MLND Capstone Project

Capstone Project

Donors Choose Application Screening

Niall O'Hara
May 4th, 2018

I. Definition

Project Overview

Founded in 2000 by a high school teacher in the Bronx, DonorsChoose.org empowers public school teachers from across the country to request much-needed materials and experiences for their students. At any given time, there are thousands of classroom requests that can be brought to life with a gift of any amount.

Each year DonorsChoose.org receive hundreds of thousands of applications that they have to decide to approve or not approve manually. This project will focus on building a ML model that will help automate this approval process to allow their staff to focus on higher value work helping teachers realise their projects. This is a binary classification problem, I will outline the proposed solution to this problem in the coming sections.

Datasets and Inputs [1]

The project dataset contains information from teachers' project applications to DonorsChoose.org including teacher attributes, school attributes, and the project proposals including application essays. My objective as outlined above is to use a combination of this data to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved. For reference the data sets used in this project can be downloaded here.

Project File Descriptions

- **train.csv** - the training set **[182,080 Rows]**
- **test.csv** - the test set **[78,036 Rows]**
- **resources.csv** - resources requested by each proposal [**1,048,856 Rows]**

Data Fields (Test & Train CSVs)

- **id -** unique id of the project application
- **teacher_id** - id of the teacher submitting the application
- **teacher_prefix** - title of the teacher's name (Ms., Mr., etc.)
- **school_state** - US state of the teacher's school
- **project_submitted_datetime** - application submission timestamp
- **project_grade_category** - school grade levels (PreK-2, 3-5, 6-8, and 9-12)
- **project_subject_categories** - category of the project (e.g., "Music & The Arts")
- **project_subject_subcategories** - sub-category of the project (e.g., "Visual Arts")
- **project_title** - title of the project

- **project_essay_1** - "Introduce us to your classroom"
- **project_essay_2** - "Tell us more about your students"
- **project_essay_3*** - "How will students use the materials you're requesting"
- **project_essay_4*** - "Close by sharing why your project will make a difference"
- **project_resource_summary** - summary of the resources needed for the project
- **teacher_number_of_previously_posted_projects** - number of previously posted applications by the submitting teacher
- **project_is_approved** - accepted (0="rejected", 1="accepted") -> train.csv only

*** *Note***: Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

Data Fields (Data Resource.csv)

Proposals also include resources requested. Each project may include multiple requested resources. Each row in resources.csv corresponds to a resource, so multiple rows may tie to the same project by id.

- **id -** unique id of the project application; joins with test.csv. and train.csv on id
- **description -** description of the resource requested
- **quantity -** quantity of resource requested
- **price -** price of resource requested

Problem Statement

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, large numbers of volunteers are needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve [1].

1. How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
2. How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
3. How to focus volunteer time on the applications that need the most assistance

The goal of this project is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

With an algorithm to pre-screen applications, DonorsChoose.org will be able auto-approve some applications quickly so that volunteers can spend their time on more nuanced and detailed project vetting processes, including doing more to help teachers develop projects that qualify for specific funding opportunities.

The problem is in the form of a classic binary classification problem. In the provided data set teachers who have been approved historically have been assigned ‘1’ and those who have not been approved have been assigned ‘0’. Our solution will entail fitting/training several classification algorithms on the teachers associated data to come up with a model which can most accurately predict whether or not future applications should be approved or not.

Metrics

As mentioned above this is a binary classification problem. Potential evaluation metrics for a problem of this type include accuracy, F1/Fbeta scores and AROC.

Because the data set in question is heavily imbalanced (i.e. 85% of teachers applications are approved) an evaluation metric like accuracy would not be a great indicator of the models success. (Example: A Naïve model which predicted that all applications are approved would have an accuracy score of 85% despite having zero intelligence!).

For this reason the accuracy of the Machine Learning model derived as part of this project will be evaluated based on the area **under the ROC curve** between the predicted probability and the observed target. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 − specificity). It can also be thought of as a plot of the Power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities) [2].

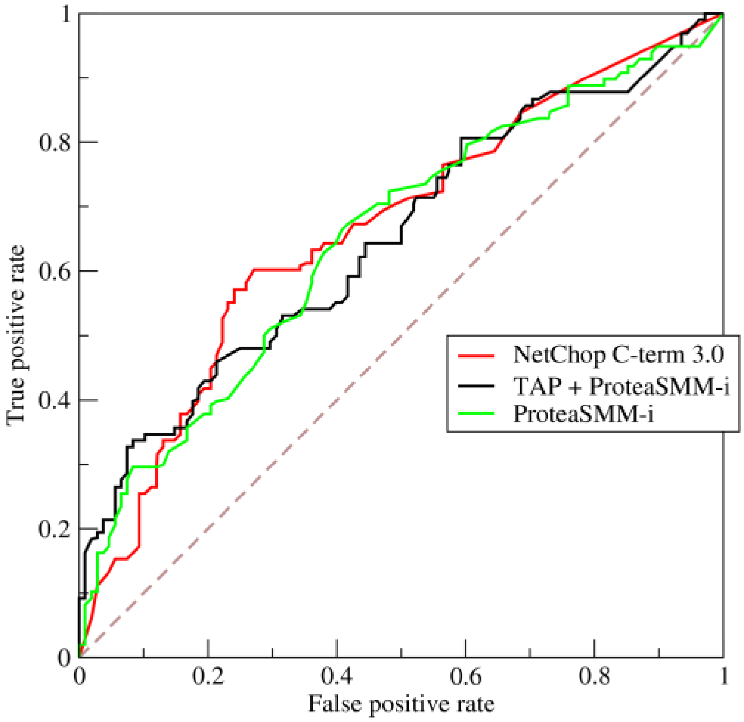
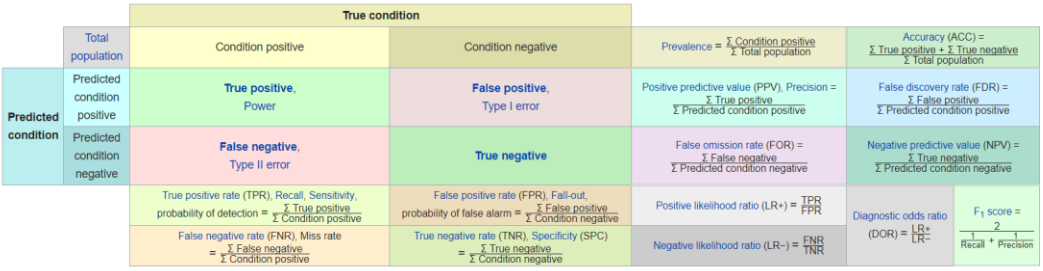


Figure 1.4. Example ROC Curve - Three predictors of peptide cleaving in the proteasome. [2]

In a binary classification like this one, the outcomes are labelled either as positive (p) or negative (n). There are four possible outcomes from a binary classifier. If the outcome from a prediction is p and the actual value is also p, then it is called a true positive (TP); however if the actual value is n then it is said to be a false positive (FP). Conversely, a true negative (TN) has occurred when both the prediction outcome and the actual value are n, and false negative (FN) is when the prediction outcome is n while the actual value is p. The four outcomes can be formulated in a 2×2 contingency table or confusion matrix, as follows:



		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma\text{ Condition positive}}{\Sigma\text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma\text{ True positive} + \Sigma\text{ True negative}}{\Sigma\text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma\text{ True positive}}{\Sigma\text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma\text{ False positive}}{\Sigma\text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma\text{ False negative}}{\Sigma\text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma\text{ True negative}}{\Sigma\text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma\text{ True positive}}{\Sigma\text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma\text{ False positive}}{\Sigma\text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma\text{ False negative}}{\Sigma\text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma\text{ True negative}}{\Sigma\text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Figure 1.5. Calculations to determine a confusion matrix or 2x2 contingency table [2]

In addition to the primary evaluation metric (Area under ROC) as part of the final evaluation I also intent to create a detailed classification report which will include measures of precision, recall, and F1scores.

II. Analysis

Data Exploration

Before commencing any feature engineering, analysis or visualisation a process of exploring the project data was undertaken. Some features of the data uncovered during this exercise will be discussed herein.

Shape of the Data

The primary training data set had 182,080 rows of data, one for each individual application made by a given teacher. In total the training data had 16 columns. 15 of these columns were associated with features of the training data (e.g. State) and the last column in the data set was the target data or result which was whether the teacher's application had been approved or not. This additional target column of data explained why the Test Data set had only 15 columns of data. This fact led me to create a new data frame called Target Data so I could separate out the training data from the target data neatly in the analysis.

The test data contains 78,035 rows of data (42.8% of training data set) and 15 features which matched the ones in the training data set on inspection. It was noted at this stage that no target data was provided for the Test data set as is typical for Kaggle competitions of this nature. As this was the case, for the model evaluation step I decided to use k-fold cross validation on the training data.

A separate table of 'Resource Data' requested by the teachers as part of their application. This table contained 4 features including the total number items ordered, their description, unit and total costs. This data set had 1,541,272 rows of data. This data set has far more entries than the main applications table as each application was often associated with multiple resource line items in the resources table. For ease of analysis it was decided to merge these two tables on the application id to establish a single training data set. Aggregations carried out on the resource requests enabled us to get a better summary picture of the teachers order requests in this final training data table.

Table 1.1. Shape of Datasets Provided

	#Rows of Data	***#Features/Columns***
Training Data	**182,080**	**16**
Test Data	**78,035**	**15**
Resource Data	**1541272**	**4**
Target Data	***182,080***	***1***

Data Types

To utilise most Machine Learning algorithms we must first process different data types (text, categorical, numeric) to get them into model ready states. To do this we must first establish the types of data contained in the data sets.

Table 1.2. Dataset Data Types

	Data Type	***Data Category***
Id	182080 non-null int 64	Numeric
Teacher_id	182080 non-null Object	Categorical
Teacher_prefix	182080 non-null Object	Categorical
School state	182080 non-null Object	Categorical
Project submitted date time	182080 non-null Object	Categorical
Project grade category	182080 non-null Object	Categorical
Project grade subject categories	182080 non-null Object	Categorical
Project grade subject subcategories	182080 non-null Object	Categorical
Project Title	182080 non-null Object	Text
Project Essay 1	182080 non-null Object	Text
Project Essay 2	182080 non-null Object	Text
Project Essay 3	182080 non-null Object	Text
Project Essay 4	182080 non-null Object	Text
Project Resource Summary	182080 non-null Object	Text
Number previously Posted Projects	182080 non-null Object	Numeric
Resource Description	182080 non-null Object	Text
Resource Quantity	182080 non-null Object	Numeric
Resource Price	182080 non-null Object	Numeric

Categorical Statistics

To get a sense for the composition of the datasets I started looking at the basic descriptive statistics of the categorical features in the training data. It became immediately apparent that a significant amount of data was missing for the Project Essay 3 and 4 (175706 NaN or 96% of the data). A decision was made on this basis to remove these features from the modelled data set.

Table 1.3. Descriptive Statistics Categorical Features

	Count	***Unique***	***Top***	***Frequency***
Id	182,080	182,080	p096258	1
Teacher_id	182,080	104414	-	74
Teacher_prefix	182076	5	mrs	95405
School state	182,080	51	CA	25695
Project submitted date	182,080	180439	2016-09-01	30
Project grade category	182080	4	Grades PreK-	73890
Project grade subject	182080	407	Literacy	15757
Project Title	182080	164282	Flexible	377
Project Essay 1	182080	147689	-	46
Project Essay 2	182080	180984	-	24
Project Essay 3	**6374**	**6359**	**-**	**2**
Project Essay 4	**6374**	**6356**	**-**	**4**
Project Resource	182080	179730	-	84
Resource Description	1540980	332928	-	3037

Numerical Statistics

I also composed the main descriptive statistics for the numeric features in the data set. The average number of previously submitted projects by a teacher is 11.24. One teacher has applied 451 times. The average number of resource items asked for as part of a order line item is 2.86 and finally the average price of resource asked for as part of the application is 50.28 dollars. It appears from this analysis that there is likely and upper cap of 9999 dollars on the prices of resources requested by any given teacher.Table 1.4. Descriptive Statistics Numerical Features

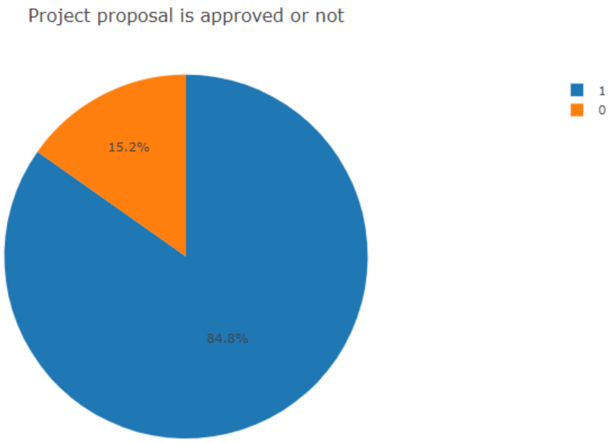
	# Previous Submitted Projects	***Quantity***	***Price***
Count	182,080	1,541,272	1,541,272
Mean	11.24	2.86	50.28
Std	28.02	7.57	144.7
Min	0.00	0.00	0.00
25%	0.00	1.00	7.90
50%	2.00	1.00	14.99
75%	9.00	2.00	39.80
Max	451.00	800	9999

Exploratory Visualization

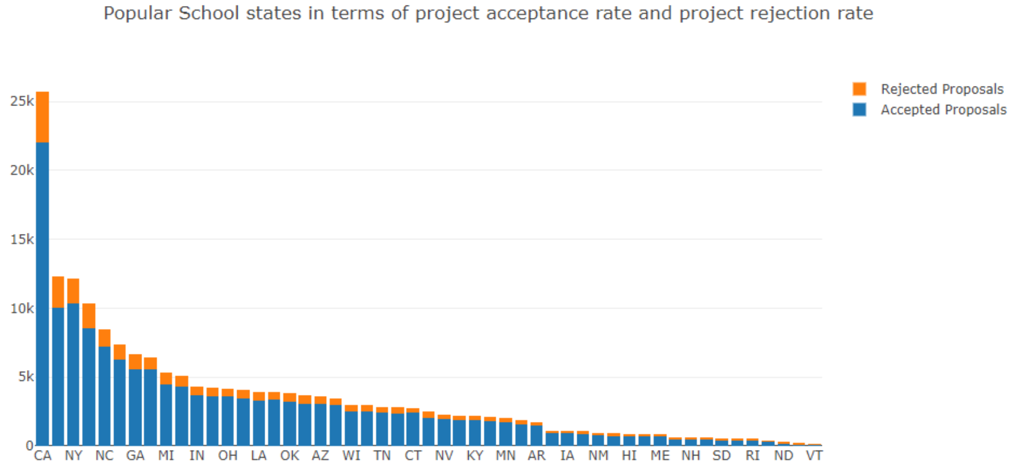
As well as exploring the data numerically via an initial statistical analysis I explored the data visually via a number of plots. For the purpose of this report I will present and discuss two.

Balance of Target Data Set

The training data is highly imbalanced with approximately 85% of the projects being approved while only 15% of the projects were not approved. Majority imbalanced class is positive. With this in mind, an evaluation metric of ‘Area under ROC curve’ is deemed suitable for a problem of this nature. Utilising an evaluation metric like accuracy on a project like this would lead to deceptively good results (e.g. a naivve predictor would have an accuracy score of 85%). This visualisation helped inform both the models and evaluation techniques I will adopt in future steps.



As well as exploring the target data set visually, I also plotted all of the main features in the data set to get a sense for their distribution and influence on the approval rates (see example plot of approval rate). See example figure below which depicts the number of applications submitted per US state and also split in terms of those which were rejected and those which were approved. It is clear from the plot that California has the highest number of submissions and you can also begin to get a sense for the split of rejected/approved ratio per state.



Algorithms and Techniques

This is a binary classification problem. The inputs include categorical, numerical and text features submitted in teacher's applications and the goal is to predict whether the application should be approved or not.

As much of the rich data in this project is in the form of long form text statements I will be using natural language processing and TF-IDF vectorization to process & extract useful features in the text data. I will supplement the categorical and numeric data set with additional features that I generate myself before encoding and scaling the data to be of a suitable format for feeding to the modelling algorithms.

For learning purposes I propose utilising four different classification algorithms with varying levels of complexity:

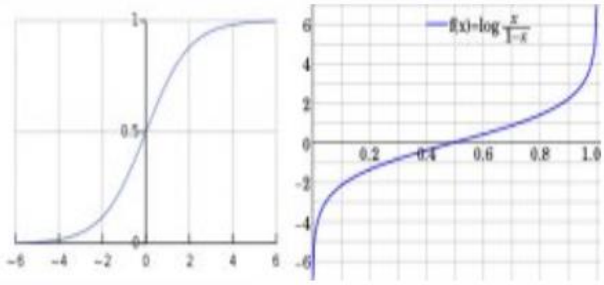
- Logistic Regression
- Random Forest
- Light GBM
- XG Boost

Logistic Regression

Logistic Regression is a popular statistical technique used to predict binomial outcomes (y = 0 or 1) such as we are facing with this project. Logistic regression predicts categorical outcomes (binomial / multinomial values of y), whereas linear Regression is good for predicting continuous-valued outcomes (such as weight of a person in kg, the amount of rainfall in cm). The predictions of Logistic Regression are in the form of probabilities of an event occurring, ie the probability of y=1, given certain values of input variables x. Thus, the results of Logistic regression range between 0-1.

Logistic Regression models the data points using the standard logistic function, which is an S- shaped curve given by the equation:

$$\frac{1}{1+e^{-x}} = \frac{e^{x}}{1+e^{x}}$$



As shown in Figure1, the logit function on the right- with a range of - ∞ to +∞, is the inverse of the logistic function shown on the left- with a range of 0 to 1.

The equation to be solved in Logistic Regression is:

$$y= \text{logit}(p) = \log(p/(1-p))= \beta_0 + \beta_1{}^*x1 + \ldots + \beta_k{}^*xk = Bt.X$$

where:

p = probability that y=1 given the values of the input features, x.

$x_1,x_2,..,x_k$ = set of input features, x.

$B_0,B_1,..,B_k$ = parameter values to be estimated via the maximum likelihood method. $B_0,B_1,..B_k$ are estimated as the 'log-odds' of a unit change in the input feature it is associated with.

B_t = vector of coefficients

X = vector of input features

Random Forest

Decision trees are supervised learning techniques often used in binary classification problems such as this one. Ensemble methods, combine several decision trees to produce better predictive performance than utilizing a single decision tree. The main principle behind the ensemble model is that a group of weak learners come together to form a stronger learner.

Bagging (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here the idea is to create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.

Random Forest is an extension over bagging. It takes one extra step where in addition to taking the random subset of data, it also takes the random selection of features rather than using all features to grow trees. When you have many random trees. It's called Random Forest.

Light GBM

Boosting is another ensemble technique to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. By combining the whole set at the end converts weak learners into better performing model. Gradient Boosting is an extension over boosting method. Light GBM is a gradient boosting framework that uses tree based learning algorithms. Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

Gradient Boosting= Gradient Descent + Boosting.

It uses gradient descent algorithm which can optimize any differentiable loss function. An ensemble of trees are built one by one and individual trees are summed sequentially. Next tree tries to recover the loss (difference between actual and predicted values).

XGBoost

XGBoost is short for “Extreme Gradient Boosting”. This is also an ensemble method that seeks to create a strong classifier (model) based on “weak” classifiers. In this context, weak and strong refer to a measure of how correlated are the learners to the actual target variable. By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor, until the training data is accurately predicted or reproduced by the model. Gradient boosting also comprises an ensemble method that sequentially adds predictors and corrects previous models. However, instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction. So, in the end, you are updating your model using gradient descent and hence the name, gradient boosting. This is supported for both regression and classification problems. XGBoost specifically, implements this algorithm for decision tree boosting with an additional custom regularization term in the objective function.

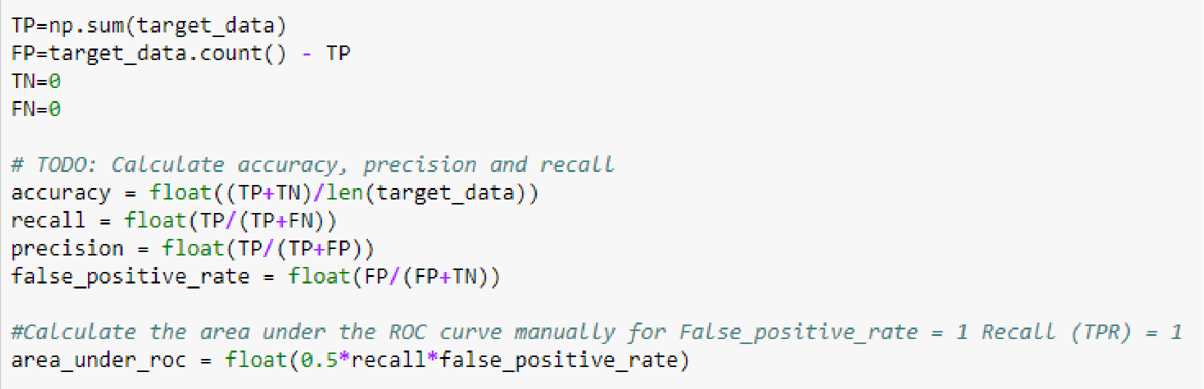
Benchmark

At an early stage in the project I set about determining an initial benchmark from which I could judge the effectiveness of my machine learning models. My benchmarking process had three components:

1. Benchmark 1->Naive Predictor **(AuROC=0.5)**
2. Benchmark 2->Simple Logistic Reg Model (Two Features) **(AuROC=0.62)**
3. Benchmark 3 -> Top Kaggle Competition Leader board **(AuROC=0.82)**

Benchmark 1 -> Naïve Predictor **(AuROC=0.5)**

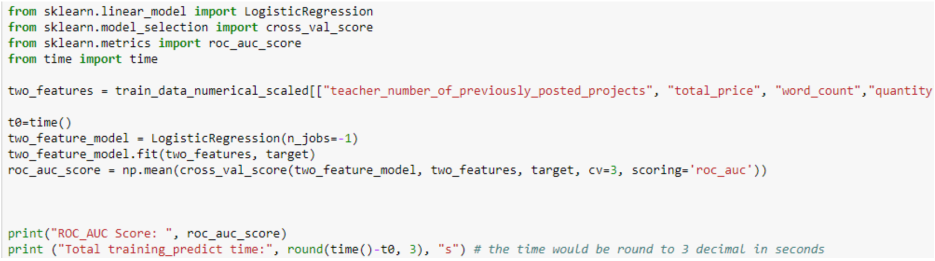
I calculated baseline evaluation metrics for a naïve predictor which predicted every single application will be approved. The resulting Area Under ROC Curve for the Naïve Predictor was equal to 0.5.



```
TP=np.sum(target_data)
FP=target_data.count() - TP
TN=0
FN=0

# TODO: Calculate accuracy, precision and recall
accuracy = float((TP+TN)/len(target_data))
recall = float(TP/(TP+FN))
precision = float(TP/(TP+FP))
false_positive_rate = float(FP/(FP+TN))

#Calculate the area under the ROC curve manually for False_positive_rate = 1 Recall (TPR) = 1
area_under_roc = float(0.5*recall*false_positive_rate)
```

Benchmark 2 -> Simple Logistic Regression Model **(AuROC=0.59)**

I wanted to also establish a baseline result for a very simple ML model. I fit a simple logistic regression model on a stripped back two feature data set. For this purpose I used two numerical features which I felt could have a large bearing on whether or not the teachers application was approved (the total cost of resources the teacher was asking for and how many previously submitted projects the teacher had). The resulting benchmark model had an AuROC value of 0.59 beating the benchmark by 0.09 with only two features.

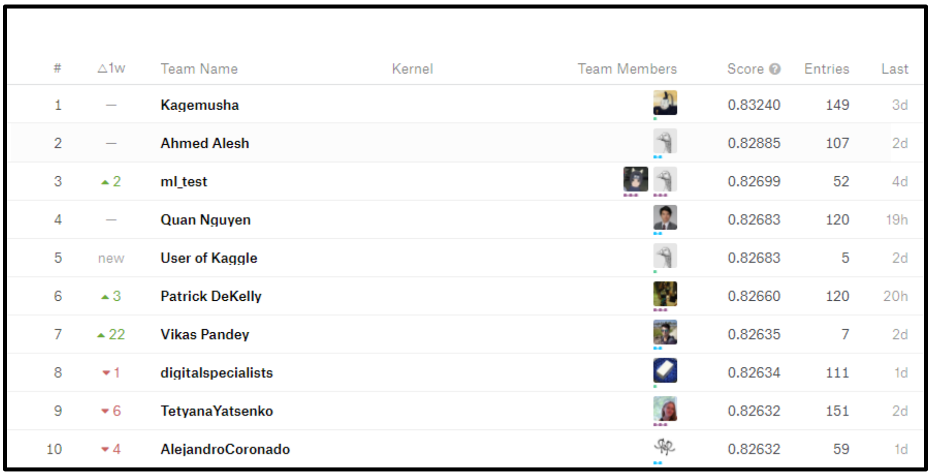


```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from time import time

two_features = train_data_numerical_scaled[["teacher_number_of_previously_posted_projects", "total_price", "word_count","quantity

t0=time()
two_feature_model = LogisticRegression(n_jobs=-1)
two_feature_model.fit(two_features, target)
roc_auc_score = np.mean(cross_val_score(two_feature_model, two_features, target, cv=3, scoring='roc_auc'))



print("ROC_AUC Score: ", roc_auc_score)
print ("Total training_predict time:", round(time()-t0, 3), "s") # the time would be round to 3 decimal in seconds
```

Benchmark 3 -> Top Score Kaggle Competition Leaderboard **(AuROC=0.83)**

Benchmark ROC scores of solutions to this problem are outlined in the competitive leader board on Kaggle. A score above a value of 0.76 would put the ML algorithm performance in the top 50% of public results. A score above a value of 0.82 would put the ML algorithm in the top 30% of public results. A score greater than 0.83240 would be the best performing ML algorithm for the problem.

Table 1.1. Public Leader board- Donors Choose Application Screening (Kaggle.com)



#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1	—	Kagemusha			0.83240	149	3d
2	—	Ahmed Alesh			0.82885	107	2d
3	▴2	ml_test			0.82699	52	4d
4	—	Quan Nguyen			0.82683	120	19h
5	new	User of Kaggle			0.82683	5	2d
6	▴3	Patrick DeKelly			0.82660	120	20h
7	▴22	Vikas Pandey			0.82635	7	2d
8	▾1	digitalspecialists			0.82634	111	1d
9	▾6	TetyanaYatsenko			0.82632	151	2d
10	▾4	AlejandroCoronado			0.82632	59	1d

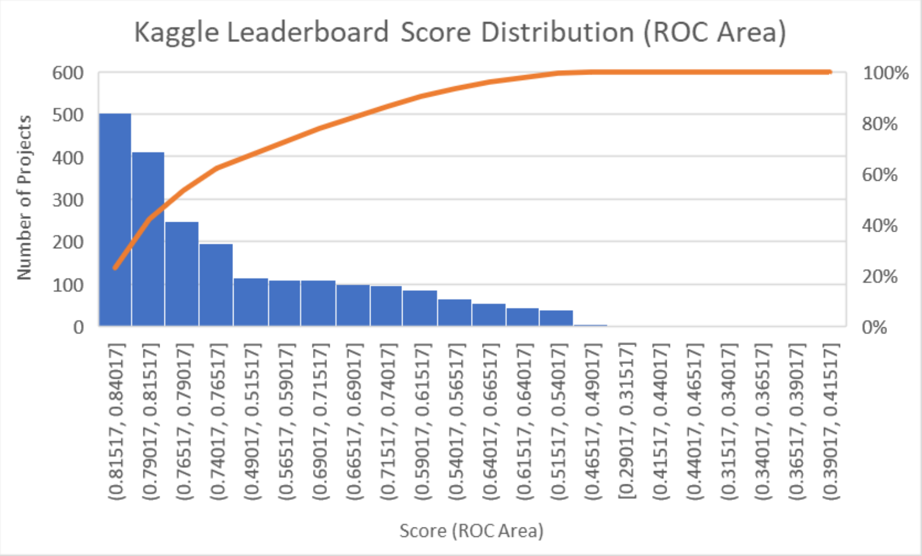


Figure 1.3. Cumulative Distribution of Kaggle leader board scores (Donors Choose)

III. Methodology

Data Pre-processing

Before fitting training any of our predictive classifiers on the given data a series of data pre-processing steps had to be undertaken to:

1. Get the data into a suitable format for feeding the training algorithms
2. Create new features from the existing data

Feature Engineering

Removing unwanted Features

As identified in the data exploration step there were some features in the data set that were missing significant amounts of data (e.g. essay 3). These features along with some others were removed from the data at an early stage. These removed features included:

- Essay 3 – Removed as over 96% of data was missing
- Essay 4 – Removed as over 96% of data was missing
- Teacher ID – Did not carry any meaningful intelligence
- Project Submitted Date Time – removed after transforming into useable numeric format

New Aggregated features

7 new features were obtained by aggregating the fields from resources data and the training data:

- Feature 1,2,3 **-** Min Price, Max Price, Mean Price: Min, Max, and Mean value of Price of resources requested.
- Feature 4,5,6 **-** Min Quantity, Max Quantity, Mean Quantity: Min, Max, and Mean value of Quantity of resources requested.
- Feature 7,8,9 **-** Min Total Price, Max Total Price, Mean Total Price: Min, Max, and Mean value of Total Price of resources requested.
- Feature 10,11,12 **-** Sum of Total Price: Total price of all the resoruces requested by the teacher in a proposal
- Feature 13 **-** Items Requested: Total unique number of items requested by the teacher in a proposal
- Feature 14 - Quantity: Total number of quantities requested by the teacher in a proposal

New Date-Time Features

Six new features were obtained obtained by extracting from project submitted datetime

- Feature 15 - Year of Submission: Value of year when the proposal was submitted
- Feature 16 - Month of Submission: Month number (values between 1 to 12) when the proposal was submitted
- Feature 17 - Week Day of Submission: Week Day value (values between 1 to 7) when the proposal was submitted
- Feature 18 - Hour of Submission: Value of time hour (values between 0 to 23) when the proposal was submitted
- Feature 19 **-** Year Day of Submission: Year Day (values between 1 to 365) when the proposal was submitted

- Feature 20 - Month Day of Submission: Month Day (values between 1 to 31) when the proposal was submitted

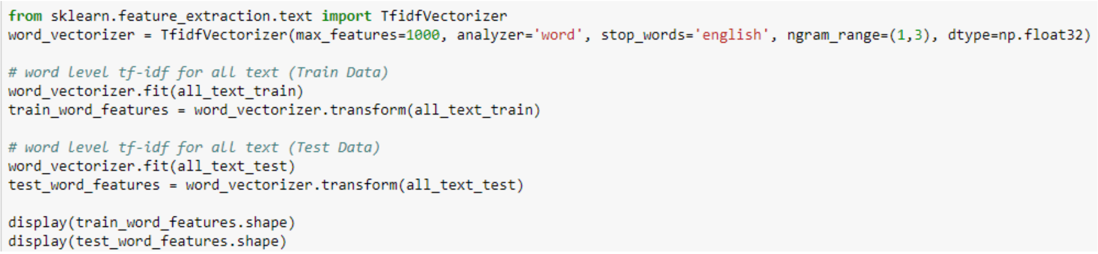
New Text Based Features

Features extracted from proposal essay text and resources description included:

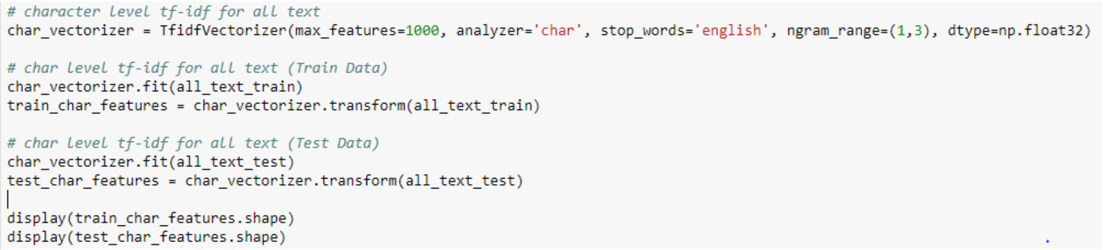
- Feature 21: Length of Essay 1 - total number of characters in essay 1 including spaces
- Feature 22: Length of Essay 2 - total number of characters in essay 2 including spaces
- Feature 23: Length of Project Title - total number of characters in project title including spaces
- Feature 24**:** Word Count in the Complete Essay - total number of words in the complete essay text
- Feature 25: Character Count in the Complete Essay - total number of characters in complete essay text
- Feature 26: Word Density of the Complete Essay - average length of the words used in the essay
- Feature 27: Puncutation Count in the Complete Essay - total number of punctuation marks in the essay
- Feature 28**:** Upper Case Count in the Complete Essay - total number of upper count words in the essay
- Feature 29: Title Word Count in the Complete Essay - total number of proper case (title) words in the essay
- Feature 30: Stopword Count in the Complete Essay - total number of stopwords in the essay

New TFIDF Word & Character Features

One of the main elements of the data pre-processing involved processing the text bodies of the teacher applications using TFIDF vectorization. In information retrieval, TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word or char is to a document in a collection or corpus [3,4]. As part of this project we will use TFIDF to determine weighting factors whose values increases proportionally to the number of times a word appears in the teacher essays while offsetting by the frequency of the word in the corpus, which will help to adjust for the fact that some words appear more frequently in general. For the purposes of this project I have limited the number of word and char features extracted to 1000.



```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer(max_features=1000, analyzer='word', stop_words='english', ngram_range=(1,3), dtype=np.float32)

# word level tf-idf for all text (Train Data)
word_vectorizer.fit(all_text_train)
train_word_features = word_vectorizer.transform(all_text_train)

# word level tf-idf for all text (Test Data)
word_vectorizer.fit(all_text_test)
test_word_features = word_vectorizer.transform(all_text_test)

display(train_word_features.shape)
display(test_word_features.shape)
```

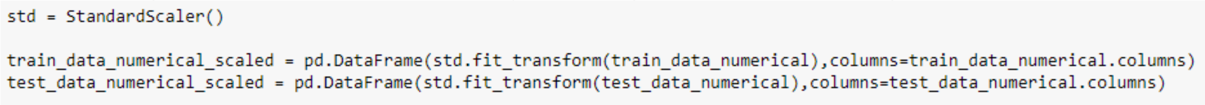


```
# character level tf-idf for all text
char_vectorizer = TfidfVectorizer(max_features=1000, analyzer='char', stop_words='english', ngram_range=(1,3), dtype=np.float32)

# char level tf-idf for all text (Train Data)
char_vectorizer.fit(all_text_train)
train_char_features = char_vectorizer.transform(all_text_train)

# char level tf-idf for all text (Test Data)
char_vectorizer.fit(all_text_test)
test_char_features = char_vectorizer.transform(all_text_test)

display(train_char_features.shape)
display(test_char_features.shape)
```

Feature Scaling & Transformation

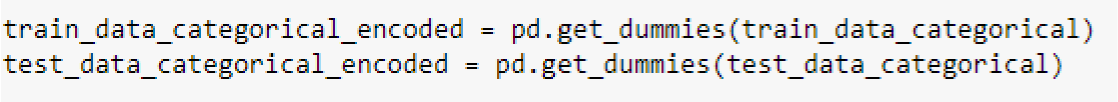
After I had augmented the initial data sets with additional features I had to scale, encode and transform the raw input data to get it into a form suitable for feeding to the training algorithms. This involved a few primary steps including:

1. Standardizing numerical features features by removing the mean and scaling to unit variance



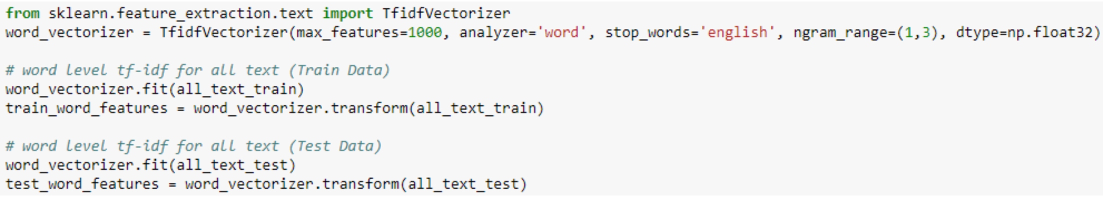
```
std = StandardScaler()

train_data_numerical_scaled = pd.DataFrame(std.fit_transform(train_data_numerical),columns=train_data_numerical.columns)
test_data_numerical_scaled = pd.DataFrame(std.fit_transform(test_data_numerical),columns=test_data_numerical.columns)
```

2. Encoding categorical features with integer representations and labels



```
train_data_categorical_encoded = pd.get_dummies(train_data_categorical)
test_data_categorical_encoded = pd.get_dummies(test_data_categorical)
```

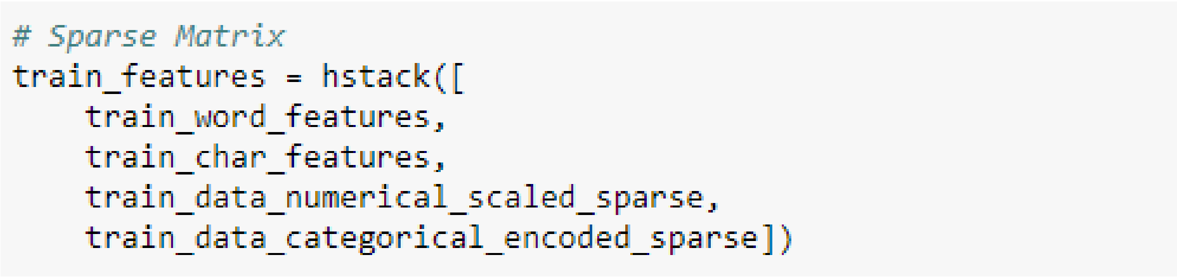
3. Creating sparse matrix representation of char/word scores (in csr format)



```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer(max_features=1000, analyzer='word', stop_words='english', ngram_range=(1,3), dtype=np.float32)

# word level tf-idf for all text (Train Data)
word_vectorizer.fit(all_text_train)
train_word_features = word_vectorizer.transform(all_text_train)

# word level tf-idf for all text (Test Data)
word_vectorizer.fit(all_text_test)
test_word_features = word_vectorizer.transform(all_text_test)
```

Combining Training Features

Once the categorical, numeric and text based features had been pre-processed into suitable formats the final step was to combine the three into a single sparse matrix using horizontal stacking.

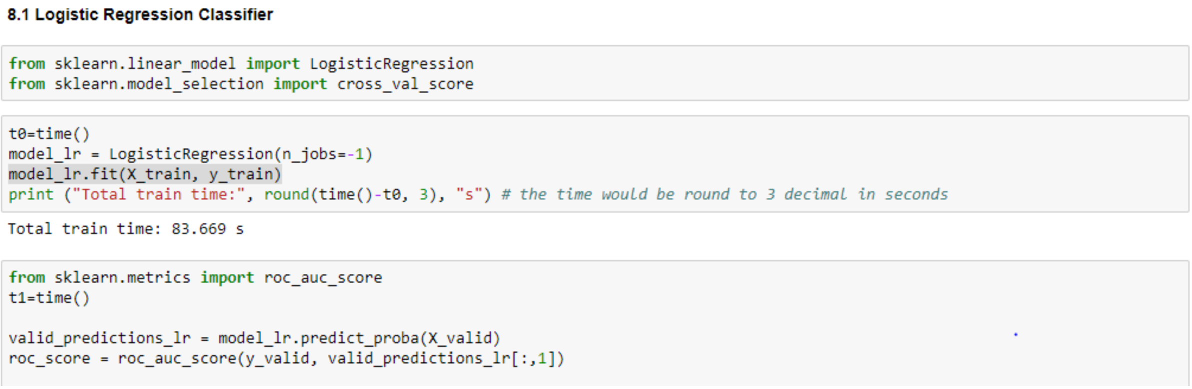


```
# Sparse Matrix
train_features = hstack([
    train_word_features,
    train_char_features,
    train_data_numerical_scaled_sparse,
    train_data_categorical_encoded_sparse])
```

Implementation

Logistic Regression Model

I initially tried to fit a simple logistic regression model to the training features. To determine the performance of the resulting predictions from the model I used a Cross Validation Score method with 3 folds and an ROC_AUC scoring function. The resulting model took 83s to fit/train and 0.17s to predict. It had a cross validation ROC_AUC_Score of 0.727 and got a score of 0.714 on the test set in the Kaggle Competition.



8.1 Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
```

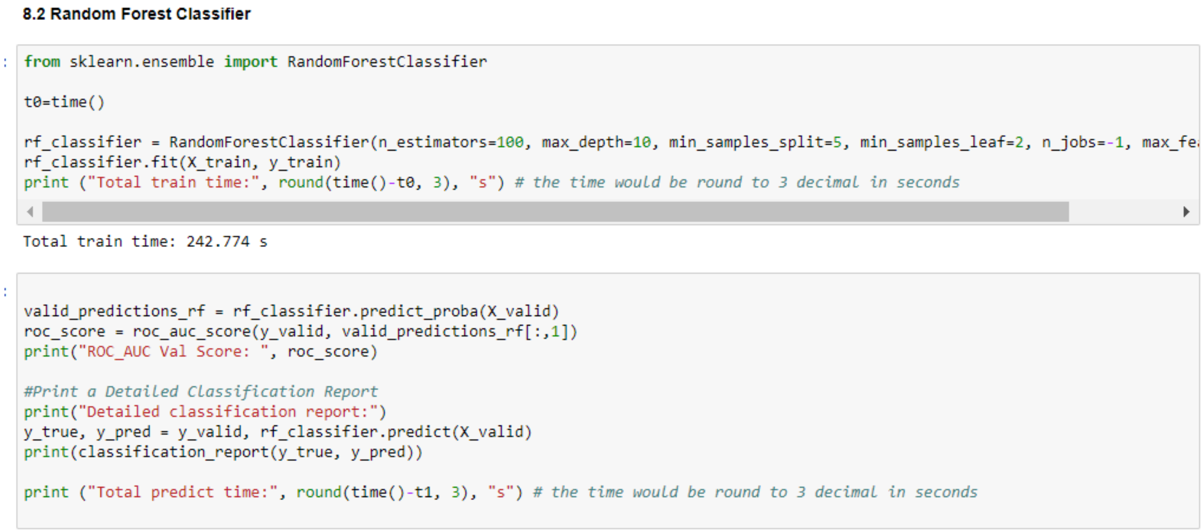
```
t0=time()
model_lr = LogisticRegression(n_jobs=-1)
model_lr.fit(X_train, y_train)
print ("Total train time:", round(time()-t0, 3), "s") # the time would be round to 3 decimal in seconds
```

```
Total train time: 83.669 s
```

```
from sklearn.metrics import roc_auc_score
t1=time()

valid_predictions_lr = model_lr.predict_proba(X_valid)
roc_score = roc_auc_score(y_valid, valid_predictions_lr[:,1])
```

Random Forest Classifier

After trying the Logistic Regression model I tried to fit, train and predict using a Random Forest Classifier. This algorithm uses a bagging technique that combines a group of weak learners to form a stronger learner. In this implementation I used 100 different decision trees with a max depth of each tree equal to 10. The minimum number of samples required to split an internal node was set as 5 and the minimum number of samples required to be at a leaf node was set to be equal to 5 also. These initial hyperparameters were chosen from suggested good ranges to use from other practitioners. The resulting model took 242s to fit/train and 1649s to predict. It had a ROC_AUC_Score of 0.72 and got a score of 0.68 on the test set in the Kaggle Competition.



8.2 Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

t0=time()

rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=5, min_samples_leaf=2, n_jobs=-1, max_fe
rf_classifier.fit(X_train, y_train)
print ("Total train time:", round(time()-t0, 3), "s") # the time would be round to 3 decimal in seconds
```

```
Total train time: 242.774 s
```

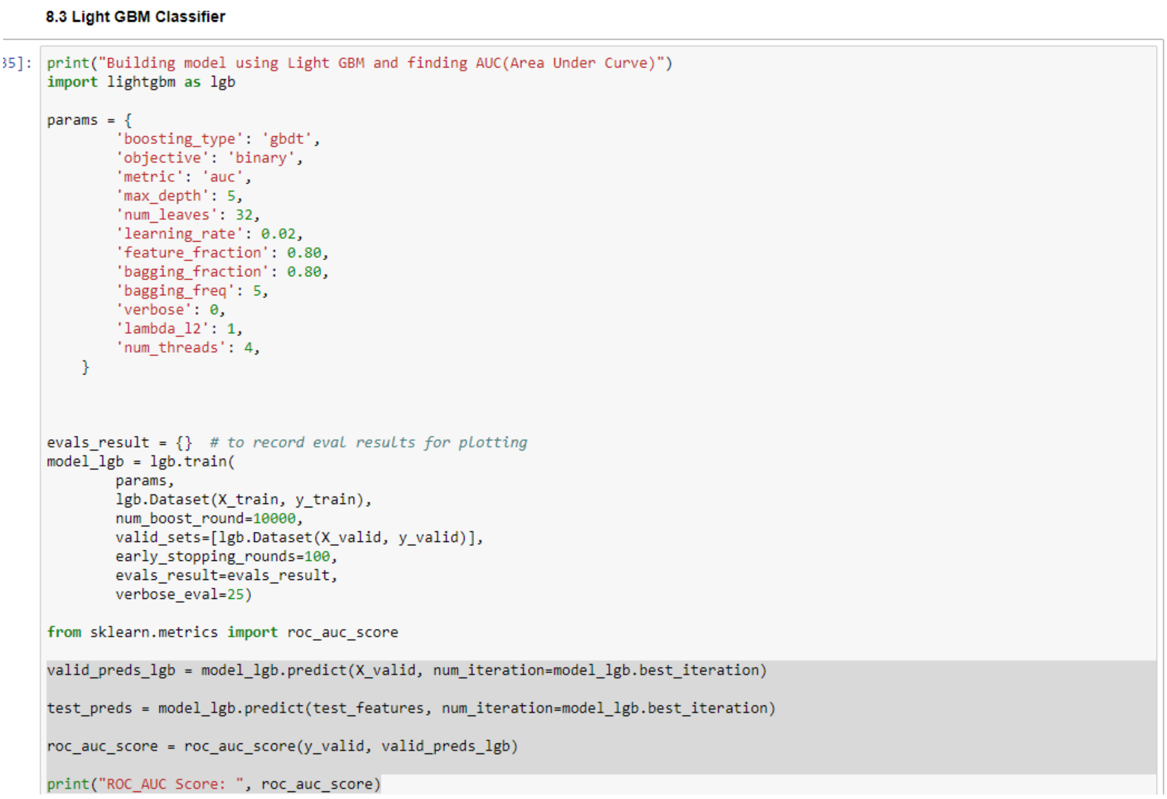
```

valid_predictions_rf = rf_classifier.predict_proba(X_valid)
roc_score = roc_auc_score(y_valid, valid_predictions_rf[:,1])
print("ROC_AUC Val Score: ", roc_score)

#Print a Detailed Classification Report
print("Detailed classification report:")
y_true, y_pred = y_valid, rf_classifier.predict(X_valid)
print(classification_report(y_true, y_pred))

print ("Total predict time:", round(time()-t1, 3), "s") # the time would be round to 3 decimal in seconds
```

Light GBM

Light GBM is a gradient boosting framework that uses tree based learning algorithms. Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

In this implementation I set the max tree depth to 5. This is an important parameter for controlling over fitting. I set the number of leaves to 32, this is the most important parameter for controlling the complexity of the tree model. To decide the number of leaves to use I followed the suggested guidelines of utilising 2^max depth. I utilised a learned rate of 0.02. The learning rate determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in estimates. Feature fraction was set at a value of 0.8. 0.8 feature fraction means Light GBM will select 80% of parameters randomly in each iteration for building trees. A bagging fraction of 0.8 specifies the fraction of data to be used for each iteration and is generally used to speed up training and prevent over fitting. An early_stopping_round value of 100 was used to speed up your analysis. The model will stop training if one metric of one validation data doesn't improve in last 100 rounds. This will reduce excessive iterations. The boosting type used was traditional gradient boosting decision trees. Potential alternatives include Random Forest, Dart or Goss.

The resulting model had a ROC_AUC_Score of 0.779 on the validation data set and and got a score of 0.747 on the test set in the Kaggle Competition.



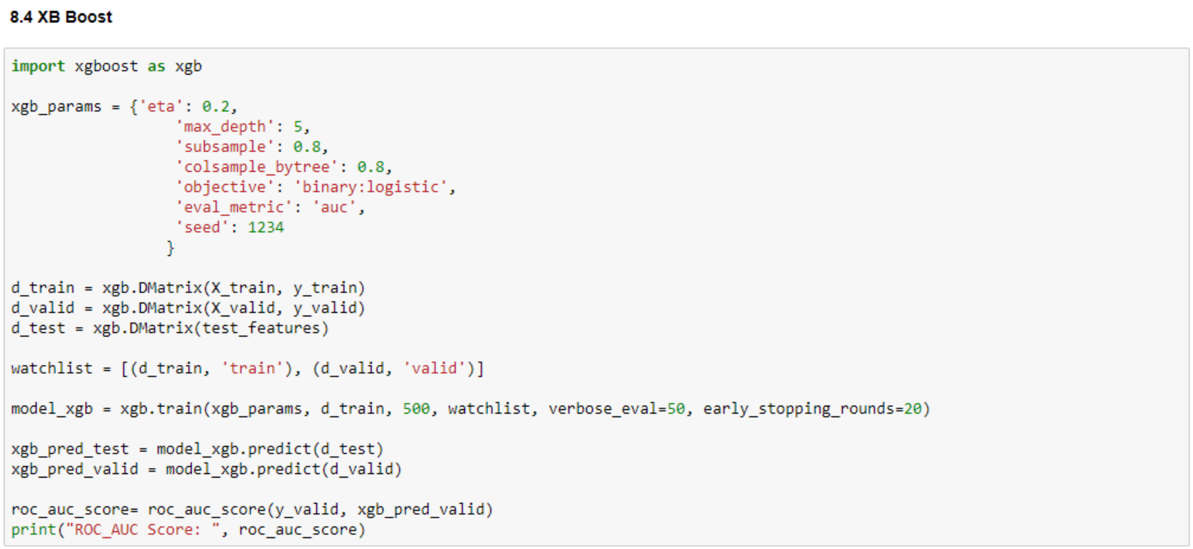
8.3 Light GBM Classifier

```
35]: print("Building model using Light GBM and finding AUC(Area Under Curve)")
     import lightgbm as lgb

     params = {
             'boosting_type': 'gbdt',
             'objective': 'binary',
             'metric': 'auc',
             'max_depth': 5,
             'num_leaves': 32,
             'learning_rate': 0.02,
             'feature_fraction': 0.80,
             'bagging_fraction': 0.80,
             'bagging_freq': 5,
             'verbose': 0,
             'lambda_l2': 1,
             'num_threads': 4,
         }



     evals_result = {}  # to record eval results for plotting
     model_lgb = lgb.train(
             params,
             lgb.Dataset(X_train, y_train),
             num_boost_round=10000,
             valid_sets=[lgb.Dataset(X_valid, y_valid)],
             early_stopping_rounds=100,
             evals_result=evals_result,
             verbose_eval=25)

     from sklearn.metrics import roc_auc_score

     valid_preds_lgb = model_lgb.predict(X_valid, num_iteration=model_lgb.best_iteration)

     test_preds = model_lgb.predict(test_features, num_iteration=model_lgb.best_iteration)

     roc_auc_score = roc_auc_score(y_valid, valid_preds_lgb)

     print("ROC_AUC Score: ", roc_auc_score)
```

XG Boost

XGBoost is short for “Extreme Gradient Boosting”. This is also an ensemble method that seeks to create a strong classifier (model) based on “weak” classifiers. In this context, weak and strong refer to a measure of how correlated are the learners to the actual target variable. By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor, until the training data is accurately predicted or reproduced by the model. Gradient boosting also comprises an ensemble method that sequentially adds predictors and corrects previous models. However, instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction. So, in the end, you are updating your model using gradient descent and hence the name, gradient boosting. This is supported for both regression and classification problems. XGBoost specifically, implements this algorithm for decision tree boosting with an additional custom regularization term in the objective function.

The resulting model had a ROC_AUC_Score of 0.764 and got a score of 0.702 on the test set in the Kaggle Competition.



8.4 XB Boost

```
import xgboost as xgb

xgb_params = {'eta': 0.2,
                  'max_depth': 5,
                  'subsample': 0.8,
                  'colsample_bytree': 0.8,
                  'objective': 'binary:logistic',
                  'eval_metric': 'auc',
                  'seed': 1234
                 }

d_train = xgb.DMatrix(X_train, y_train)
d_valid = xgb.DMatrix(X_valid, y_valid)
d_test = xgb.DMatrix(test_features)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

model_xgb = xgb.train(xgb_params, d_train, 500, watchlist, verbose_eval=50, early_stopping_rounds=20)

xgb_pred_test = model_xgb.predict(d_test)
xgb_pred_valid = model_xgb.predict(d_valid)

roc_auc_score= roc_auc_score(y_valid, xgb_pred_valid)
print("ROC_AUC Score: ", roc_auc_score)
```

Refinement

Having determined from the initial model analysis that the Light GBM model was likely to give the best performing predictive classifications for this given project, the next step was to tune the Hyper Parameters of the model to see could we improve it's performance. Initially I tried to use grid SKLearns 'GridSearchCV' method to tune a number of important parameters in the model but ran into difficulty with it's implementation. In the end I studied the underlying principles of the algorithm to determine likely ways to improve the performance of the model. These alterations to the original model included:

Table 1.6. Tuned Light GBM model Hyperparameters

	Base Case Light GBM Model	***Tuned Model***
Max Depth	5	7
Number of Leaves	32	120
Learning Rate	0.02	0.01

I increased the maximum tree depth from 5 to 7. In line with this change I also increased the number of leaves from 32 to 120 in line with a general rule of thumb of (2^max_depth). And finally I reduced the learning rate from 0.02 to 0.01. In general I was trying to improve the complexity, fit and accuracy of the model without risking overfitting. The results of both the base model and the final tuned model are presented in table 1.4 below and the AUC_ROC metric improvement during training is presented in figure 1.4.

Table 1.7. Comparison of Base Case Vs Tuned Light GBM classifier

	Base Model Performance	***Tuned Model Performance***
ROC_AUC (Eval Metric)	0.779	0.778
Kaggle Competition Score	0.747	0.750

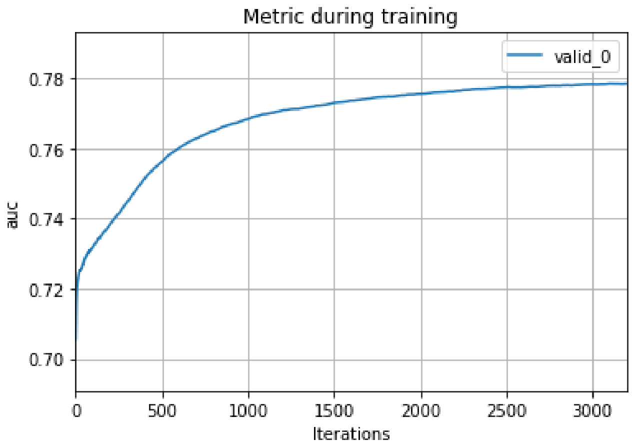


Figure 1.6 Improvement in AUC_ROC score during training tuned Light GBM Classifier

IV. Results

Model Evaluation and Validation

The final tuned model utilised for the purposes of predicting whether or not a teacher's application should be approved or not was that of a Light GBM model. The tuned hyperparameters of the final model where (Max Depth->7 and Number Leaves->120 and Learning Rate ->0.01). The final tuned model had a ROC_AUC score of 0.778 and achieved a score of 0.750 in the final Kaggle competition. This was the highest score achieved by any of my models.

A summary of the classification performance of the various models explored as part of this project are outlined in table 1.8 below. As can be seen from the results my final tuned model had the best Kaggle competition score of 0.750.

Table 1.8. Comparing the models evaluated as part of this project

	Logistic Regression	***Random Forest***	***Light GBM***	***XGBoost***	****Tuned LB Model***
ROC_AUC	0.727	0.722	0.779	0.765	0.778
Kaggle Score	0.714	0.68	0.747	0.702	0.750

Justification

The final tuned Light GBM model had a ROC_AUC score of 0.750 in the Kaggle Competition. This out performed the Naïve benchmark by 0.25 and the untuned Logistic Regression model by 0.003. In terms of the overall Kaggle competition this entry lagged the winning score by 0.078. My final submission placed my in the top 50% of scores in the Kaggle competition.

Table 1.9. Model Evaluation & Validation

	ROC_AUC
Naïve Benchmark	0.5
Logistic Regression (2 Feature Model)	0.592
Light GBM Classifier	0.747
Light GBM Classifier (Tuned)	0.750
Kaggle Competition Winner	0.828

V. Conclusion

Free-Form Visualization

The ROC AUC Score is the corresponding score to the ROC AUC Curve. It is simply computed by measuring the area under the curve, which is called AUC. A classifiers that is 100% correct, would have a ROC AUC Score of 1 and a completely random classiffier would have a score of 0.5. A plot of the receiving operating characteristic is presented for the four models considered as part of this project. It is clear in Figure 1.8 that the area under the LGBM curve is greatest from the plot with an AUC score of 0.78.

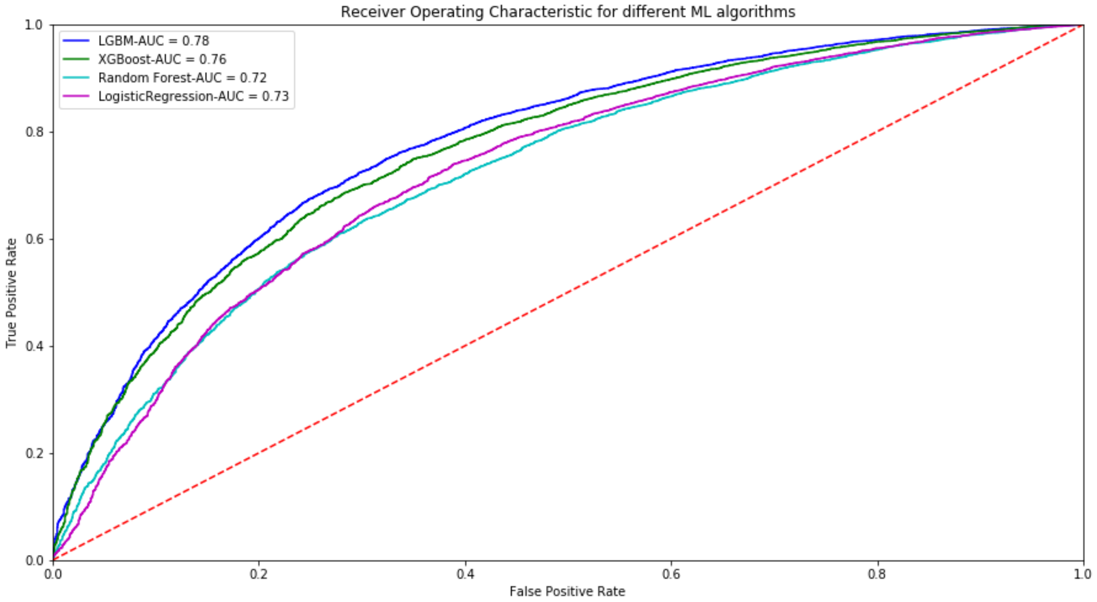


Figure 1.8 ROC Characteristic for ML algorithms trialed as part of this proejct

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned. The precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall). A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly. A plot of the precision-recall curve for the final tuned Light GBM model is presented in figure 1.9 below.

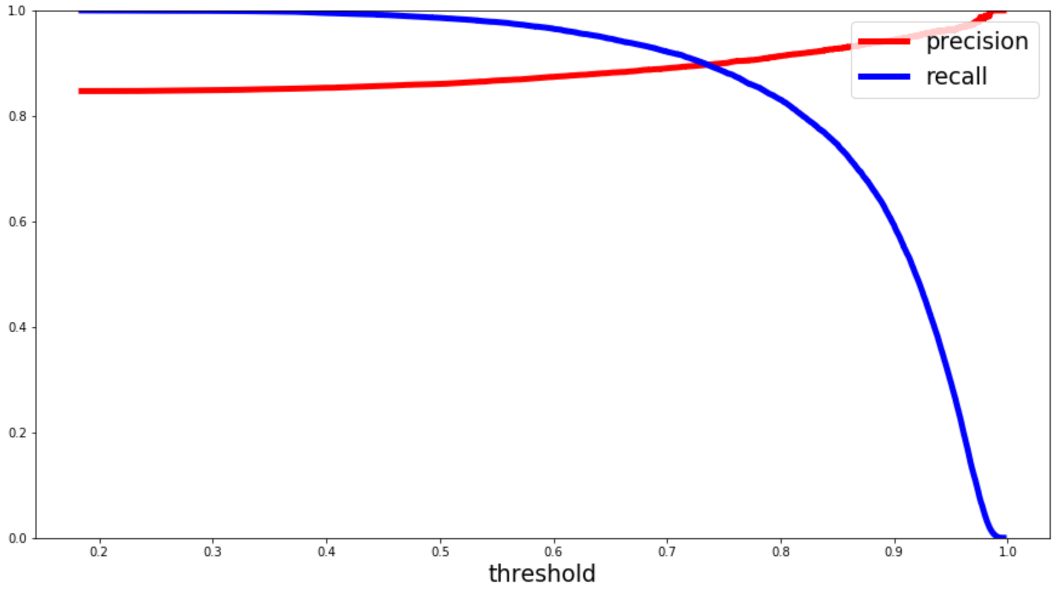


Figure 1.9 Precision-Recall curve for final Light GBM model

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and public data set was found
2. The data was downloaded and imported into a Jupiter notebook for analysis
3. The data was explored statistically and visually to determine interesting characteristics, informative trends and anomalies/outliers.
4. Extensive feature engineering was then carried out to remove erroneous data and outliers, create new features from the existing data and extract features from data of different types (e.g. TFIDF extraction was used to obtain features from the text features in the data set).
5. The next step of the process involved pre-processing the data to get it into the right format for feeding the machine learning models. This involved scaling the numerical features, encoding the categorical features and using TFIDF vectorization to extract useful text features.
6. These pre-processed data (categorical, numeric, text) was then combined into a single training data using horizontal matrix appending.
7. Before evaluating any machine learning models a simple benchmark was created using a Naïve Predictor and a simple two feature logistic regression model to get a feel for what a non-intelligent solution looked like.
8. Four separate binary classifiers where trained to establish their base performance levels including a Logistic Regression Classifier, Random Forest Classifier, Light GBM and XGBoost classifier. These models were evaluated and compared using their ROC_AUC scores to determine the most promising model
9. The best performing model from the previous step (Light GBM) was then tuned to establish the optimal set of Hyper parameters
10. The final evaluation steps on the tuned model involved plotting its' ROC curve, Precision-Recall curve

The most difficult part of the project was dealing with memory issues that arose when running the pre-processing and models on my computer. The large memory taken up by the TFIDF feature sparse matrices and also the process of stacking the numerical, categorical and text data features led to my computer freezing multiple times. While this was frustrating it forced me to really learn about how data is stored in memory and how to control memory issues during the process. This process made me appreciate the power of smart, simple models and also the power of utilising cloud based compute power for future projects. Because I had never used a Light GBM classifier before I had to do a lot of reading to understand how that works on how to model it effectively. I also found tuning the light GBM model using Grid Search CV difficult and in the end reverted to tuning manually based on what I had learned about the model.

The most interesting aspect of the project was the use of TFIDF vectorization to extract useful features from the text input data in the project. This was something I had not done before but something I will likely use in a lot of future projects. I also found the process of incrementally building up the complexity of my input features and algorithms an interesting way to get a feel for a project/problems complexity and the relative power of the techniques being employed.

Improvement

To improve the results I obtained in my implementation for this project I would consider the following:

- *Training my models on a AWS server with higher computer power and memory. In certain situations I had to limit aspects of my input data and training model to stop my computer from freezing or crashing due to memory issues. This extra compute power would for example allow me to Increase the number of Character and Word features included in my Input models*
- *I would explore the use of a multi-channel deep learning model in an attempt to improve the results of my predictions. From the Kaggle leaderboard it appears that this was the top performing model but one which I did not get to implement.*
- *I would like to explore the use of model tuning for all three models considered in the Model Selection stage. In this example I considered three base case models with assumed parameters and took the best model to the tuning stage. I feel that in certain circumstances with tuning some of the models I disgarded at this step may well have performed best after tuning. Something to consider for again.*
- *In doing this project I had to run my Jupyter Notebook more than 50 times. Everytime I re-opened the jupyter notebook I had to do all of the intensive pre-processing (e.g. TFIDF and model training) to get the data ready in my notebook. This proved to be very time consuming and wasteful to have to do this every time. I am going to look for solutions to this in the future. I believe creating saved dumps of key result sets may well be the way to go with this from what I have read.*
- *I would also like to explore extracting even more features from the text data in the project. I looked at basic things like length of text, word and character features but I would like to extend this exploration to features like the number of Stop Words, Punctuation marks, Capital Letters, Misspellings etc. I think that these subtle features could well be a strong indication of application quality and the resulting chances of approval.*
- *I would look at splitting aspects of my project into separate Jupyter notebooks and scripts (example: I would include my data visualisation in a separate notebook to avoid the notebook getting too long and daunting).*
- *In the end I tuned the final Light GBM model manually. In the future I would like to get Grid Search CV to work so that I could explore even more Hyper parameters.*

References

[1] “Kaggle Competition: Donors Choose Application Screening”, 17/04/18, https://www.kaggle.com/c/donorschoose-application-screening

[2] “Receiver operating characteristic”, 16/04/18, 1https://en.wikipedia.org/wiki/Receiver_operating_characteristic,

[3] “tf-idf”, 16/04/18, https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[4] Breitinger, Corinna; Gipp, Bela; Langer, Stefan (2015-07-26). "Research-paper recommender systems: a literature survey". International Journal on Digital Libraries. 17 (4): 305–338. doi:10.1007/s00799-015-0156-0. ISSN 1432-5012.