Predicting Funding Outcome of Projects on Kickstarter.com

Daocun Yang

Autumn 2020

Introduction

In this project, I aim to build a machine learning model to predict if a project listed on kickstarter.com (Kickstarter) can achieve its fund-raising goal. I'm modeling the problem as a binary classification problem, in which the outcome of the prediction will be either "Success" (1) or "Failure" (0).

A quick introduction to Kickstarter: Kickstarter is the leading crowd-funding platform for creative projects, which includes projects ranging from film, games, and music to art, design, and technology [1]. As of December 2019, Kickstarter has received more than $4.6 billion in pledges from 17.2 million backers to fund 445,000 projects [2].

The project will be meaningful in that it can be a helpful tool for multiple groups of people. First of all, it can give potential project creators an idea of the likeliness of success if they were to fund their projects on Kickstarter or similar crowd-funding platforms. In addition, it can also provide some insights to business students or people who study crowd-funding on how the various factors of a project can contribute to the outcome of crowd-funding.

Related Works

Plenty of literature exists for solving supervised learning and binary classification problems. I consulted the textbook *The Elements of Statistical Learning* by Hastie et al [3], and the course notes of CS229 by professor Andrew Ng [4]. The book *Speech and Language Processing* by Jurafsky and Martin [5] also has a great chapter on logistic regression. Regarding implementation using scikit-learn, the book *Python Machine Learning* [6] and *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow* [7] both contain some information which I found helpful. I also consulted Léon Bottou's paper "Stochastic Gradient Descent Tricks" [8] to better understand the learning rate used by SGDClassifier in scikit-learn.

Dataset

Description of Dataset

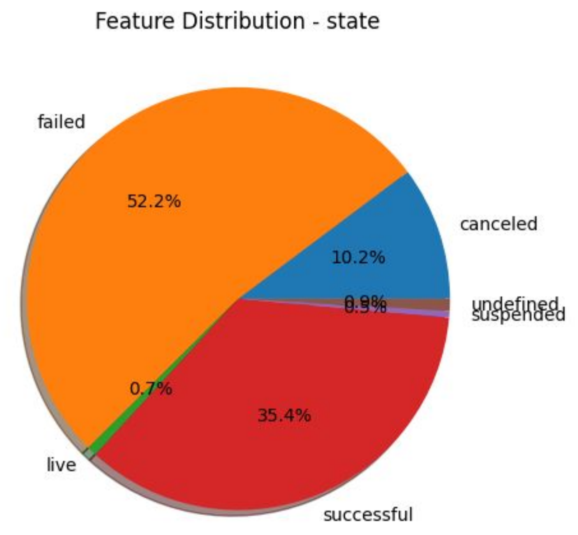
I chose to use the "Kickstarter Projects" [9] dataset found on Kaggle, in specific the data contained in the file "ks-projects-201801.csv". This is a fairly large dataset, which contains a total of 378661 rows, where each row represents a project.

A row of the original input data contains 15 columns: ID, name, category, main_category, currency, deadline, goal, launched, pledged, state, backers, country, usd pledged, usd_pledged_real, usd_goal_real. An example row of data is shown next:

1000064918,The Beard,Comic Books,Comics,USD,2014-11-08,1500.00,2014-10-09 22:27:52,395.00,failed,16,US,395.00,395.00,1500.00

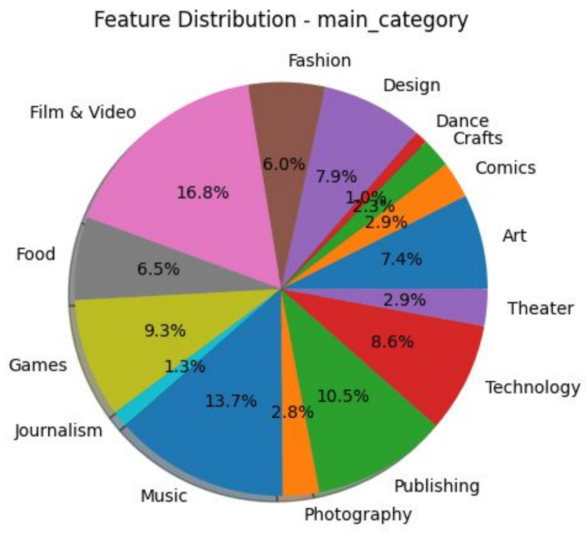
In particular, the “state” column contains the funding outcome of the project. In the above example, it is “failed”. Other possible states include “successful”, “canceled”, “live” and “undefined”. For simplicity, I decided to only focus on the ones that are “successful” or “failed”.

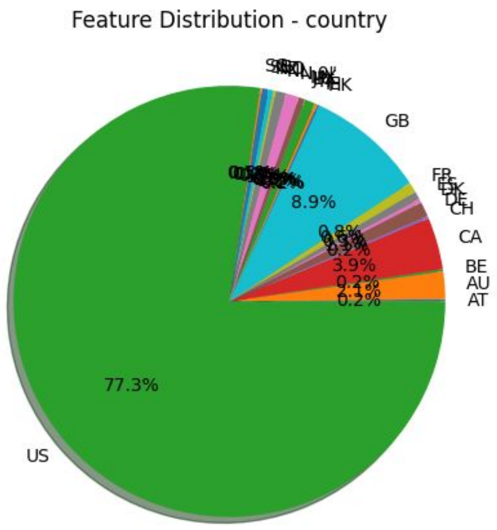
Below is a pie chart that shows the feature distribution of the “state” column:



For model training, I chose to include the following existing features: main_category, deadline, launched, state, backers, country, usd_goal_real. “main_category” is a categorical feature which has 15 distinct categories (e.g. “Technology”, “Film & Video”, “Publishing” etc.). “launched” and “deadline” correspond to the start and end date of the funding period, respectively. “backers” and “usd_goal_real” are both numerical features, which represents the number of backers and the amount in US dollars that the project aims to raise. “Country” is a categorical feature which has 22 distinct values, representing the origin country of the project.

The next two following graphs show the feature distribution of “main_category” and “country” columns:





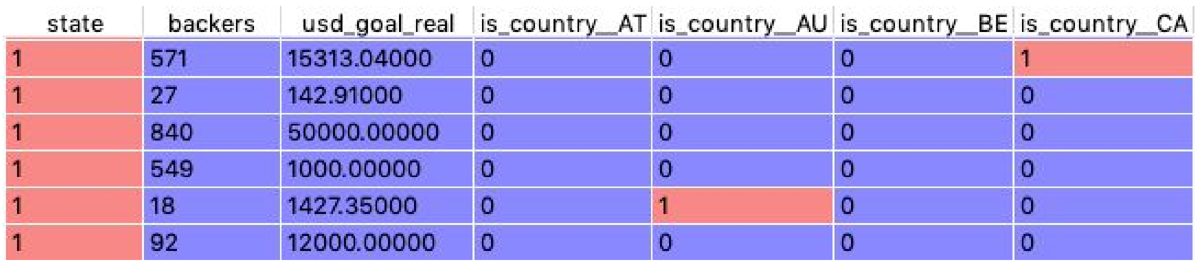
Preprocessing
The pre-processing of the dataset contains five steps in total.

First of all, I dropped the features that are irrelevant and redundant. For example, the ID of a project doesn't contain much useful information and can be dropped. Columns such as "goal" and "pledged" are redundant with respect to "usd_goal_real" and "usd_pledged", since the former just represents the same amount in a different currency (e.g. Canadian Dollars, etc), and can be dropped.

Secondly, since I'm only interested in projects whose "state" column is either "successful" or "failed", I dropped the rows in the datasets where the "state" value is neither "successful" nor "failed" (e.g. "undefined", "canceled", etc.). Having further examined the data, I also found the “country” feature of a small subset of rows (i.e. approximately 4,000) has a value of “N,0", which appears to be malformed; these rows were also excluded from subsequent modeling and testing.

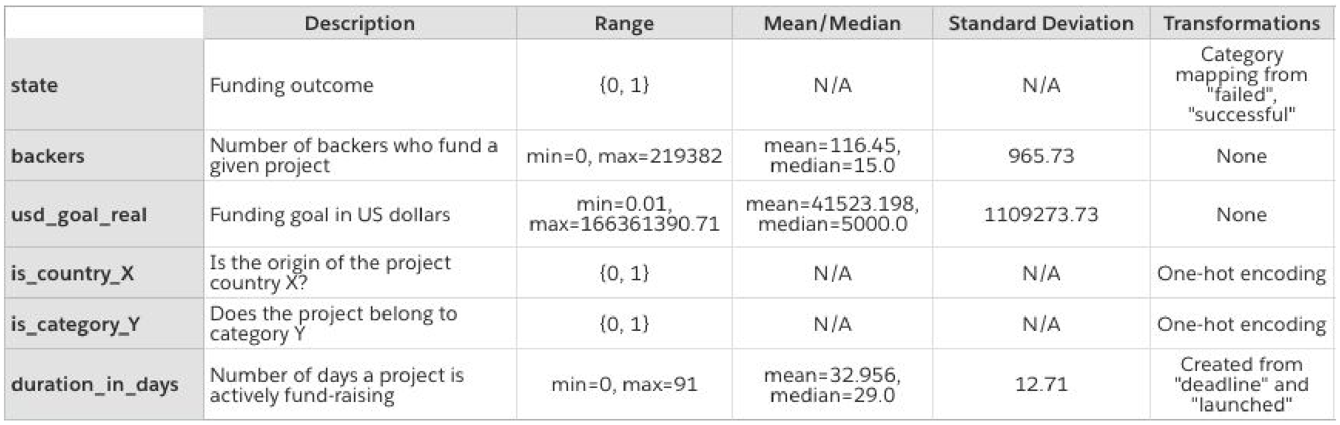
As the next step, I decided to create a new integer feature, “duration_in_days”, based on the difference in days between the two dates found in “deadline” and “launched”. This tells us for how many days a project is active on Kickstarter, which I believe can be useful during model training and prediction — will a project raise more money if its fund-raising period lasts longer? After creating this new feature, I dropped both “deadline” and “launched” columns.

I then went on to encode categorical features, and transformed the features “main_category”, “country”, and “state” from string representations into integer features. I encoded “successful” to 1, and “failed” to 0 for “state”. As for category and country, initially I simply used ordinal encoding for both features. For example, “US” was encoded as 0, “CA” as 1, etc.. But later I realized that doing so can impose an ordinal relationship where no such relationship may exist. Therefore I further applied one-hot encoding to “main_category” and “country”, and created features such as “is_country_US”, “is_country_CA”, “is_category_Technology”, etc.. There are 15 distinct values for “main_category”, and 22 for “country”. These 37 columns, combined with “state”, “backers”, “usd_goal_real”, and “duration_in_days” lead to a total number of 41 features.



state	backers	usd_goal_real	is_country__AT	is_country__AU	is_country__BE	is_country__CA
1	571	15313.04000	0	0	0	1
1	27	142.91000	0	0	0	0
1	840	50000.00000	0	0	0	0
1	549	1000.00000	0	0	0	0
1	18	1427.35000	0	1	0	0
1	92	12000.00000	0	0	0	0

Example rows of processed data (only showing a subset of all columns)



	Description	Range	Mean/Median	Standard Deviation	Transformations
state	Funding outcome	{0, 1}	N/A	N/A	Category mapping from "failed", "successful"
backers	Number of backers who fund a given project	min=0, max=219382	mean=116.45, median=15.0	965.73	None
usd_goal_real	Funding goal in US dollars	min=0.01, max=166361390.71	mean=41523.198, median=5000.0	1109273.73	None
is_country_X	Is the origin of the project country X?	{0, 1}	N/A	N/A	One-hot encoding
is_category_Y	Does the project belong to category Y	{0, 1}	N/A	N/A	One-hot encoding
duration_in_days	Number of days a project is actively fund-raising	min=0, max=91	mean=32.956, median=29.0	12.71	Created from "deadline" and "launched"

Descriptions of features in processed dataset

The last step is to shuffle the data, and split the data into training and test set. I decided to use 70% (232026 rows) of the remaining data as training data, and 30% (99439 rows) for testing.

After processing, the size of the dataset was reduced from 378661 to 331465.

Evaluation Metric

Since this is a supervised learning problem in which the correct labels are given, I chose to use accuracy and F1 score as the evaluation metrics.

Let TP denote the number of true positives, TN the number of true negatives, FP false positives, and FN false negatives. Accuracy is then calculated by:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total size of data}}$$

For F1 score, it's given by:

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Baseline

I implemented the baseline model with two approaches. The first one is to simply predict the mode or majority label. The mode of the "state" column for both training and test set is 0 (i.e. "failed"). For this baseline model, the accuracy is simply the percentage of failed projects among all projects in the test set, which is 59.9%.

The other approach is to build a baseline model using each of the features of "backers", "usd_goal_real", and "duration_in_days". I'm curious about which one of these features has the most correlation with the success or failure of the funding outcome. Using logistic regression, I trained the model on each of the features separately, resulting in three models. I discovered the one feature that yields the best prediction is "backers", the number of people who fund the project. The accuracy is 0.795, and the F1 score is 0.845. Which is pretty good for a simple model.

Approach

To tackle this supervised machine learning problem. I explored numerous algorithms available in scikit-learn. Some of the main algorithms that I tried applying are logistic regression with stochastic gradient descent (SGD), k-Nearest Neighbor (kNN), decision tree and random forest.

Logistic Regression

Since the problem I'm facing is a binary classification problem, one natural approach that comes to mind is logistic regression, which is intended for binary classification. For logistic regression, the formula of the hypotheses is given by:

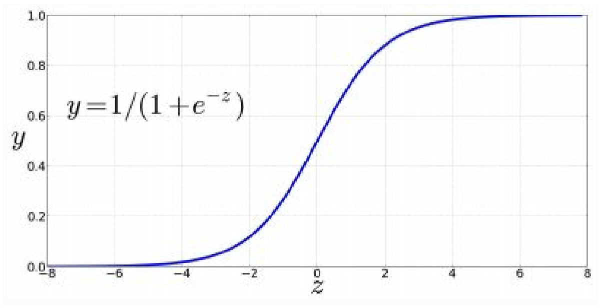
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where θ are the weights, and

$$g(z) = \frac{1}{1 + e^{-z}}$$

In the context of this specific binary classification problem, each x is a row vector of 41 columns. θ is a one-dimensional weight vector where each weight corresponds to a feature in x.

The second formula g(z) shown above is the sigmoid function, which “squashes” the input and maps real-valued input to the range of 0 and 1. This characteristic makes logistic regression a better choice than linear regression, since the latter can output a value that’s greater than 1 or smaller than 0.



The sigmoid function [10]

The cost function of logistic regression is given by:

$$\mathrm{Cost}(h_\theta(x), y) = -y\ \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

which expresses, for an observation x, how close the classifier output is to the correct output y (which is 0 or 1).

Having chosen logistic regression as the model, the next thing to be determined is the optimization algorithm, from which we can find the optimal weights. The algorithm is stochastic gradient descent (SGD), which minimizes the cost function by computing its gradient after each training example. Given the size of my training data, it’s important to use the stochastic flavor of gradient descent algorithm; otherwise it may take a long time for training to be complete. The general formula of gradient descent is given by:

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x;w)$$

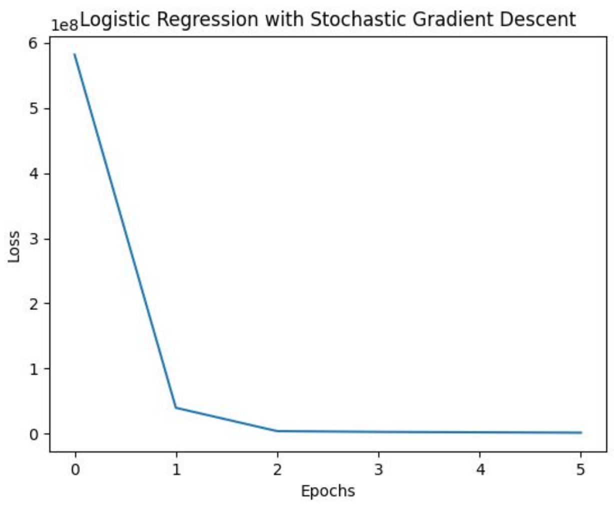
Where the learning rate η determines how much we should move the weights w in each step.

For implementation, I used the SGDClassifier class available in scikit-learn. SGD Classifier uses the “optimal” learning rate by default, which is given by the following formula:

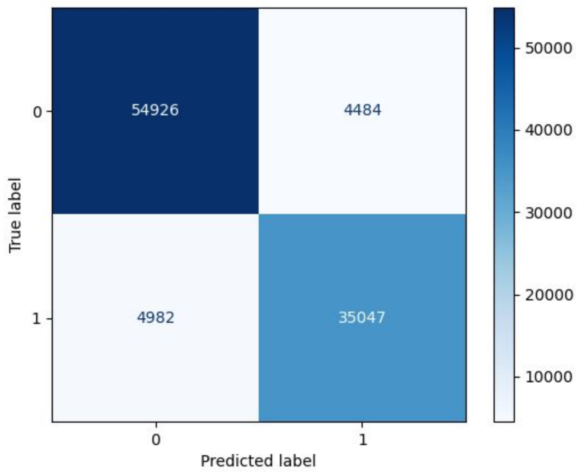
$$\eta^{(t)} = \frac{1}{\alpha(t_0 + t)}$$

where t is the time step, alpha = 0.0001 by default, and t0 is chosen by a heuristic proposed by Leon Bottou.

With SGD, convergence is observed within 8 epochs on the training data. I also noticed the loss decreases dramatically even after the initial first epoch, as shown in the following plot:



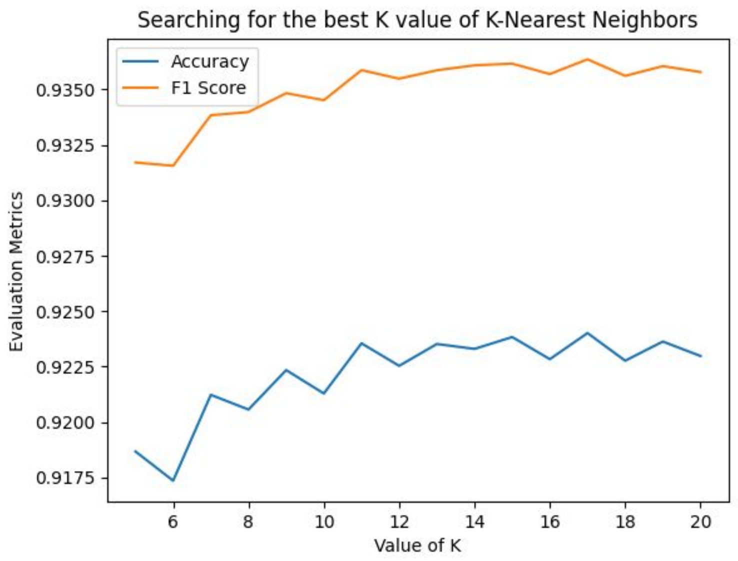
The accuracy is around 0.90, and the F1 score is 0.92. Below is the confusion matrix of a sample run of the logistic regression predictor.



k-Nearest Neighbors

The next algorithm I tried is k-Nearest Neighbors (kNN). Unlike logistic regression, kNN doesn't require learning and optimizing the weights. In kNN, we go through each point in the test data, and find the k data points in the training data that are the closest to it (where the distance is calculated based on a predetermined metric). We then predict the majority label of the k nearest neighbors for the test data point.

As suggested by the name of the algorithm, an important task when applying kNN is to find the right k that yields good performance. With the help of GridSearchCV in the scikit-learn framework, I did a grid search with several different values of k, and found k = 17 to be a local optima.



In addition to k, I also grid-searched for the best distance metric. The result is Bray Curtis Distance, which is given by the following formula:

$$\frac{\sum_{i=1}^{n} |x_i - y_i|}{\sum_{i=1}^{n} (x_i + y_i)}$$

With kNN, the accuracy on training data has improved to around 0.924, with a F1 score of 0.936.

Decision Trees
The next class of algorithms I explored is tree-based algorithms, the simplest one being decision trees. With decision trees, the idea is to recursively partition the data into two parts: the NO part and the YES part. The idea here is that we want the results after splitting to be as unambiguous or pure as possible; the purer it is, the more confident we are in assigning a class label to it. We partition until some predetermined stopping criterion is met. For example, the algorithm halts when the tree grows to a certain depth, or when the impurity decrease is no longer significant.

From grid search, I determined the optimal max depth of the tree to be 11. I also compared the performance of using gini impurity and entropy, respectively, as the splitting measurement, and found entropy to be the better one. To calculate entropy, we use the following formula:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Where pi is the probability of the i-th class. A small entropy value is better than a large one.

With decision trees, the accuracy on training data has improved to around 0.929, with a F1 score of 0.940.

Random Forest

Another class of algorithms that I explored is ensemble algorithms. The idea with ensemble algorithms is that it can be easy for one classifier to overfit or err, but if we combine multiple classifiers into a meta-classifier, we are more likely to get better generalization performance than any single classifier alone.

Random forest is an ensemble algorithm in that it combines multiple decision trees. Each decision tree in the ensemble is built from a sample drawn with replacement from the training set. Unlike decision trees in which we look at all features when partitioning, in random forest we only look at a subset of all features, and choose the best one to split on among this subset. This characteristic adds randomness; without the randomness, the different trees in the forest will tend to use the same features that perform well. The intuition here is to have diverse decision trees in the forest and minimize correlations between the trees, so that each tree adds some unique value to the ensemble.

From running grid search on the number of decision trees in the forest, max depth of the trees, and splitting criteria, I discovered the best performance is achieved with 70 decision trees in the forest, with max depth of each tree being 20, and splitting criteria entropy.

With random forest, the accuracy on training data has improved slightly to around 0.935, with a F1 score of 0.945.

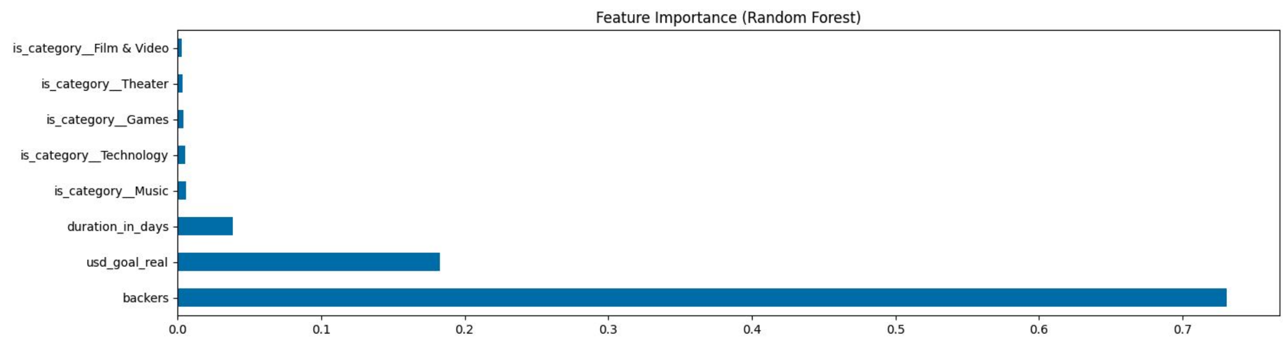
Other Approaches

In addition to the several methods discussed above, I also briefly explored a few other methods using classifiers available in scikit-learn. For example, HistGradientBoostingClassifier for gradient boosting, MLPClassifier for neural network, etc.. Given my rather superficial understanding with these methods, I wasn't able to employ these methods effectively to see significant improvement in performance.

Results & Analysis

Overall, the best accuracy and F1 scores are obtained on the testing data when random forest is employed. The result is an accuracy of approximately 0.94, and F1 score 0.95.

From looking at the learned weights in decision trees and random forest, I found the feature that matters the most for predicting the funding outcome to be the number of backers, followed by the amount of funding the project tries to raise. The results make sense, since one may think the more backers a project has, the more popular it is, and thus it's more likely for the project to get the support it needs to be successful. This result also corroborates with the best-performing baseline model, where the only feature is backers.



Future Work

To further improve the performance of the classifier, the first thing to try is to use different features. For example, the dataset contains a “name” feature which corresponds to the name of a project, which is currently not used. In the future, I can try to apply word2vec or other techniques in natural language processing on this feature and find out if correlations exist between certain words in project names and the funding outcome. Another feature to include can be the year in which a project is launched.

Looking at the two categorical features “main_category” and “country”, from which a total of 37 columns were created as the results of one-hot encoding, I’m considering aggregating the values and reducing the number of features. For example, it might be a good idea to group values such as “Art”, “Film & Video”, “Design”, “Fashion” into a single sub-category, since they are all somewhat related to art. Similarly, countries can be grouped by continents. By reducing the number of features, we reduce the size of data, which can in turn help reduce training time.

As I learn more about machine learning, I will deepen my understanding of the various models I explored in this project. With more knowledge and experience, I will be able to employ these models more effectively, and have clearer ideas on what to look for regarding grid search and hyper-parameter tuning.

Link to Github repo: https://github.com/daocunyang/Kickstarter-Kaggle

References

[1] *Kickstarter and Taxes.* Kickstarter. https://www.kickstarter.com/help/taxes

[2] Kickstarter (2020, November 3). In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Kickstarter

[3] Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009

[4] Andrew Ng, Tengyu Ma. *CS229 Lecture Notes.* http://cs229.stanford.edu/notes2020spring/cs229-notes1.pdf

[5] Daniel Jurafsky, James H. Martin. *Speech and Language Processing, 3rd edition draft.* 2019 https://web.stanford.edu/~jurafsky/slp3/5.pdf

[6] Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning, 3rd edition*. 2019

[7] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition.* 2019

[8] Léon Bottou. Stochastic Gradient Descent Tricks. 2012.

[9] *Kickstarter Projects.* Kaggle. https://www.kaggle.com/kemical/kickstarter-projects

[10] From “Speech and Language Processing, 3rd edition draft” by Daniel Jurafsky, James H. Martin. Chapter 5, page 3. 2019
https://web.stanford.edu/~jurafsky/slp3/5.pdf