# Analysis Report Decision Tree

The Data set of engineering students have following features obtained from the web with addition of few subject fields. It has 1000 Patterns and 52 features.

It also has two classes [6] and[7]

gender

race/ethnicity

parental level of education lunch

test preparation course m1 score

m2 scorep

physics score

m3 score

m4 score

DS score

CO score

LD score

SOM score

ED score

OS-2 score

:

:

:

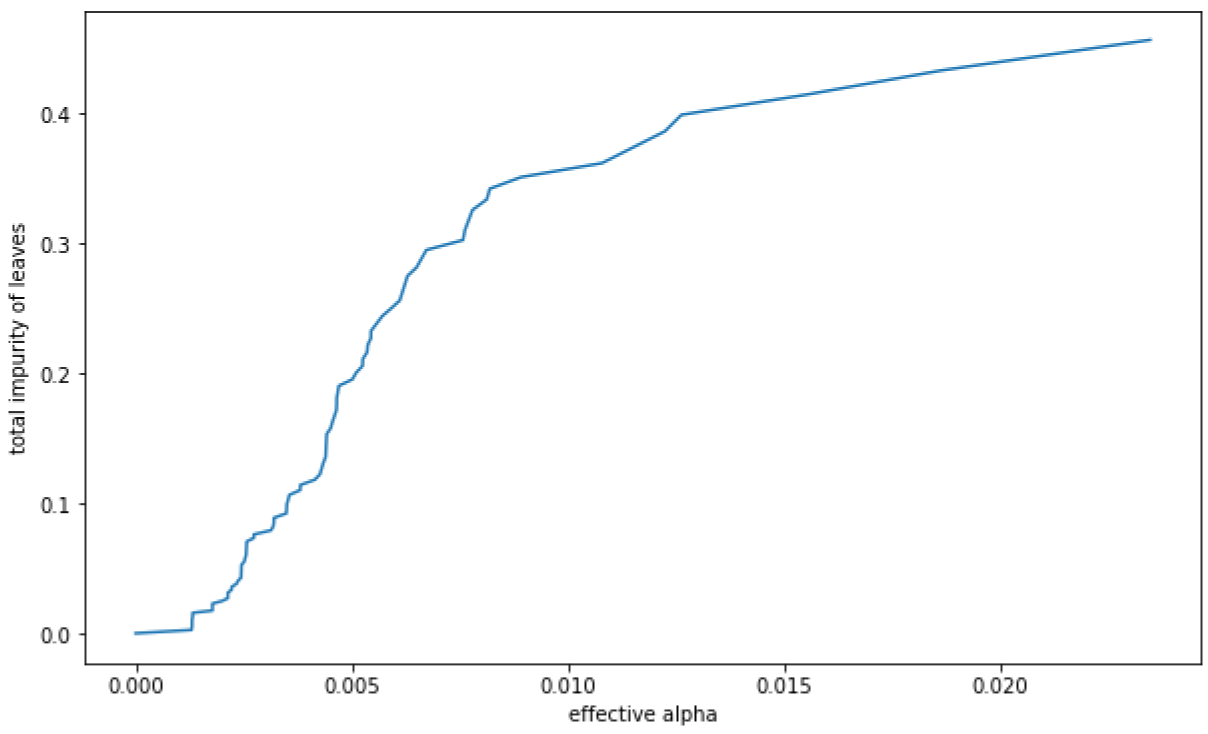
SUB30

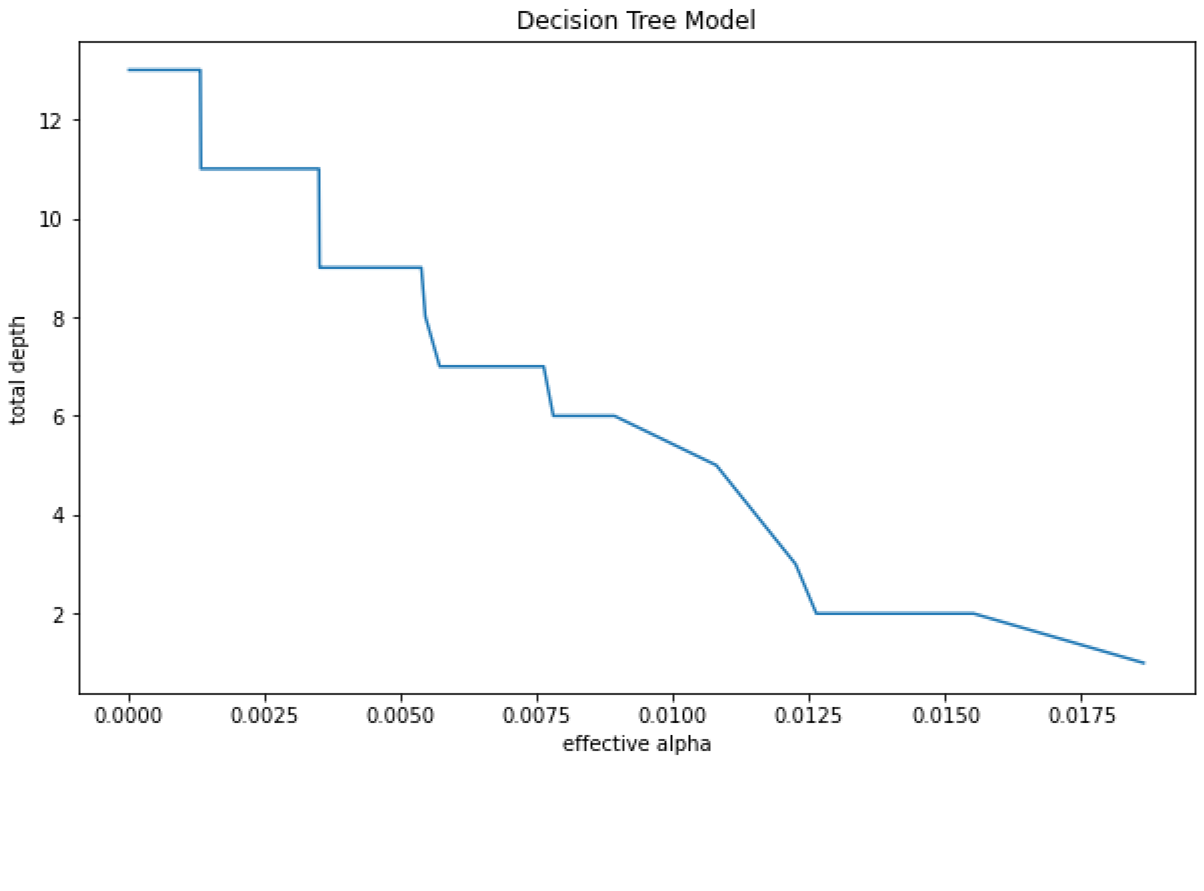
SUB31

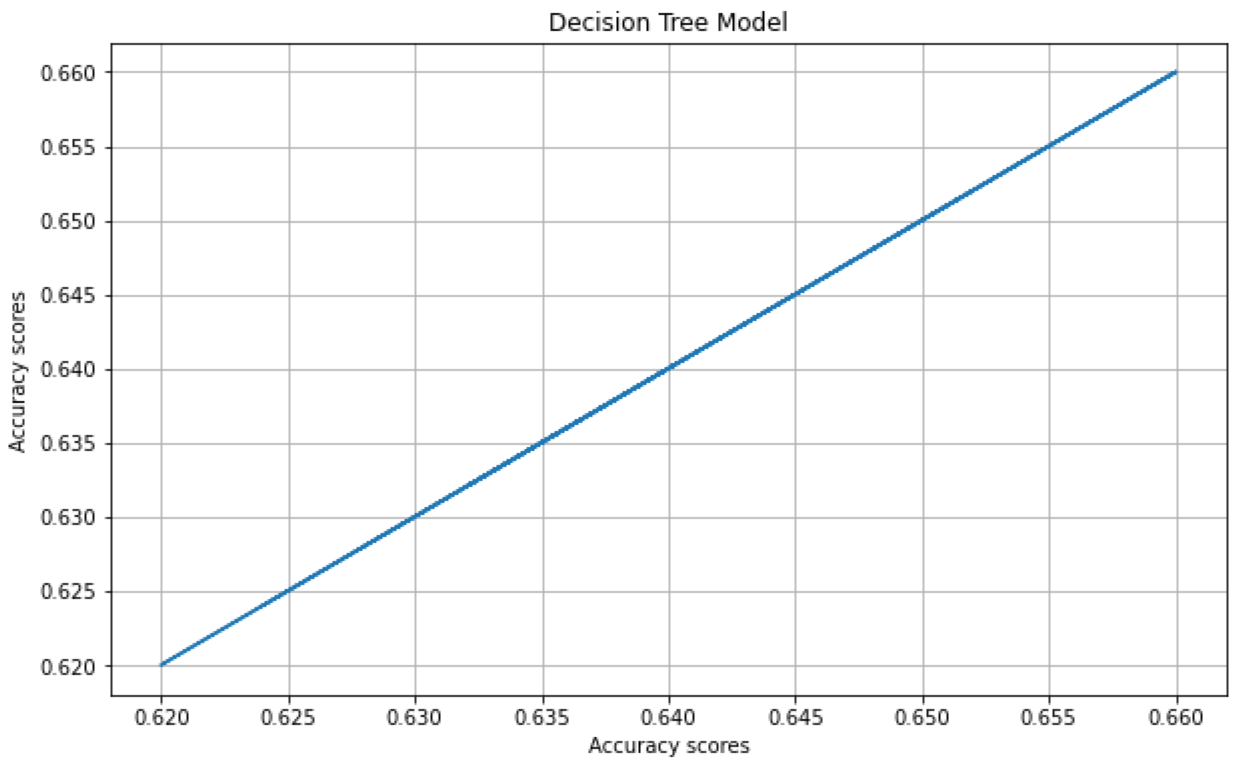
**Results:**

```
accuracy score per decision tree-------> 0.6

accuracy score per decision tree-------> 0.596

accuracy score per decision tree-------> 0.596

accuracy score per decision tree-------> 0.592

accuracy score per decision tree-------> 0.592

accuracy score per decision tree-------> 0.596

acy score per decision tree-------> 0.66

accuracy score per decision tree-------> 0.66
```

**Decision Tree accuracy = 0.71**


total impurity of leaves
0.0
0.1
0.2
0.3
0.4
0.000
0.005
0.010
0.015
0.020
effective alpha


Decision Tree Model
total depth
2
4
6
8
10
12
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
effective alpha



In [5]:

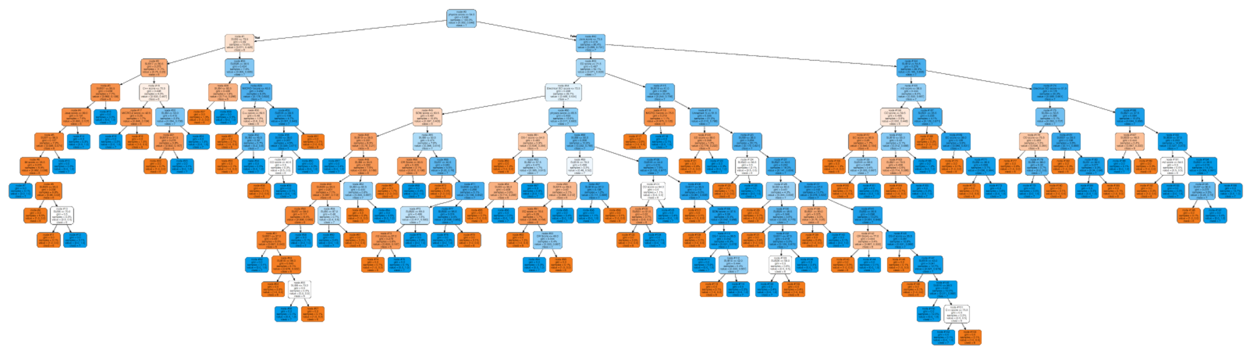
```
print(f" accuracy = {accuracy_score(y_test, global_clf.predict(x_test))}")
```

# Decision Tree accuracy = 0.716

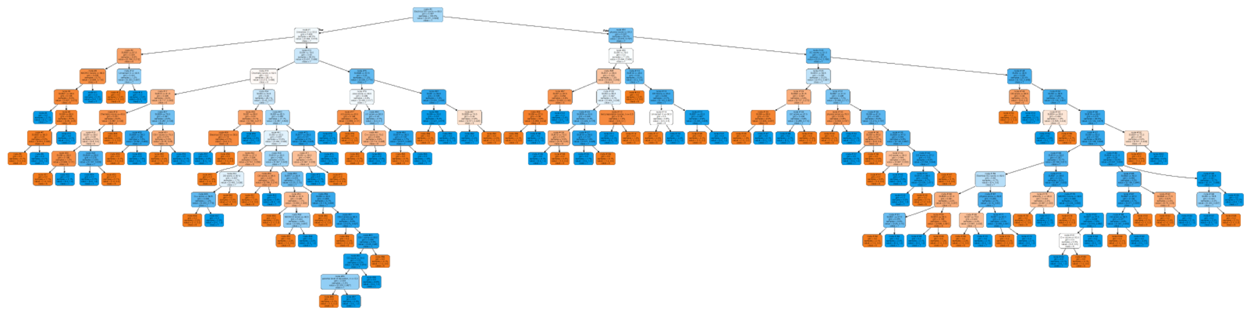
The decision Tree created to classify the Percentical /Results of student engineering data.

Ran the algorithm shuffling the data multiple times.

Observed the splitting as shown below:



More details about the split is in the ZIP folders.
Used GINI entropy for classification.



Overfitting happens when the learning algorithm continues to develop hypotheses that reduce training set error at the cost of an increased test set error. There are several approaches to avoiding overfitting in building decision trees.

- Minimal cost complexity pruning recursively finds the node with the “weakest link”. The weakest link is characterized by an **effective alpha,** where the nodes with the smallest effective alpha are pruned first.

- As alpha increases, more of the tree is pruned, which increases the total impurity of its leaves as shown above in the diagram.

- Number of nodes and tree depth decreases as alpha increases.

Plotted DT for different values of alpha.

```
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X, y_train)
    clfs.append(clf)
```

**Advantages of Decision trees:**
Easy to interpret the decision rules
Nonparametric so it is easy to incorporate a range of numeric or categorical data layers
Robust with regard to outliers in training data.

## Analysis Report Random Forest

### Accuracy of Random Forest 0.78

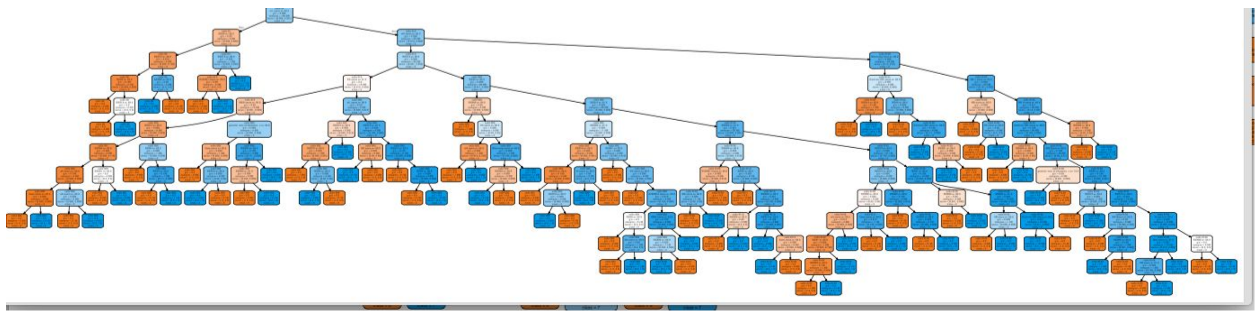
Random forests are an ensemble method used for classification. In Random Forest, we grow multiple trees as opposed to a single tree in Decision Tree model. I. One of the major

problems of Decision Tree is overfitting which gives us a very bad predictive model and adding multiple trees in the randomforest introduces randomness which in turn gets rid of

overfitting and gives us a very superior predictive model. To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The

forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees

*RandomForestRegressor(n_estimators=estimator controls* ***n_estimators :***
*This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher numbers of trees give you better performance but makes your code slower. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable.*



Used different estimator values and plotted the tress and also calculated the Accuracy

**Plotted Trees with different estimators - seen that higher estimator values , accuracy is increased and also after certain value remains constant.**

```
estimator 2

accuracy scores = 0.528

estimator 3

accuracy scores = 0.66

estimator 9

accuracy scores = 0.676

estimator 11

accuracy scores = 0.712

estimator 13

accuracy scores = 0.724

estimator 15

accuracy scores = 0.74

estimator 30

accuracy scores = 0.76

 done

accuracy scores = 0.688

-

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.588

estimator 3

accuracy scores = 0.644

estimator 6

accuracy scores = 0.688

estimator 10

accuracy scores = 0.72
```

```
estimator 22

accuracy scores = 0.724

estimator 23

accuracy scores = 0.728

estimator 39

accuracy scores = 0.732

estimator 68

accuracy scores = 0.736

 done

accuracy scores = 0.68

f FFFF ----->>  {Y}

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.564

estimator 3

accuracy scores = 0.68

estimator 8

accuracy scores = 0.7

estimator 9

accuracy scores = 0.716

estimator 10

accuracy scores = 0.728

estimator 13

accuracy scores = 0.732

estimator 21

accuracy scores = 0.748

estimator 23
```

```
accuracy scores = 0.764

estimator 34

accuracy scores = 0.772

estimator 35

accuracy scores = 0.776

estimator 38

accuracy scores = 0.788

 done

accuracy scores = 0.772

f FFFF ----->>  {Y}

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.504

estimator 3

accuracy scores = 0.624

estimator 4

accuracy scores = 0.656

estimator 5

accuracy scores = 0.66

estimator 8

accuracy scores = 0.7

estimator 10

accuracy scores = 0.716

estimator 13

accuracy scores = 0.728

estimator 15

accuracy scores = 0.744
```

```
estimator 18

accuracy scores = 0.756

estimator 33

accuracy scores = 0.764

estimator 48

accuracy scores = 0.776

 done

accuracy scores = 0.756

f FFFF ----->>  {Y}

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.6

estimator 3

accuracy scores = 0.608

estimator 6

accuracy scores = 0.632

estimator 7

accuracy scores = 0.696

estimator 9

accuracy scores = 0.704

estimator 11

accuracy scores = 0.716

estimator 14

accuracy scores = 0.74

estimator 18

accuracy scores = 0.744

estimator 36
```

```
accuracy scores = 0.748

estimator 59

accuracy scores = 0.76

 done

accuracy scores = 0.732

f FFFF ----->>  {Y}

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.508

estimator 3

accuracy scores = 0.608

estimator 4

accuracy scores = 0.616

estimator 5

accuracy scores = 0.628

estimator 7

accuracy scores = 0.664

estimator 9

accuracy scores = 0.668

estimator 10

accuracy scores = 0.676

estimator 11

accuracy scores = 0.688

estimator 13

accuracy scores = 0.728

estimator 16

accuracy scores = 0.748
```

```
 done

accuracy scores = 0.732

f FFFF ----->>  {Y}

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.548

estimator 3

accuracy scores = 0.572

estimator 4

accuracy scores = 0.644

estimator 5

accuracy scores = 0.684

estimator 10

accuracy scores = 0.712

 done

accuracy scores = 0.664

f FFFF ----->>  {Y}

Automatically created module for IPython interactive environment

estimator 2

accuracy scores = 0.596

estimator 3

accuracy scores = 0.616

estimator 5

accuracy scores = 0.7

estimator 8

accuracy scores = 0.704

estimator 15
estimator 7

accuracy scores = 0.684

estimator 12

accuracy scores = 0.7

estimator 13

accuracy scores = 0.704

estimator 15

accuracy scores = 0.712

estimator 18

accuracy scores = 0.716

estimator 21

accuracy scores = 0.74

estimator 26

accuracy scores = 0.744

estimator 37

accuracy scores = 0.752

estimator 49

accuracy scores = 0.76

estimator 59

accuracy scores = 0.78

 done

accuracy scores = 0.748

 Accuracy of Random Forest 0.78
```

## Analysis Report XGBoost

## Accuracy of XGBoost = 0.748

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction.

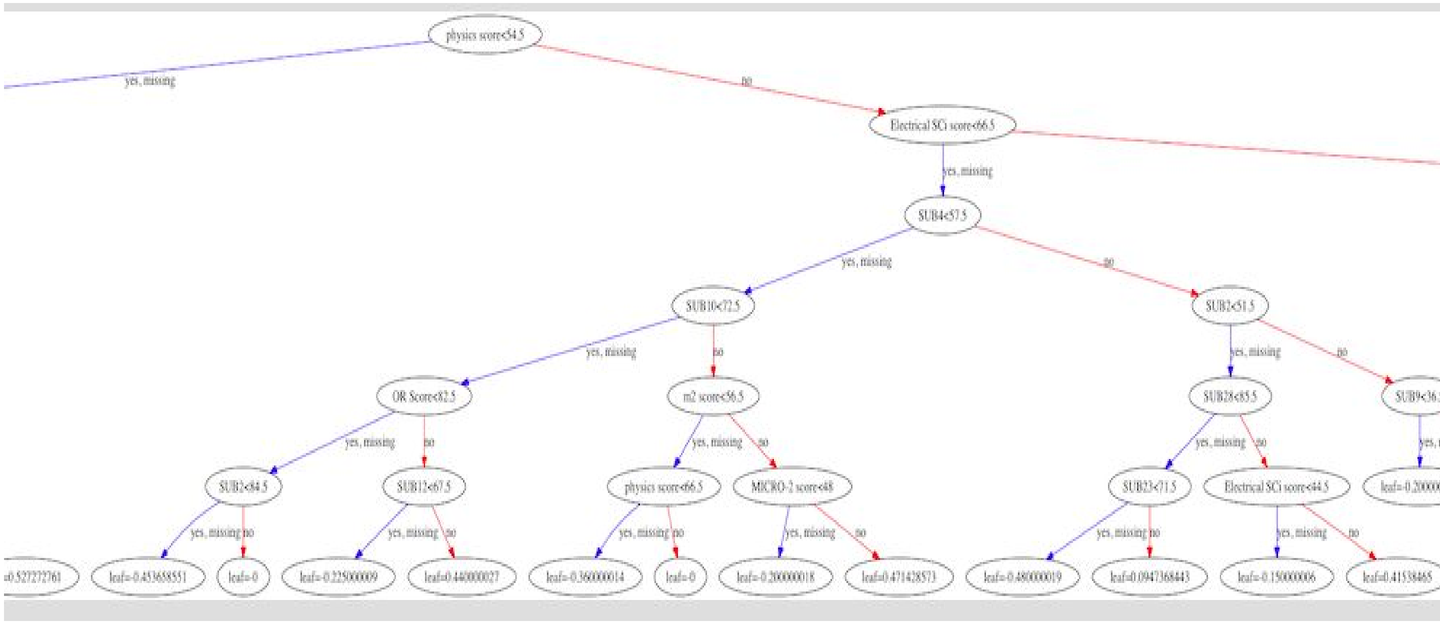
In order to calculate a prediction, XGBoost sums predictions of all its trees. The number of trees is controlled by n_estimators argument and is set to 100,1000 etc. Each tree is not a great predictor on it's own, but by summing across all trees, XGBoost is able to provide a robust estimate.

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
optimized_GBM
 GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=None,

                                     gpu_id=None, importance_type='gain',

                                     interaction_constraints=None,

                                     learning_rate=0.1, max_delta_step=None,

                                     max_depth=None, min_child_weight=None,

                                     missing=nan, monotone_constraints=None,

                                     n_estimators=1000, n_jobs=None,

                                     num_parallel_tree=None, random_state=None,

                                     reg_alpha=None, reg_lambda=None,

                                     scale_pos_weight=None, seed=0,

                                     subsample=0.8, tree_method=None,

                                     validate_parameters=None, verbosity=None),

             n_jobs=-1,

             param_grid={'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5]},

             scoring='accuracy')
```

Out[19]:

```
GridSearchCV(cv=5,

             estimator=XGBClassifier(base_score=None, booster=None,

                              colsample_bylevel=None,

                              colsample_bynode=None,

                              colsample_bytree=0.8, gamma=None,

                              gpu_id=None, importance_type='gain',

                              interaction_constraints=None,

                              learning_rate=0.1, max_delta_step=None,
                              max_depth=None, min_child_weight=None,
                              missing=nan, monotone_constraints=None,
                              n_estimators=1000, n_jobs=None,
                              num_parallel_tree=None, random_state=None,
                              reg_alpha=None, reg_lambda=None,
                              scale_pos_weight=None, seed=0,
                              subsample=0.8, tree_method=None,
                              validate_parameters=None, verbosity=None),
             n_jobs=-1,
             param_grid={'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5]},
             scoring='accuracy')
```



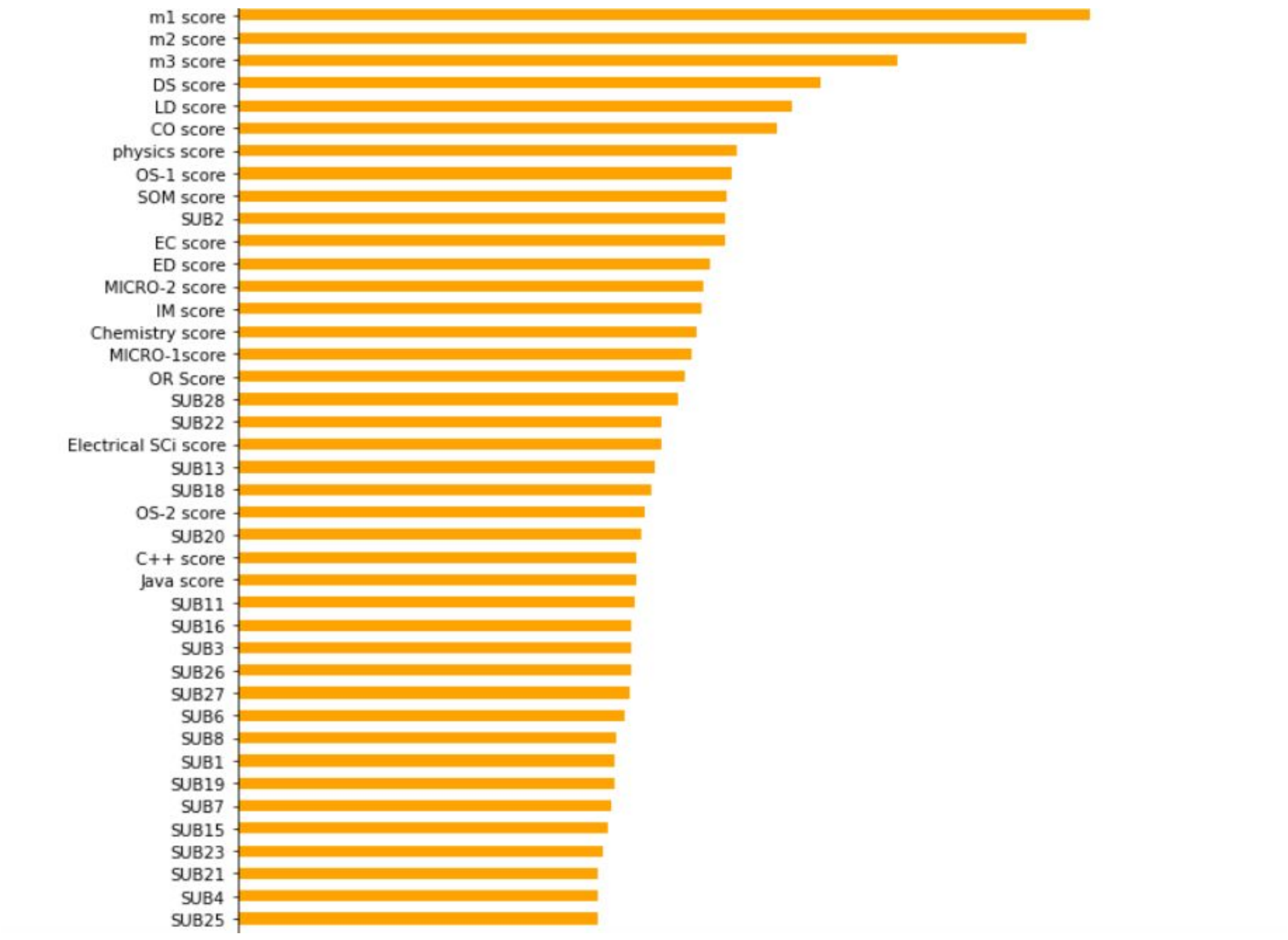
```
cv_results.tail(5)
```

Out[22]:

```
importances = final_gb.get_fscore()
importances
```

Also I have configured to know what subjects are more important for processing.

```
{'Unnamed: 5': 599,
 'physics score': 224,
 'SUB2': 219,
 'm3 score': 296,
 'SUB25': 162,
 'SUB11': 178,
 'EC score': 219,
 'OS-2 score': 183,
 'MICRO-1score': 204,
 'Electrical SCi score': 190,
 'SUB28': 198,
 'MICRO-2 score': 209,
 'OR Score': 201,
 'SUB20': 181,
 'SUB12': 134,
 'SUB8': 170,
 'Java score': 179,
 'SUB18': 186,
 'SOM score': 220,
 'SUB27': 176,
 'm1 score': 383,
 'SUB10': 153,
 'SUB29': 146,
 'SUB14': 158,
 'SUB4': 162,
 'SUB16': 177,
 'SUB24': 161,
 'IM score': 208,
'SUB5': 158,
'SUB7': 168,
'DS score': 262,
'CO score': 242,
'C++ score': 179,
'SUB31': 144,
'SUB13': 187,
'SUB23': 164,
'SUB3': 177,
'SUB9': 154,
'ED score': 212,
'ethnicity_n': 69,
'SUB26': 177,
'SUB19': 169,
'SUB1': 169,
'SUB15': 166,
'Chemistry score': 206,
'LD score': 249,
'SUB30': 155,
'm2 score': 354,
'SUB17': 133,
'OS-1 score': 222,
'SUB21': 162,
'SUB22': 190,
'SUB6': 174,
'parental level of education_n': 61,
'lunch_n': 17,
'test preparation course_n': 35,
'gender_n': 13}
```



This will tell us which features were most important in the series of trees and also tuned other parameters to get better accuracy.

**booste**

- gbtree: tree-based models

It is observed for the same dataset XGBoost accuracy is better than random forest and random forest is better than Decision Tree