Machine Learning Engineer Nanodegree

Kickstarter Success Prediction

Sujit Horakeri

February 28, 2018

I. Definition

Project Overview

Kickstarter is a community of more than 10 million people comprising of creative, tech enthusiasts who help in bringing creative project to life. Till now, more than $3 billion dollars have been contributed by the members in fueling creative projects. The projects can be literally anything – a device, a game, an app, a film etc.

Kickstarter works on all or nothing basis i.e. if a project doesn't meet its goal, the project owner gets nothing. For example: if a project's goal is $500. Even if it gets funded till $499, the project won't be a success.

I'm trying to build a binary classification model here to find if a project will reach its intended goal of fund amount or not.

From the sited articles below, it is clear that to build a good classification model, we would need to consider variety of features like name, description, goal of the project, duration etc. Also, algorithms like

Random Forest seem to have the best accuracy among others.

Related Works:

https://cseweb.ucsd.edu/classes/wi17/cse258-a/reports/a108.pdf

https://www.stat.berkeley.edu/~aldous/157/Old_Projects/Haochen_Zhou.pdf

https://www.kaggle.com/codename007/funding-successful-projects

The data is gotten from:

https://www.kaggle.com/codename007/funding-successful-projects/data

https://www.hackerearth.com/problem/machine-learning/funding-successful-projects/description/

Dataset Details:

project_id: unique id of project

name: name of the project

desc: description of project

goal: the goal (amount) required for the project

keywords: keywords which describe project

disable communication: whether the project authors has disabled communication option with people donating to the project

country: country of project author

currency: currency in which goal (amount) is required

deadline: till this date the goal must be achieved (in unix timeformat)

state_changed_at: at this time the project status changed. Status could be successful, failed, suspended, cancelled etc. (in unix timeformat)

created_at: at this time the project was posted on the website (in unix timeformat)

launched_at: at this time the project went live on the website (in unix timeformat)

backers_count: no. of people who backed the project

final_status: whether the project got successfully funded (target variable – 1, 0)

There should not be any missing data issues as the dataset contains very less number of null/nan values.

There are 3 null items in the name columns of training data and 9 missing values in the desc column.

While the test data has 4 missing desc entries

There is an imbalance in the target variable distribution as there are more entries with unsuccessful values than successful. Below are the value counts

Unsuccessful (0): 73568

Successful (1): 34561

Problem Statement

This is a binary classification problem. The aim here is to create a prediction model to predict whether the project will be funded or not i.e., we need to determine if the kickstarter project will be a success or if it won't be a success.

The inputs here are:

'project_id', 'name', 'desc', 'goal', 'keywords', 'disable_communication', 'country', 'currency', 'deadline', 'state_changed_at', 'created_at', 'launched_at'

The output should be:

'final_status' – 0 //failure

1 //success

The variable backers_count is only available in the training data and is not present in the testing data and hence cannot be used for training the model.

To solve this problem, I would initially start with a naïve predictor which predicts all the projects to be successfully funded. I will then move on to create a benchmark model which uses Random Forest algorithm. Finally, I'll try to outscore the benchmark model using LightGBM model. Both the benchmark model and lightGBM model would be cross validated along with tuning the hyperparameters so that we have a model which generalizes to unseen data.

Metrics

The evaluation metric proposed in the competition is accuracy score, I'll be using accuracy score as the evaluation metric.

Accuracy is defined as the percentage of the predictions that were accurate.

Accuracy = (true positives + true negatives) / total population

We could also use other metrics like F score in combination with Accuracy score as accuracy score alone might be misleading sometimes. And since we have an imbalance in the distribution of the target variable in the dataset at hand, it is better to consider both Accuracy and F1 score as the evaluation metric.

Fscore is the harmonic mean of precision and recall:

(1+β^2)*precision*recall / ((β^2*precision + recall)

Where,

Precision = True Positives/ (True Positives + false positives)

Recall = True Positives/ (True Positives + false negatives)

II. Analysis

Data Exploration

Data Sample

Let us look at how the data looks like

	project_id	name	desc	goal	keywords	disable_communication	country	currency	deadline	state_changed_at	created_at	launched_at	backers_count	final_status
0	kkst1451568084	drawing for dollars	I like drawing pictures. and then i color them...	20.0000	drawing-for-dollars	False	US	USD	1241333999	1241334017	1240600507	1240602723	3	1
1	kkst1474482071	Sponsor Dereck Blackburn (Lostwars) Artist in ...	I, Dereck Blackburn will be taking upon an inc...	300.0000	sponsor-dereck-blackburn-lostwars-artist-in-re...	False	US	USD	1242429000	1242432018	1240960224	1240975592	2	0
2	kkst183622197	Mr. Squiggles	So I saw darkpony's successfully funded drawin...	30.0000	mr-squiggles	False	US	USD	1243027560	1243027818	1242163613	1242164398	0	0
3	kkst597742710	Help me write my second novel.	Do your part to help out starving artists and ...	500.0000	help-me-write-my-second-novel	False	US	USD	1243555740	1243556121	1240963795	1240966730	18	1
4	kkst1913131122	Support casting my sculpture in bronze	I'm nearing completion on a sculpture, current...	2000.0000	support-casting-my-sculpture-in-bronze	False	US	USD	1243769680	1243770317	1241177914	1241180541	1	0

Figure 1 Sample Data

Data Dimensions

Train data dimensions: (108129, 14)

Test data dimensions: (63465, 12)

Basic Data Info:

Train:

```
RangeIndex: 108129 entries, 0 to 108128
Data columns (total 14 columns):
project_id               108129 non-null object
name                     108126 non-null object
desc                     108120 non-null object
goal                     108129 non-null float64
keywords                 108129 non-null object
disable_communication    108129 non-null bool
country                  108129 non-null object
currency                 108129 non-null object
deadline                 108129 non-null int64
state_changed_at         108129 non-null int64
created_at               108129 non-null int64
launched_at              108129 non-null int64
backers_count            108129 non-null int64
final_status             108129 non-null int64
dtypes: bool(1), float64(1), int64(6), object(6)
memory usage: 10.8+ MB
```

Figure 2 Train Info

Test:

```
RangeIndex: 63465 entries, 0 to 63464
Data columns (total 12 columns):
project_id              63465 non-null object
name                    63465 non-null object
desc                    63461 non-null object
goal                    63465 non-null float64
keywords                63465 non-null object
disable_communication   63465 non-null bool
country                 63465 non-null object
currency                63465 non-null object
deadline                63465 non-null int64
state_changed_at        63465 non-null int64
created_at              63465 non-null int64
launched_at             63465 non-null int64
dtypes: bool(1), float64(1), int64(4), object(6)
memory usage: 5.4+ MB
```

Figure 3 Test Info

Stats Summary

Train:

	goal	deadline	state_changed_at	created_at	launched_at	backers_count	final_status
count	108129.0000	108129.0000	108129.0000	108129.0000	108129.0000	108129.0000	108129.0000
mean	36726.2288	1380248498.0049	1380152995.7698	1374036857.7694	1377299004.7093	123.5167	0.3196
std	971902.7052	42702221.2209	42664018.4447	42723097.6779	42944212.6260	1176.7452	0.4663
min	0.0100	1241333999.0000	1241334017.0000	1240335335.0000	1240602723.0000	0.0000	0.0000
25%	2000.0000	1346732388.0000	1346695335.0000	1340057670.0000	1343917387.0000	2.0000	0.0000
50%	5000.0000	1393628400.0000	1393567218.0000	1384444520.0000	1390870008.0000	17.0000	0.0000
75%	13000.0000	1415719201.0000	1415547766.0000	1409622898.0000	1412807179.0000	65.0000	1.0000
max	100000000.0000	1433096938.0000	1433096940.0000	1432325200.0000	1432658473.0000	219382.0000	1.0000

Figure 4 Train Stats Summary

Test:

	goal	deadline	state_changed_at	created_at	launched_at
count	63465.0000	63465.0000	63465.0000	63465.0000	63465.0000
mean	35323.7193	1459008907.9198	1458278292.7510	1451771216.3865	1456134610.4284
std	1206678.0607	16388498.2018	15712109.0084	19838269.1118	16419208.1925
min	1.0000	1433116800.0000	1428068650.0000	1266343242.0000	1427939719.0000
25%	2000.0000	1444521936.0000	1444418822.0000	1438097683.0000	1441755870.0000
50%	6000.0000	1458414817.0000	1458254471.0000	1452243288.0000	1455634804.0000
75%	20000.0000	1472586766.0000	1470671818.0000	1466366170.0000	1469649982.0000
max	100000000.0000	1490915735.0000	1490914927.0000	1490228268.0000	1490297448.0000

Figure 5 Test Stats Summary

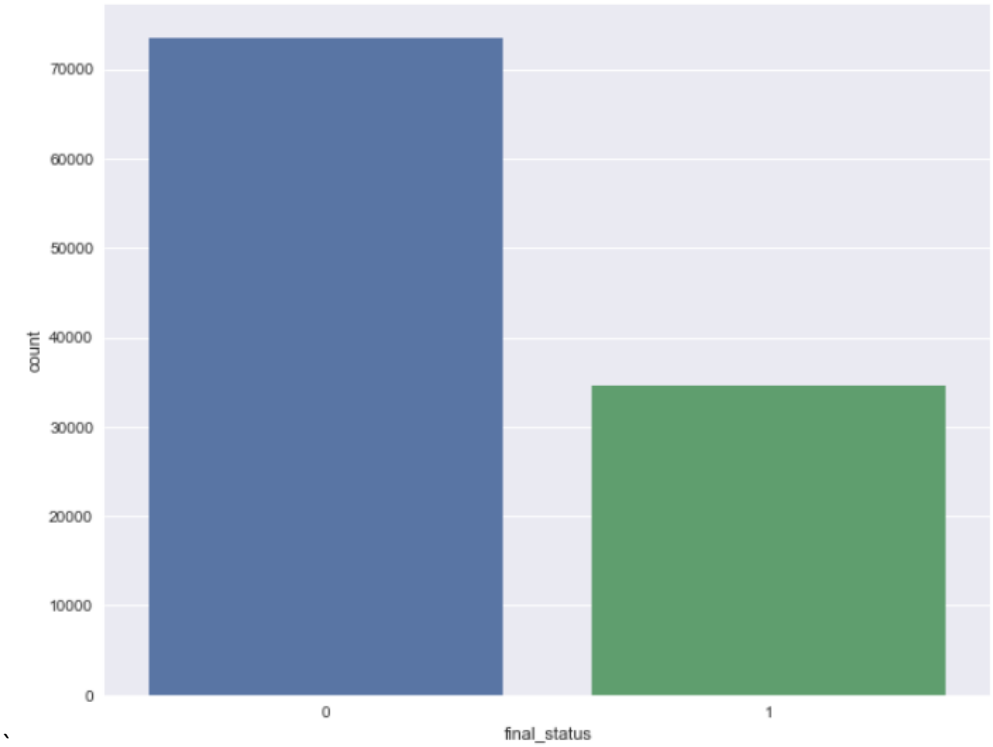
From the above data exploration process, we can note that:

- The **backers_count** column is missing in the test dataset, and hence cannot be used to train the model.
- Most of the columns have no null values except **Name** and **desc** columns, which is clear from the **info** about the dataset. This should be taken care by adding in empty spaces for the respective fields.
- The **goal** variable seems to have outliers as it has a very high std. deviation
- The percentiles of **goal** have a low values in comparison to std. deviation suggests that most of the values are nearer to the min value than the max, viz., the distribution is mostly positive skewed.
- A lower mean value in comparison to the max suggests the need to use power transform on the **goal** variable.
- There is definitely an imbalance in the target variable as there are more instance with 0 value than 1.
- The unix timestamp variable could be used as a features by themselves and they could also be converted to datetime format and we could engineer some useful features from the same.

Exploratory Visualization

Target Variable:

Let’s see if we have any imbalance in the target variable.



`

Figure 6 Target Variable Value Counts

There is an imbalance to some extent in the target variable distribution. The counts are as below:

final_status (0): 73568

final_status (1): 34561

Feature Correlation

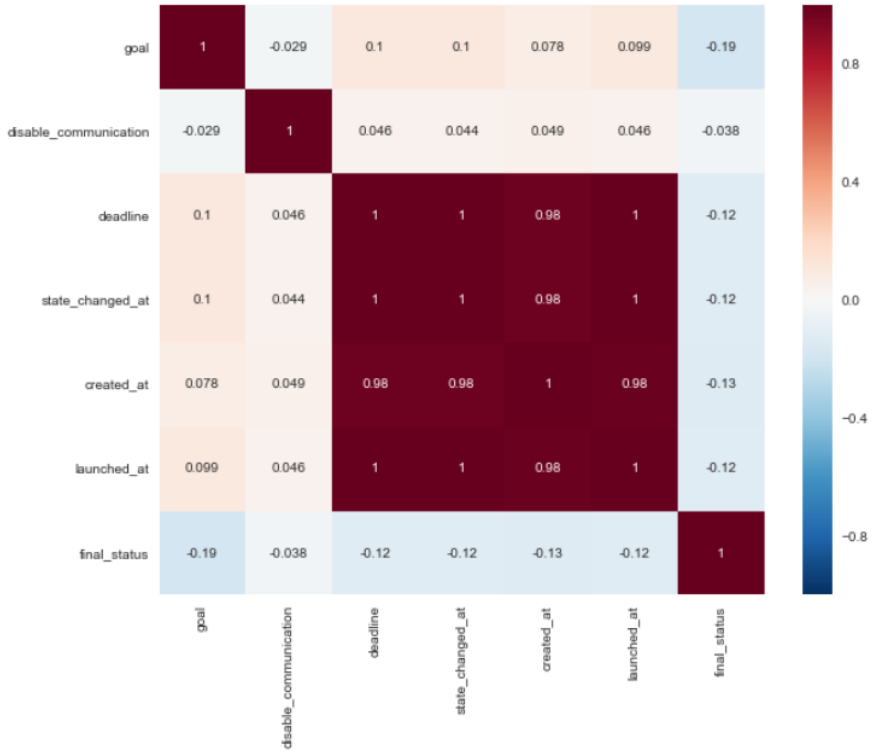


Figure 7 Correlation between variables

There seems to be a complete correlation between **deadline**, **state_changed_at** and **launched_at**, we could decide to drop two of the columns in case we end up with more features than the training sample size. But, in this case we have quite a large training dataset and hence we decide to keep the variables.

There aren't any heavily correlated features to the target variable and hence one thing is pretty clear that we cannot use a linear regression model here.

Float Variables

The standard deviation of variable **goal** is pretty high and seems obvious that there could be outliers.

Let’s look at its distribution:

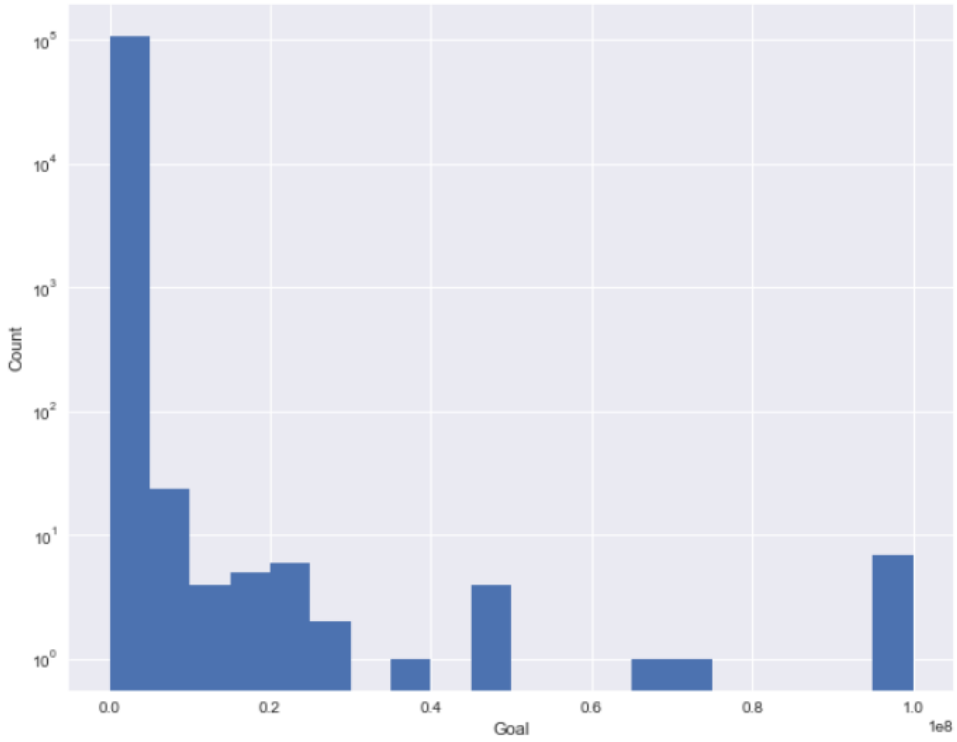


Figure 8 Goal Distribution

The outliers are prominent but there are a few of them. Removing them might not be the best possible option, let's log transform the data and see if that fixes the distribution to a more normal form.

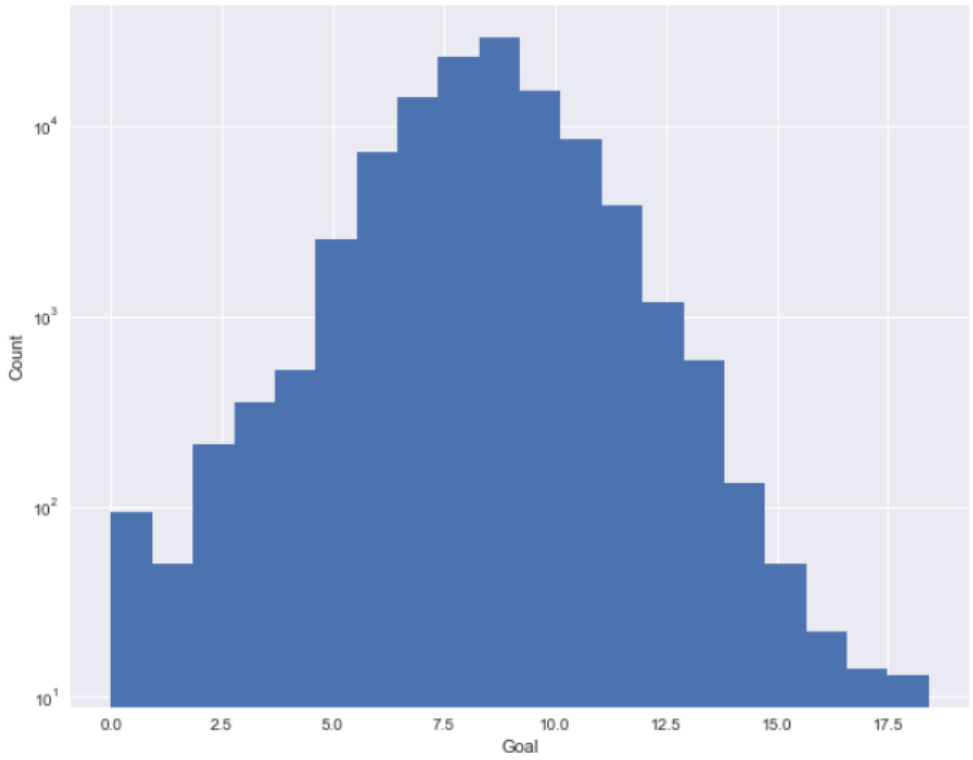
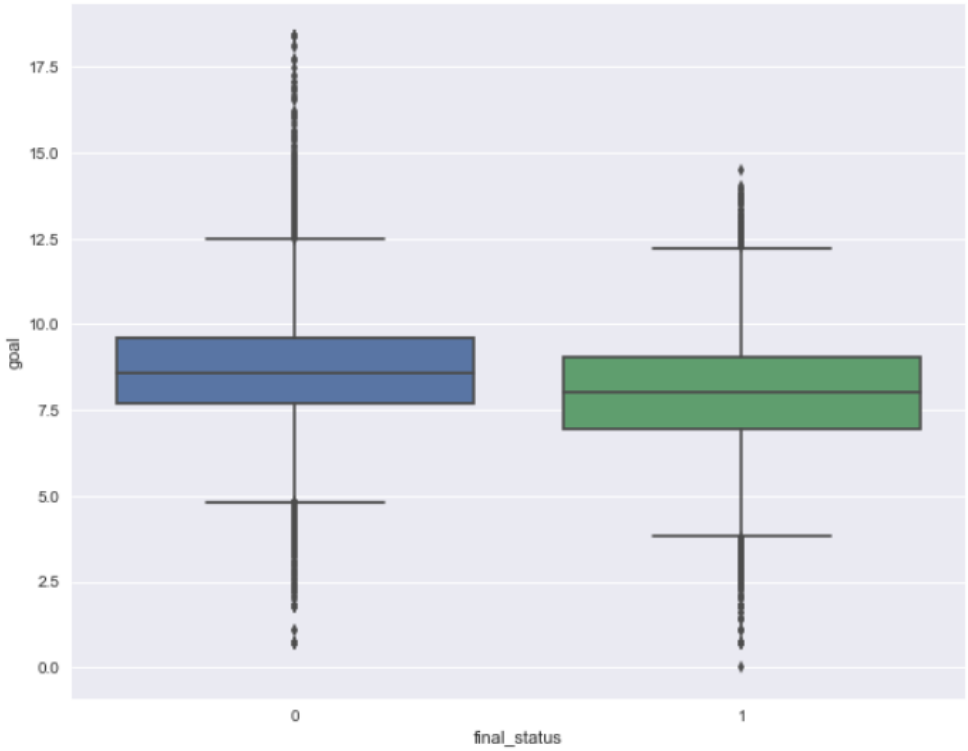


Figure 9 Goal Histogram Log Transformed

The outliers are taken care of after log transforming and now we have a more uniform distribution of the variable.

Below plot visualizes the relation between the **target variable** and **goal** feature. We can notice that most of the projects with high goals are unsuccessful. There are many outliers in the dataset, but since we have log transformed the goal variable, the effect should be minimal and we are not dropping any records due to this reason.



Categorical Variables

There are two categorical variables in the dataset, **country** and **currency**.

Let's visualize how the country countplot to see the different country data available.

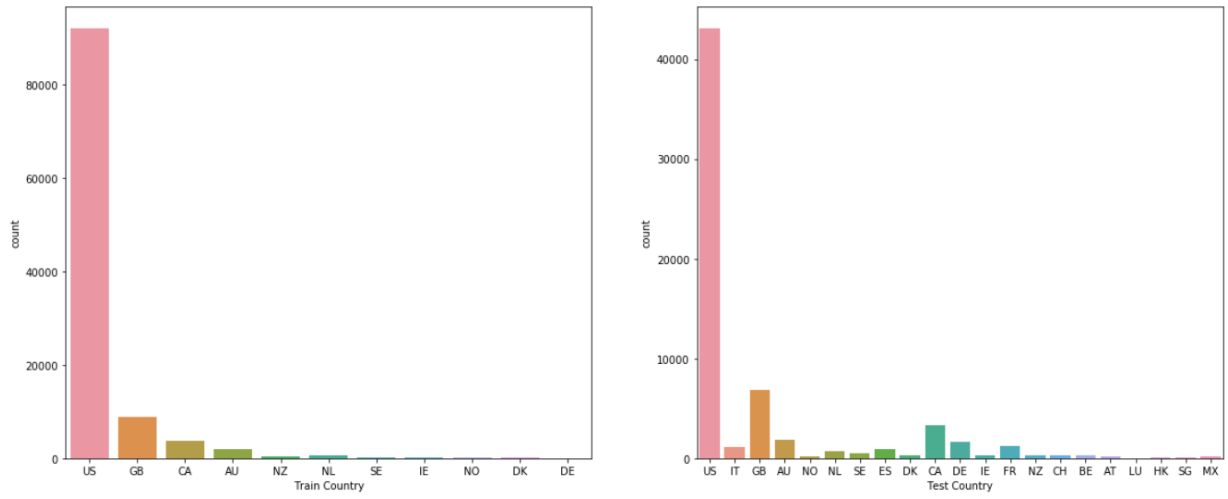


Figure 10 Country Countplot

There seems to be a lot of projects from **US** in both the **train** and **test** dataset. The test dataset also has more number of countries than the train. This would make it difficult for us to have any feature engineered with respect to the target variable.

Let us confirm the characteristics of **final_status** wrt **country** before coming to any judgement.

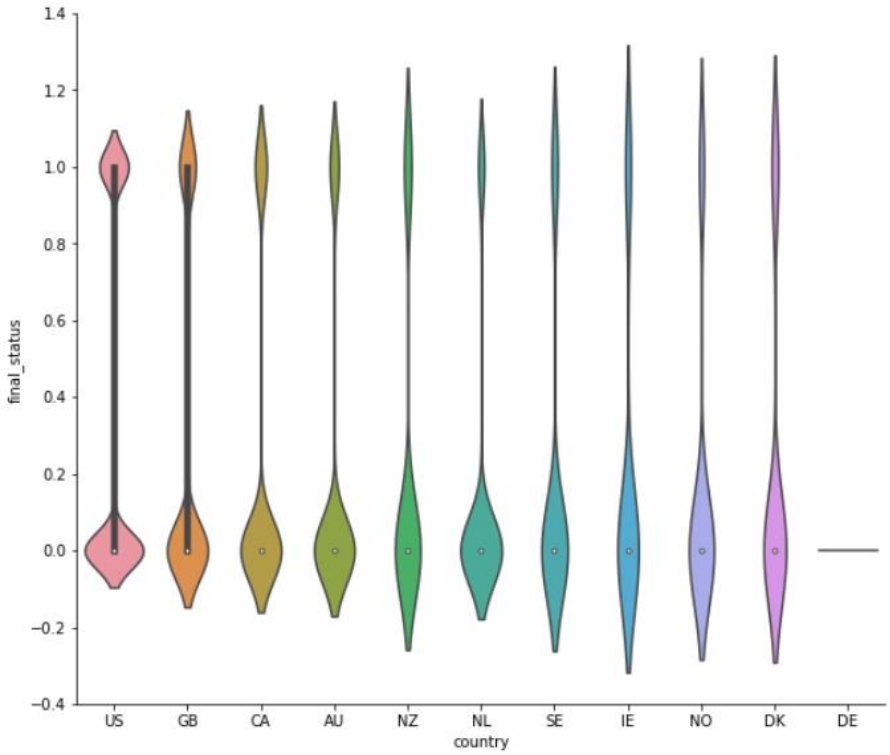


Figure 11 Country-Final_status FactorPlot

The distribution of target variable stays imbalanced regardless of the country the project is from. One more interesting thing to note is that the projects from DE are not successful at all.

Boolean Variables

There is one Boolean variable in the dataset, **disable_communication**. Let us see how is it distributed with respect to the **final_status** variable

disable_communication	**final_status**	
False	0	73245
	1	34561
True	0	323

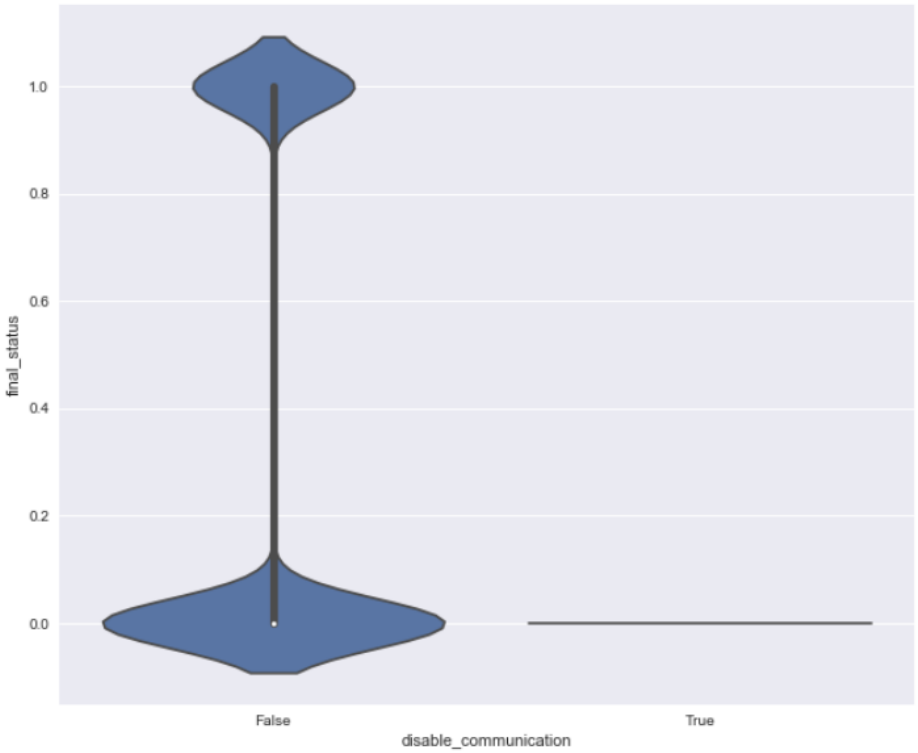


Figure 12 Disable_communication-Final_status FactorPlot

One thing we notice here is that, if the communication is disabled, the **final_status** is always unsuccessful.

Text Features

There are three text features, **name, desc** and **keywords**. Let's create a new text feature named all_text, which is basically a concatenation of all the text features.

We create two features from each of the text variables, number of characters and number of words in the text.

Basic Stats Summary

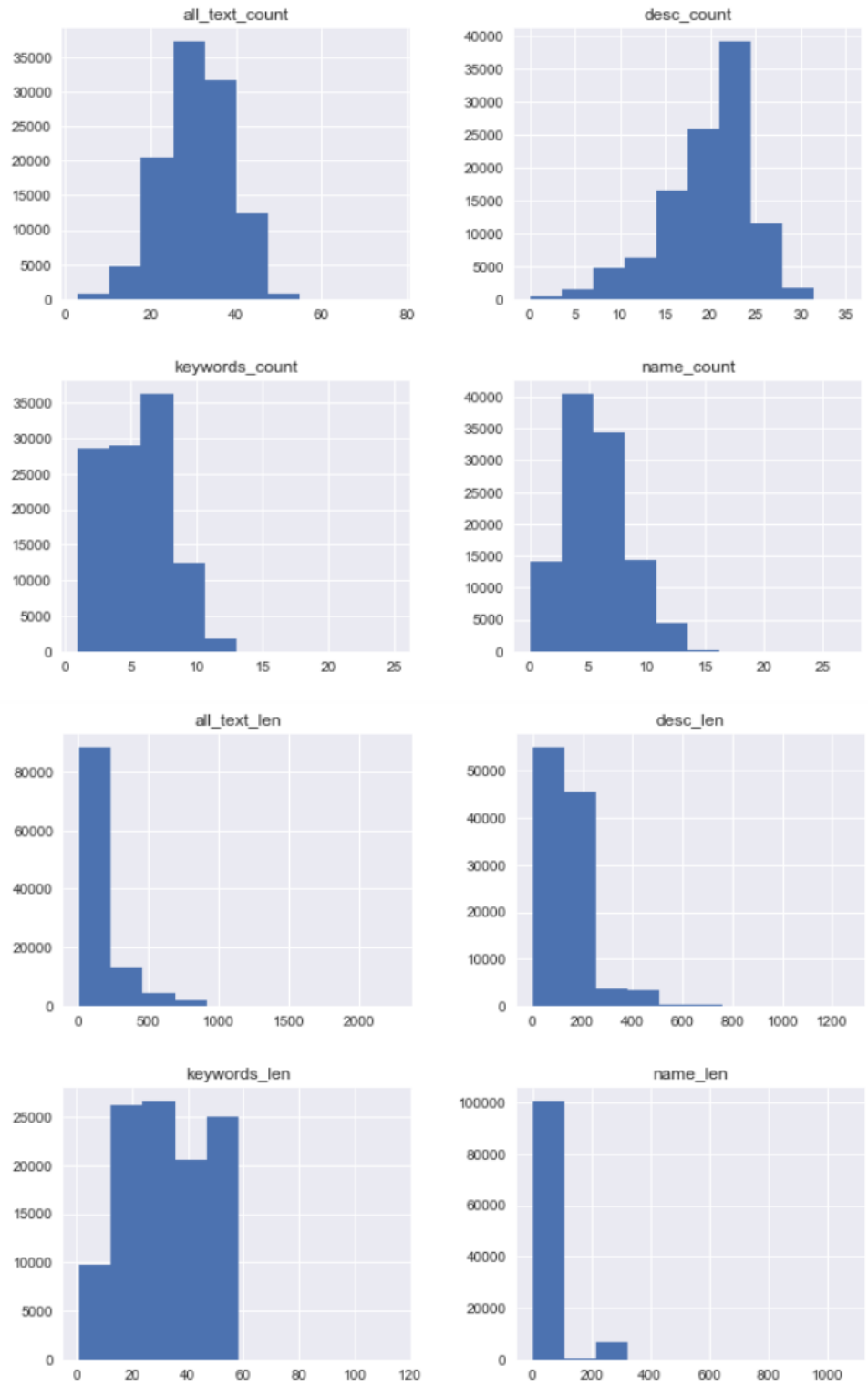
Train Text Word Count

	name_count	desc_count	keywords_count	all_text_count
count	108129.0000	108129.0000	108129.0000	108129.0000
mean	5.6886	19.5180	5.4261	30.6327
std	2.7666	4.8977	2.5192	7.6607
min	0.0000	0.0000	1.0000	3.0000
25%	3.0000	17.0000	3.0000	26.0000
50%	5.0000	20.0000	5.0000	31.0000
75%	8.0000	23.0000	7.0000	36.0000
max	27.0000	35.0000	25.0000	77.0000

Train Text Character Length

	name_len	desc_len	keywords_len	all_text_len
count	108129.0000	108129.0000	108129.0000	108129.0000
mean	51.6716	133.9949	31.6551	219.3217
std	68.7189	79.3774	13.7262	121.8186
min	0.0000	0.0000	1.0000	13.0000
25%	21.0000	106.0000	20.0000	159.0000
50%	35.0000	126.0000	31.0000	190.0000
75%	51.0000	133.0000	45.0000	229.0000
max	1076.0000	1268.0000	115.0000	2268.0000

Let’s check out the histogram of each of the above engineered features:



From the above histogram, it is clear that we need to log transform features **keywords_count, name_len, desc_len, keywords_len, all_text_len** to get rid of the skewness.

We'll use tfidf vectorizer and find out the most weighted words and add these as features.

Below are the top 10 words:

Desc

	feature	tfidf
0	women	6.7178
1	zombi	6.6864
2	graphic	6.6779
3	organ	6.6779
4	iphon	6.6728
5	photographi	6.6711
6	apparel	6.6643
7	real	6.6526
8	school	6.6444
9	danc	6.6427

Name

	feature	tfidf
0	stand	7.1367
1	sport	7.1233
2	beer	7.1233
3	dragon	7.1206
4	scienc	7.1101
5	word	7.1101
6	portrait	7.1049
7	meet	7.1049
8	green	7.1022
9	car	7.0971

Keywords

	feature	tfidf
0	car	6.3279
1	farm	6.3243
2	wood	6.2960
3	camera	6.2925
4	beer	6.2833
5	die	6.2810
6	deck	6.2775
7	hip hop	6.2684
8	press	6.2639
9	user	6.2628

There are some repeated words and perhaps using any of the three vectorized features would suffice to the model in case we want to limit the number of features used.

Algorithms and Techniques

I have chosen a Naïve predictor as a baseline model since we have an imbalance in the data, for instance, choosing all success or all failure would lead us to quite a good accuracy because of the imbalance, but the fscore would be crucial in this case as it accounts for more than just accuracy.

I'm using Random Forest algorithm as a benchmark model as it is commonly used in problems involving text features, since it can account to different text features and we would have included multiple trees in the RF algorithm. These individual trees would in turn extract information from the individual text fearures.

https://pdfs.semanticscholar.org/9b2f/84d85e5b6979bf375a2d4b15f7526597fc70.pdf

I will also use LightGBM since it is a powerful ensemble algorithm which is very fast. Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

I will use TFIDF vectorizer to get the most important words in the corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in

general. Nowadays, tf-idf is one of the most popular term-weighting schemes; 83% of text-based recommender systems in the domain of digital libraries use tf-idf.

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Benchmark

I've used Random Forest as a benchmark algorithm. The **accuracy** and **Fscore** for the same has been mentioned below:

Accuracy Score: 0.6957366133357995

Fscore: 0.45850066934404288

III. Methodology

Data Preprocessing

Below mentioned are the preprocessing steps followed in this project:

- I dropped the **backers_count** column from the train dataset as it is not available in the test dataset.
- To reduce the outliers and make the distribution of **goal** feature more normal, **log transform** it.
- Convert unix time columns, **deadline, state_changed_at, launched_at** and **created_at**, to datetime.
- Generate **early_campaign_duration** and **campaign_duration** using the below subtraction:
- Find cancelled kickstarter projects
- Create a feature goal/day based on **campaign_duration** calculated above and log transform the same.
- Generate **hour, weekday, date of month, hour-country, weekday-country** and **date of month-country** features from all the time features.
- Generate an **all_text** feature by concatenating all the text features (name, desc and keywords).
- Generate a feature for all text columns which describe their character length and number of words.
- Use TFIDF vectorizer to find the most important words from each of name, desc and keywords features. The below code represents this process. Here is the overview of what the code does:
  1. Clean the sentence to remove punctuations, digits and irregular tabs. Then convert to lower case.
  2. Combine the train and test data text fields for each of the text features and convert them to series of list.
  3. Split the words, apply snowball stemming, ignore words with less than three characters and join the words back to form a list.

4. Use tfidif vectorizer to find the most important words and also remove the stopwords while doing the same so that they don't feature in the most important words.
5. Combine these individual import words from each of name, desc and keywords into a single dataframe.

```
# this function cleans punctuations, digits and irregular tabs. Then converts the sentences to lower
def sentence_clean(word):
    p1 = re.sub(pattern='(\W+)|(\d+)|(\s+)',repl=' ',string=word)
    p1 = p1.lower()
    return p1

# creating a Series from train and test for text columns
kick_desc = pd.Series(train['desc'].tolist() + test['desc'].tolist()).astype(str)
kick_name = pd.Series(train['name'].tolist() + test['name'].tolist()).astype(str)
kick_keywords = pd.Series(train['keywords'].tolist() + test['keywords'].tolist()).astype(str)

# instantiate snowballstemmer
stemmer = SnowballStemmer(language='english')

# clean descriptions
kick_desc = kick_desc.map(sentence_clean)
# split words
kick_desc = [[x for x in x.split()] for x in kick_desc]
# apply stemming
kick_desc = [[stemmer.stem(x) for x in x] for x in kick_desc]
# ignore words with less than Three characters
kick_desc = [[x for x in x if len(x) > 2] for x in kick_desc]
# join the words with space separation
kick_desc = [' '.join(x) for x in kick_desc]

# apply the above steps for name
kick_name = kick_name.map(sentence_clean)
kick_name = [[x for x in x.split() ] for x in kick_name]
kick_name = [[stemmer.stem(x) for x in x] for x in kick_name]
kick_name = [[x for x in x if len(x) > 2] for x in kick_name]
kick_name = [' '.join(x) for x in kick_name]

# apply the above steps for keywords
kick_keywords = kick_keywords.map(sentence_clean)
kick_keywords = [[x for x in x.split()] for x in kick_keywords]
kick_keywords = [[stemmer.stem(x) for x in x] for x in kick_keywords]
kick_keywords = [[x for x in x if len(x) > 2] for x in kick_keywords]
kick_keywords = [' '.join(x) for x in kick_keywords]

# instantiate TfidfVectorizer for each of name, desc and keywords
tfidf_desc = TfidfVectorizer(max_features=400, ngram_range=(1,4), stop_words='english')

tfidf_name = TfidfVectorizer(max_features=150, ngram_range=(1,2), stop_words='english')

tfidf_keywords = TfidfVectorizer(max_features=250, ngram_range=(1,3), stop_words='english')

# fit_transform to each of the above dataframes
all_desc = tfidf_desc.fit_transform(kick_desc).todense()
all_name = tfidf_name.fit_transform(kick_name).todense()
all_keywords = tfidf_keywords.fit_transform(kick_keywords).todense()
# delete kick_desc, kick_name, kick_keywords
```

```
del kick_desc, kick_name, kick_keywords

# convert to dataframes
all_desc_df = pd.DataFrame(all_desc)
all_name_df = pd.DataFrame(all_name)
all_keywords_df = pd.DataFrame(all_keywords)
# rename variables
all_desc_df.rename(columns= lambda x: 'variable_desc_'+ str(x), inplace=True)
all_name_df.rename(columns= lambda x: 'variable_name_'+ str(x), inplace=True)
all_keywords_df.rename(columns= lambda x: 'variable_keywords_'+ str(x), inplace=True)
del all_desc, all_name, all_keywords

# concat all the above Three dataframes and delete them once combined
combine = pd.concat([all_desc_df, all_name_df, all_keywords_df],axis=1)
del all_desc_df, all_name_df, all_keywords_df
```

- Combine these newly engineered features into test and train data.
- Label encode categorical variables, *currency,country, deadline_hour_country, deadline_weekday_country, deadline_day_country, created_at_hour_country, created_at_weekday_country, created_at_day_country, launched_at_hour_country, launched_at_weekday_country, launched_at_day_country, state_changed_at_hour_country, state_changed_at_weekday_country, state_changed_at_day_country*

Implementation

The goal of this implementation was to make the most of the data present in the dataset and build a robust model. I started with preprocessing the data as we would do with any project. There was a significant preprocessing of the data involved in this use case, as it had a combination of various types of data, text, bool, categorical and float.

As mentioned in the preprocessing step above, text processing was one of the main tasks that I did since there was no particular category to the projects and hence had to rely on extraction useful information from the text features so that our learner could make the most out the information it got. That was perhaps one of the challenging stuff that I encountered while working on this project.

Since I was using the TFIDF vectorizer, it meant that I would have a lot of features for my model, and this would easily lead to memory errors. And hence had to limit the number of features used to 800 text features.

In addition to these text features a lot of features were extracted from the date features, the initial date feature were a unix timestamp. I converted these into datetime format and then generated weekday, date of month and hour features for each of the date features. These were then grouped based on the country the project was submitted from.

```
# Generate Weekday, day, hour, Weekday-Country, day-country and hour-
country features from all the time features - train
time_cols = ['deadline', 'created_at', 'launched_at',  'state_changed_at']
```

```
for time in time_cols:
    weekday = []
    hour = []
    day = []
    for x in train.loc[:, time].tolist():
        weekday += [datetime.datetime.fromtimestamp(x).weekday()]
        hour += [datetime.datetime.fromtimestamp(x).hour]
        day += [datetime.datetime.fromtimestamp(x).day]
    train[time + '_' + 'weekday'] = weekday
    train[time + '_' + 'hour'] = hour
    train[time + '_' + 'day'] = day

for time in time_cols:
    train[time + '_' + 'hour_weekday'] = train[time + '_' + 'hour'].astype(str) + '_' + train[time + '_' + 'weekday'].astype(str)
    train[time + '_' + 'hour_country'] = train[time + '_' + 'hour'].astype(str) + '_' + train['country'].astype(str)
    train[time + '_' + 'weekday_country'] = train[time + '_' + 'weekday'].astype(str) + '_' + train['country'].astype(str)
    train[time + '_' + 'day_country'] = train[time + '_' + 'day'].astype(str) + '_' + train['country'].astype(str)
```

Below mentioned are the three kinds of predictors that I used:

Naïve Predictor

A naive predictor is where we could simply predict that all the projects get successful. The scores for the same are as below:

Accuracy score: 0.3196,

F-score: 0.3700

Benchmark Model

For the benchmark model, I used sklearn's Random Forest ensemble algorithm. Below mentioned are the steps followed to achieve this:

- Split the train data into X_train, y_train, X_test and y_test. I used a 80-20 split percentage
- Use the train data to help the model learn.
- Calculate the train/test accuracy and f1 score. Import sklearn's metrics methods for the same.
- To make the model more robust, I turned to hyperparameter tuning along with cross validation. I used sklearn's GridSearchCV for the same. I also used ShuffleSplit to shuffle and split the data for the cross validation process.
- Calculate the best results (best model, best scores) from the cross validation.

LightGBM Model

The **lightgbm** model uses the same steps as mentioned above except that the classifier used is from **lightgbm** module.

Refinement

The initial lightgbm model with default parameters scored an accuracy and f1 score more than that of the benchmark model. However, I used GridsearchCV to cross validate and tune the hyperparameters.

I tuned num_leaves, subsample, subsample_freq and colsample_bytree parameters of the lightgbm model. There was quite a good improvement in the fscore.

Below are the scores for both the unoptimized and optimized model.

Unoptimized Model

Accuracy Score: 0.72607047072967723

Fscore: 0.54351671678754643

Optimized Model

Accuracy Score: 0.72778137427170997

Fscore: 0.55309007232084162

IV. Results

Model Evaluation and Validation

We used three types of predictions in this project, the **Naïve** predictor, a benchmark **Random Forest** model and the model with highest accuracy and Fscore, the **lightGBM** model.

Both the Random forest and lightGBM models have been trained with 80% of the train data and 20% of them is used to validate the model and check how the models behave to unseen data.

The Random Forest and lightGBM models have been cross validated using 3 splits and the best models has been selected respectively after tuning the hyperparameters.

Let's look at the effect of the hyperparameter values

1. Random Forest model

	Accuracy Score	**Fscore**
Default parameter values	0.69157495607139552	0.42925026399155231
After hyperparameter tuning	0.6957366133357995	0.45850066934404288

2. LightGBM model

	Accuracy Score	**Fscore**
Default parameter values	0.72607047072967723	0.54351671678754643
After hyperparameter tuning	0.72778137427170997	0.55309007232084162

Let us see how these models perform in terms of the time taken to train and predict:

1. Random Forest

	train_time	**pred_time**
Default parameter values	11.58334469795227	7.591559648513794
After hyperparameter tuning	9.473100185394287	6.503999948501587

2. LightGBM

	train_time	**pred_time**
Default parameter values	17.370043516159058	8.898366212844849
After hyperparameter tuning	40.74390006065369	12.605600118637085

We notice from the tables above that the even though the lightGBM produces a higher accuracy and fscore, it is relatively slow when compared to the benchmark Random Forest model.

The higher training time is because of the fact that we are using a relatively higher number of estimators (100) while the benchmark Random Forest model uses 10 estimators. Also, the number of leaves has been set to a high value of 80.

Now, to explain the robustness of the final model or how well the model generalizes to unseen data, let me consider the train accuracy and test accuracy. I used an 80-20 train-test split to train the model and validate it. Below are the results of the same:

	Accuracy	**Fscore**
Train	0.78820387732217378	0. 68576782422099303
Test	0.72778137427170997	0.55309007232084162

As we can see the model is quite robust and generalizes quite well even though ensemble models are generally prone to overfitting.

Justification

I first developed the benchmark model which used Random Forest algorithm. This model was then improved by tuning the hyperparameters which produced a significant improvement in the accuracy score and fscore for test set.

The final model which is the lightGBM model produced significant improvement in the accuracy score and fscore even with the default parameters. These scores were further improved by tuning the hyperparameters.

The final comparison of the scores are as tabulated below

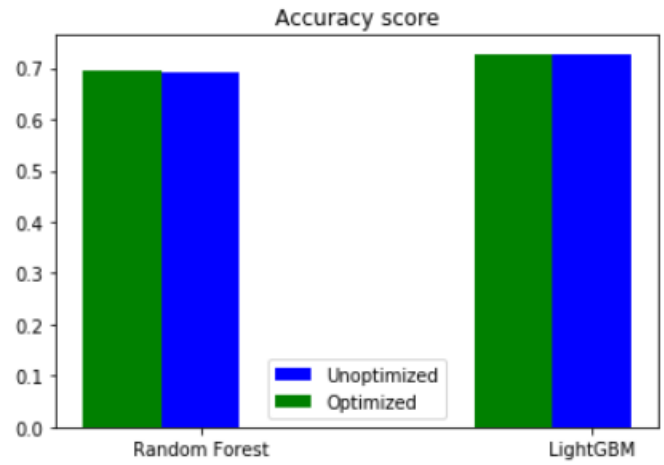
	Naïve	Random Forest	LightGBM
Accuracy score	0.3196	0.6957	0.72778
F score	0.3700	0.4585	0.55309

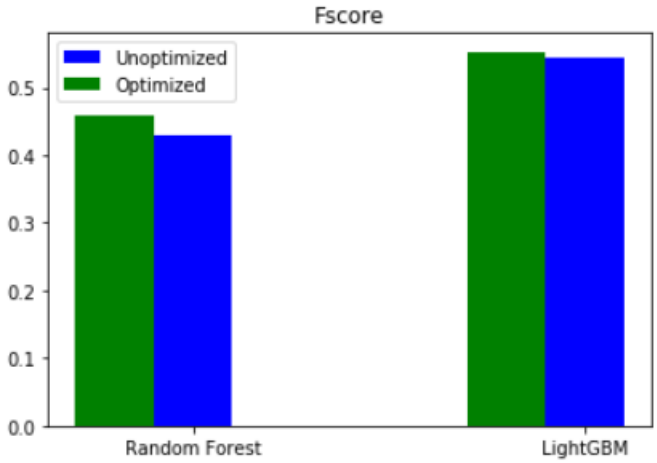
Based on the scores of benchmark model and the final model, it is quite clear that the lightGBM model outperformed the benchmark model by quite a margin both in terms of Accuracy score and Fscore.

V. Conclusion

Free-Form Visualization

As we can see in the below bar graphs, the accuracy scores of the final model is higher than that of the benchmark model and so is the case with Fscore.





Reflection

With the very limited features that we have in hand to predict the success or failure of a kickstarter campaign, it was not so easy a task. As there were no history of a users' project success or the types of project that are usually preferred by funders, there was no single feature which could directly relate to the result of the project.

The text features to an extent increased the accuracy of the model as they compensated to the lack of 'type of project' feature. Though the text features were used in limited number due to memory restrictions, it did increase the accuracy and fscore by quite a margin.

The model predicts if the kickstarter project will get funded or not correctly for about 72.7% of the time, which given the data is not too bad, but it cannot really be used as an actual predictor to let users know about how their project would do before hand to help them improve on their campaign. Maybe if we have more info about the **categories, backers_count** perhaps this would significantly increase the accuracy and could as well be used for official purposes.

This project gave me a hands on some aspects of natural language processing, particularly, how to work with text data. I also learned the concepts of TFIDF. One more thing that I learned is the importance of feature engineering. How do I efficiently use the data at hand and generate features from the same, which is perhaps the most important aspect of applied machine learning.

Improvement

The prediction could definitely be improved and by no means is the best result that is possible. Below are the list of improvements I think could be made to have a better Kickstarter prediction:

1. Better quality of dataset with more features like category of project and backers_count for both test and train data.
2. We could use more number of important words obtained from the TFIDF vectorizer.
3. We could model stacking to have a better scores so that the strengths of different algorithms could be used for the final prediction.

These are some of the improvements that I think could be made to make sure that the model has a good accuracy and f1 score so that it can be used in real time projects.