-0.693326	0.622907	1.087562	-0.381742	0.589204	1.365479	1.179295	0.968218	0.728563	0.000000	1.083158	0.043429	1.154854	0.094859

The above figure shows the first five rows of the Dataframe with Features and the class Label.

Splitting the Dataframe as Features and Class Labels:



```
[6] Features=PM_DF.drop('Class_Label',axis=1)
```

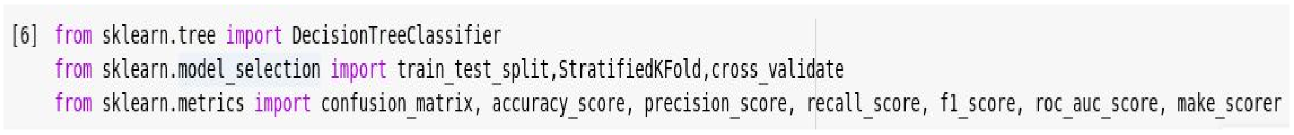
```
[7] Features.head()
```

	lepton 1 pT	lepton 1 eta	lepton 1 phi	lepton 2 pT	lepton 2 eta	lepton p
0	0.972861	0.653855	1.176225	1.157156	-1.739873	-0.8743
1	1.667973	0.064191	-1.225171	0.506102	-0.338939	1.6725
2	0.444840	-0.134298	-0.709972	0.451719	-1.613871	-0.7686
3	0.381256	-0.976145	0.693152	0.448959	0.891753	-0.6773
4	1.309996	-0.690089	-0.676259	1.589283	-0.693326	0.6229

The Features and the class labels are separated from the data frame where the features are **Independent Variables(Predictors)** and Class Label is a **Dependent variable (Target).**

1.2 Decision Tree Classifier

Decision Tree classifier is a Class of Supervised Machine Learning Algorithms which takes Independent variables as input and Classifies the dependent variable as an output. The class label may be either Binary (**Binary Classification**) or it may be a Multi-Class label (**Multi-Class Classification**).



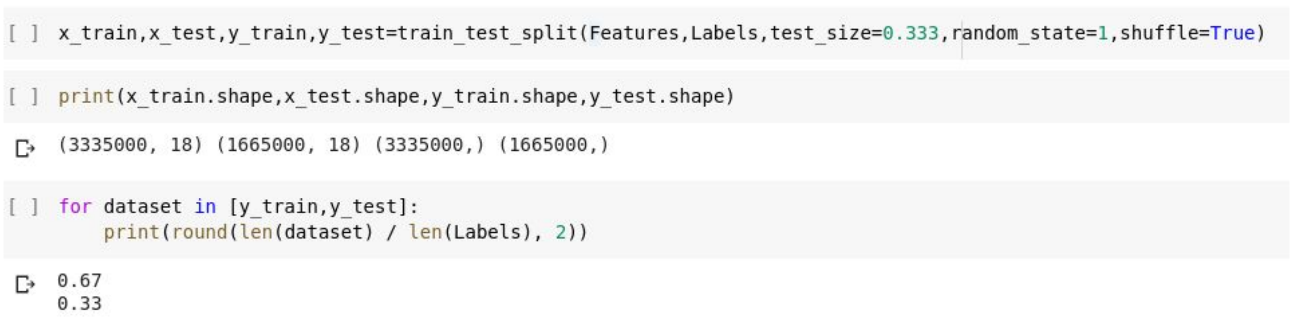
```
[6] from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split,StratifiedKFold,cross_validate
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, make_scorer
```

The Decision Tree algorithm can be imported from the sci-kit Learn package which is a Machine learning package in Python. A Binary Decision Tree Classifier is imported since the Classification is a Binary Classification Problem.

Splitting the Dataset into Test Data and the Train Data:

The First phase of Machine Learning is splitting the data into Training data and Test data where the training data is fed into the ML model and the test data is used to evaluate the performance of the model bypassing the unseen data.

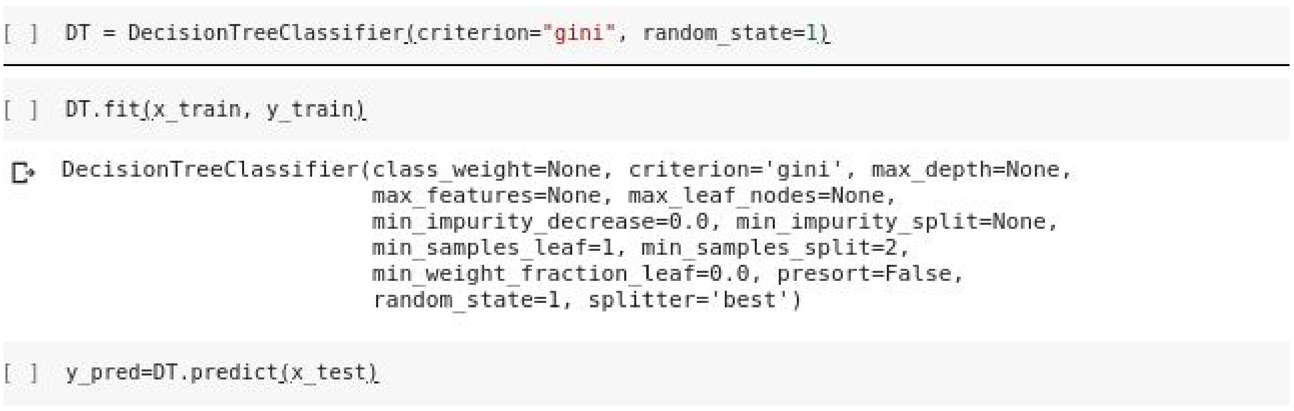
The main purpose of splitting the dataset as a train data and test data is to prevent the model from the problem of overfitting and may not perform well on the unseen data.



```
[ ]  x_train,x_test,y_train,y_test=train_test_split(Features,Labels,test_size=0.333,random_state=1,shuffle=True)

[ ]  print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)

     (3335000, 18) (1665000, 18) (3335000,) (1665000,)

[ ]  for dataset in [y_train,y_test]:
         print(round(len(dataset) / len(Labels), 2))

     0.67
     0.33
```

The Dataset was split using the sci-kit learn package using the train_test_split method where 33% of the data was split into the test data and the remaining 67% as the training data.

Training and Testing the Model :



```
[ ]  DT = DecisionTreeClassifier(criterion="gini", random_state=1)

[ ]  DT.fit(x_train, y_train)

     DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort=False,
                            random_state=1, splitter='best')

[ ]  y_pred=DT.predict(x_test)
```

DT is the Decision Tree Model where criterion and random_state are passed as the hyperparameters. The first hyperparameter is the criterion which is a function used to measure the quality of the split. The criteria considered for this dataset were ‘Gini’ for the Gini impurity and random_state randomly picks a data point in the dataset.

The Formula for Gini Impurity :

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

The training data is fed to the model using the fit method and the model is tested using the test data gives y_pred as the Model Predictions I,e after training the model test data is used for predicting model performance on Unseen data.

Evaluation On the Model Using Evaluation Metrics such as the Confusion Matrix :

The confusion matrix describes the performance of the classifier on a set of test data for which the true values are Know.

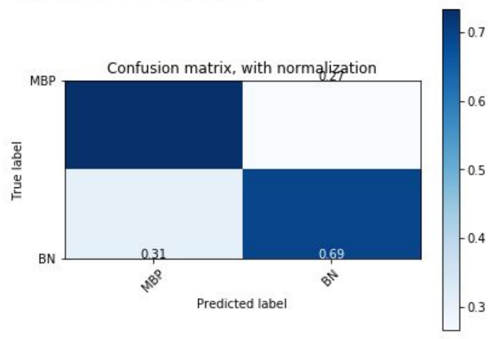
```
tn, fp, fn, tp = confusion_matrix(y_test, Predictions).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  662528
False Positives:  240168
False Negatives:  232635
True Positives:  529669
```

Plotting Confusion Matrix with Normalization :

```
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['MBP','BN'],
                      title='Confusion matrix, with normalization')
```

```
Normalized confusion matrix
[[0.73394365 0.26605635]
 [0.30517353 0.69482647]]
```



Evaluation Metrics Resulst Of Decsion Tree Classifier :

```
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy {:0.2f}%:".format(Accuracy))

Precision = tp/(tp+fp)
print("Precision {:0.2f}".format(Precision))


Recall = tp/(tp+fn)
print("Recall {:0.2f}".format(Recall))


f1 = (2*Precision*Recall)/(Precision + Recall)
print("F1 Score {:0.2f}".format(f1))
```

```
Accuracy 71.26%:
Precision 0.68
Recall 0.69
F1 Score 0.69
```

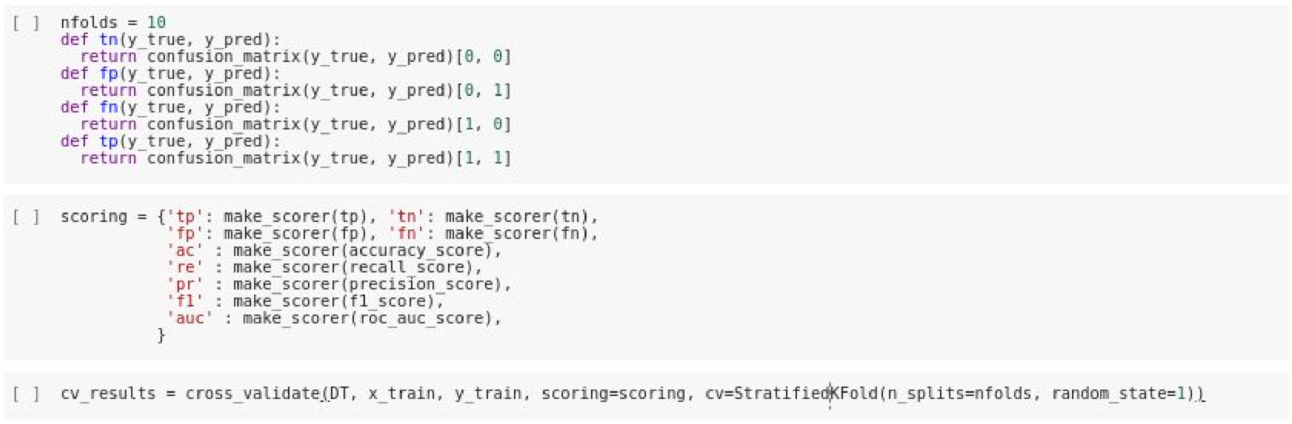
```
DT_SA.score(x_test,y_test)
```

```
0.7125618181818182
```

The accuracy of the Model on evaluating the test data is almost 72%.

Validation

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.



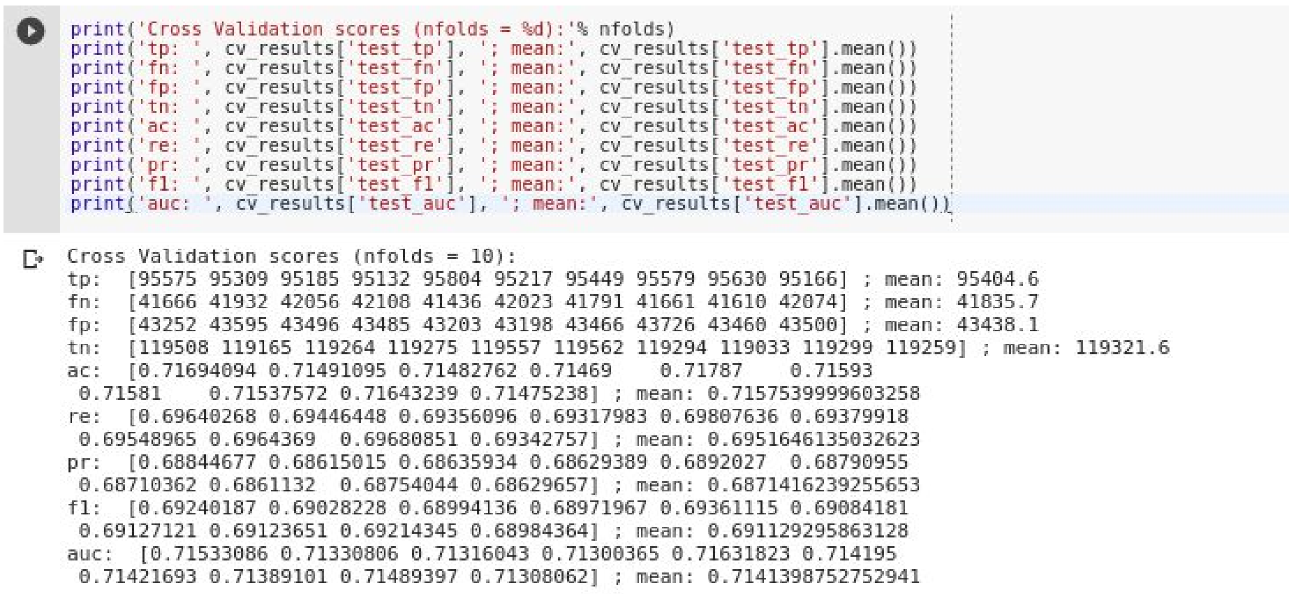
```
[ ] nfolds = 10
    def tn(y_true, y_pred):
      return confusion_matrix(y_true, y_pred)[0, 0]
    def fp(y_true, y_pred):
      return confusion_matrix(y_true, y_pred)[0, 1]
    def fn(y_true, y_pred):
      return confusion_matrix(y_true, y_pred)[1, 0]
    def tp(y_true, y_pred):
      return confusion_matrix(y_true, y_pred)[1, 1]
```

```
[ ] scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
               'fp': make_scorer(fp), 'fn': make_scorer(fn),
               'ac' : make_scorer(accuracy_score),
               're' : make_scorer(recall_score),
               'pr' : make_scorer(precision_score),
               'f1' : make_scorer(f1_score),
               'auc' : make_scorer(roc_auc_score),
              }
```

```
[ ] cv_results = cross_validate(DT, x_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

10 -fold cross-validation is used where the dataset is divided into 10 groups in which 9(k-1) groups are used for training the model and another group is used as the test data used for evaluation.

Cross-validation Results :



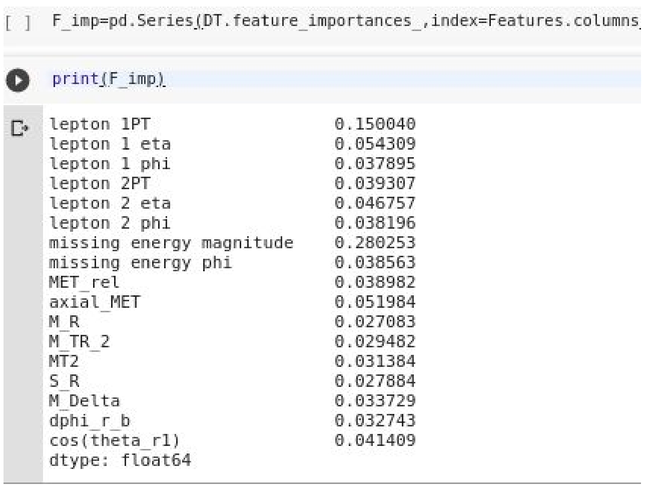
```
print('Cross Validation scores (nfolds = %d):'% nfolds)
print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())
```

```
Cross Validation scores (nfolds = 10):
tp:  [95575 95309 95185 95132 95804 95217 95449 95579 95630 95166] ; mean: 95404.6
fn:  [41666 41932 42056 42108 41436 42023 41791 41661 41610 42074] ; mean: 41835.7
fp:  [43252 43595 43496 43485 43203 43198 43466 43726 43460 43500] ; mean: 43438.1
tn:  [119508 119165 119264 119275 119557 119562 119294 119033 119299 119259] ; mean: 119321.6
ac:  [0.71694094 0.71491095 0.71482762 0.71469    0.71787    0.71593
 0.71581    0.71537572 0.71643239 0.71475238] ; mean: 0.7157539999603258
re:  [0.69640268 0.69446448 0.69356096 0.69317983 0.69807636 0.69379918
 0.69548965 0.6964369  0.69680851 0.69342757] ; mean: 0.6951646135032623
pr:  [0.68844677 0.68615015 0.68635934 0.68629389 0.6892027  0.68790955
 0.68710362 0.6861132  0.68754044 0.68629657] ; mean: 0.6871416239255653
f1:  [0.69240187 0.69028228 0.68994136 0.68971967 0.69361115 0.69084181
 0.69127121 0.69123651 0.69214345 0.68984364] ; mean: 0.691129295863128
auc:  [0.71533086 0.71330806 0.71316043 0.71300365 0.71631823 0.714195
 0.71421693 0.71389101 0.71489397 0.71308062] ; mean: 0.7141398752752941
```

The results of the Cross-validation shows different Evaluation metrics such as Accuracy, Precision, Recall, F1 Score, and Area Under Curve.

Results obtained from Training the Model :

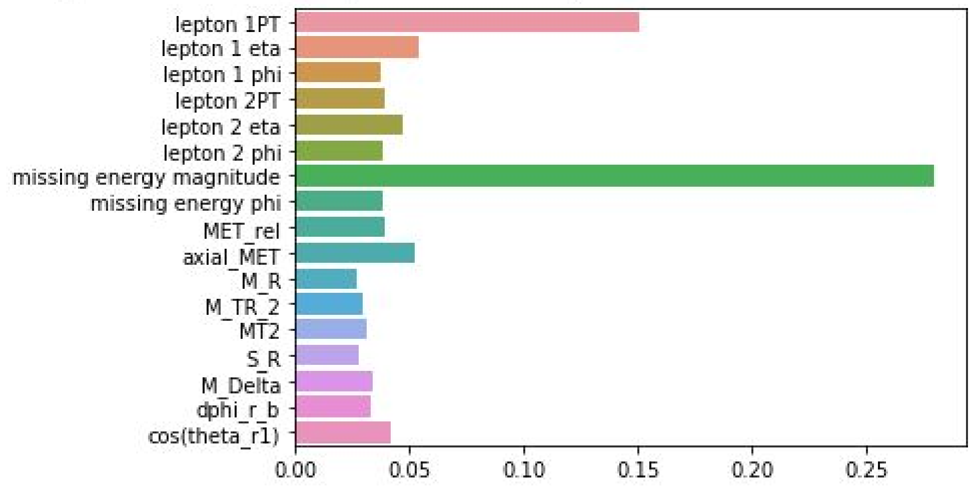
1. Feature Importance



```
[ ] F_imp=pd.Series(DT.feature_importances_,index=Features.columns

print(F_imp)

lepton 1PT                    0.150040
lepton 1 eta                  0.054309
lepton 1 phi                  0.037895
lepton 2PT                    0.039307
lepton 2 eta                  0.046757
lepton 2 phi                  0.038196
missing energy magnitude      0.280253
missing energy phi            0.038563
MET_rel                       0.038982
axial_MET                     0.051984
M_R                           0.027083
M_TR_2                        0.029482
MT2                           0.031384
S_R                           0.027884
M_Delta                       0.033729
dphi_r_b                      0.032743
cos(theta_r1)                 0.041409
dtype: float64
```

Plotting the Feature Importances :

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe124c444e0>
```

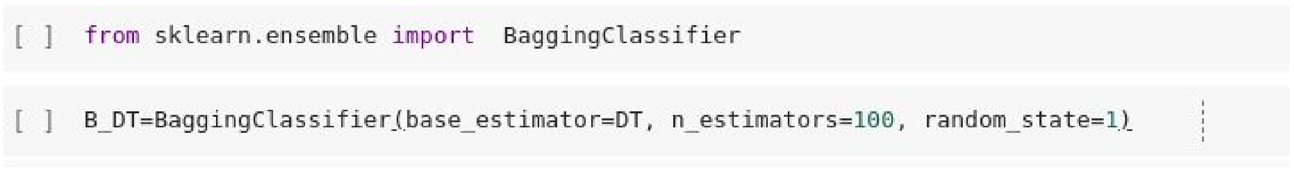


The above diagram shows the importance of all the 18 features which are used to classify the Class Label.

1.3 Ensemble of Decision Tree Classifiers

In the Ensemble of Decision Tree Classifiers, multiple models are trained in order to obtain better results. Bagging and Boosting are some of the major kinds of ensemble Learning Algorithms.

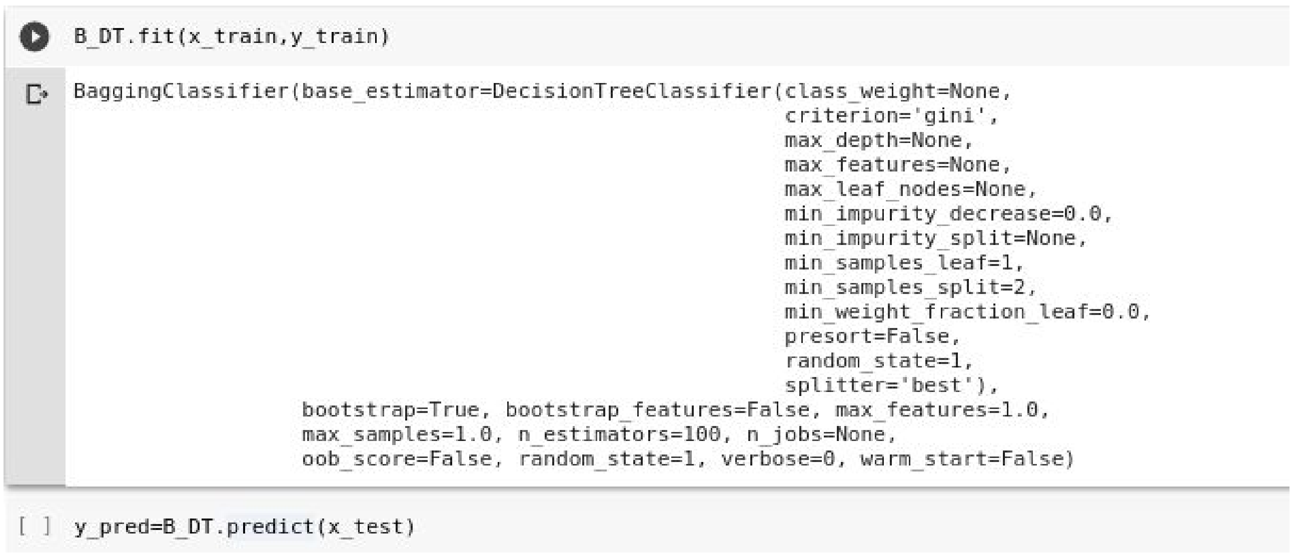
1. Bagging Classifier



```
[ ] from sklearn.ensemble import  BaggingClassifier

[ ] B_DT=BaggingClassifier(base_estimator=DT, n_estimators=100, random_state=1)
```

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions either by voting or by averaging to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a decision tree, by introducing randomization into its construction procedure and then making an ensemble out of it.

Testing and Training :



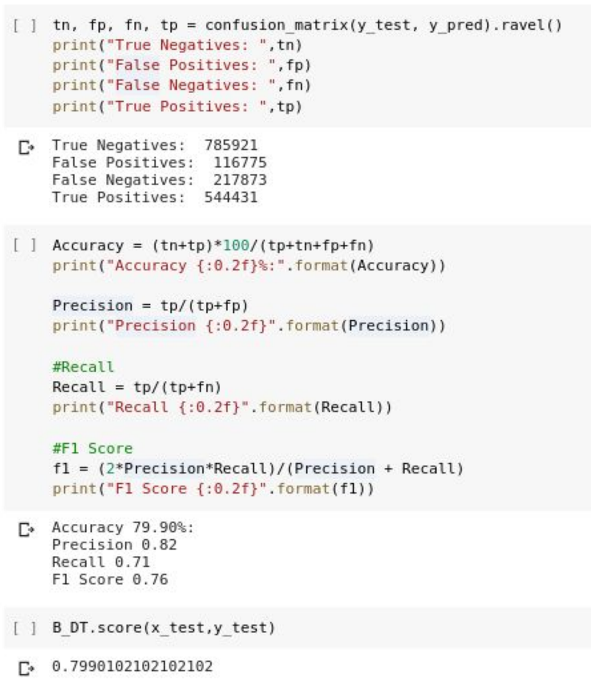
```
B_DT.fit(x_train,y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                         criterion='gini',
                                                         max_depth=None,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort=False,
                                                         random_state=1,
                                                         splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=100, n_jobs=None,
                  oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
y_pred=B_DT.predict(x_test)
```

The training data is fed to the model using the fit method and the model is tested using the test data gives y_pred as the Model Predictions I,e after training the model test data is used for predicting model performance on Unseen data. Decision Tree Classifier is used as the Base estimator. The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.

Evaluation On the Model Using Evaluation Metrics such as the Confusion Matrix :



```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  785921
False Positives:  116775
False Negatives:  217873
True Positives:  544431
```

```
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy {:0.2f}%:".format(Accuracy))

Precision = tp/(tp+fp)
print("Precision {:0.2f}".format(Precision))

#Recall
Recall = tp/(tp+fn)
print("Recall {:0.2f}".format(Recall))

#F1 Score
f1 = (2*Precision*Recall)/(Precision + Recall)
print("F1 Score {:0.2f}".format(f1))
```

```
Accuracy 79.90%:
Precision 0.82
Recall 0.71
F1 Score 0.76
```

```
B_DT.score(x_test,y_test)
```

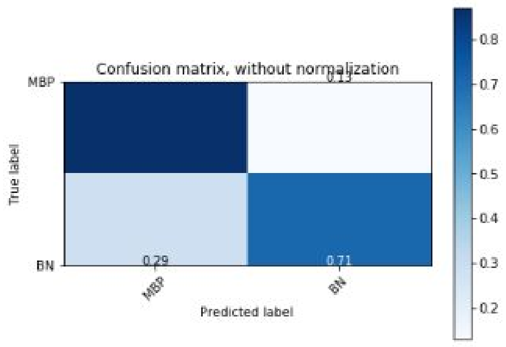
```
0.7990102102102102
```

The Accuracy of the model on evaluating the test data is almost 80%

Confusion Matrix For Bagging Decision Tree Classifier :

```
[ ]  plt.figure()
     plot_confusion_matrix(cnf_matrix, classes=['MBP','BN'],
                           title='Confusion matrix, without normalization')
```

```
Normalized confusion matrix
[[0.87063751 0.12936249]
 [0.28580855 0.71419145]]
```



Validation :

Other Ensemble Methods Used Were Random Forest Classifier (Bagging) and Gradient Boosting To get Better AUC (Area Under Curve) and increase the model Performance.

1. **Random Forest Algorithm**

```
[3]  RF_model=RandomForestClassifier(random_state=1,criterion='gini',n_estimators=100)
```

The Number of Base estimators used was 100 and used the Gini index as the criteria.

Training and Testing :

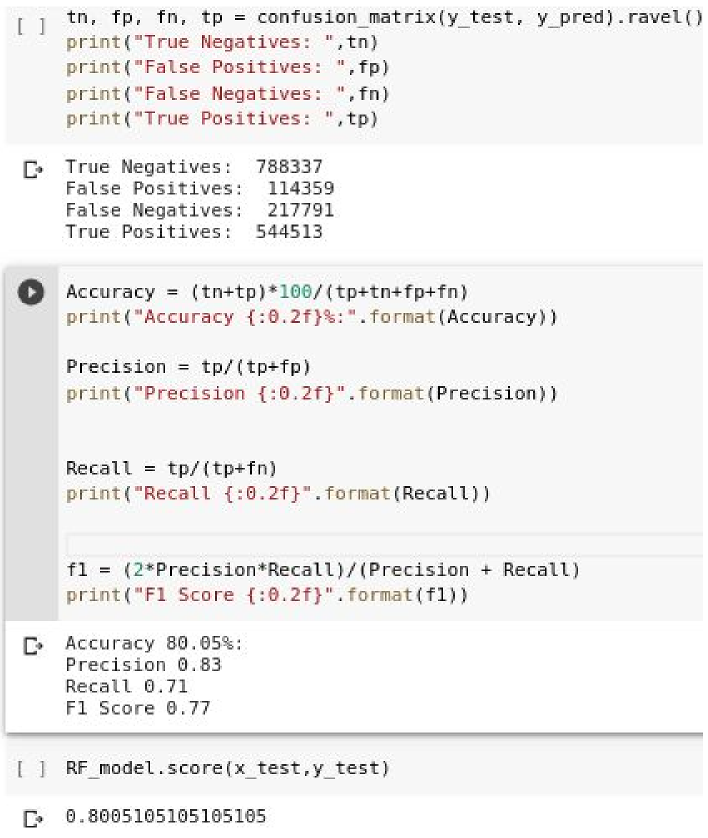
```
[ ]  RF_model.fit(x_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

```
[ ]  y_pred=RF_model.predict(x_test)
```

Results obtained from the Random Forest algorithm :

1. **Evaluation Metrics (Confusion Matrix and Accuracy).**



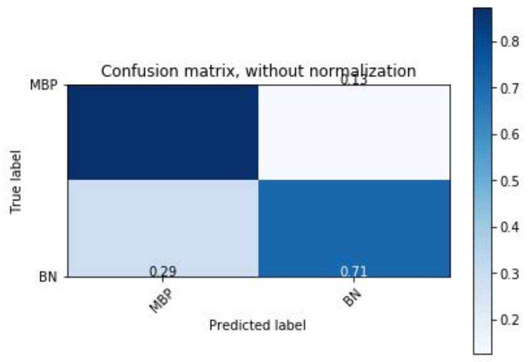
```
[ ] tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)

True Negatives:  788337
False Positives:  114359
False Negatives:  217791
True Positives:  544513
```

```
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy {:0.2f}%:".format(Accuracy))

Precision = tp/(tp+fp)
print("Precision {:0.2f}".format(Precision))


Recall = tp/(tp+fn)
print("Recall {:0.2f}".format(Recall))


f1 = (2*Precision*Recall)/(Precision + Recall)
print("F1 Score {:0.2f}".format(f1))

Accuracy 80.05%:
Precision 0.83
Recall 0.71
F1 Score 0.77
```

```
[ ] RF_model.score(x_test,y_test)

0.8005105105105105
```

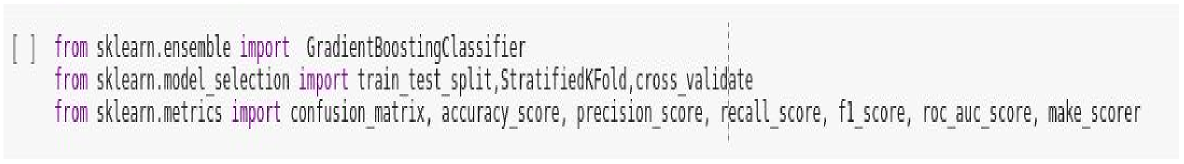
2. **Confusion Matrix Of Random Forest Algorithm :**

```
Normalized confusion matrix
[[0.87331394 0.12668606]
 [0.28570098 0.71429902]]
```



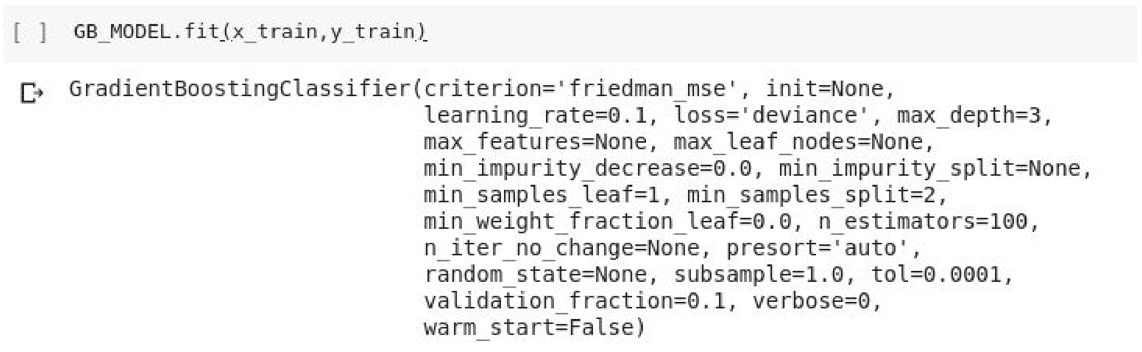
The Accuracy of the model on evaluating the test data is almost 81%

2. Gradient Boosting Algorithm



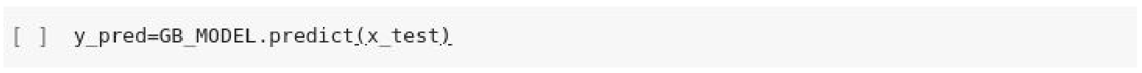
```
[ ] from sklearn.ensemble import  GradientBoostingClassifier
    from sklearn.model_selection import train_test_split,StratifiedKFold,cross_validate
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, make_scorer
```

Training and Testing :



```
[ ]  GB_MODEL.fit(x_train,y_train)
```

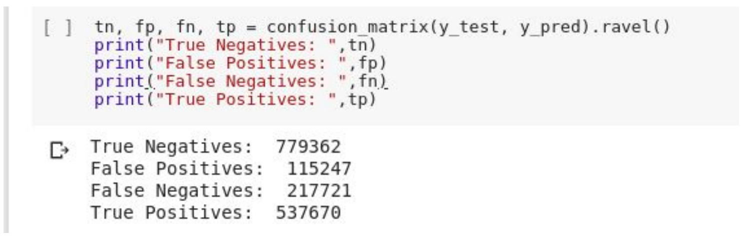
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```



```
[ ]  y_pred=GB_MODEL.predict(x_test)
```

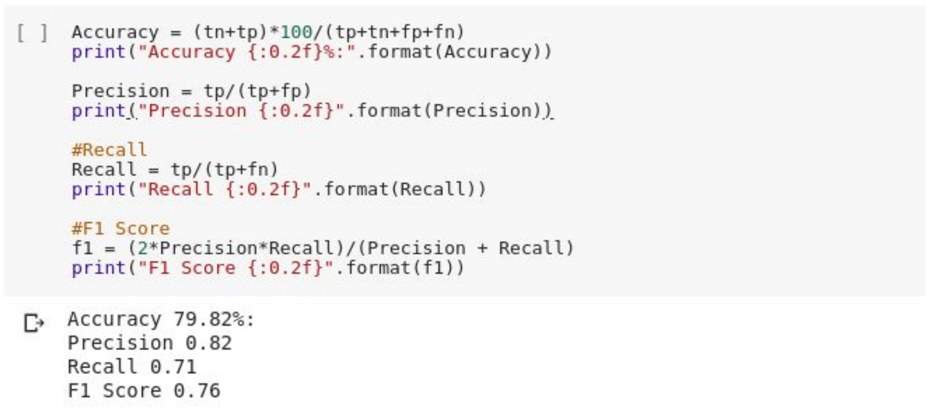
Results obtained from the Random Gradient Boosting Algorithm :

1. **Evaluation Metrics Such as Confusion Matrix and Accuracy, Precision, etc.**



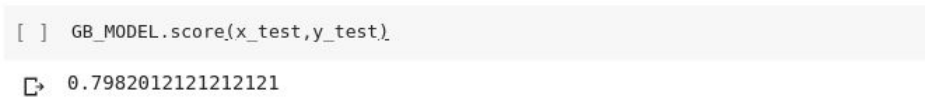
```
[ ]  tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
     print("True Negatives: ",tn)
     print("False Positives: ",fp)
     print("False Negatives: ",fn)
     print("True Positives: ",tp)
```

```
True Negatives:  779362
False Positives:  115247
False Negatives:  217721
True Positives:  537670
```



```
[ ]  Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
     print("Accuracy {:0.2f}%:".format(Accuracy))

     Precision = tp/(tp+fp)
     print("Precision {:0.2f}".format(Precision))

     #Recall
     Recall = tp/(tp+fn)
     print("Recall {:0.2f}".format(Recall))

     #F1 Score
     f1 = (2*Precision*Recall)/(Precision + Recall)
     print("F1 Score {:0.2f}".format(f1))
```

```
Accuracy 79.82%:
Precision 0.82
Recall 0.71
F1 Score 0.76
```



```
[ ]  GB_MODEL.score(x_test,y_test)
```

```
0.7982012121212121
```

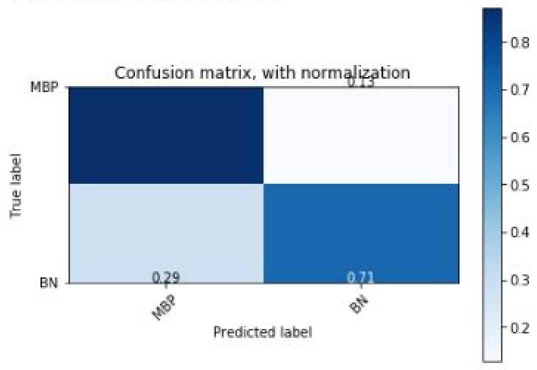
The Accuracy of the Model is almost 80%.

Confusion Matrix :

```
[ ] cnf_matrix = confusion_matrix(y_test, y_pred)
```

```
[ ] plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=['MBP','BN'],
                          title='Confusion matrix, with normalization')
```

```
Normalized confusion matrix
[[0.87146005 0.12853995]
 [0.28744054 0.71255946]]
```



The Accuracies of the Random Forest and Gradient Boosting almost resulted in the same but gave better AUC which is the best Classification Metric for Classification algorithms.

1.4 Decision Tree Classifier with Selected Data Attributes

The Dataset has 18 Independent Variables in which the last ten attributes were derived from the first 8 kinetic Data attributes.
so In this section, the DT algorithm has to be applied to the Selected data Attributes as given by the course leader.

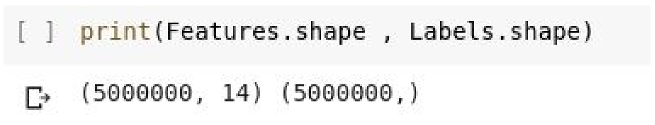
According to me Parameter Elimination by doing Statistical Tests would have been a better approach than the Selected data Attributes, where based on the hypothesis testing Certain Parameters could have been eliminated and the model could be trained on the Final important Features.

```
[ ] Features=DTSA_DF.drop(columns=['Class_Label','M_TR_2','MT2','M_Delta_R','dPhi_r_b'])
```

```
[ ] Features.head()
```

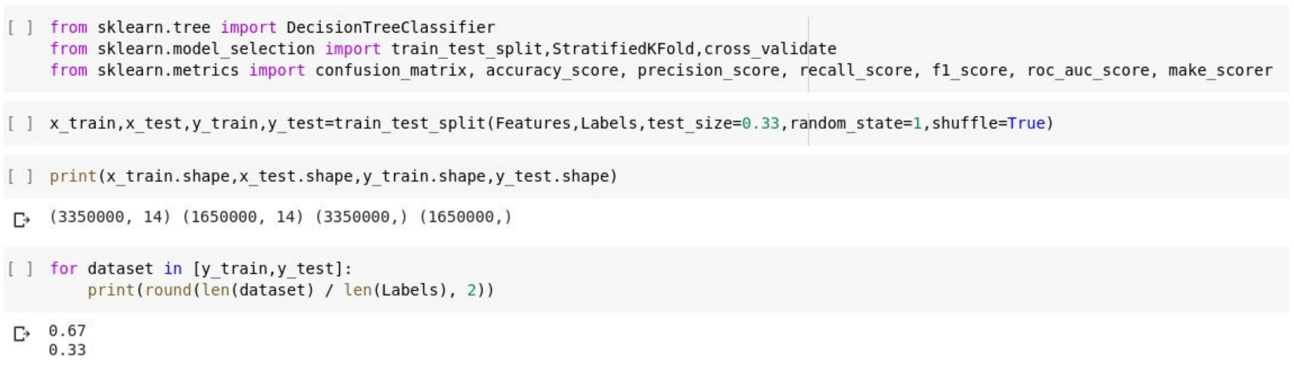
	lepton 1 pT	lepton 1 eta	lepton 1 phi	lepton 2 pT	lepton 2 eta	lepton 2 phi	missing energy magnitude	missing energy phi	MET_rel	axial MET	M_R	R	S_R	cos(theta_r1)
0	0.972861	0.653855	1.176225	1.157156	-1.739873	-0.874309	0.567765	-0.175000	0.810061	-0.252552	1.921887	0.410772	1.932632	0.040714
1	1.667973	0.064191	-1.225171	0.506102	-0.338939	1.672543	3.475464	-1.219136	0.012955	3.775174	1.045977	0.481928	0.448410	0.377584
2	0.444840	-0.134298	-0.709972	0.451719	-1.613871	-0.768661	1.219918	0.504026	1.831248	-0.431385	0.526283	1.587535	0.603498	0.180910
3	0.381256	-0.976145	0.693152	0.448959	0.891753	-0.677328	2.033060	1.533041	3.046260	-1.005285	0.569386	1.582217	0.761215	0.090719
4	1.309996	-0.690089	-0.676259	1.589283	-0.693326	0.622907	1.087562	-0.381742	0.589204	1.365479	1.179295	0.728563	1.083158	0.094859

The shape of the Features and Labels :



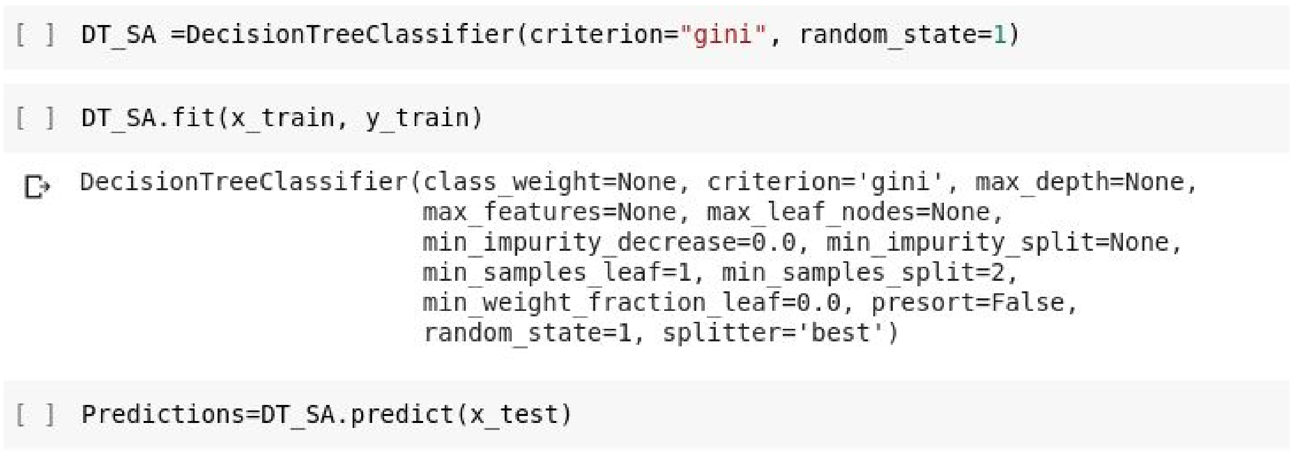
```
[ ]  print(Features.shape , Labels.shape)

     (5000000, 14) (5000000,)
```

Training and Testing :



```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split,StratifiedKFold,cross_validate
    from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, make_scorer

[ ] x_train,x_test,y_train,y_test=train_test_split(Features,Labels,test_size=0.33,random_state=1,shuffle=True)

[ ] print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)

    (3350000, 14) (1650000, 14) (3350000,) (1650000,)

[ ] for dataset in [y_train,y_test]:
        print(round(len(dataset) / len(Labels), 2))

    0.67
    0.33
```

Similar to the Decision Tree on all attributes, the Training and Test was split into 67% and 33%.

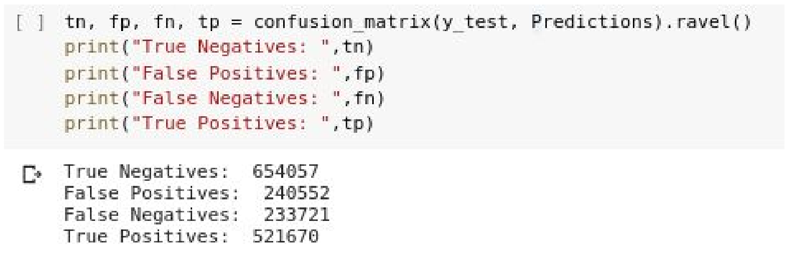


```
[ ]  DT_SA =DecisionTreeClassifier(criterion="gini", random_state=1)

[ ]  DT_SA.fit(x_train, y_train)

     DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort=False,
                            random_state=1, splitter='best')

[ ]  Predictions=DT_SA.predict(x_test)
```

DT_SA represents the Decision Tree Model on Selected Attributes.

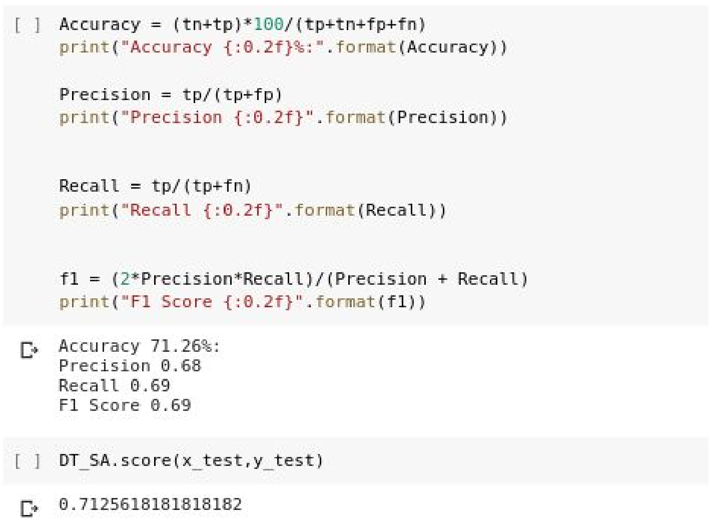
The training data is fed to the model using the fit method and the model is tested using the test data gives y_pred as the Model Predictions I,e after training the model test data is used for predicting model performance on Unseen data. Decision Tree Classifier is used as the Base estimator. The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.

Evaluation On the Model Using Evaluation Metrics such as the Confusion Matrix :



```
[ ]  tn, fp, fn, tp = confusion_matrix(y_test, Predictions).ravel()
     print("True Negatives: ",tn)
     print("False Positives: ",fp)
     print("False Negatives: ",fn)
     print("True Positives: ",tp)
```

```
True Negatives:  654057
False Positives:  240552
False Negatives:  233721
True Positives:  521670
```



```
[ ]  Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
     print("Accuracy {:0.2f}%:".format(Accuracy))

     Precision = tp/(tp+fp)
     print("Precision {:0.2f}".format(Precision))


     Recall = tp/(tp+fn)
     print("Recall {:0.2f}".format(Recall))


     f1 = (2*Precision*Recall)/(Precision + Recall)
     print("F1 Score {:0.2f}".format(f1))
```

```
Accuracy 71.26%:
Precision 0.68
Recall 0.69
F1 Score 0.69
```

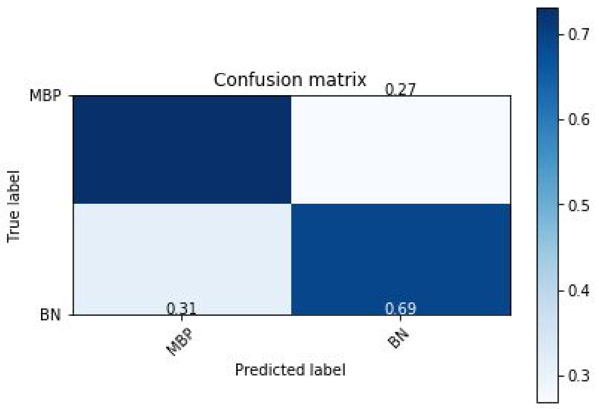
```
[ ]  DT_SA.score(x_test,y_test)
```

```
0.7125618181818182
```

Evaluating the test data the model yielded an Accuracy of 71.26%

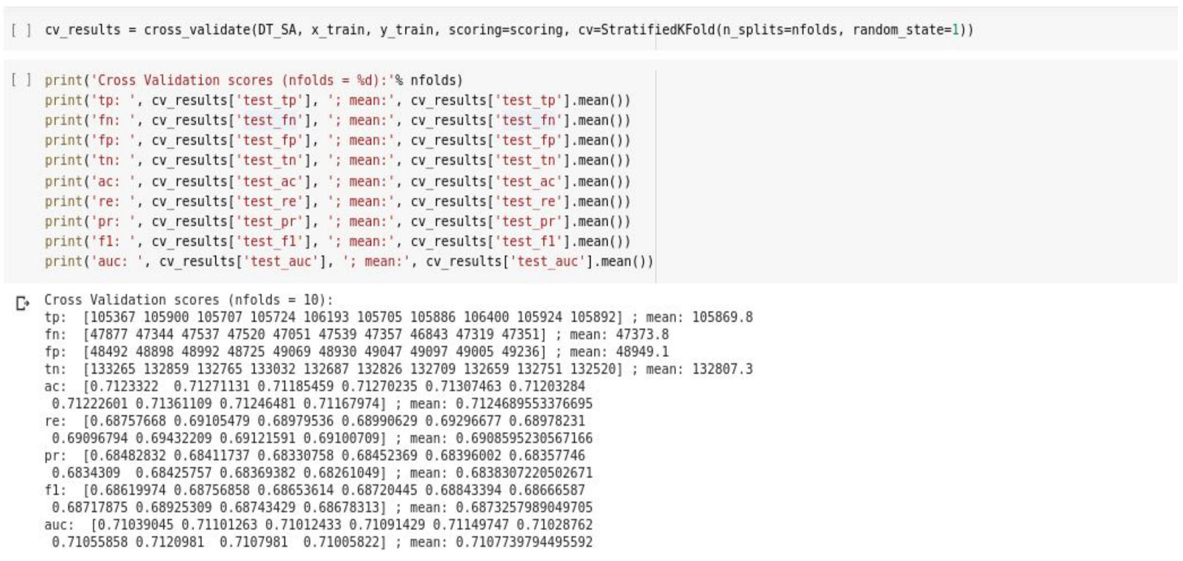
Confusion Matrix :

```
Normalized confusion matrix
[[0.73110934 0.26889066]
 [0.309404   0.690596  ]]
```



Validation :

Performing 10 fold Stratified Cross-Validation On the Selected data Attributes :



```
[ ] cv_results = cross_validate(DT_SA, x_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

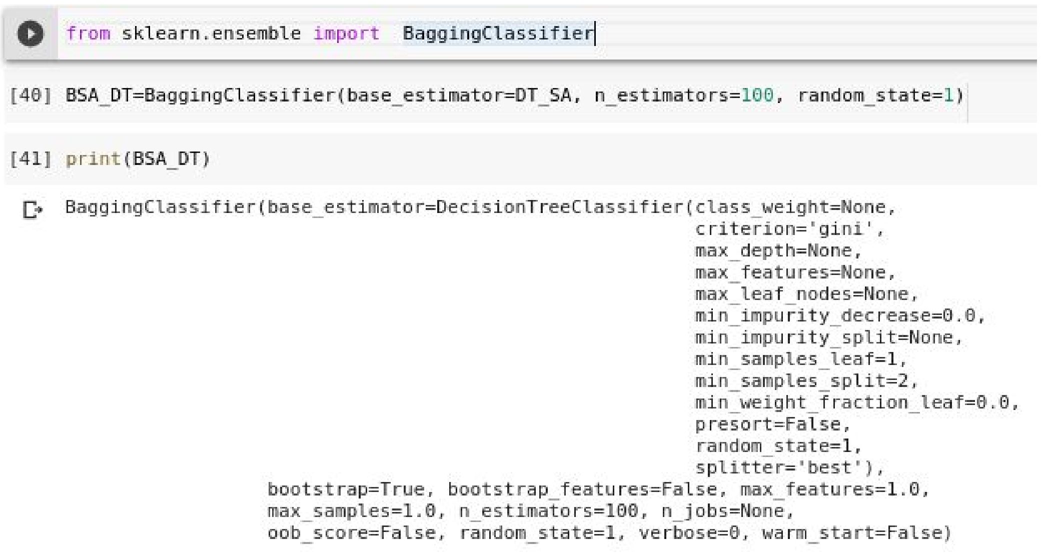
```
[ ] print('Cross Validation scores (nfolds = %d):'% nfolds)
    print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
    print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
    print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
    print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
    print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
    print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
    print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
    print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
    print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())
```

```
Cross Validation scores (nfolds = 10):
tp:  [105367 105900 105707 105724 106193 105705 105886 106400 105924 105892] ; mean: 105869.8
fn:  [47877 47344 47537 47520 47051 47539 47357 46843 47319 47351] ; mean: 47373.8
fp:  [48492 48898 48992 48725 49069 48930 49047 49097 49005 49236] ; mean: 48949.1
tn:  [133265 132859 132765 133032 132687 132826 132709 132659 132751 132520] ; mean: 132807.3
ac:  [0.7123322  0.71271131 0.71185459 0.71270235 0.71307463 0.71203284
 0.71222601 0.71361109 0.71246481 0.71167974] ; mean: 0.7124689553376695
re:  [0.68757668 0.69105479 0.68979536 0.68990629 0.69296677 0.68978231
 0.69096794 0.69432209 0.69121591 0.69100709] ; mean: 0.6908595230567166
pr:  [0.68482832 0.68411737 0.68330758 0.68452369 0.68396002 0.68357746
 0.6834309  0.68425757 0.68369382 0.68261049] ; mean: 0.6838307220502671
f1:  [0.68619974 0.68756858 0.68653614 0.68720445 0.68843394 0.68666587
 0.68717875 0.68925309 0.68743429 0.68678313] ; mean: 0.6873257989049705
auc:  [0.71039045 0.71101263 0.71012433 0.71091429 0.71149747 0.71028762
 0.71055858 0.7120981  0.7107981  0.71005822] ; mean: 0.7107739794495592
```

The Results of the Cross-validation can be used to compare the significance perf

1.5 Ensemble of Decision Tree Classifiers with Selected Data Attributes

The same approach is used for the Ensemble of Decision tree Classifiers with Selected Data Attributes where Bagging Tree Classifier is used as the Ensemble Algorithm and Decision Tree Model from the selected Attributes is used as the Base estimator.



```
from sklearn.ensemble import  BaggingClassifier
```

```
[40] BSA_DT=BaggingClassifier(base_estimator=DT_SA, n_estimators=100, random_state=1)
```

```
[41] print(BSA_DT)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                        criterion='gini',
                                                        max_depth=None,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=2,
                                                        min_weight_fraction_leaf=0.0,
                                                        presort=False,
                                                        random_state=1,
                                                        splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=100, n_jobs=None,
                  oob_score=False, random_state=1, verbose=0, warm_start=False)
```

n_estimators are given as 100 which is the number of trees in the Bagging Classifier and random state is given as 1.

Training and Testing :

```
[ ]  BSA_DT.fit(x_train,y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                        criterion='gini',
                                                        max_depth=None,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=2,
                                                        min_weight_fraction_leaf=0.0,
                                                        presort=False,
                                                        random_state=1,
                                                        splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=100, n_jobs=None,
                  oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
[ ]  Predictions=BSA_DT.predict(x_test)
```

Evaluation Metrics obtained by testing on the test data :

```
tn, fp, fn, tp = confusion_matrix(y_test, Predictions).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

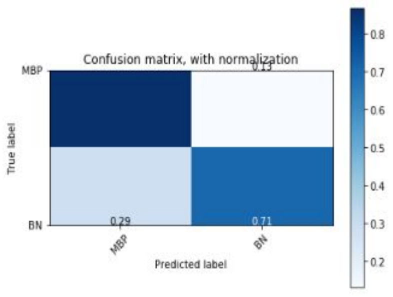
```
True Negatives:  776849
False Positives:  117760
False Negatives:  218120
True Positives:  537271
```

Confusion Matrix :

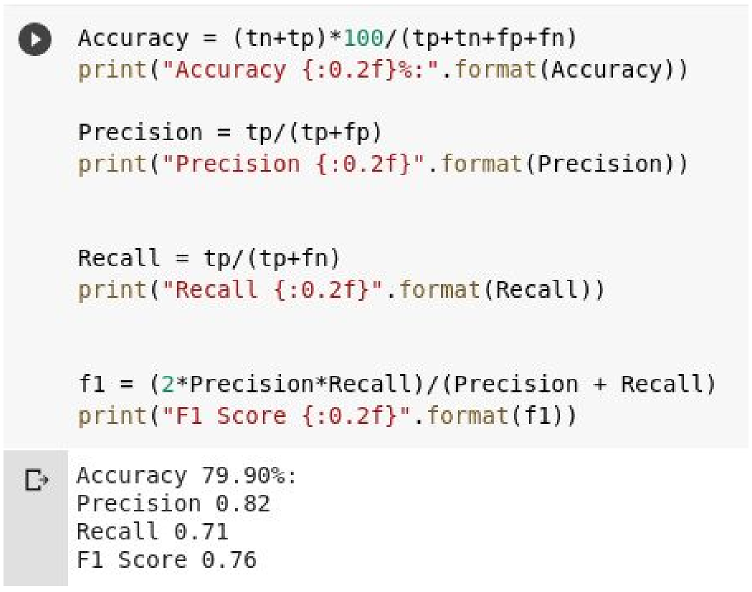
```
[ ]  cnf_matrix = confusion_matrix(y_test, Predictions)
```

```
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['MBP','BN'],
                      title='Confusion matrix, with normalization')
```

```
Normalized confusion matrix
[[0.86836707 0.13163293]
 [0.28875112 0.71124888]]
```



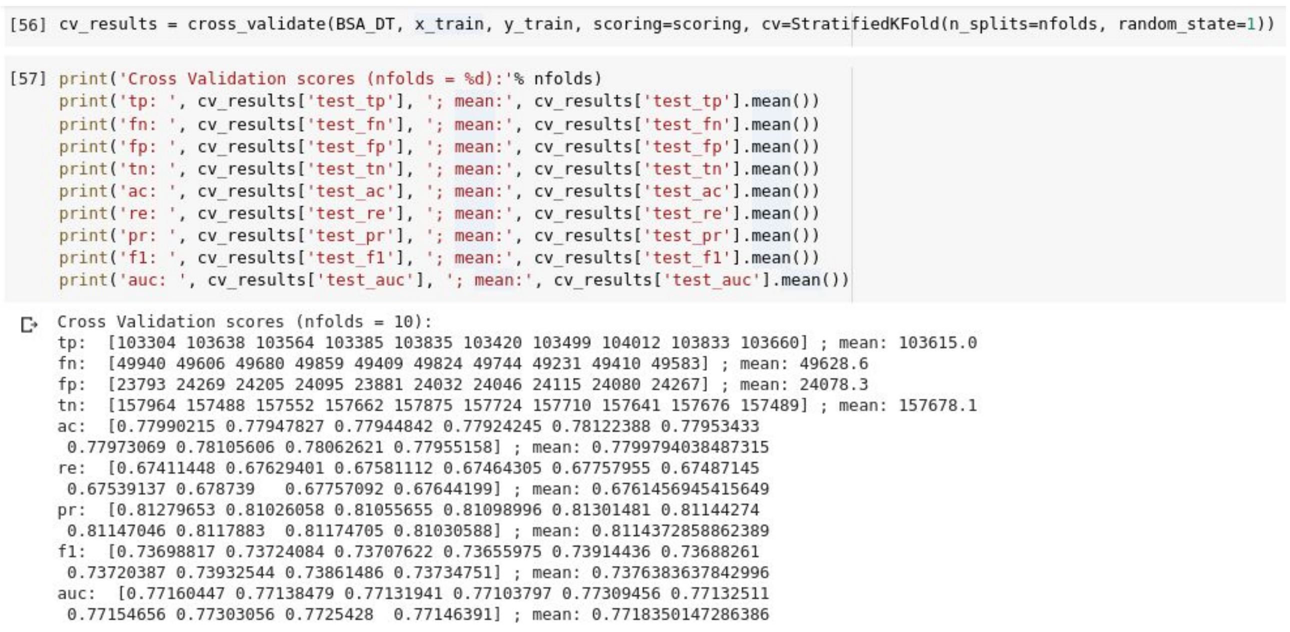
Accuracy :



```
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy {:0.2f}%:".format(Accuracy))

Precision = tp/(tp+fp)
print("Precision {:0.2f}".format(Precision))


Recall = tp/(tp+fn)
print("Recall {:0.2f}".format(Recall))


f1 = (2*Precision*Recall)/(Precision + Recall)
print("F1 Score {:0.2f}".format(f1))
```

```
Accuracy 79.90%:
Precision 0.82
Recall 0.71
F1 Score 0.76
```

Validation :



```
[56] cv_results = cross_validate(BSA_DT, x_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

```
[57] print('Cross Validation scores (nfolds = %d):'% nfolds)
     print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
     print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
     print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
     print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
     print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
     print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
     print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
     print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
     print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())
```

```
Cross Validation scores (nfolds = 10):
tp:  [103304 103638 103564 103385 103835 103420 103499 104012 103833 103660] ; mean: 103615.0
fn:  [49940 49606 49680 49859 49409 49824 49744 49231 49410 49583] ; mean: 49628.6
fp:  [23793 24269 24205 24095 23881 24032 24046 24115 24080 24267] ; mean: 24078.3
tn:  [157964 157488 157552 157662 157875 157724 157710 157641 157676 157489] ; mean: 157678.1
ac:  [0.77990215 0.77947827 0.77944842 0.77924245 0.78122388 0.77953433
 0.77973069 0.78105606 0.78062621 0.77955158] ; mean: 0.7799794038487315
re:  [0.67411448 0.67629401 0.67581112 0.67464305 0.67757955 0.67487145
 0.67539137 0.678739   0.67757092 0.67644199] ; mean: 0.6761456945415649
pr:  [0.81279653 0.81026058 0.81055655 0.81098996 0.81301481 0.81144274
 0.81147046 0.8117883  0.81174705 0.81030588] ; mean: 0.8114372858862389
f1:  [0.73698817 0.73724084 0.73707622 0.73655975 0.73914436 0.73688261
 0.73720387 0.73932544 0.73861486 0.73734751] ; mean: 0.7376383637842996
auc:  [0.77160447 0.77138479 0.77131941 0.77103797 0.77309456 0.77132511
 0.77154656 0.77303056 0.7725428  0.77146391] ; mean: 0.7718350147286386
```

1.6 Comparative Evaluation of the Classifiers

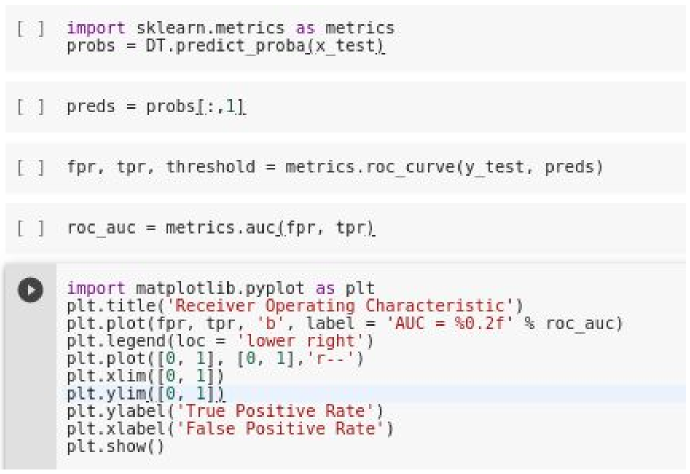
Evaluating the machine learning algorithm is an essential part of Data analysis. The model may give satisfying results when evaluated using a metric such as accuracy_score but may give poor results when evaluated against other metrics such as logarithmic_loss or any other such metric. Most of the time we use classification accuracy to measure the performance of our model, however, it is not enough to truly judge our model.

Classification Accuracy

It is the ratio of the number of correct predictions to the total number of input samples. It works well only if there are an equal number of samples belonging to each class.

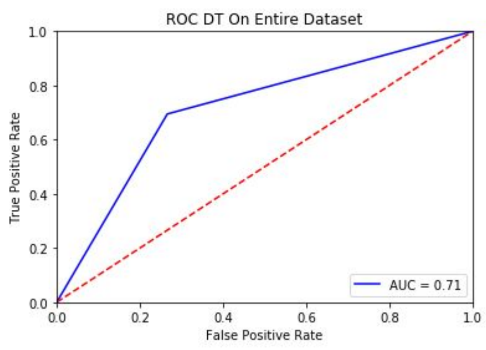
Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problems. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.

AUC Curve For Decision Tree on All Attributes :



```
[ ]  import sklearn.metrics as metrics
     probs = DT.predict_proba(x_test)

[ ]  preds = probs[:,1]

[ ]  fpr, tpr, threshold = metrics.roc_curve(y_test, preds)

[ ]  roc_auc = metrics.auc(fpr, tpr)

     import matplotlib.pyplot as plt
     plt.title('Receiver Operating Characteristic')
     plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
     plt.legend(loc = 'lower right')
     plt.plot([0, 1], [0, 1],'r--')
     plt.xlim([0, 1])
     plt.ylim([0, 1])
     plt.ylabel('True Positive Rate')
     plt.xlabel('False Positive Rate')
     plt.show()
```

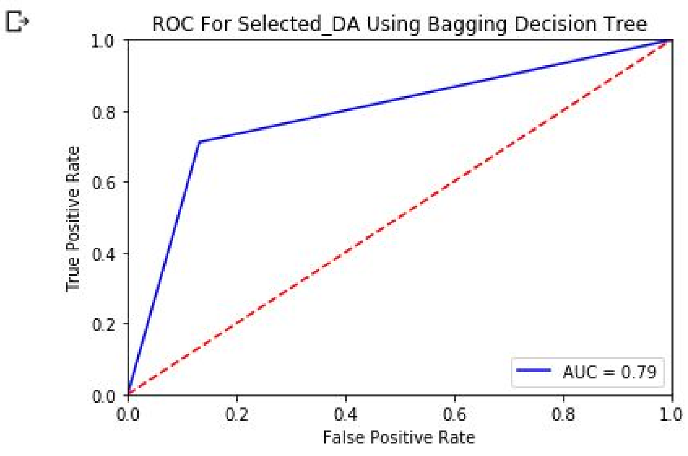
Code For plotting the AUC Curve or ROC



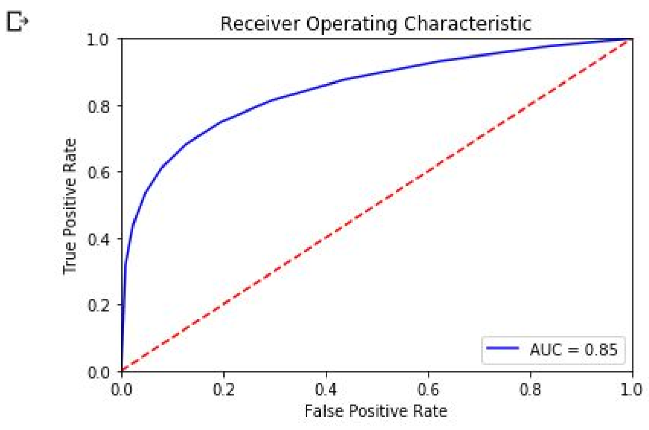
It has an AUC of 0.71, so the model classifies the two classes(made by particle/background noise) 71% correctly.

AUC Curve For Bagging Decision Tree on All Attributes :

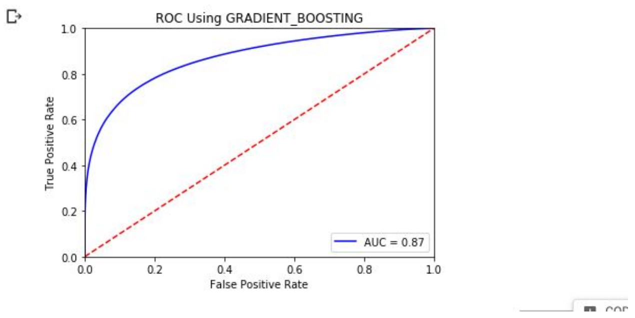
1.7 Conclusion



AUC for Random Forest Algorithm :



AUC for Gradient Boosting on Entire Dataset.



Even though Accuracies were similar the models had huge difference in their AUCs. Hence among all the Above four algorithms Gradient boosting was able to classify the classes 87% correctly.

Hypothesis Testing to check the statistical Significance of the Machine Learning Model Using Area under Curve Values obtained from Cross-validation Results.

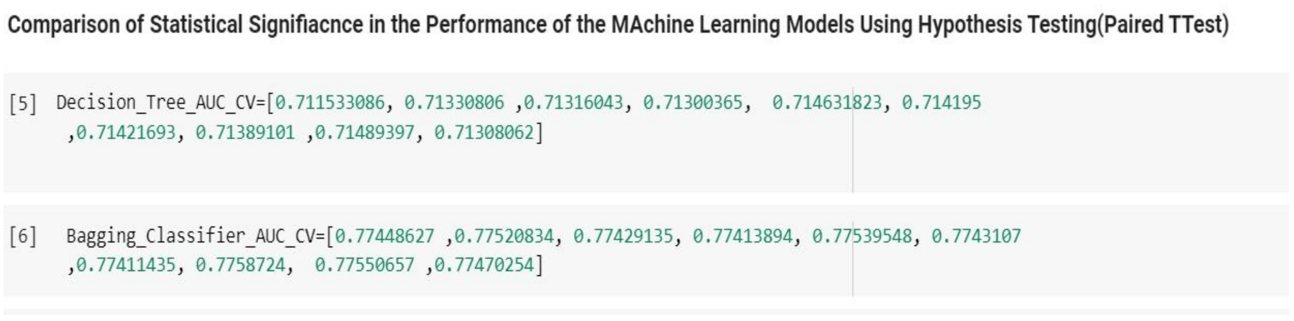
Comparing machine learning methods and selecting a final model is a common operation in applied machine learning.

Models are commonly evaluated using resampling methods like k-fold cross-validation from which mean skill scores are calculated and compared directly. Although simple, this approach can be misleading as it is hard to know whether the difference between mean skill scores is real or the result of a statistical fluke.

Statistical significance tests are designed to address this problem and quantify the likelihood of the samples of skill scores being observed given the assumption that they were drawn from the same distribution. If this assumption, or null hypothesis, is rejected, it suggests that the difference in skill scores is statistically significant.

Null Hypothesis: The performance between models are statistically significant.

Paired T-Test :

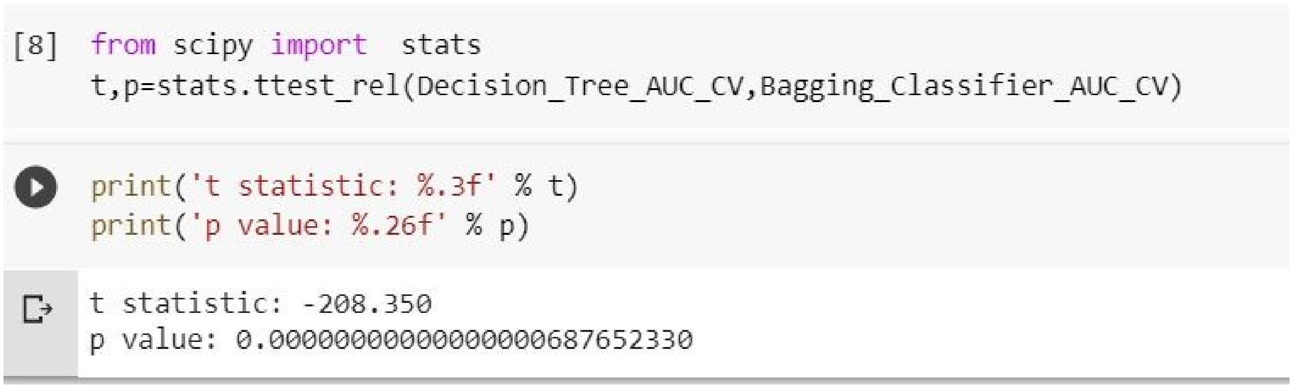


Comparison of Statistical Signifiacnce in the Performance of the MAchine Learning Models Using Hypothesis Testing(Paired TTest)

```
[5] Decision_Tree_AUC_CV=[0.711533086, 0.71330806 ,0.71316043, 0.71300365,  0.714631823, 0.714195
     ,0.71421693, 0.71389101 ,0.71489397, 0.71308062]
```

```
[6]  Bagging_Classifier_AUC_CV=[0.77448627 ,0.77520834, 0.77429135, 0.77413894, 0.77539548, 0.7743107
     ,0.77411435, 0.7758724,  0.77550657 ,0.77470254]
```

The above figure shows the AUC Results obtained from the Cross-Validation of DT and Bagging DT on all attributes.

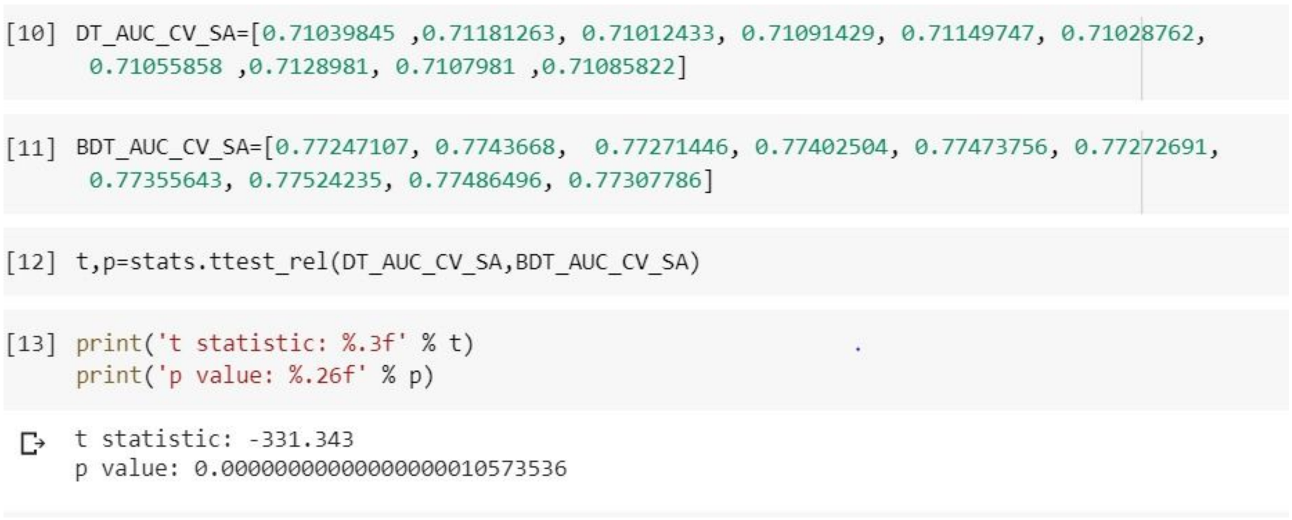


```
[8] from scipy import  stats
    t,p=stats.ttest_rel(Decision_Tree_AUC_CV,Bagging_Classifier_AUC_CV)
```

```
print('t statistic: %.3f' % t)
print('p value: %.26f' % p)
```

```
t statistic: -208.350
p value: 0.00000000000000000687652330
```

Since the p-value is less than the threshold significance of 0.05, the null hypothesis is accepted and there is Statistical Significance in the performance of the two models.



```
[10] DT_AUC_CV_SA=[0.71039845 ,0.71181263, 0.71012433, 0.71091429, 0.71149747, 0.71028762,
      0.71055858 ,0.7128981, 0.7107981 ,0.71085822]
```

```
[11] BDT_AUC_CV_SA=[0.77247107, 0.7743668,  0.77271446, 0.77402504, 0.77473756, 0.77272691,
      0.77355643, 0.77524235, 0.77486496, 0.77307786]
```

```
[12] t,p=stats.ttest_rel(DT_AUC_CV_SA,BDT_AUC_CV_SA)
```

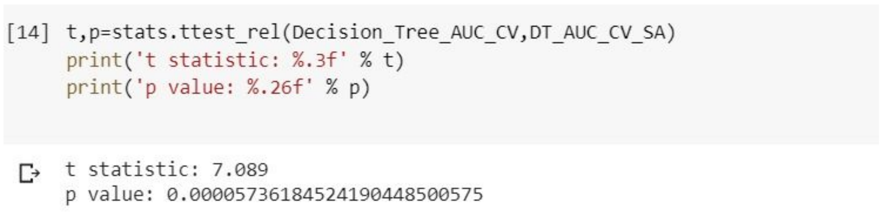
```
[13] print('t statistic: %.3f' % t)
     print('p value: %.26f' % p)
```

```
t statistic: -331.343
p value: 0.00000000000000000010573536
```

The above figure shows Hypothesis testing on Selected Attributes between the DT and Bagging DT Models.

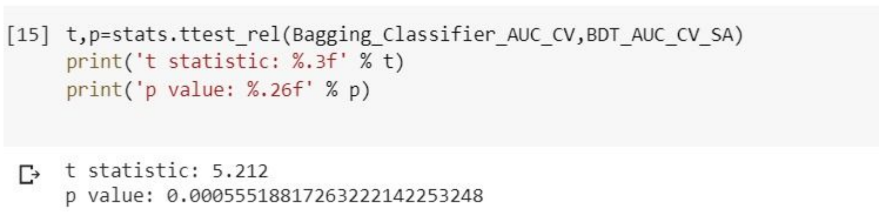
P-value is less than 0.05 hence there is a statistical significance in the performance of these models.

Hypothesis Test between DT on all Attributes and Selected Attributes and also between the bagging Classifiers.



```
[14] t,p=stats.ttest_rel(Decision_Tree_AUC_CV,DT_AUC_CV_SA)
     print('t statistic: %.3f' % t)
     print('p value: %.26f' % p)
```

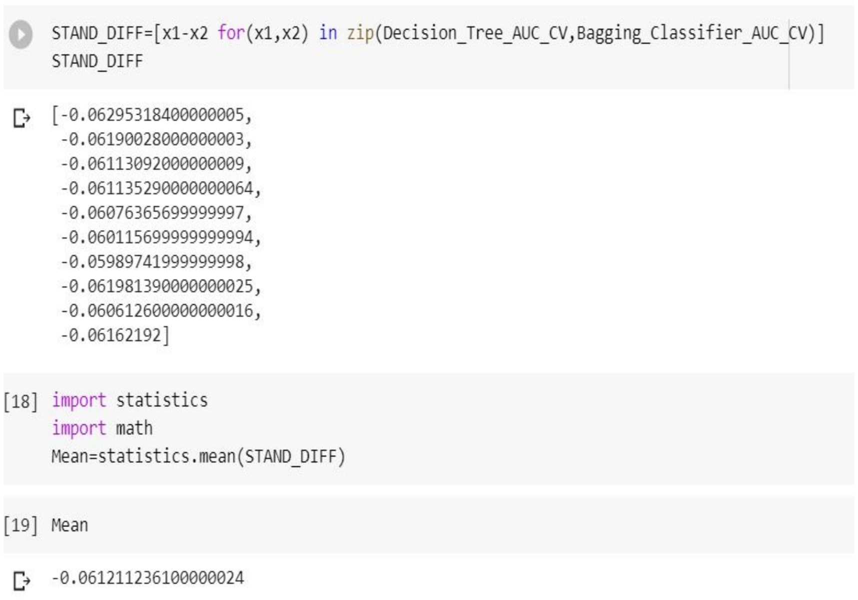
```
t statistic: 7.089
p value: 0.00005736184524190448500575
```



```
[15] t,p=stats.ttest_rel(Bagging_Classifier_AUC_CV,BDT_AUC_CV_SA)
     print('t statistic: %.3f' % t)
     print('p value: %.26f' % p)
```

```
t statistic: 5.212
p value: 0.00055518817263222142253248
```

Manual Calculation :



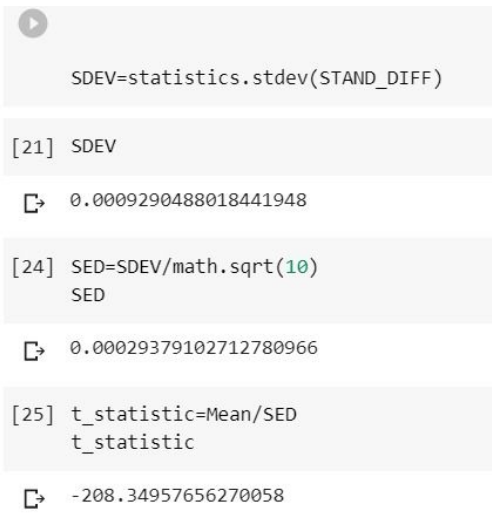
```
STAND_DIFF=[x1-x2 for(x1,x2) in zip(Decision_Tree_AUC_CV,Bagging_Classifier_AUC_CV)]
STAND_DIFF
```

```
[-0.06295318400000005,
 -0.06190028000000003,
 -0.06113092000000009,
 -0.061135290000000064,
 -0.06076365699999997,
 -0.060115699999999994,
 -0.05989741999999998,
 -0.061981390000000025,
 -0.060612600000000016,
 -0.06162192]
```

```
[18] import statistics
     import math
     Mean=statistics.mean(STAND_DIFF)
```

```
[19] Mean
```

```
-0.061211236100000024
```



```
SDEV=statistics.stdev(STAND_DIFF)
```

```
[21] SDEV
```

```
0.0009290488018441948
```

```
[24] SED=SDEV/math.sqrt(10)
     SED
```

```
0.00029379102712780966
```

```
[25] t_statistic=Mean/SED
     t_statistic
```

```
-208.34957656270058
```

1.7 Conclusion

AUC and ROC are important evaluation metrics for calculating the performance of any classification model's performance. Therefore getting to know how they are calculated is as essential as using them. Statistical significance tests are an important tool to help to interpret the results from machine learning experiments. Additionally, the findings from these tools can help you better and more confidently present your experimental results and choose the right algorithms and configurations for your predictive modeling problem.