Predicting Graduate Admissions

Amal Imdad Alhaq

February 9th, 2019

Definition

Project Overview

Graduate admissions are not quite accurate science and the admissions process took lots of time. Faculty committees, sitting around conference tables for hours on end, have abundant of data to decide about applicants, including their transcripts, personal statements, GPA, letters of recommendation and GRE scores. Problem is, it's not always clear just what the data mean. The admission systems vary widely from country to country, and sometimes from institution to institution. That makes it easy for biases to slip in undetected [1].

Concerns about diversity in graduate programs are well-founded. But standardized tests like the GRE are not what's holding the academy back from attaining greater diversity. The problem emerges instead in those long meetings in conference rooms. The faculty has plenty of applicants and limited time to make important decisions, all while navigating departmental politics and seeking to raise their program's prestige [2]. By machine learning, the whole decisions can be done faster with more accuracy to help the admission department to crack this problem.

Problem Statement

All the admission process is depending on faculty to deal with massive applications that they can accept in a limited time, that take more human endeavors. whereas the admission relies on many features, for instance, GRE score, TOEFL scores, university rating, Letter of Recommendation, Statement of Purpose and Research Experience, that all use numeric values, anticipating the graduate admission can decrease the time and helping the admission committee by giving them the possible candidates. On the other hand, helping students in shortlisting universities with their profiles. The predicted output gives them a fair idea about their chances for a particular university.

Solution Statement

Supervised learning can be used for this problem. It uses regression or classification of machine learning algorithms to predict whether a particular candidate has a chance to admit to university or not. We can build a logistic regression model predicting the admission [3]. The most common solution to such problem is Regression methods.

1. Linear Regression

2. Polynomial Regression.

3. Logistic Regression

4. Regularization

Regression method is a form of predictive modeling technique which investigates the relationship between a dependent (target) and independent variables. This technique is used for forecasting between the variables. For example, the relationship between GRE Scores and Chance of Admit.

But there are other methods that is suitable for this problem too. Some of them listed next:

- AdaBoost classifier

- Random Forest

- Stochastic Gradient Descent Classifier (SGDC)

To select the best model, need to evaluate them. Random forest algorithm can use both for classification and the regression kind of problems. As the name suggest, this algorithm creates the forest with a number of trees. In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results. [4]. AdaBoost is a popular boosting technique which helps combine multiple “weak classifiers” into a single “strong classifier”. Graduate Admission would seem that using accuracy and F-score as a metric for evaluating a particular model's performance would be appropriate.

Evaluation Metrics

The chosen models will be evaluated my Evaluation metrics which are F-score and the accuracy. The f-score formula is

$$F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

It uses true positive TP, true negative TN, false positive FP, and false negative FN.

1. the recall is measure number the true positive TP over the sum of the true positive and false negative.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

2. precision is defined as . you can see that Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

3. Where the accuracy measures the ratio of the number of correct predictions to the total number of predictions (the number of test data points)

$$accuracy = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalsePositive + FlaseNegative}$$

In our problem I use the f-score function that provided by SciKit-learn.

Analysis

Data Exploration

The dataset is obtained from Kaggle Repository. It is inspired by the UCLA Graduate Dataset. The dataset is owned by Mohan S Acharya [3], and it made publicly available online through the website: https://www.kaggle.com/shraban020/predicting-admission-by-logistic-regression/data . The dataset has 500 records updated month ago. It has 9 attributes are described as follows:

1. GRE Scores (out of 340)

2. TOEFL Scores (out of 120)

3. University Rating (out of 5)

4. Statement of Purpose "SOP" (out of 5)

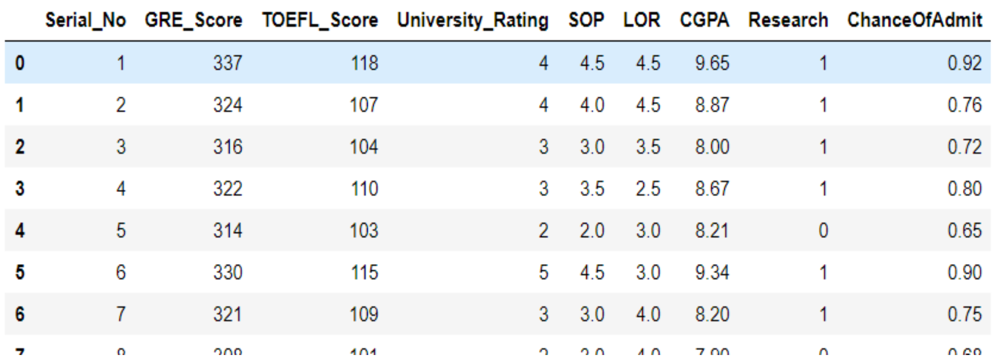
5. Letter of Recommendation Strength "LOR" (out of 5)

6. Undergraduate GPA (out of 10)

7. Research Experience (either 0 or 1)

8. Chance of Admit (ranging from 0 to 1)

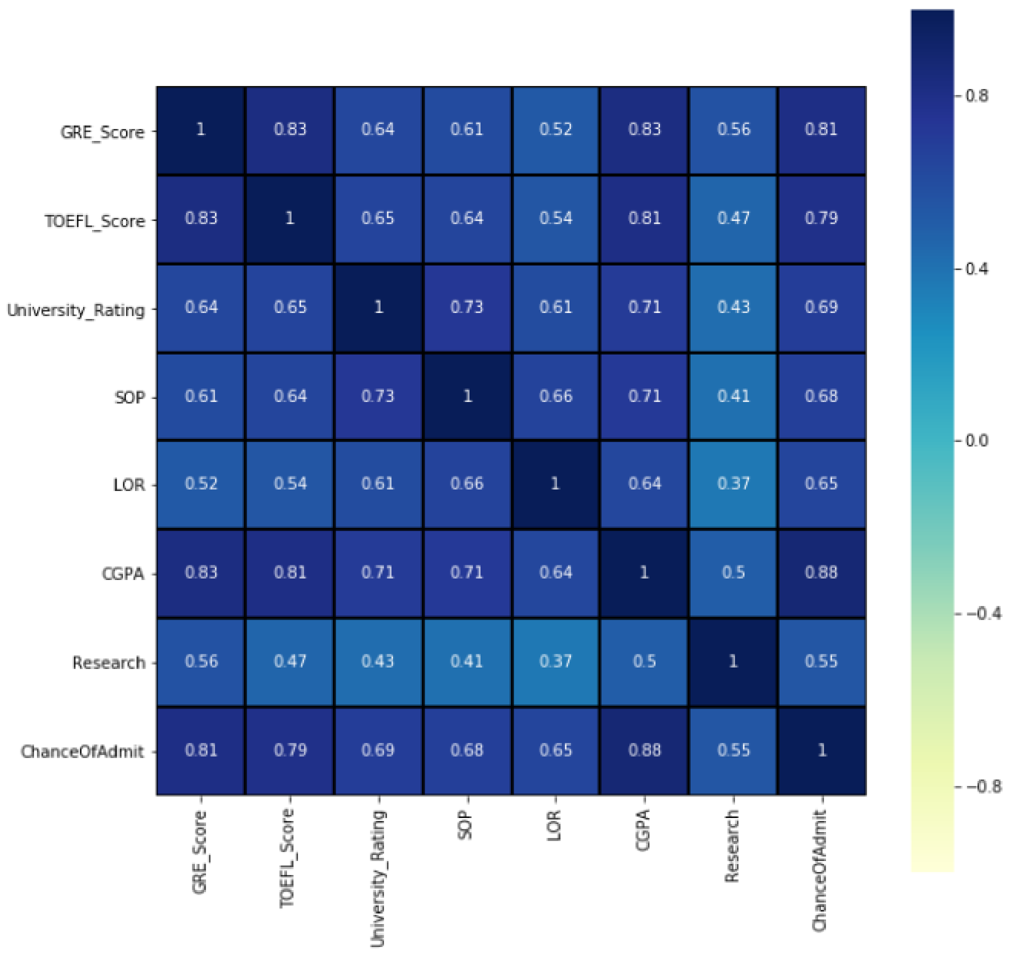
The characteristics of the dataset or input are a numerical data contains discrete and continues value. And these features are appropriate given the context of the problem [3].



	Serial_No	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	ChanceOfAdmit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65
5	6	330	115	5	4.5	3.0	9.34	1	0.90
6	7	321	109	3	3.0	4.0	8.20	1	0.75
7	8	308	101	2	3.0	4.0	7.90	0	0.68

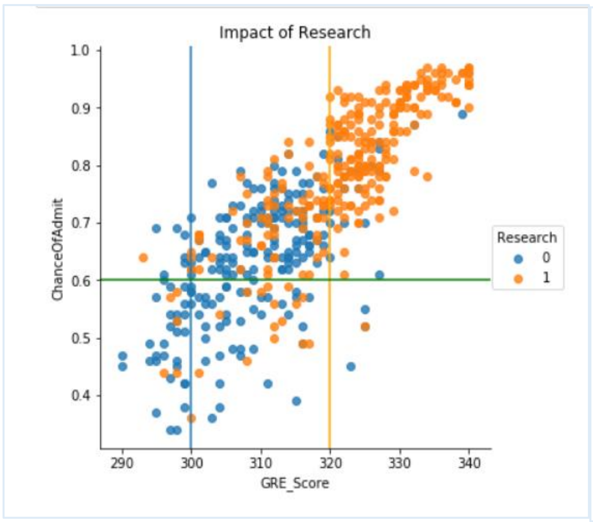
Exploratory Visualization

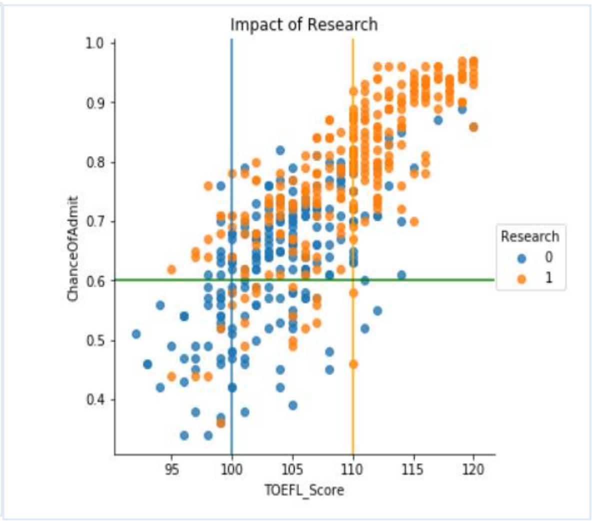
Note that the last column from this dataset "ChanceOfAdmit" is our target label, all other columns are features except "Serial_No", it has no effect on the chance of admission to the University. We better drop the column of serial number from the dataset. Then finding out correlations between the features and our target value chance of admit to university. The next figure will show the correlation:

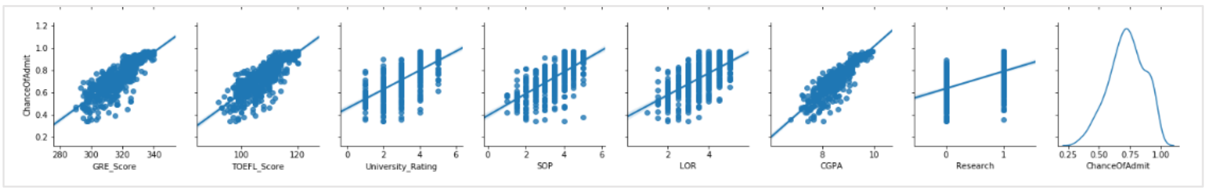


	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	ChanceOfAdmit
GRE_Score	1	0.83	0.64	0.61	0.52	0.83	0.56	0.81
TOEFL_Score	0.83	1	0.65	0.64	0.54	0.81	0.47	0.79
University_Rating	0.64	0.65	1	0.73	0.61	0.71	0.43	0.69
SOP	0.61	0.64	0.73	1	0.66	0.71	0.41	0.68
LOR	0.52	0.54	0.61	0.66	1	0.64	0.37	0.65
CGPA	0.83	0.81	0.71	0.71	0.64	1	0.5	0.88
Research	0.56	0.47	0.43	0.41	0.37	0.5	1	0.55
ChanceOfAdmit	0.81	0.79	0.69	0.68	0.65	0.88	0.55	1

As shown above, we noticed that the research is not highly correlated with the chances of admission, having done research leads to better GRE and TOEFL score which have high correlation with the chances of admission. I draw a graph that show the impact of research with getting high GRE score and TOFEL score.







From the correlation shown above, I think GRE, TOFEL and CGPA are tho most important features of the dataset.

Algorithm and Techniques

The given dataset is a typical supervised learning problem, it has a numerical data. for which tree type models perform a lot better than the rest. We don't know which algorithms would be fit for this problem. That's why we select three algorithms to evaluate. They were selected based on the features, values' datatype and the nature of the problem

1. **Logistic Regression**

   It is a technique borrowed by machine learning from the field of statistics. Logistic regression algorithm also uses a linear equation with independent predictors to predict a value [5]. It's good in describing data and explaining the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables, as we want to know the relationship between the features such as the TOFEL score with the chance of admission to the university.

   Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

   $y = e^{(b0 + b1*X)} / (1 + e^{(b0 + b1*X)})$

2. **Random forest**

   Random forest algorithm can be used for both classification and regression problems. As the name suggest, this algorithm creates the forest with number of trees. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

   To perform prediction using the trained random forest algorithm uses the below pseudocode.

   1. Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)

2. Calculate the votes for each predicted target.
3. Consider the high voted predicted target as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created trees.

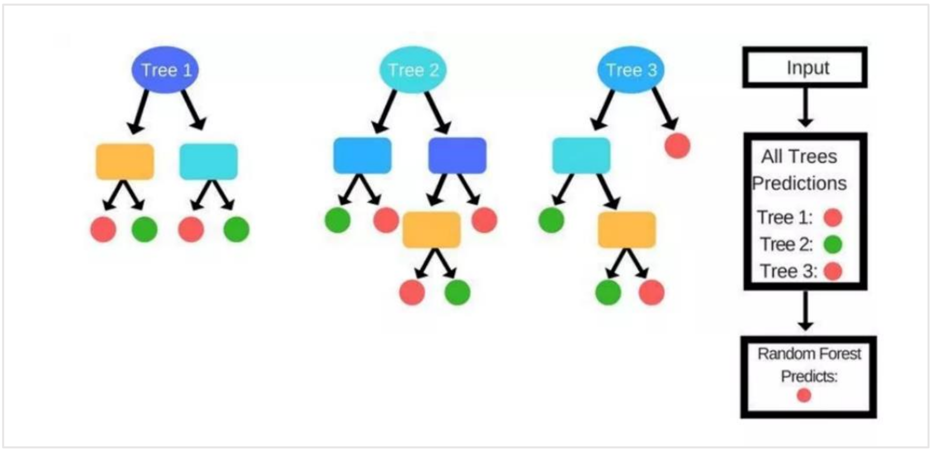


Figure1: random forest algorithm overview

Suppose let’s say we formed 100 random decision trees to from the random forest. Each random forest will predict different target (outcome) for the same test feature. Then by considering each predicted target votes will be calculated. Suppose the 100 random decision trees are prediction some 3 unique targets x, y, z then the votes of x is nothing but out of 100 random decision tree how many trees prediction is x. Likewise for other 2 targets (y, z). If x is getting high votes. Let’s say out of 100 random decision tree 60 trees are predicting the target will be x. Then the final random forest returns the x as the predicted target [4].

3. Ensemble Model (AdaBoost)

Boosting is an ensemble technique that attempts to create a strong classifier from number of weak classifiers. This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added [6].

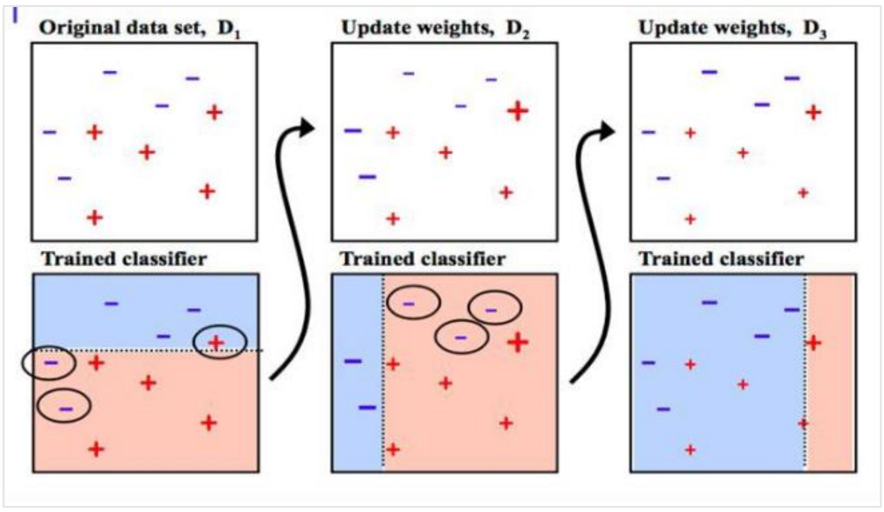


Figure2: Algorithm AdaBoost Example

Predictions are made by calculating the weighted average of the weak classifiers.

For a new input instance, each weak learner calculates a predicted value as either +1.0 or -1.0. The predicted values are weighted by each weak learner stage value. The prediction for the ensemble model is taken as the sum of the weighted predictions. If the sum is positive, then the first class is predicted, if negative the second class is predicted [7].

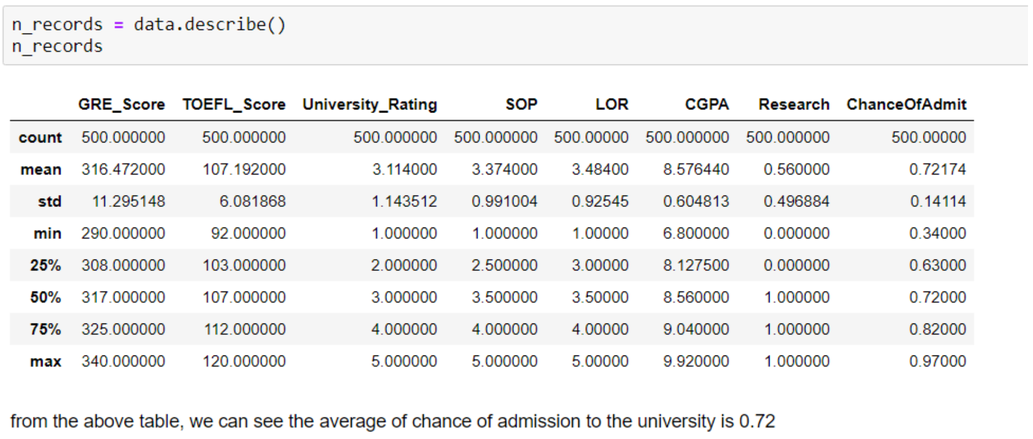
Benchmark Model

While I was searching for this problem, I found someone uses logistic regression for prediction. So, I use it in addition of other models to compare between them and decide which model should be used for our problem. I use AdaBoost method with regression as a weaker learner to predict the graduate admission, as well as Random forest classifier. I use the three methods mothed that are currently available in sci-kit-learn.

Methodology

Data Preprocessing

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured.

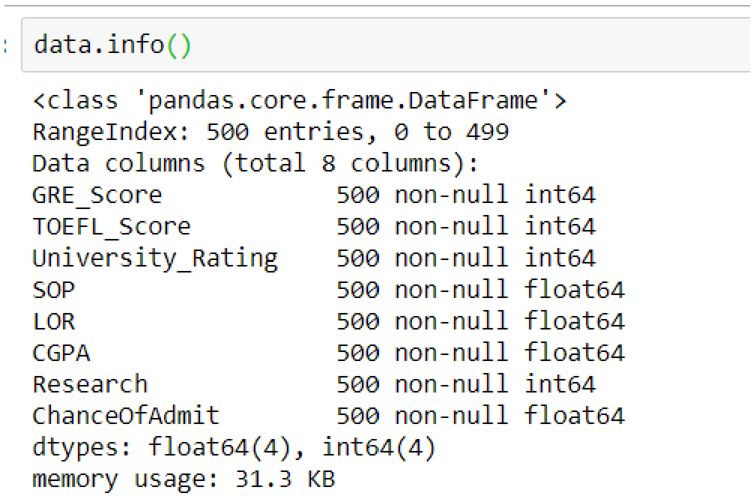


```
n_records = data.describe()
n_records
```

	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	ChanceOfAdmit
count	500.000000	500.000000	500.000000	500.000000	500.00000	500.000000	500.000000	500.00000
mean	316.472000	107.192000	3.114000	3.374000	3.48400	8.576440	0.560000	0.72174
std	11.295148	6.081868	1.143512	0.991004	0.92545	0.604813	0.496884	0.14114
min	290.000000	92.000000	1.000000	1.000000	1.00000	6.800000	0.000000	0.34000
25%	308.000000	103.000000	2.000000	2.500000	3.00000	8.127500	0.000000	0.63000
50%	317.000000	107.000000	3.000000	3.500000	3.50000	8.560000	1.000000	0.72000
75%	325.000000	112.000000	4.000000	4.000000	4.00000	9.040000	1.000000	0.82000
max	340.000000	120.000000	5.000000	5.000000	5.00000	9.920000	1.000000	0.97000

from the above table, we can see the average of chance of admission to the university is 0.72

Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain features that must be adjusted.



```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
GRE_Score            500 non-null int64
TOEFL_Score          500 non-null int64
University_Rating    500 non-null int64
SOP                  500 non-null float64
LOR                  500 non-null float64
CGPA                 500 non-null float64
Research             500 non-null int64
ChanceOfAdmit        500 non-null float64
dtypes: float64(4), int64(4)
memory usage: 31.3 KB
```

We're splitting the data into features and target label, the target label in our dataset is the chance of admission, and the other labels is our features. Since the dataset features are numerical values, it's good practice to perform some type of scaling (normalizing). Applying a scaling to the data does not change the shape of each feature's distribution for instance " CGPA " or " LOR ", however, normalization ensures that each feature is treated equally when applying supervised learners. Then applying one-hot encoding on the features. The target label was needed to replace the value with 1 when the chance admission is more than or equal to 72 % which is the mean of chance of admission values. Replace the value with zero when the chance of admission score is less than 72%. After the data become clean and normalize, it's time to be splite the data into training and test sets. 80% of the data will be used for training and 20% for testing.

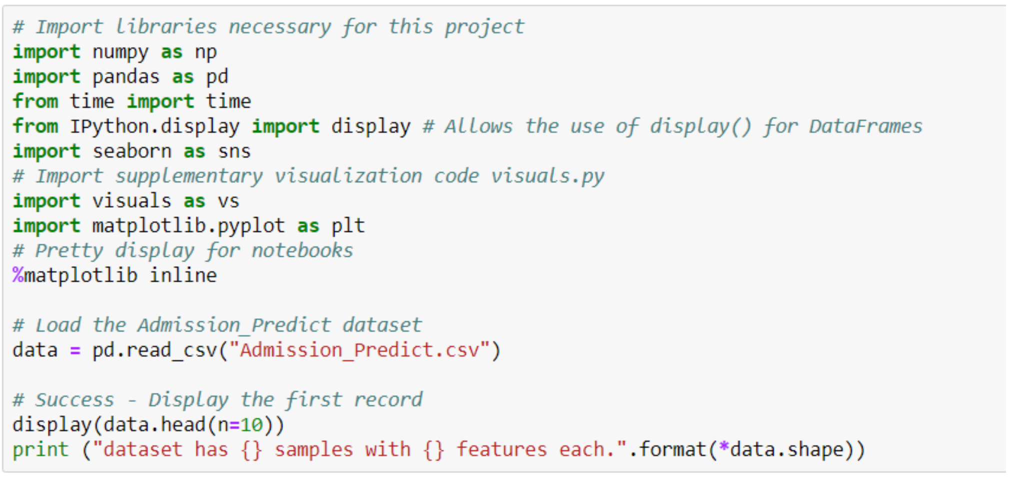
Implementation

The key steps to implementing this model are detailed in the accompanying python code and JuPyter notebooks, below is a brief summary of the nuances in the approach:

1. **Data cleaning and normalizing**

We load the data using pandas library. loading the data was straightforward, it was stored in standard csv format. However, it was important to understand the column names, types and distributions.

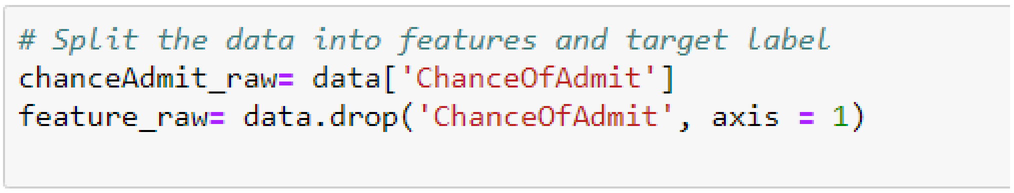
loading necessary Python libraries and loading the Admission_Predict data.



```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
from time import time
from IPython.display import display # Allows the use of display() for DataFrames
import seaborn as sns
# Import supplementary visualization code visuals.py
import visuals as vs
import matplotlib.pyplot as plt
# Pretty display for notebooks
%matplotlib inline

# Load the Admission_Predict dataset
data = pd.read_csv("Admission_Predict.csv")

# Success - Display the first record
display(data.head(n=10))
print ("dataset has {} samples with {} features each.".format(*data.shape))
```

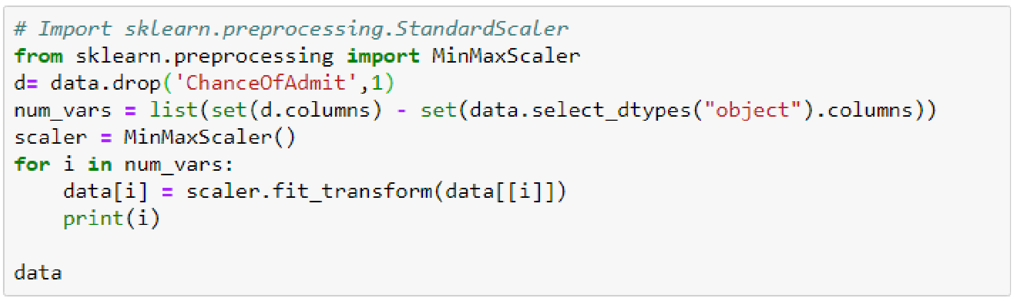
Checking missing values, it was then no missing value. For the serial Number, it has not any impact to the prediction, so dropping that column was done.

Since we would like to implement a binary classification algorithm, I decided to drop the rows with target value "ChanceOfAdmit". Now, we have only two target classes to predict. We extract the independent and dependent variables from the dataset.

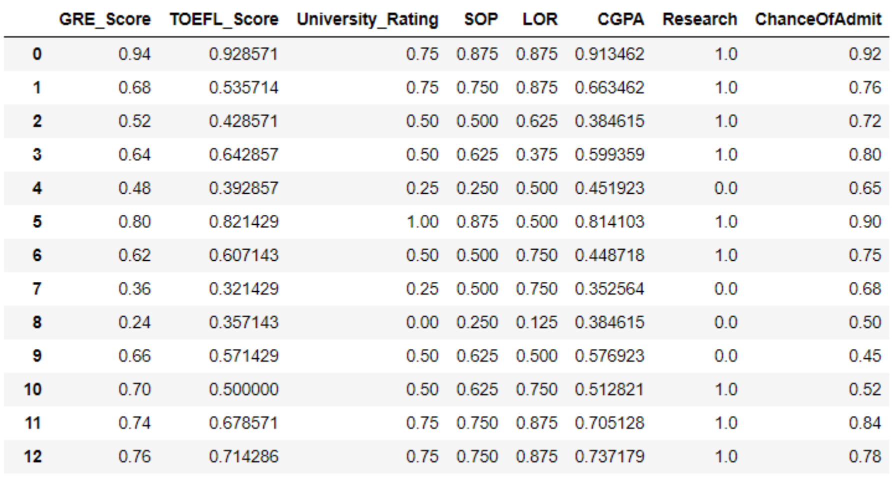


```
# Split the data into features and target label
chanceAdmit_raw= data['ChanceOfAdmit']
feature_raw= data.drop('ChanceOfAdmit', axis = 1)
```

It is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution, however, normalization ensures that each feature is treated equally when applying supervised learners. After normalizing, the data become in range [0,1].



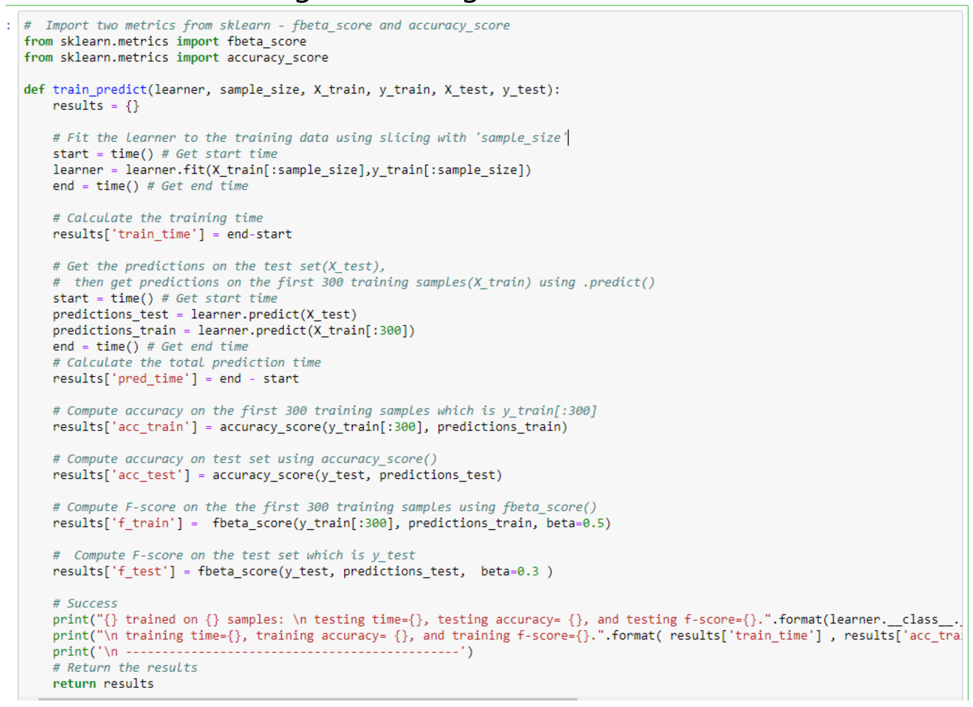
```
# Import sklearn.preprocessing.StandardScaler
from sklearn.preprocessing import MinMaxScaler
d= data.drop('ChanceOfAdmit',1)
num_vars = list(set(d.columns) - set(data.select_dtypes("object").columns))
scaler = MinMaxScaler()
for i in num_vars:
    data[i] = scaler.fit_transform(data[[i]])
    print(i)

data
```



	GRE_Score	TOEFL_Score	University_Rating	SOP	LOR	CGPA	Research	ChanceOfAdmit
0	0.94	0.928571	0.75	0.875	0.875	0.913462	1.0	0.92
1	0.68	0.535714	0.75	0.750	0.875	0.663462	1.0	0.76
2	0.52	0.428571	0.50	0.500	0.625	0.384615	1.0	0.72
3	0.64	0.642857	0.50	0.625	0.375	0.599359	1.0	0.80
4	0.48	0.392857	0.25	0.250	0.500	0.451923	0.0	0.65
5	0.80	0.821429	1.00	0.875	0.500	0.814103	1.0	0.90
6	0.62	0.607143	0.50	0.500	0.750	0.448718	1.0	0.75
7	0.36	0.321429	0.25	0.500	0.750	0.352564	0.0	0.68
8	0.24	0.357143	0.00	0.250	0.125	0.384615	0.0	0.50
9	0.66	0.571429	0.50	0.625	0.500	0.576923	0.0	0.45
10	0.70	0.500000	0.50	0.625	0.750	0.512821	1.0	0.52
11	0.74	0.678571	0.75	0.750	0.875	0.705128	1.0	0.84
12	0.76	0.714286	0.75	0.750	0.875	0.737179	1.0	0.78

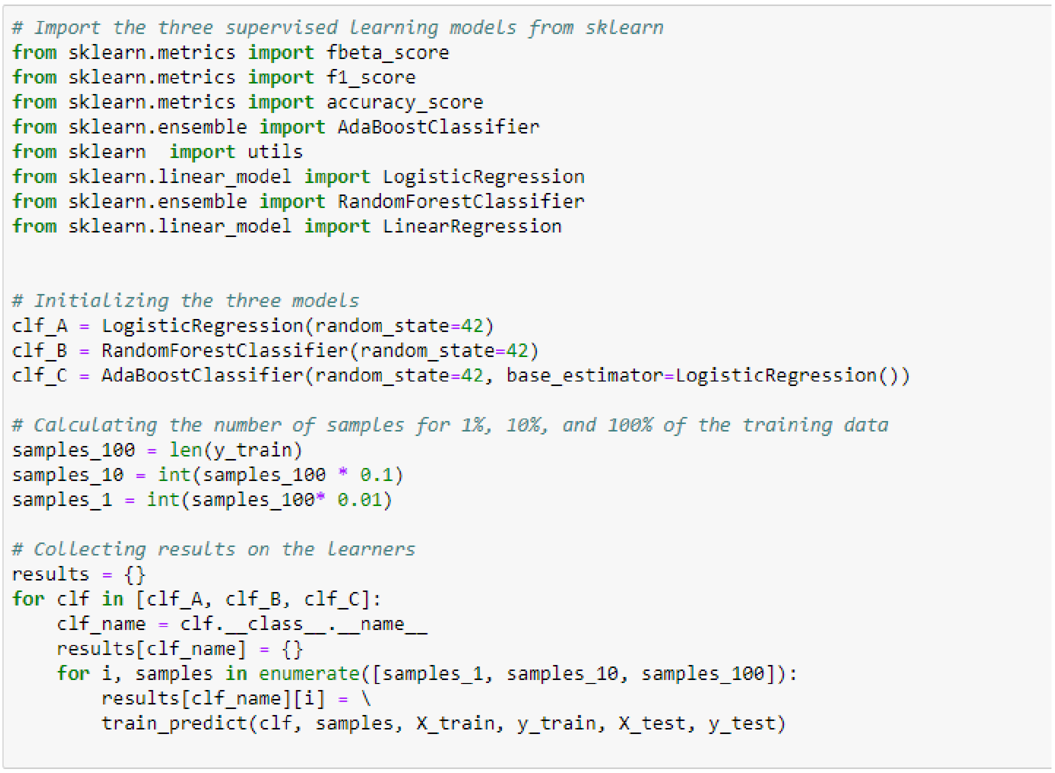
2. Model Evaluation

To evaluate the model we first create a training and predicting pipeline that allows to quickly and effectively train models using different sizes of training data and perform predictions on the testing data. I also calculate the training and prediction time, and calculate the training and testing accuracy. As well as the f-score of training and testing.



```
: #  Import two metrics from sklearn - fbeta_score and accuracy_score
  from sklearn.metrics import fbeta_score
  from sklearn.metrics import accuracy_score

  def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
      results = {}

      # Fit the learner to the training data using slicing with 'sample_size'
      start = time() # Get start time
      learner = learner.fit(X_train[:sample_size],y_train[:sample_size])
      end = time() # Get end time

      # Calculate the training time
      results['train_time'] = end-start

      # Get the predictions on the test set(X_test),
      #  then get predictions on the first 300 training samples(X_train) using .predict()
      start = time() # Get start time
      predictions_test = learner.predict(X_test)
      predictions_train = learner.predict(X_train[:300])
      end = time() # Get end time
      # Calculate the total prediction time
      results['pred_time'] = end - start

      # Compute accuracy on the first 300 training samples which is y_train[:300]
      results['acc_train'] = accuracy_score(y_train[:300], predictions_train)

      # Compute accuracy on test set using accuracy_score()
      results['acc_test'] = accuracy_score(y_test, predictions_test)

      # Compute F-score on the the first 300 training samples using fbeta_score()
      results['f_train'] =  fbeta_score(y_train[:300], predictions_train, beta=0.5)

      #  Compute F-score on the test set which is y_test
      results['f_test'] = fbeta_score(y_test, predictions_test,  beta=0.3 )

      # Success
      print("{} trained on {} samples: \n testing time={}, testing accuracy= {}, and testing f-score={}.".format(learner.__class__.
      print("\n training time={}, training accuracy= {}, and training f-score={}.".format( results['train_time'] , results['acc_tra
      print('\n ----------------------------------------------')
      # Return the results
      return results
```

Then I import the three supervised learning models that I've discussed in the previous section (Algorithms and techniques), I train the models over different sample size, the whole training data for 100%, 10 % and 1% of the training data.



```python
# Import the three supervised learning models from sklearn
from sklearn.metrics import fbeta_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn  import utils
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LinearRegression


# Initializing the three models
clf_A = LogisticRegression(random_state=42)
clf_B = RandomForestClassifier(random_state=42)
clf_C = AdaBoostClassifier(random_state=42, base_estimator=LogisticRegression())

# Calculating the number of samples for 1%, 10%, and 100% of the training data
samples_100 = len(y_train)
samples_10 = int(samples_100 * 0.1)
samples_1 = int(samples_100* 0.01)

# Collecting results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        results[clf_name][i] = \
        train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

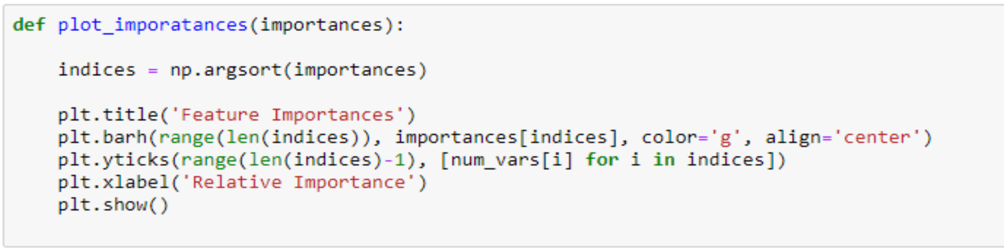
4. **Best Model selection**

   See Best Model selection in section "Result and conclusion" in "Model Evaluation" that already described.

5. **Features importance**

   In Supervised learning problem, it's better to determine which of the features contribute most to the prediction. By focusing on the relationship between only a few critical features and the target label. In the case of this project, that means we wish to identify a small number of features that most strongly predict whether a student have a chance of admission to university less or more than 72%.

   feature_importance_ attribute that available in random forest classifier provided by scikit learn. This function ranks the importance of features according to the chosen classifier.



```python
def plot_imporatances(importances):

    indices = np.argsort(importances)

    plt.title('Feature Importances')
    plt.barh(range(len(indices)), importances[indices], color='g', align='center')
    plt.yticks(range(len(indices)-1), [num_vars[i] for i in indices])
    plt.xlabel('Relative Importance')
    plt.show()
```

```
#  Train the supervised model on the training set using .fit(X_train, y_train)
lr_model = RandomForestClassifier(random_state=42).fit(X_train, y_train)

importances = lr_model.feature_importances_
```

Refinement

We prepared the data by splitting feature and target/label columns and also check for the quality of given data and perform data cleaning. To check if the model we created is any good, we split the data into training and validation sets to check the accuracy of the models thus know the best model. We split the given training data in two, 80% of which was used to train our models and 20% we held back as a validation set.

After choosing the best model for our dataset which is Random Forest, we perform a grid search optimization for the model over the entire training set (X_train and y_train) by tuning one parameter to improve upon the untuned model's F-score.

```
# Create the parameters list to tune, using a dictionary.
parameters = {"n_estimators":[16, 32, 64, 100, 200,500,1000,2000] }
```

Results and Conclusion

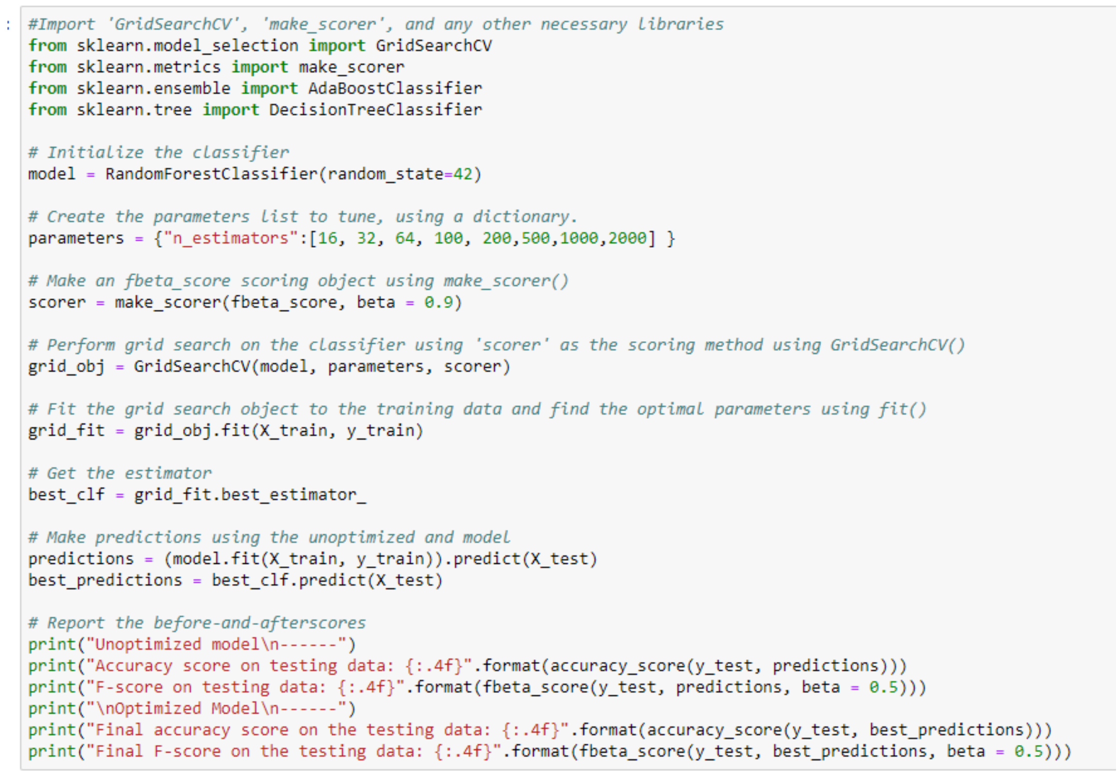
Model Evaluation

For each model, evaluate both the training subset and testing set using accuracy metrics accuracy_score and f-score fbeta_score from sklearn.metrics to decide the best model that should be used. And the time of training and testing was calculated. After investigating three different supervised learning algorithms, it can be determined which is best at modeling the data.

Based on the evaluation performed, I found that the Random Forest Classifier was the best one because it has largest f-score and accuracy when sample data was 100% of the training set, as well as it took smallest testing time. the testing accuracy was 0.86 and testing f-score was 0.8634978671541 and the testing time was 0.008084774017333984 which is better than the two others. So Random Forest classifier was the best model for our dataset.

- **Model Tuning**

After choosing the best model for our problem and tuning at least one parameter to improve upon the untuned model's F-score.

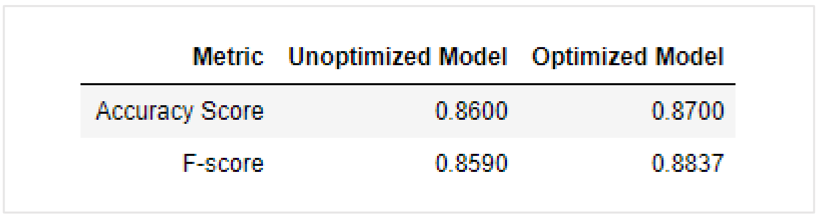


```
#Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Initialize the classifier
model = RandomForestClassifier(random_state=42)

# Create the parameters list to tune, using a dictionary.
parameters = {"n_estimators":[16, 32, 64, 100, 200,500,1000,2000] }

# Make an fbeta_score scoring object using make_scorer()
scorer = make_scorer(fbeta_score, beta = 0.9)

# Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()
grid_obj = GridSearchCV(model, parameters, scorer)

# Fit the grid search object to the training data and find the optimal parameters using fit()
grid_fit = grid_obj.fit(X_train, y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
predictions = (model.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)

# Report the before-and-afterscores
print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5)))
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5)))
```

as a result, we got the accuracy and f-score of unoptimal and optimal model is so close to each other. the differentiation between unoptimal and optimal modal is 0.01 for the accuracy score, and 0.0247 for f-score, so the two models are good, but the optimal model is better.

Justification

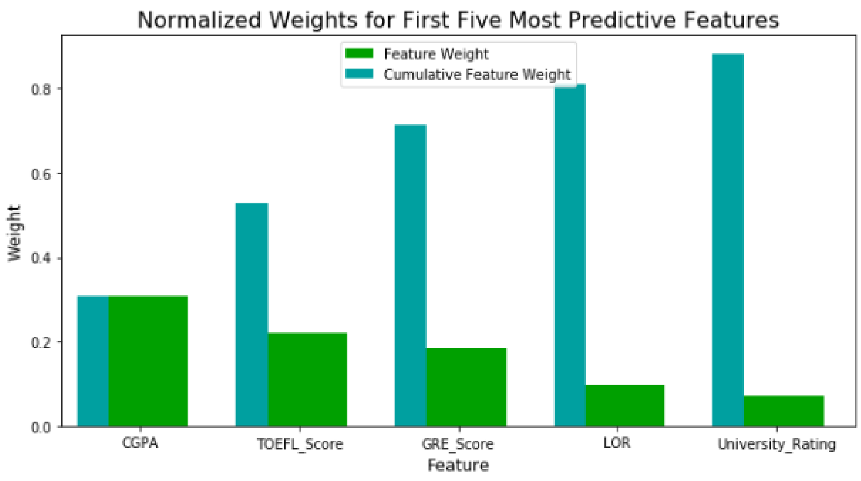
There is leeway for improvement on the final results, the tuned final model made small improvement over the untuned model. There could be other ways to improve the score, especially by selecting a subset of features using the ranking obtained from observing feature importance ranking and then performing the same exercise we did as describe above in " Model Tuning " section. But for now, this is out of the space of this report. This is what we got from the tuning step:



Metric	Unoptimized Model	Optimized Model
Accuracy Score	0.8600	0.8700
F-score	0.8590	0.8837

Free-Form Visualization

After the feature importance selection is applied, we can see that the CGPA, TOFEL_score, GRE_score, LOR and University_Rating are the most important features in admission dataset.



Reflection

The most important, challenging, and time-consuming part of the problem and was to pick an algorithm that could be best suited for the problem that we choose to solve. With prior knowledge and also long searching on supervised learning algorithms, I decided to use the three models explained previously. I thought the logistic regression is the best for our problem, but I observed from the outcome of accuracy on training data that performed the best out of others. The next part that I faced a problem when calculating the f-score. f1_score and precision_recall_fscore throw an error for some class labels, the problem was because of continues type of y-train and multiclass type of x-train, so it takes my time to try to solve this issue.

Improvement

So far, tuning one parameter (n_estemator) for the model is performed to improve our model. Also, we use a subset of all the available features in the data with fewer features required to train by reducing the feature space, as expected the training and prediction time is much lower at the cost of performance metrics but the accuracy and f-score become low. So, one of the improvements we could to do to get better performance is to perform tuning other parameters not tuned in this project, for example, the max_depth or n_jobs. In addition, we will use AUC (Area Under Curve) as the evaluation metric. Our target value is binary since we replace each value over than 0.72 with 1 and less than 0.72 with 0, so AUC is a good way for evaluation for this type of problems. Whereas I have limited time for this project, I found that what I did for improvement was ok, but of course, I will improve the performance of this model once I have more time.

References

[1] "The Problem in Graduate Admissions Is Culture, Not Testing - ETS Open Notes," *Open Notes*. [Online]. Available: https://news.ets.org/stories/problem-graduate-admissions-culture-not-testing/.

[2] L. Cassuto, "Inside the Graduate-Admissions Process," *The Chronicle of Higher Education*, 01-Feb-2016. [Online]. Available: https://www.chronicle.com/article/Inside-the-Graduate-Admissions/235093.

[3] *RSNA Pneumonia Detection Challenge | Kaggle*. [Online]. Available: https://www.kaggle.com/shraban020/predicting-admission-by-logistic-regression/data.

[4] "How the random forest algorithm works in machine learning," *Dataaspirant*, 01-Oct-2017. [Online]. Available: http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing.

[5] R. Gandhi and R. Gandhi, "Introduction to Machine Learning Algorithms: Logistic Regression," *Hacker Noon*, 28-May-2018. [Online]. Available: https://hackernoon.com/introduction-to-machine-learning-algorithms-logistic-regression-cbdd82d81a36.

[6] 1052926731387809, "A Tour of The Top 10 Algorithms for Machine Learning Newbies," *Towards Data Science*, 20-Jan-2018. [Online]. Available: https://towardsdatascience.com/a-tour-of-the-top-10-algorithms-for-machine-learning-newbies-dde4edffae11.

[7] "Boosting and AdaBoost for Machine Learning," *Machine Learning Mastery*, 22-Sep-2016. [Online]. Available: https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning.