Predicting Kickstarter Project Success

Kevin Robb (4033), Logan Elliott (5033)
Short Supervised Learning Project
(Dated: December 14, 2020)

1. INTRODUCTION

Kickstarter.com is a website on which individuals or small groups can propose a product idea and receive crowdfunding support to fund production and manufacturing. Creators post different pledge tiers and goals, and only receive the money if the fundraising goal is achieved. Where traditionally a budding entrepreneur would need to propose their concept to a bank official and receive a loan, Kickstarter allows anyone to show their idea to the public at large, and gain direct support. When people pledge their money to a campaign, they are expressing their interest in the product or experience, as well as their confidence in the creators to produce and deliver. Predicting a venture's outcome can help potential donors to evaluate the chance of success, and can help creators themselves model their campaign in a way that predicts the best outcome.

We propose in this project to create an intelligent algorithm which can predict the results of a Kickstarter project using readily available data including the project title, date, fundraising window, and category. We hope to achieve at least 60% accuracy with our models, and will be comparing them to a human's guesses, a random agent, and the more optimal scikit-learn implementations.

2. DATA PRE-PROCESSING

Our data is the Kickstarter Projects dataset available on Kaggle, which contains over 300,000 examples [2]. We wrote R code to pre-process this data into a form that is more suitable for our purposes.

The base dataset has many different types, including Numeric (*ID*, *goal*, *pledged*, *backers*, *usd_pledged*, *usd_pledged_real*, *usd_goal_real*), String (*name*, *category*, *main_category*, *currency*, *country*), Datetime (*deadline*, *launched*), and lastly the label (*state*).

We first transform *state* into a numeric variable to make our comparisons simpler:

$$target = \begin{cases} 0 & \text{"failed"} \\ 1 & \text{"successful"} \end{cases} \quad (1)$$

Some examples have a non-binary *state*, such as campaigns that are still in progress or were cancelled. We throw all these out, and use only states with explicit success or failure.

We want the agents to have access only to information that would be available before the campaign starts, so we throw out *pledged*, *backers*, *usd_pledged*, and *usd_pledged_real*. It would defeat the purpose of training a predictive model if it knows how much has been raised in pledges. We also throw out the *ID* since it should not be used for decisions.

Notice this has removed most of the numeric variables, and our learning methods will have very little to work with if we aren't smart; thus, we extract numeric information from as many of the other attributes as we can in order to maximize the amount of information our agents can use. This is the novelty in our approach.

We first look at the campaign's title, *name*. From this we can extract *title_length*, *title_punc*, and *title_caps_ratio*, which store numerically the length, number of punctuation marks, and ratio of letters that are uppercase. We can then throw out *name*, while still having some relevant information from it that can be characterized and realistically used.

We next look at the datetime variables, *deadline* and *launched*. We transform *launched* into *launched_epoch*, which is an integer representing the launch date as seconds from the epoch. We then do the same with *deadline*, and subtract the two epoch times to get *open_epoch*, representing the number of seconds for which the campaign was live.

Our next focus is the *currency* and *country*. We don't expect these variables to be important in the decision-making process, but we are curious if there is a noticeable difference in the success rate of projects based in the United States (the majority) from those outside the US. We thus convert these variables into the pure binary (1 or 0) *currency_is_usd* and *country_is_us* for our model to use.

The last variables to address are *category* and *main_category*. It is not possible to check all possible combinations of categories, since there are fifteen main categories and far more subcategories. At first, we just threw out these variables. We then included *main_category* by using one-hot encoding. This gives us a variable for each category which is 1 if the example is in that category and 0 if it is not. These variables are denoted by the names of the categories, which are *Art*, *Comics*, *Crafts*, *Dance*, *Design*, *Fashion*, *Film & Video*, *Food*, *Games*, *Journalism*, *Music*, *Photography*, *Publishing*, *Technology*, and *Theater*. The inclusion of these categories improved the decision tree's accuracy by 2-3 percentage points.

Thus our final dataset for the agents is fully numeric, and is composed of the variables *goal*, *usd_goal_real*, *title_length*, *title_punc*, *title_caps_ratio*, *currency_is_usd*, *country_is_us*, *launched_epoch*, *open_epoch*, the fifteen categories stated above, and the *target*.

We split the new nearly 300,000 row dataset into three partitions to be used for training our agents: 60% for training, 20% for validation, and 20% for testing. After repeatedly training with the training data to tune the model, we will train with both the training and validation data and test its accuracy on the testing data.

3. HYPOTHESES AND COMPARISONS

We will be comparing our implementations to a random agent, a human agent, and the optimal scikit-learn implementations available online [1]. We will also compare our implementations, Decision Trees and Logistic Regression, to each other.

- H_0: Decision Trees will outperform the random agent.
- H_1: Logistic Regression will outperform the random agent.
- H_2: Decision Trees will perform better than a human on the test set.
- H_3: Logistic Regression will perform better than a human on the test set.
- H_4: Decision Trees will achieve 80% accuracy.
- H_5: Logistic Regression will achieve 80% accuracy.
- C_0: Decision Trees will come within 15% of the "optimal" performance by the scikit-learn tree classifier using Gini indexing.
- C_1: Logistic Regression will come within 15% of the "optimal" performance by the scikit-learn regression algorithm.
- C_2: Decision Trees will outperform Logistic Regression.

4. LEARNING IMPLEMENTATIONS

Before creating our agents, we want to see how they compare to the "optimal" implementations available through scikit-learn. This tells us what levels of accuracy we could reasonably expect, and we will discuss more on this performance in Section 5. We will also compare our agents to a random classifier and to a "human agent;" for this, we wrote a Python program that simply prompts the user with a line of data and requests a prediction, which we catalogue and use to calculate accuracy at the end. In terms of the learning methods, we are implementing Decision Trees and Logistic Regression.

Decision Trees are a fascinating supervised learning technique in that they provide a viewable structure which makes clear to the reader which variables are being used for classification. The relative importance of each variable is clear by their place in the hierarchy; variables closer to the root node are a better predictor of the target value. This means we are able to compare not only the performance (accuracy) of our method against the optimal, but we can see the different sets of variables used to make the decisions.

We use Gini indexing, calculating a float between 0 and 1 rating the classification effectiveness of a given variable and given threshold; a Gini value of 0 represents purity, where all rows in a partition have the same label, and a Gini of 0.5 represents a random mixture. We thus seek to minimize Gini and separate the data as uniformly as possible. We loop through a node's rows, separating it into two children based on every variable, trying every single value of that variable in the dataset as the threshold. When we find the partition with the best gini value, we finalize the split of data into two child nodes, and repeat the procedure for each child.

For example, say we split our data on *goal*. We find that $goal \leq 1000$ gives the best Gini value. Then we create child nodes c1 with all rows for which $goal \leq 1000$ and c2 with all other rows. Then maybe within c1, $title\_length \leq 30$ gives the best split, while for c2, $launched\_epoch \leq 1379046650$ gives the best. We follow this process, splitting recursively (not allowing reuse of variables) until hitting our max tree depth of 5 or finding a pure node in which all rows have the same label. Because of the deterministic and exhaustive nature of this classification method, we will reproduce the same tree on every run of a dataset.

To create this structure, we make several custom Python classes, including DecisionTree, Node, and NodeStorage, which we utilize with a separate Driver file that grabs the data and runs everything. The NodeStorage class is used for display of a created tree and file I/O, such as storing a tree in a file and later recovering it to use for generating more predictions. We referenced [3] for developing a procedure for this that works in all cases. Each Node stores its dataset before it has children, and then stores the variable and threshold corresponding to its partition. Terminal nodes will instead store a decision for the target value based on the majority of its rows' label. For making predictions then, we can simply send an example to the root node, and it will follow the tree recursively to a terminal node,

returning the final prediction.

Logistic Regression, on the other hand, gives a probabilistic rather than deterministic prediction. We take the prediction for a given example as 1 if the agent returns something greater than 0.5. The main principles of logistic regression follow those laid out in class. Each row in our dataset is an x_i with an associated label y_i. N is the total number of items in our dataset. We calculate the gradient decent over our full dataset several times to converge to our weight values. To do this, we use the following procedure:

1. Initialize weights to 0.

2. For each x_i in the dataset, compute

$$\nabla_{x_i} = \frac{y_i \cdot x_i}{1 + e^{y_i \cdot w_i^T \cdot x_i}} \tag{2}$$

3. Calculate

$$\nabla_w \hat{R}_s = -\frac{1}{N} \cdot \sum_i^N \nabla_{x_i} \tag{3}$$

4. Update weights as

$$w = w - n \cdot \nabla_w \hat{R}_s \ , \tag{4}$$

where n is the learning rate.

5. Repeat until Δw is small or max time is reached.

A keen reader might ask what happens if $x_i \cdot w \cdot y_i$ is large? Wouldn't that result in an overflow? This is exactly right. Our dataset has several large values that we didn't regularize. Instead of regularizing, we set the gradient to be close to

$$\begin{cases} 1 & -x_i \cdot w \cdot y_i > 30 \\ 0 & -x_i \cdot w \cdot y_i < 30 \end{cases} \tag{5}$$

This prevented the overflow error and helped converge quickly. It's worth noting that even the scikit-learn implementation does something similar [1]. It applies regularization by default which we did not change.

The only parameter we choose to use was a learning rate of $n = 0.1$ in our implementation. Everything else was derived from the data. For the scikit-learn implementation, we used a grid search over different regularization parameters and solvers. In the end we used the default solver of *lbfgs* and $C = 0.001$.

We train all agents separately since it is a non-competitive task.

5. RESULTS AND DISCUSSION

We will first discuss the performance of the heuristic agents. We expect the random agent to perform at an average of 50% accuracy, since the state to predict is binary. We get something fairly close to this on any given run, due to the law of large numbers and our dataset having very many entries; it is consistently in the range $\{0.494, 0.504\}$ on our testing set. We expect similar performance for a human, since the data doesn't give much information at face value. On a trial of 100 examples, our human achieved 52% accuracy. These are the two thresholds we hope to beat with our learning implementations.

The results for the decision tree are very interesting. The optimal scikit-learn implementation gives an accuracy of 64.02%, which is lower than we had hoped, but still significant. Our implementation of the decision tree gives an accuracy of 63.09%, which is very close to optimal! By our metrics, we can confirm hypotheses H_0, H_2, and C_0, since we have beaten the heuristic agents and come close to the optimal accuracy.

The benefit of using decision trees is that we have not only a structure for making predictions, but a visual representation of the deliberate choices for each variable and the associated threshold value. Additionally, our method of exhaustively building the decision tree makes it deterministic: running the code again on the same dataset will produce exactly the same tree. Because of this, the fact that our tree did not precisely match the optimal implies a slight difference in the logic or methodology, resulting in some small changes. Since we allow a max tree depth of 5, the trees can get fairly large, so we will discuss the variables which were deemed to be the most important, and show simplified trees in FIG. 1 and 2. The captions contain links to view the full trees.

Both the optimal tree and our tree agree that the four most important variables are *usd_goal_real*, *goal*, *launched_epoch*, and *open_epoch*, but they disagree on the ordering of the hierarchy. Our tree has less complexity lower in the tree, whereas the optimal does use some other variables, as seen in FIG. 1.

Logistic regression achieved very similar results as the decision trees. The scikit-learn implementation achieved a testing accuracy of 60.45%. Our implementation achieved a very similar score of 59.69% and doesn't take too much longer to train! We ran this well over 30 times and these results remained static.

As we hoped, logistic regression also outperformed the random and human agents, satisfying our hypotheses H_1 and H_3. Additionally our model was within 1% of the scikit-learn implementation, so we have more than satisfied C_1.

If we compare logistic regression and decision trees,

```
|usd_goal_real <= 15002.38
|-- goal <= 4726.50
|-- -- launched_epoch <= 1399331776
|-- -- -- open_epoch <= 2498396
|-- -- -- -- Fashion == 0
|-- -- -- open_epoch > 2498396
|-- -- -- -- Publishing == 0
|-- -- launched_epoch > 1440044224
|-- -- -- open_epoch < 2499461
|-- goal > 4726.50
|-- -- title_punc >= 1
|-- -- -- Music == 1
|-- -- -- -- open_epoch <= 4439779
```

FIG. 1. Optimal tree simplified to show only paths to a classification of 1. The full tree can be seen here.

```
|goal <= 9600
|-- open_epoch <= 2491141
|-- -- launched_epoch > 1404823293
|-- -- -- usd_goal_real <= 748.37
|-- -- launched_epoch <= 1399075564
|-- -- -- usd_goal_real <= 4115.62
```

FIG. 2. Our tree simplified to show only paths to a classification of 1. The full tree can be seen here.

we see that they are very similar in their performance. They are also incredibly similar in how close our individual implementations of them are. We think the consistency across models and implementations shows that it is not necessarily the models' fault for not achieving our H_4 or H_5, as we will discuss below. It is worth noting in regards to C_2 that decision trees do outperform logistic regression by a small margin, but they are very comparable.

We have not achieved H_4 or H_5, because we overestimated the predictive power that this dataset would grant. We may have seen lower accuracies than expected due to information loss during the transformation to numeric variables. Perhaps we could have extracted more or better metrics from the campaign *name*, maybe the subcategories would have made a big difference, or maybe we needed more complex preprocessing which took greater advantage of statistical theory. The reason we think is most likely is the lack of some important information in the dataset in the first place; a campaign must be structured well, with a good pitch video, polished graphics, well-written descriptions, and extensive pledge goals and options. These qualitative metrics can't really be measured or objectively quantified, but they make a difference in the consumer's perception of a campaign, and thus their likelihood to contribute.

We also did a more through analysis of other notebooks reporting accuracies in excess of 85%, such as one by Chia [4]. They report a logistic regression accuracy of 90% and a decision tree accuracy of 89%. A closer examination of this implementation shows that they do significantly more data preproceessing and use some variables we threw out. The most significant changes is they use the amount of backers the project had and a custom category days to deadline. We believe had we used these metrics in our process we would have been more accurate. They also use a strong statistical background to preprocess the data, regularize, test cross correlation, and select the highest correlated variables.

Our goals with the project differ from this group. We want to be able to make predictions before a campaign is launched, while their success relies on information only available during or after a campaign. They predict the success of campaigns currently running, so they have more data they can use reasonably.

Overall, we believe this project was a good exploration of supervised learning methods. It was also an apt reminder that an algorithm can only achieve upwards of 90% accuracy if it's in addition to good data and careful planning.

6. CONTRIBUTIONS

Kevin and Logan discussed ideas for how to process the data and decided on the best path forward. Logan wrote initial pre-processing code in Python, and Kevin wrote some more robust R scripts that were quicker and more adaptable for creating differently sized datasets for testing before moving on to the real deal. Logan implemented Logistic Regression and wrote code to run the scikit-learn version. Kevin did the same for Decision Trees, and created a custom class structure to fully customize the nodes and tree. Lastly, Kevin wrote simple Python scripts for the random and human agents. All of our code is stored on GitHub here.

7. RELATED WORK

Unlike our RL project domain, Kickstarter campaigns aren't a well-published area of research. We found a few student papers and several medium articles, but decided to explore related work that go more in-depth with our models, rather than necessarily looking at the same domain.

Logistic regression has been used in several instances to predict attributes. It is particularly apt at processing large sums of data and making predictions that would take humans a long time. Gen Hori published a paper in 2018 discussing their use of logistic regression to predict

the factors that contributed to dropping out of school [5]. The formulae laid out in the paper are largely identical to the those we used in this project. They calculate a true gradient and then calculate the Hessian, as we do in Eq. 3. The equations look slightly different but prove similar to our approach. Shockingly, their performance was similar to ours, as they were reporting 70% accuracy on their test set. The paper serves to confirm that our method and implementation matches the published literature. It contains a very similar process to ours and concludes results along the same lines.

Similarly to the previous paper discussed, Prasetio and Harlili published a paper using logistic regression to predict (European) football match results [6]. They reported an accuracy of around 69.5%. They were able to use logistic regression to discover the significant variables and to make accurate predictions. This follows very similarly to what we did in our analysis. The most interesting part of their paper is they used data from the video game FIFA to improve their results. We are interested if we could have improved our results similarly with a supplementary dataset, if one were to exist for our domain. It is interesting to see that the results from Hori and Prasetio/Harlili were close to our accuracy rates. The data we analyzed from some of the other projects with our dataset seems to have much higher accuracy rates than those seen even in real applications of Logistic Regression. It seems to be that higher fidelity data and better predicting factors, like dollars pledged, have a huge influence on accuracy, which makes sense.

A very old paper published in the Harvard Business Review proposes many potential uses for Decision Trees, and describes situational ways to think of them [7]. The author holds that this method of intelligent classification will be in common use by businessmen, managers, and investors before too long. They describe decision problems in terms of "payoffs," risks, and rewards, and represent current decisions and future decisions as dependent on one another, with much uncertainty mixed in. In this way, a business' entire management can be reduced to a tree of decisions, so this is a very important topic to be addressing. The paper gets into not only decision trees, but decision-event chains which tie together possible outcomes and distinct actions. This is a very good way to begin representing a problem in a domain similar to ours before moving into the implementation aspect.

8. SUMMARY AND FUTURE WORK

We were able to implement both Decision Trees and Logistic Regression from scratch, and achieve statistically significant performance. We confirmed all Hypotheses other than H_4 and H_5, since our dataset did not give us the predictive power that we anticipated. This was a fun project, and if we were to try and improve it, the main areas we would look next are smarter data pre-processing and better optimization of the algorithms themselves. The Decision Tree takes several hours to train on the full dataset, since it exhaustively checks all possible variable and threshold values, but is beat out by the scikit-learn implementation in only a few minutes. It would be interesting to look deeper to discover the inefficiencies in our code and find the cleverness that allows theirs to run so much faster. We have discussed in Section 5 possible changes to the preprocessing that would result in higher accuracy, and it would also be interesting to see other supervised learning algorithms applied to this same dataset.

[1] Scikit-learn, *Machine Learning in Python*, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[2] M. Mouillé, *Kickstarter Projects Dataset*, version 7.9, updated 2018. Available at https://www.kaggle.com/kemical/kickstarter-projects.

[3] Ritambhara Technologies, *Storing Binary Tree in a file*, 2018. Available at https://www.ritambhara.in/storing-binary-tree-in-a-file/.

[4] K. Chia, *Predicting project success with ML models*, 2020. Available at https://www.kaggle.com/kesterchia/predicting-project-success-with-ml-models

[5] G. Hori, *Identifying Factors Contributing to University Dropout with Sparse Logistic Regression*, 2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI), Yonago, Japan, 2018, pp. 430-433, doi: 10.1109/IIAI-AAI.2018.00091.

[6] D. Prasetio and D. Harlili, *Predicting football match results with logistic regression*, 2016 International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA), George Town, 2016, pp. 1-5, doi: 10.1109/ICAICTA.2016.7803111.

[7] J. Magee, *Decision Trees for Decision Making*, Harvard Business Review, from the Magazine, July 1964. Available at https://hbr.org/1964/07/decision-trees-for-decision-making