Crowdfunding: Predicting Kickstarter Project Success

Using over 220, 000 project proposals scraped from Kickstarter, we predict whether a Kickstarter project proposal will succeed or fail in achieving its fundraising goal using only information from project launch. We evaluate the performance of various machine learning models for these predictions based on the project category, fundraising goal, and short descriptions of the proposed product. We achieve an accuracy rate of 83% with our best model.

1 INTRODUCTION

Kickstarter, founded in 2009, is an online platform that acts as a centralized marketplace to connect "creators" with the capital to pursue their visions. Creators post their project ideas on the platform and collect the necessary funding from their "backers." Often, these donations take on the form of an investment, with the creator promising premium content for the backers. As of Nov. 2020, over 500,000 projects have been launched on Kickstarter, with over five billion dollars raised.

In this paper, we explore the use of Kickstarter project data to predict campaign success. In particular, we answer the question: given information about a Kickstarter project's fundraising goal, category of product, brief project descriptions, and more, can we accurately predict the outcome of the campaign? We employ a variety of learning models, including logistic, lasso, and ridge regression, random forests, neural nets, and gradient boosting algorithms to answer this.

2 RELEVANT LITERATURE

There is a diverse literature focusing on the success of Kickstarter projects. [Genevsky 2017] approach the topic through the lens of neuroscience, analyzing brain activity in an fMRI, while many papers in linguistics have used quantitative and qualitative lenses to focus on the role of specific phrases in a project description. Within the computer science literature, classification accuracy ranges between the low 70s and the high 80s (in percent) depending on the variables included, methods used, and the timing of the project since launch (see references for a complete list). For example, [Greenberg 2013] see the best results using decision trees while [Chen 2013] utilize a Support Vector Machine (SVM).

However, a substantial portion of the literature focuses on project success probabilities over time, incorporating temporally-dynamic information such as the amount raised or social media interactions (see [Etter 2013], [Mollick 2015], and [Zhao 2017]). Of the few papers that focus on pre-launch factors, a variety of methods and features are used. [Cheng 2019] formulate a multi-modal deep learning algorithm using both text and images while [Desai 2015] use natural language processing (NLP) methods to classify project outcomes. While the impact of post-launch factors are certainly important, our interest lies more in predicting the outcome of a project given its initial static properties. This has important implications for both creators and Kickstarter – as such knowledge would enable creators to more efficiently allocate their time and effort while also allowing the platform to prioritize projects with a higher likelihood of a success.

3 DATA & FEATURES

3.1 Data

Projects on Kickstarter combine a variety of data types. Each project has a funding goal and a duration. Projects generally contain text descriptions and backstories as well as visual information, such as photos and videos. Moreover, projects are connected to user profiles (i.e., creators) that contain short biographies, previous Kickstarter activity, and external links to various social media platforms. Kickstarter also displays the amount of money raised to date and information about the backers, which includes their location, whether they are new or returning Kickstarter users, as well as any comments they might have left on the project page.

We use a well-maintained repository containing data for over 250,000 Kickstarter projects. The authors designed a robot to crawl Kickstarter each month and scrape the labeled HTML output into CSV files. This output contains all the information we use for this analysis, including the funding goal, project categories, and a short project blurb. Although the data is stored as a CSV, many of these features are stored as JSON strings, which we expand to obtain our desired project variables. We attempt only to select information that would be available at the time of launch. However, since our data comes from a monthly snapshot, if a creator were to edit their project metadata after launch, we would not be able to detect this. Luckily, many project aspects cannot be changed after launch, including the funding goal. Due to technological constraints with the HTML scraper, our dataset does not contain the longer-form project description ("story").

For our main dataset, we use the projects contained at the above repository starting with the April 2019 link up to the April 2021 link. However, these files are largely cumulative and thus contain significant overlap. We deduplicate projects by removing rows that share the same project name, blurb, launch date and deadline. Our resulting data sample contains 221,248 Kickstarter completed Kickstarter campaigns. Given this relatively large sample, we use a 70-30 train test split. Thus, our train and test sample contain 154,873 and 66,375 projects, respectively. Our selected metadata

includes the target funding goal in USD, the launch date and deadline of the project, the project blurb, as well as categorical information such as the project category and the location of the creator. Metadata that would not be available at the time of launch, such as the number of backers, the amount of funding received thus far, time remaining, etc. is excluded.

Table 1. Summary statistics of numeric variables, train sample

Statistic	Outcome	Goal (USD)	Blurb Length	Name Length
Mean	0.6	33,943.73	108	35
Median	1	4,820.28	120	34
Min	0	0.01	1	1
Max	1	112,339,607	151	85

Table 2. Summary statistics of categorical variables, train sample

Variable	Unique Levels	PpL Mean	PpL Min	PpL Max
Currency	15	10,325	14	106,524
Country	25	6,195	13	106,524
Category	171	906	5	5,280
Parent Category	15	9,660	1,218	19,527
Location	13,771	11	1	6,905
Location Type	9	17,191	1	143,863

Note: PpL abbreviates Projects per Level.

3.2 Categorical Encodings

As discussed above, each Kickstarter project is associated with a variety of categorical variables, including the project category (e.g. Board Games, Documentary, etc.) and the parent category (e.g. Gaming, Film, etc.), the creator's city and the country of origin as well as the currency of the donation. We design a rich encoding for these categorical features.

First, we calculate target mean encodings for each categorical variable, that is, given a categorical x and target y, we replace each distinct level j in x with its conditional mean of y: $\bar{y}_j = \sum_{i=1}^{n} \frac{1_{x_i=j}\, y_i}{1_{x_i=j}}$. Levels with fewer than 1,000 projects are replaced with the overall, unconditional mean. To avoid data leakage, all estimates are obtained "out-of-fold", i.e. for a given project in fold 1, the mean encoding is calculated based on the average of the project outcome in folds 2-5. Second, we introduce "out-of-fold" mean encodings of the same categorical features using the project funding goal. We also specify the difference between a project's funding goal and the mean goal for each categorical as features to make this relative difference explicit. Finally, we introduce one-hot encodings of the project category and parent category variables. We do not add these for every categorical variable as we hypothesize the two category variables to be most important and we wish to preserve the cardinality of our feature space, e.g. one-hot encoding the location variable would introduce over 10,000 additional features, as seen in Table 2.

3.3 Text Encodings

Moreover, our dataset contains a small description ("blurb") for each project. At the most basic level, we use the character length of this blurb as a feature. In addition, we employ a variety of text-modeling approaches to extract valuable information from the blurbs. These methods include unigram count matrices (with a tf-idf transformation), Latent Dirichlet Allocation (LDA), word embeddings (with Word2Vec), and sentiment analysis to provide a positivity rating of the blurb (with CoreNLP). All methods received a processed version of the blurbs, with stopwords and non-alphanumeric characters removed.

3.3.1 Unigram Counts. The simplest text encoding approach is to construct a counts matrix of the occurrences of every unigram (token) in our corpus for each project blurb. We then introduce a tf-idf transformation that weights this matrix according to the relative frequency of a token within a blurb (tf) and across blurbs (idf). For example, a word that is common to all blurbs (say "Kickstarter") would not convey much information. However, the dimensionality of such a matrix (equal to the cardinality of tokens) is substantially too large. Thus, we compressed this to a single feature by running a Multinomial Naive Bayes classifier on the counts matrix (see Section 4.1 for more information) to obtain the probability of project success.

3.3.2 Latent Dirichlet Allocation. LDA is a generative model that allows for the unsupervised classification of texts according to a predefined number of clusters ("topics"). The model assumes each project blurb is a "bag of words" and that there exist two latent variables – the topic distribution for each blurb and the topic for each word – which influence the selection of words in a blurb. LDA imposes a (often symmetric) Dirichlet prior on the topic mixture proportions and runs an EM algorithm, marginalizing over the latent variables. Thus, the model learns a set of weights for each topic, which we can use to predict the probability that each project blurb belongs to each topic. In turn, we can use these project-specific topic probabilities as features in our main model.

3.3.3 Word Embeddings. Word embedding is a NLP process that captures the similarity of semantic meanings between words. The main intuition is that words that occur frequently together are likely related. There are two different architectures used: 1. Continuous Bag of Words (CBOW) where neighboring words in a sliding window are used to predict the center word and 2. skip-grams which does the reverse. Both architectures use stochastic gradient descent to update the probability of the target word(s) given the word(s) in the window. Training such a model yields a low-dimensional vector representation capturing the semantic difference between words, such that similar words have similar

vectors. Thus, we can compute the average word vector for each project based on the words in the blurb and use this as a feature.

3.3.4 Sentiment Analysis. The “attitude" or “feel" of a blurb can significantly impact whether a user donates to a campaign. Here, we use Stanford CoreNLP to analyze the sentiment of each blurb and assign it a score between 0 (least positive) and most positive (4). The score is used as a feature in our models to confirm whether there exists a positive correlation between more positively-framed campaigns and success rate. We analyze each sentence in a blurb independently and provide a score for the overall blurb that is the average sentiment across all sentences in a given blurb.

4 MODELS

We disccuss the models we use with some brief mathematical background here. We define our task as a binary classification problem, where we predict whether a Kickstarter project will “succeed" (1) or "fail" (0) in meeting its fundraising goal. For all specifications with continuous outputs, e.g. linear or logistic regression, we use a threshold of 0.5 to obtain our predictions.

4.1 Multinomial Naive Bayes

Multinomial Naive Bayes is a simple supervised learning algorithm based on the assumption that token occurrences are drawn from a multinomial distribution and are conditionally independent given the outcome. The model is parameterized by ϕ_y as well as vectors $\phi_{w|y=0}$ and $\phi_{w|y=1}$ where w is an token in the corpus. These estimates are obtained by counting the relative frequencies:

$$\phi_{w|j} = \frac{\sum_{i=1}^{n} 1_{y_i=j} x_{i,w}}{\sum_{i=1}^{n} \sum_{w} 1_{y_i=j} x_{i,w}}, \quad \phi_y = \frac{\sum_{i=1}^{n} 1_{y_i=1}}{n}$$

where $x_{i,w}$ represents the number of of occurrences of token w in observation i and $j \in \{0, 1\}$.

4.2 Linear Probability Models

4.2.1 Ordinary Least Squares. The least-squares cost function is defined as $J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$, where $h_\theta(x^{(i)}) = \theta^T x^{(i)}$. We minimize this with gradient descent. Namely, we start with some initial θ and repeatedly perform the update $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ until convergence, where α is the learning rate.

4.2.2 Lasso and Ridge Regression. Lasso and Ridge regression follow the same paradigm as OLS with added penalties in the cost function. Specifically, Lasso adds the $l1$ penalty, $\lambda \sum_{j=1}^{D} |w_j|$, while Ridge adds the $l2$ penalty, $\lambda \sum_{j=1}^{D} w_j^2$.

4.3 Logistic Regression

Logistic Regression works similarly to the linear regression methods but here, instead of using a linear hypothesis function, we use the sigmoid function $h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)}$.

4.4 Support Vector Machines

SVM finds the maximum margin hyperplane in our feature space where an outcome is assigned according to $y = \text{sign}(K(w, x) + b)$ where a negative y value indicates “failure", while a positive y value indicates a “success". This problem can be reduced to an optimization problem where we seek to find $\min_{w,b} \frac{1}{2}||w||^2$ subject to the constraint $y^{(i)}(w^T x^{(i)} + b) \geq 1$ for $i = 1, ..., n$ where the kernel $K(w, x)$ is the inner product of w and x.

4.5 Random Forest

Random Forest is an ensemble method that "bags" decision trees, combining their individual predictions by "voting" (in classification problems) to obtain an overall inference. The randomness introduced by the algorithm – e.g. choosing from a random subset of the overall features at a given split – prevents the individual trees from being too similar, thereby allowing it to explore more of the overall feature space, and thus ensuring the subsequent aggregation of the trees is more robust.

4.6 Gradient Boosting

Gradient Boosting is an ensemble method that finds the optimal hypothesis function in an iterative manner by fitting a weak learner on the residual at each step, thereby attempting to correct the errors from the previous step. We use gradient tree boosting which fits a decision tree at each iteration. As this will be our main model, we present this more explicitly: for a differentiable loss function L and an initial hypothesis function f_0, gradient boosting updates at step m by calculating

$$r_{i,m} = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)},$$

fitting a decision tree $g_m(x_i)$ to these residuals, and setting $f_m(x_i) = f_{m-1}(x_i) + \lambda \gamma_m g_m(x_i)$ where γ_m represents the step multiplier and λ the learning rate. We use a binary log loss function defined as $L_i = y_i \log f(x_i) + (1 - y_i) \log(1 - f(x_i))$.

4.7 Dense Neural Network

Neural networks allow for the learning of complex non-linear models through a series of connected layers. In a dense neural network, every node is fully connected to all of the nodes in the previous layer. Layer l receives the input ("forward-propagation") $a^{[l]} = f(\sum_{i=1}^{n} w_i a^{[l-1]} + b)$ where f specifies the activation function. We use a Scaled-Exponential Linear Unit (SELU) activation

$$\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha \exp(x) - \alpha & \text{if } x \leq 0 \end{cases}$$

and a sigmoid activation. The neural net then "back-propagates" using the Adam optimizer with a binary log loss function.

5 EXPERIMENTS

5.1 Model Evaluation Metrics

To evaluate ouor model performance, we report the overall accuracy $\boldsymbol{A}$, the precision $\boldsymbol{P}_j$ and recall $\boldsymbol{R}_j$ for both outcome classes, and finally the F-1 score **F1**. These are defined as

$$\mathbf{P}_j = \frac{TP_j}{TP_j + FP_j}, \quad \mathbf{R}_j = \frac{TP_j}{TP_j + FN_j}, \quad \mathbf{F1} = 2 \cdot \frac{\mathbf{P}_1 \cdot \mathbf{R}_1}{\mathbf{P}_1 + \mathbf{R}_1},$$

where $j \in \{0, 1\}$ and TP_j represents the number of true positives, TN_j true negatives, FP_j false positives, and FN_j false negatives, all for class j. While the accuracy places an equal importance on TP and TN, the F-1 score places a greater emphasis on false predictions (FN and FP). Given that our data is not especially unbalanced and the costs associated with a false prediction are not grave (like, say, with coronavirus testing), we believe that accuracy is an important, intuitive metric. However, the F-1 score is more interesting from a business perspective, as it more harshly penalizes false predictions, and places less value on true negatives. Thus, we consider both throughout the discussion of the results.

5.2 Samples

We have five main samples, each including the same project metadata-related features but with different blurb encodings:

(1) Metadata
(2) Metadata + Naive Bayes (NB)
(3) Metadata + NB + CoreNLP Sentiment (NLP)
(4) Metadata + NB + Latent Dirichlet Allocation (LDA)
(5) Metadata + NB + Word2Vec (W2V)

All samples consist of the project metadata, as the project blurb observed in our data is not rich enough to perform well on the prediction task independently. Nonetheless, we believe that even this limited text description likely contains relevant signals that can make an impact on the margins.

5.2.1 Text Models. We now detail the construction of these text models and include their accuracy to illustrate their baseline performance. First, for our Naive Bayes classifier we use unigram occurrences and apply a tf-idf transformation. Fitting this to the train data, we obtain probabilistic predictions for the success of each project. Naive Bayes attains a classification accuracy on the test sample of 0.70. Second, we use Stanford CoreNLP to annotate the blurbs, requesting a sentiment value between 0 and 4 for each blurb, averaging across all sentences in a given blurb for the overall score. Fitting a simple logistic regression between the sentiment score and project outcome, yields a 0.6 accuracy on the test data. Third, for the construction of our LDA model, we drop all tokens, which occur in fewer than 10 blurbs or more than 35% of blurbs (conceptually similar to tf-idf). We specify 20 latent topics with identical priors, as this is on the order of Kickstarter's parent project categories (15) and it has the best predictive accuracy (compared with 5, 10, 50 and 100 topics) with a score of 0.60. Finally, for our word embeddings, we use Google's pretrained W2V model with 300 dimensions from a 100 billion word Google News corpus. The training architecture of this model is unknown (i.e. CBOW or skip-grams). For each blurb, we compute the average of the associated W2V vectors. Using these features, a logistic regression achieves an accuracy of 0.69. As we will illustrate in the next section, while the performance of these text methods individually is quite poor, when combined with the external project metadata we see small performance increases. Thus, there is certainly some signal in the project blurbs that the text models can detect.

5.3 Results

We present the results of our models, as discussed in Section 4, trained on Data Samples 1 and 2 in Tables 3 and 4, respectively. In both samples, gradient boosting performs best on both of our main metrics, accuracy and F-1 score, followed closely by random forest and neural net. See Figure 1 for a heatmap of the gradient boosting predictions. Aside from the Lasso, the remaining models all have fairly similar performances. All of our models perform worst on $\mathbf{P}_0$, due to the presence of false negatives (as seen in bottom left of Figure 1 as well). Note, input features for the neural net and logistic regression are standardized with zero-mean and unit variance. For SVM, we use min-max standardization to improve computational efficiency, as we ran into an issue of lack of convergence and long run times. We ran the remaining methods with and without feature standardization, which yielded identical results – other than for the lasso whose performance on standardized data suffered tremendously. All models are weighted to ensure equal outcome class balance, as our initial sample is slightly unbalanced, and all metrics are calculated on the test data.

Table 3. Model Results from Data Sample 1 (Metadata)

Model	**A**	**F1**	$\mathbf{P}_1$	$\mathbf{P}_0$	$\mathbf{R}_1$	$\mathbf{R}_0$
G. Boosting	0.81	0.83	0.90	0.72	0.77	0.88
R. Forest	0.80	0.82	0.91	0.70	0.74	0.89
Neural Net	0.80	0.82	0.90	0.70	0.75	0.88
SVM	0.79	0.83	0.81	0.74	0.84	0.71
Logistic	0.78	0.80	0.88	0.67	0.72	0.86
Ridge	0.77	0.79	0.90	0.66	0.70	0.88
OLS	0.77	0.79	0.90	0.66	0.70	0.88
Lasso	0.70	0.74	0.78	0.61	0.70	0.71

The most important features in all of our models and samples are the outcome and USD goal encoded categories as well as the Naive Bayes predictions. Indeed, there is no additional gain for including the other text-based features, as seen in Figures 2 and 3, which display the average accuracy and F-1 score achieved by our main specification, gradient boosting, on each data sample across 50 random seeds. While performance is even across samples, computational efficiency is not. Indeed, Samples 3-5 have 1, 20,

Table 4. Model Results from Data Sample 2 (Metadata & NB Bayes)

Model	A	F1	P_1	P_0	R_1	R_0
G. Boosting	0.83	0.85	0.89	0.75	0.81	0.84
Neural Net	0.82	0.84	0.88	0.74	0.80	0.84
R. Forest	0.82	0.84	0.88	0.74	0.80	0.84
Logistic	0.80	0.83	0.86	0.72	0.79	0.80
SVM	0.80	0.83	0.82	0.76	0.85	0.71
Ridge	0.80	0.82	0.87	0.71	0.78	0.83
OLS	0.80	0.82	0.87	0.71	0.77	0.83
Lasso	0.71	0.75	0.79	0.62	0.72	0.71

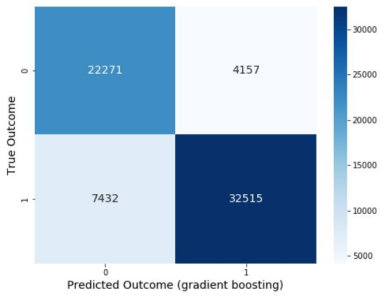


Fig. 1. Heatmap of true v. predicted outcomes

and 300 more features than Sample 2 respectively. Thus, Sample 2 and 3 are our preferred samples.

Fig. 2. Gradient boosting accuracy by data sample across 50 seeds

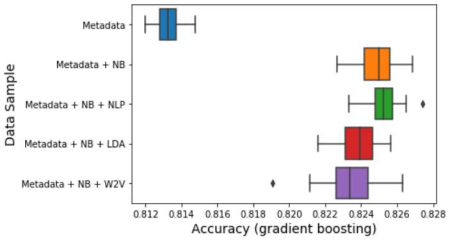
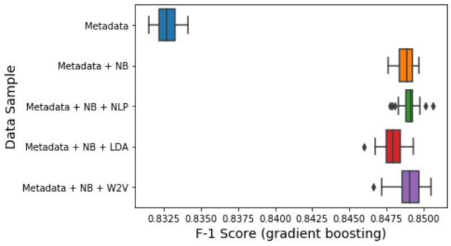


Fig. 3. Gradient boosting F-1 score by data sample across 50 seeds



5.4 Parameters

For all models, optimal parameters were obtained by running a grid search algorithm using five fold cross validation on the train sample. The performance of both ridge and lasso regression was not very sensitive to the regularization parameter, producing nearly identical results, and we ultimately chose 0.75. For logistic regression, we use a $l2$ penalty with C, the inverse of the regularization strength, equal to 0.1. We use LinearSVC for SVM with squared hinge loss as our loss function and a $l2$ penalty with $C = 1$. We constructed our neural net to have one hidden layer with an output dimension of 25 and a SELU activation function. The final layer uses a sigmoid activation function. We train it with a mini batch size of 3 over 30 epochs. For the random forest, we run 200 estimators, where each tree has a maximum depth of 55 and each leaf has a minimum sample size of 10. At each split, we use a Gini impurity criterion and consider approximately 15 random features. Finally, in our gradient boosting implementation, we set the proportion of rows and features available to each learner at 0.75 and 0.2. We use large trees with a maximum depth of 55 and a maximum number of leaves of 400. Lightgbm uses a binning procedure to obtain optimal splits, which we control by setting the maximum number of feature bins at 500.

6 CONCLUSION

We find that the gradient boosting algorithm provides us with our best results, consistently across all samples. Random forest and neural net also performed very well, though were computationally significantly more expensive. We note that SVM does not perform as well as we might expect from the existing literature, which we attribute to our large sample size. The size of the dataset is likely a reason for our regression models performing up to par with some of the other, more robust models.

Our best performing features were the categorical mean encodings and the Naive Bayes predictions, underscoring the importance of feature engineering. While we experimented with more aggressive feature selection, we did not observe any significant performance benefits. Our models also did not exhibit any serious signs of overfitting, as they generalized extremely well to the test set. One potential explanation for this is that our training dataset is large, particularly compared with previous crowdfunding papers that have sample sizes on the order of 30,000 projects.

In our future work, we hope to obtain longer project “stories" and use more advanced text encoding methods, such as Bidirectional Encoder Representations from Transformers (BERT). BERT, allows the model to evaluate the context of a word based on all of its surroundings using Masked LM (MLM), which predicts “masked" words in a sequence using their surroundings, and Next Sentence Prediction (NSP) that predicts whether the first sentence in a pair of sentences precedes the second. In training BERT, the combined loss of MLM and NSP is minimized, thus optimizing for the understanding of context. We also wish to incorporate image and video project metadata in our models, potentially specifying a Convolutional Neural Network, as we hypothesize that visual cues are likely more important to the casual viewer than long-form text descriptions.

7 CONTRIBUTIONS

Sauren initially spent his time looking for a good dataset to use for the project as well as reading lots of relevant literature to understand the pitfalls of other projects and where other projects succeeded. His knowledge about the dataset came in handy when we could not find where our models were going wrong. Further down the road, he wrote the code that scaled the data for usage in the SVM model, which he also wrote and tuned. In addition, he wrote the CoreNLP code that analyzed the sentiment of the blurbs and included it as a feature in the dataset. Finally, he spearheaded writing the paper, in particular with regards to writing about the models and the introductory text.

Benjamin, at the beginning, spent his time importing, cleaning, and processing the data, including selecting which data attributes to keep in our data analysis. After cleaning the data to a state in which we could work with it, he developed the categorical mean and one-hot encodings as well as the the first three text encodings (NB, LDA, and W2V). He created a data pipeline we could all use which was helpful for ensuring we were all working with the same data. He wrote and tuned the majority of the sklearn models, including (but not limited to) the regressions, the gradient boosting algorithm, and the neural network. Finally, he has spent time doing some error analysis and generating plots that we used in our report above.

Joel (not enrolled in CS 229) initially played a key role in determining what analyses and features might be of interest to us as we began the data analysis. He then helped to import and clean data. Finally, he contributed to the NLP analysis of project blurbs. Specifically, he worked on a BERT model that informed the concluding section of the report.