# Predicting Crowdfunding Success with Heterogeneous Ensemble Classification

Gavin Hardin (gchardin)
North Carolina State University
gchardin@ncsu.edu

Joshua Roddy (jbroddy)
North Carolina State University
jbroddy@ncsu.edu

Andrew Stewart (astewar6)
North Carolina State University
astewar6@ncsu.edu

Noah Eggenschwiler (ndeggens)
North Carolina State University
ndeggens@ncsu.edu

## 1 INTRODUCTION AND BACKGROUND

### 1.1 Problem Statement

With the expansion of crowdfunding websites such as Kickstarter (where users can pitch their ideas and ask others for donations), the option to crowdfund capital to get a project launched has become more accessible to both individuals and businesses. As a result, it has become easier than ever to get your project funded, provided you are lucky enough to find the investors. However, it may not be pure luck that determines whose ideas come to fruition.

Despite this widespread availability, only about 40% of projects are successful in reaching their funding goal [5]. With these odds, every investor in a project is at a high risk of not seeing a return on their investment, and it is difficult to ascertain those projects that will reach or surpass their goal from those that will fail. This difficulty presents itself from the project organizer's perspective as well: if the odds are high that a project will fail, their effort in advertising and developing the product will be put to waste. To address this uncertainty, we aim to design a model that can predict whether a Kickstarter project will be successful in reaching its crowdfunding goals, given its name and several other attributes.

The primary goal of this classification model is to predict the success (and the extent of success) of a Kickstarter project using attributes of the project available before it is launched, including its name, monetary goal, category, and proposed fundraising time. These attributes are decided upon before the project is started, and so predicting success based on them could provide a rough estimate of the outcome of one's project, allowing them to make changes to increase its odds of succeeding. We chose not to use attributes in the dataset that would not be available at the time of launch, such as the final number of backers and amount pledged, as knowing these would allow one to easily predict the project's outcome and serve as a form of look-ahead bias.

### 1.2 Related Work

Classifying short texts, like titles of Kickstarter projects, is inherently different than classifying longer works and documents because of the lack of material to work with. Researchers from Texas A&M's Department Electrical and Computer Engineering and SocialCredits, Ltd. examined these differences and how they change the typical methods used in Machine Learning [8]. They concluded that using a logistic regression classifier with Word2Vec (which is employed by Doc2Vec) was much more accurate for the classification of short texts than other methods. This research offers valuable guidelines for how we should go about creating a model for the project titles.

Rania et al. analyzed topic modeling methods on short texts like Facebook posts, and illustrated the struggles that typical methods like PCA have in applying meaningful conclusions to small texts, which contributed to our decision to use Doc2Vec in our model examining the titles of projects. [1].

Richard Maclin and David Opitz, from the Computer Science Departments at the Universities of Minnesota and Montana, respectively, studied popular methods for producing ensembles and evaluated their effectiveness when used on conjunction with neural networks and decision trees [6]. They found that an ensemble produced using Bagging "almost always" outperforms a single classifier, while an ensemble produced by Boosting can "greatly outperform" both Bagging and (in turn) a single classifier on some datasets. However, they found that Boosting may actually perform worse than both Bagging and a single classifier when working with some datasets, as it is particularly susceptible to overfitting on noisy data. By default, Bagging is generally useful, but for some data sets, Boosting can be more accurate. We will consider their analysis when producing an ensemble of our feature models, evaluating which method results in the best accuracy for our data.

Tuarob et al. explored heterogeneous ensemble classification in the context of health-related social media messages, an objective close to ours [7]. The authors analyzed short, informal messages posted to Twitter and Facebook, and attempted a binary classification over the messages by labeling them as health-related or non-health related. Their objective was to produce a classification that surpassed bag of words approaches. Their experiments consisted of training a heterogeneous ensemble of base classifiers, including naive Bayes and support vector machines, on different feature types identified from the social media datasets and evaluating their performance when combined with various ensemble techniques. These techniques include majority voting and weighted probability averaging (WPA). Their results indicated that WPA resulted in the best performance on their short-text dataset, which influenced our decision to use WPA in building our ensemble predictions.

Fernandez-Blanco et al. also analyzed crowdfunding success, focusing mainly on projects launched on Kickstarter [2]. They identified several contributory factors relating to the success of a project posted on the Kickstarter website, and clustered the projects from their dataset based on similarities in the characteristics of the attributes of the projects. We hope to go beyond this work by not only analyzing the existing projects, but by building a predictive

model that could estimate the degree of success of an entirely new project.

## 2 METHOD

### 2.1 Approach

Below we explain our approach of using two models, as well as the design and structure of the models.

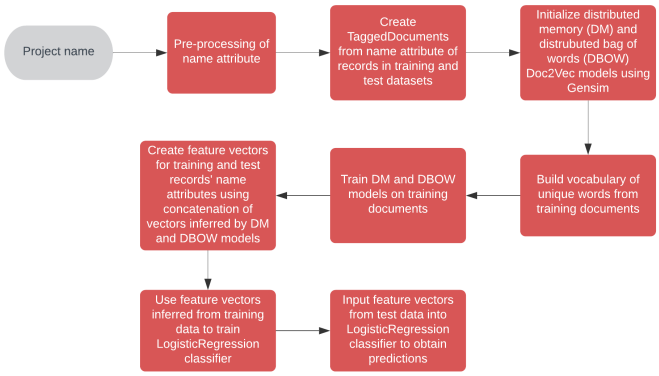


**Figure 1: Pipeline for Designing and Displaying Doc2Vec Project Predictions**

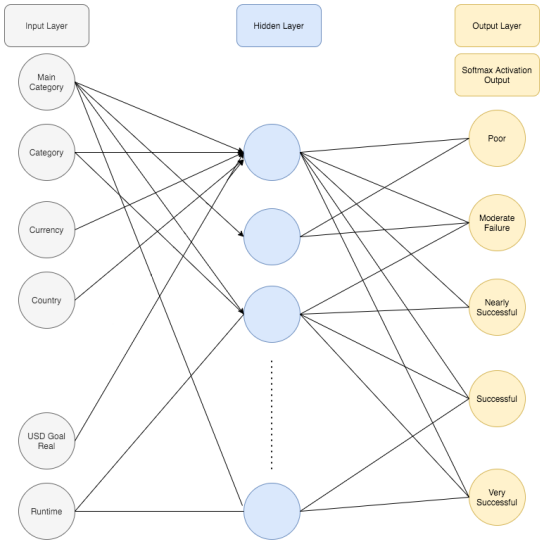


**Figure 2: Multi-Layer Perceptron using Softmax Activation for Multi-Class Prediction**

#### 2.1.1 *Classification Models.*

(1) Our first model determines the success of a project based on its name. Figure 1 illustrates the pipeline that we used to design this model. We first preprocess the data that is contained within the dataset by removing stopwords – words that frequently appear in regular English speech, such as "the", "a", "for", and others. Additionally, we convert all text to lowercase to ensure that capitalization of a word does not differentiate a word from another occurrence of that word. Then we tokenize the string into discrete words, and remove any punctuation that is within the name, under the assumption that punctuation surrounding a word should not provide any impact on success that the word itself would not already provide. After preprocessing, a set of Gensim TaggedDocuments are created from the training records and test records, consisting of the tokenized and preprocessed name attribute along with a tag corresponding to the record's index in the dataset. The tagged documents created from the training dataset are then used to train both a distributed memory (DM) and distributed bag of words (DBOW) Doc2Vec model, using Gensim's implementation of both, with each having a vector size of 150 elements. A vocabulary of the unique tokens found in the training documents is built by the DM and DBOW models, and then both are trained on the training documents for 20 epochs. Once trained, we use both models to infer vectors for the records in the training and test datasets, and combine the DM and DBOW vectors by concatenating them into a final feature vector for each document. The feature vectors created from the training dataset are then used for training a multinomial logistic regression classifier. Once trained, this logistic regression model is used to produce predictions on the test dataset by being fed the feature vectors created by the concatenated Doc2Vec inference from the test dataset, returning a predicted class (from "Poor" to "Very successful") for each test record.

(2) The second model that is used for classification is a Multi-Layer Perceptron (MLP), shown in Figure 2. The MLP has a six feature input: Main Category, Category, Currency, Country, USD Goal Real (which is just the listed goal of a project converted into US Dollars), and Runtime (the period of time the project will be accepting donations). This is a multi-class classification model, with one hidden layer that has a ReLu activation function, and 12 nodes. Finally, the output layer consists of five nodes, one for each type of class or categorization of a project's performance: Poor, Moderate Failure, Nearly Successful, Successful, and Very Successful. In order to account for a multi-class output, a softmax activation function is used within the output layer. A softmax activation function allows for a probability distribution among the five output nodes, showing how confident the model is in its prediction.

#### 2.1.2 *Ensemble Classification.*

(1) Stacked Generalization: We attempted to evaluate stacked generalization – using the outputs of our two classifier models as input to train a final classifier – but differences in the format of data required by each sub-classifier made this infeasible, and so we moved on to another approach. Specifically, the training data required for the multi-layer perceptron

required one-hot encoded training targets, while the logistic regression classifier requires the training targets to be categorically encoded as a one-dimensional array. The structure of the pipelines we created for the stacking classifier made simultaneously adhering to both requirements difficult; while we explored using a secondary library to transform the targets within the MLP pipeline, we were not able to reach a solution.

(2) Weighted Probability Averaging: To combine the two models and create an ensemble prediction, we used weighted probability averaging (WPA). After training the two models separately on the subsets of project features relevant to each model, we used the trained models to estimate the probability that an instance belongs in each individual class. We then took a weighted average of these predictions, using weights for each model that we identified through a process that optimized the F1-score for the predictions, and output the class with the highest probability of being correct as the final prediction for each sample.

#### 2.1.3 *Tuning Hyperparameters: Multi-Layer Perceptron.*

(1) Learning Rate: Learning Rate began at a default of 0.001, and was tuned this to a slower learning rate of .0001, in an effort for better results.

(2) Epochs: The number of Epochs began at 25, and continued to increase to 40 as accuracy increased, while being tested to make sure that the model was not overfitting.

#### 2.1.4 *Tuning Hyperparameters: Logistic Regression Classifier.*

(1) Inverse of Regularization Strength: To determine the best value of C (the inverse of the regularization strength – a penalty added to reduce overfitting), we performed 5-fold stratified cross-validation using a grid of 10 potential C values on a logarithmic scale between 1e-4 and 1e4. We retained the best value of C identified through the cross-validation and used it for our final logistic regression classifier.

#### 2.1.5 *Model Evaluation.*

The metrics that we used to evaluate our models.

(1) Dataset: We split out dataset using the standard 75/25 rule, having 75% of our data for training purposes, and 25% for testing.

(2) Evaluation Metrics: In order to understand how well our models performed, we looked at the accuracy, weighted F1 score, as well as the accuracy and F1 score based on the model's ability to predict if a project would result in "Success" (meets its goal) or "Failure", rather than the specific degree or extent of success and failure.

## 2.2 Rationale

One of the most notable, yet complex, attributes of an instance in the Kickstarter dataset is the project's "name", which provides some of the more immediately noticeable commonalities between any number of projects or instances.

Regarding to the idea of having two models (one for "name" and one for the remaining attributes) to predict and classify instances, the "name" attribute on its own seemed too complex to include as part of another model. A user has a lot of free reign over what they call their project, so the name of any instance can vary greatly based off of a user's vernacular or experience in marketing a product (not even considering things such as typos or buzz words with a limited lifespan). Therefore, trying to quantify and include the "name" attribute in a model with all the other, more straightforward attributes, would dilute the subtle differences between the names of two instances.

#### 2.2.1 *Classification Models.*

(1) Doc2Vec-trained Logistic Regression Model: We chose to encode the name attribute of each project using Doc2Vec in order to preserve ordering and semantic information from the project's name. Rather than a simple bag-of-words method, we chose this approach because we assumed similar words being used in a project's name could correlate similarly with the success of the project – e.g., that projects using similar buzzwords could have a closely related outcome. In addition, we assumed the ordering of the words could serve as a correlate to grammatical correctness, which could also affect the project's outcome. We follow the approach of Le and Mikolov by combining vectors created using distributed bag of words and distributed memory [3], as we observed slightly greater accuracy on testing data by training our logistic regression classifier on the concatenated vector versus that created by the distributed memory model alone.

(2) Multi-Layer Perceptron Model: Using a multi-class classifier is mainly useful for the probability distribution of the output. When using ensemble techniques, the probability will show how confident the model is, and gives a better understanding on how to weigh its choice. We use one-hidden layer for simplicity, as training deep neural networks increases training time as well as computational power. Additionally, the ReLu (Rectified Linear) activation function was used in that hidden layer as it tends to be more computationally efficient when compared to the sigmoid activation function. The model was built using Keras, and then a Scikit-Learn wrapper was used in order to have all of the functionality of the Scikit-Learn Library. For our loss function, we used Categorical Cross Entropy, also known as Softmax Loss, as it is the default for training Multi-Class Classification models. Finally, in order to update our weights, we used the Adam Optimization algorithm, as it tends to be well suited for larger datasets, and computationally efficient. With the restraints of not having powerful computing resources, we tried to implement a light-weight neural network, and many of our choices for the structure of the neural network are reflected in this approach.

#### 2.2.2 *Ensemble Classification.*

(1) Weighted Probability Averaging: We chose to use weighted probability averaging as our method for ensemble classification because we wanted to combine the strengths of each classifier, with each being trained on a different part of the dataset's features. Other ensemble methods such as multi-staging or reverse multi-staging simply operate in order on the classifiers, passing the decision to the next classifier in

the chain depending on the previous classifier's response. This approach does not evenly consider the outputs of the classifiers, and in some cases would only result in one classifier's prediction being taken into account. Especially in our multi-class context, these simple staged approaches would not work as well: by combining the multiple class probabilities output by each classifier, we better integrate the complexities of the prediction space. Additionally, WPA allowed us to easily experiment with the weights of each classifier (in effect, how much we should trust each) and weight a lower-accuracy model less, while still taking its prediction into account.

*2.2.3 Tuning Hyperparameters.*

(1) Learning Rate: The learning rate is extremely important, as too small of a learning rate can make the model achieve a local minima, which would be detrimental to its accuracy. However, using too large of a learning rate would result in unstable training.
(2) Epochs: Varying the number of epochs is very important, as it allows us to train the model under different conditions. Increasing the epoch size enables a greater accuracy, but preventing overfitting, which could be caused by too much of an increase, is also a key concern, as overfitting leads to a deceptive report of performance on training data, while underperforming on test data.
(3) Inverse of Regularization Strength: Varying the C value is relatively important to preventing the logistic regression model from overfitting to the training data: though this results in a lower accuracy on the training data, it prevents the model from overfitting to the specifics of the data provided during training and subsequently results in a higher performance on data outside of the training set.

*2.2.4 Model Evaluation.*

(1) Dataset: Since we had tens of thousands of datapoints, we were able to split the data 75% training, and 25% testing data. Because we trained a neural network as part of our work, which required a large amount of training data, we decided that 75/25 was a good compromise between having enough data to train our models and having enough test data to form an accurate evaluation.
(2) Evaluation Metrics: We used accuracy as to make sure that the model is scoring properly, however, since a large amount of our dataset contains 'Poor' Results, it was not the single most important metric. F1 Score, Precision, and Recall helped us consider whether the model was actually accurately classifying the dataset, or if it is simply achieving high accuracy due to the dataset imbalance. We choose to focus on weighted f1 score (defined as the average of metrics for each class weighted by the support of that class) in order to account for class imbalance in our dataset.

# 3 PLAN AND EXPERIMENT

## 3.1 Dataset

*3.1.1 Description.* The dataset we used for this task was taken from Kaggle (*Kickstarter Projects*, created by Mickaël Mouillé) [5], and contains 378,661 entries corresponding to Kickstarter projects that were completed or underway from 2009-04-21 to 2018-01-02.

Each entry in the dataset contains these attributes:

- Internal Kickstarter ID
- Name of the project
- Main category of the project
- Subcategory of the project
- Currency the project accepted contributions in
- Date of the project's deadline
- Date of the project's launch
- Monetary goal of the project
- Amount pledged towards the project
- The state of the project at the time of collection (one of "successful", "failed", "canceled", "undefined", or "live")
- Number of backers
- Country the project originates from
- Conversion of the pledged amount into USD (by Kickstarter, and by Fixer.io)
- Conversion of goal amount into USD (by Fixer.io)

As is to be expected, some of the records are missing data for at least one of these attributes. In particular, the conversion of the pledged amount into USD by Kickstarter was missing from 3,797 entries in the dataset; however, none were missing the goal value computed/converted by Fixer.io.

*3.1.2 Preprocessing.* We performed several preprocessing strategies on our dataset, including feature creation, feature reduction, filtering of records from the dataset, and cleaning of data.

The metric we used as the class label, or target attribute, that our classifier attempts to predict is the percent by which a project is able to raise funds in comparison to its goal amount, using the USD conversions of the project goal and amount pledged. Rather than simply telling us if a project is predicted to meet its goal or not, this allowed us to further determine specifics such as how close a failed project was to success, or how much extra funding over the goal was a successful project able to raise.

In order to best determine the extent to which a project will succeed or fail, rather than trying to predict the exact amount of money over/under its goal a project will receive (i.e. with a continuous value output), the "success" of a project was discretized, based on the ratio between the amount of money pledged to it and the listed goal ("amount pledged" / "goal"). To discretize this "percent goal raised" attribute, a K-means clustering was performed, with different values for the hyperparameter "k" being tested.

The K-means clustering was used initially to discretize the instances, and some tuning of the hyperparameter 'k' was performed to find the optimal number of "bins"/clusters to make in order to best divide the instances based on "percent goal raised". However, the clusters created by the algorithm were detrimentally unbalanced. Of the 5 clusters created from the entire dataset, one bin contained 99.9% of all the instances, which essentially rendered discretizing useless, since the model would "learn" that almost every instance maps to that one dependent variable.

To address this, we instead manually discretized the attribute into the following bins:

- Below 25% of goal raised [Poor]

- Between 25-75% of goal raised [Moderate failure]
- Between 75-100% (exclusive) of goal raised [Nearly successful]
- Between 100-125% of goal raised [Successful]
- Greater than 125% of goal raised [Very successful]

This formed a more even distribution, and allowed meaning to be assigned to each of the bins, in that they form clean categorical separations of the amount of success of a project.

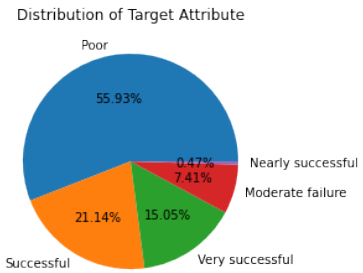


**Figure 3: Distribution of discretized percent success attribute**

In addition to creating this discrete ordinal attribute, we combined the deadline and launched attributes into a single integer "runtime" attribute, marking the number of days the project was active.

Since several instances were missing some number of attributes, and we had a considerably large enough dataset to work with, we dropped any instances/projects that were missing values for one or more attributes. Additionally, we removed records in which the goal was listed as less than 10 (US) dollars, under the assumption that there were a considerable number of projects created as jokes or for purposes other than seeking backing for a legitimate project. Most projects that are of a noteworthy scale require much more starting capital than just ten dollars. Additionally, the fact that such a goal is so small would make it more likely to get funded, as the average person is much more likely to throw 10 dollars towards some project than they are to throw 100 dollars or more to the same project.

To clean the name field before its transformation, we removed from the name the suffix "(Canceled)" where it was found. This was in order to preserve the utility of those records which had a status of canceled. Projects within the dataset have likely been canceled for several reasons, and leaving the suffix in their name would pollute the training of the classifier that operates on the name field, as it would learn that "(Canceled)" correlates with a project having low success, rather than learning based on the project name itself.

*3.1.3 Preprocessing for Multi-Layer Perceptron.* Preprocessing for the Multi-Class Classifier consisted of all of the preprocessing strategies mentioned previously, however in order to accurately train the multi-layer perceptron, the dataset needed to be balanced. Roughly 56% of the training data contains a 'Poor' Result, and only about 0.47% of the dataset contains 'Nearly Successful' Projects. While we could not completely balance the dataset, we used under-sampling on the 'Poor' Results, only taking about 80,000 of the datapoints, instead of the roughly 157,000 datapoints that were in the training dataset. This slightly balances the dataset, resulting in a more even distribution, with the 'Poor' Results only accounting for about 44% of the dataset. Additionally, in order to train the network, we needed to have the predicted output (y value) to contain five values, one for each of the neurons within the output layer of the model. To achieve this, we used Label-Encoding. Additionally, since most of the data within our dataset contained string values, they needed to be converted in order to pass through the model. Therefore, we also label-encoded the values that were contained within the Category, Main-Category, Currency, and Country columns.

## 3.2 Hypothesis

Initially, one assumption was that if a project was fully funded (achieved its goal), it would have a significant chance of continuing to accumulate extra backing as time went on. However, as pointed out by one article that also studied the factors that contributed to a project's success, most projects that do meet their goal tend to hit a plateau in funding; an investor would not be as inclined to donate (unless they have some special interest or received something in return for an investment) past the goal [2]. Elaborating on that observation further, the same article claimed that most Kickstarter campaigns resulted in one of two outcomes: they either raised significantly less than the goal (assumedly because prospective investors did not want to invest in something that was not even close to its goal), or reached their goal but did not earn any more past that point [2].

Working off of those assumptions, we predicted that a longer time frame (from conception of the project to deadline) would provide marginally better chances of a project reaching its goal (not necessarily better chances of it *exceeding* its goal), but that the benefit will dwindle after some point, in that a project running for 61 days will not have significantly greater success than a similar project that is open 60 days.

The listed goal of a project, on the other hand, is much more likely to have some relation to the success of a project. A large project being proposed by a startup company with a much higher goal fundamentally has to incur more investors and investments than a smaller, low-budget passion project. Furthermore, since the degree of success is measured in proportion to the amount needed to meet that goal, instances with higher goals have to raise more than those with smaller goals in order to reach the same level of success (for example a project with a goal of $100 has to raise $125 to exceed its goal by 25%, while a project with a goal of $500 has to raise $625 to exceed its goal by the same percentage). As a result of this, there most likely is an inversely proportional relationship between a project's goal and the degree of success, in that a project with a loftier goal is much less likely to achieve success, and a smaller goal makes it more likely a project will succeed.

We expected that creating a classifier based off of these attributes (along with others from the dataset) would allow us to correctly predict whether a Kickstarter project will succeed or fail early on in its fundraising campaign, and also expected to be able to predict the degree to which a project was a success or failure (albeit less

accurately or consistently then simply predicting success versus failure).

## 3.3 Experimental Design

*3.3.1 Experiment Description.* The experiments we performed to test our hypotheses were a series of evaluations of our trained models on a testing dataset that we held out from the original Kickstarter dataset. To create the training and test datasets, we split the original dataset into 75% reserved for training and 25% reserved for testing, using stratified sampling to preserve the relative class frequencies in both the training and test sets. We decided to use stratified sampling because of the large class imbalance in the data, with the frequency of the "Poor" class dwarfing that of the other target classes.

Our experiments were designed to answer the question of whether it is possible to build a relatively accurate model to predict a crowdfunding project's success before it is launched. Towards this goal, we trained our models and evaluated them on the test dataset on their accuracy, weighted f1 score, as well as f1 scores for each target class. Because of the stratified sampling performed, we felt that this evaluation would serve as an accurate analysis of the models' ability to classify different groups of projects. Because neither precision nor recall are much more important than the other in terms of predicting a project as successful, we primarily looked at the weighted f1 score of each model to determine its performance, with this metric taking both measures into account.

Because our final goal was to attempt to build an ensemble classification method, the baselines we compared the ensemble classification against were the evaluations of both contributing models on their independent tests. We compared the performance of the ensemble classification against that of the logistic regression and multi-layer perceptron classifiers to determine whether the ensemble approach was more successful than that of either model alone.

*3.3.2 Building Doc2Vec-based Logistic Regression.* Steps to build the model that classifies project success based solely off of the "name" attribute (these steps were influenced by the approach taken by Susan Li [4]):

(1) Remove stopwords from the "name" string of each instance, tokenize the name string, and perform the rest of the preprocessing described in the *Preprocessing* section as necessary
(2) Provide the "name" of each instance as input to the Doc2Vec models to produce a vocabulary of words used over all documents, and a document vector for each instance
(3) Train the Doc2Vec model based off of a subset of the dataset, designated the training dataset
(4) Create a function that produces concatenated feature vectors – an encoding/representation of the names (and any keywords they may contain) and any commonalities between them, using distributed memory and distributed bag of words models – when given the training data instances and the trained Doc2Vec models as input
(5) Use the previously mentioned function to create feature vectors for each document in the training dataset
(6) Train a logistic regression classifier using the feature vectors, in order to determine what class (or what discretized "success" category) the instance corresponding to that feature vector belongs to
(7) Feed the test dataset (and Doc2Vec models) to the function that creates feature vectors (from step 4) to produce feature vectors for the test data
(8) Input the test feature vectors to the logistic regression classifier (that was trained in step 6), to make predictions of what class each test instance belongs to
(9) Compare the predictions made on the test dataset to their actual classifications

*3.3.3 Building Multi-Layer Perceptron.* The following details the steps we took to build the Multi-Layer Perceptron.

(1) Preprocessing and label encoding the data. This allows us to smoothly pass through our data in order to train the model.
(2) Designing the model using Keras. In this step we created a method that designs the structure of the model, and returns that model as a Keras object. Within the method the model is also compiled with the details on the loss function, optimizer, and learning rate.
(3) Wrap the model as a 'KerasClassifier' in order to use scikit-learn methods. Keras has a wrapper class that allows the user to then call scikit-learn accuracy metrics, and since we had more experience using scikit-learn, we used this wrapper function. The KerasClassifier wrapper is also where hyper-parameters are set, such as number of epochs and batch size.
(4) Train and test model. After creating the Classifier, we trained and tested the model. We then used sklearn's classification report to display our accuracy, F1 score, and more. We used this information to determine which hyper-parameters to tune.
(5) Tuning Hyper-Parameters and retraining the model. This step includes changing hyper-parameters such as the number of epochs, in order to determine if we could achieve better results.

*3.3.4 Ensemble Creation and Tuning.* To create the final ensemble predictions, we built a function that would take the probability estimates predicted by each model and weight them according to arbitrary parameters passed in, then average the weighted predictions and output as the final prediction the class which had the highest averaged probability for each sample. We then used this function to evaluate the performance of the weighted probability average based ensemble prediction on the test dataset, initially using arbitrary weights assigning 40% weight to the Doc2Vec-based logistic regression model and 60% weight to the multi-layer perceptron. We then performed a search over the possible model weightings (from 5% logistic regression model weight to 95% weight, stepping by 5%) to determine the weights corresponding to the best performance of the ensemble prediction on the test dataset, and compared this to the arbitrarily-weighted ensemble predictions.

# 4 RESULTS

## 4.1 Results

As outlined in the Problem Statement, the main goal of our model was to predict how projects will fare by the end of their fundraising period, using attributes available before the launch of the corresponding Kickstarter campaign. Our first experiment was wholly concerned with the using the title of the Kickstarter project alone. The model created to perform natural language processing for this name attribute produced vectors for each title that were then used to train and create the logistic regression predictions seen in Figure 4.

These results, as well as the results discussed in Figures 5-7, are separated by each class a project could fall under: Poor, Moderate Failure, Nearly Successful, Successful, and Very Successful. Below are the model's F1-scores for each, which analyzes how well this model performs in predicting each category. This first model obtains predictions with a weighted F1-score of 50%.

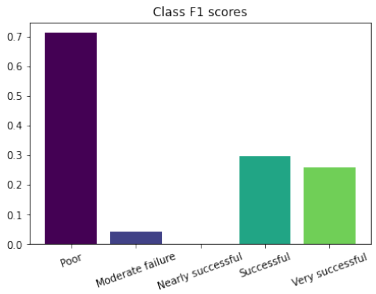


**Figure 4: Logistic Regression Performance from Project Titles**

Our second experiment was based solely on the additional data that the Kickstarter dataset provided for each project, including the time a campaign has set for fundraising, its goal amount, and category. The data was compiled into a model and used to build a multi-layer perceptron, the performance of which is displayed in Figure 5, again separated by categorical prediction performance, which graphs the predictor's performance (F1-score) in each category. The perceptron had an F1-score of 46% weighted over all categories and an accuracy of 49%.

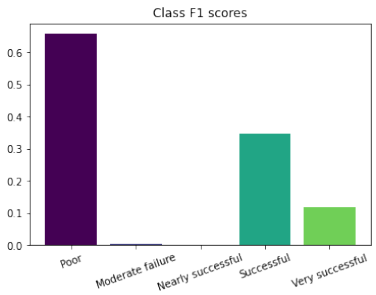


**Figure 5: Additional Attribute Perceptron Performance**

After creating the first two models, we combined our predictors into an ensemble using arbitrary weights of 40% for the language model and 60% for the multi-layer perceptron. The resulting ensemble ended up with a total weighted F1-score of 50.8%, and categorical accuracy of 58%, as seen in Figure 6. This means that around 60% of the projects had their success, and degree of success/failure, correctly predicted.

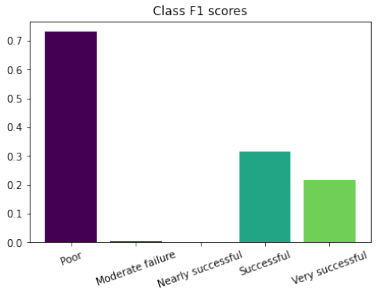


**Figure 6: Arbitrarily Weighted Ensemble Performance**

The final experiment took the title model and the additional attribute perceptron and, rather than using arbitrary weights, found the ensemble weighting that produces the greatest possible performance. Figure 7 presents this model's performance, which includes a weighted f1-score of 51%, and a non-categorical (combined accuracy - only predicting success or failure) accuracy of 68%. Since all four predictors are analyzed using the same metrics, we can make in depth comparisons among them as we evaluate and discuss our results.

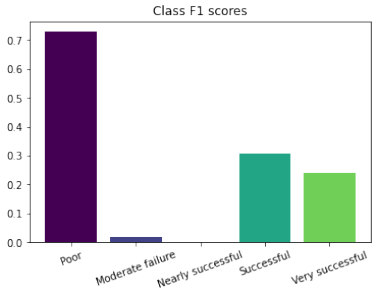


**Figure 7: Final Ensemble Performance**

For comparison, we have also included for each model the weighted F1 score, accuracy, and combined F1-score and combined accuracy produced by grouping the target classes and predictions into a binary success or failure.

| | Weighted F1 | F1 (Binary "Success/Fail") | Accuracy | Combined Accuracy (Binary "Success/Fail") |
|---|---|---|---|---|
| Multi-Layer Perceptron | 0.4598 | 0.4918 | 0.4934 | 0.6158 |
| Logistic Regression Classifier | 0.5027 | 0.4216 | 0.5503 | 0.6578 |
| Ensemble Classifier | 0.5099 | 0.4170 | 0.5732 | 0.6752 |

**Figure 8: Comparative Table of Metrics for Each Classifier**

## 4.2 Discussion

The results from the first experiment and Figure 4, upon first glance, appear to be rather poor if you look at only the sub 30% f-scores for the "Successful" and "Very Successful" categories, but when the degree of success or failure is not considered, the accuracy increases by over 10% for just the title prediction. And, this continues for each of the tests, with similar accuracy increases for each, so predictions being marked as inaccurate, are still predicting failure or success correctly a majority of the time.

The regression model for the "name" feature performed surprisingly well on its own, as its F1-score and accuracy were both over 50%. While never specifically mentioned in the hypothesis, when planning our experiments, we anticipated that project titles may play a significant role in the success of fundraising campaigns, but the results of this model suggest that we could have considered their names even more heavily than we initially thought.

The predictor for the additional attributes in the data did not perform as well as we expected it to, with both its F1-score and accuracy below 50%. Before starting our experiments, we assumed that the perceptron would easily outperform the text model, but we can definitively conclude that this assumption was incorrect. As Figure 5 shows, the perceptron performed much worse on predicting "Very Successful" projects versus "Successful" ones relative to the other models. This is perhaps due to the lack of additional attributes in the dataset, but more testing would be needed to determine what exactly causes the model to perform poorly, and with a 62% combined accuracy (only considering success or failure, rather than degree of either), the model is still usable.

The arbitrarily weighted ensemble and the final ensemble both performed similarly well, with the final ensemble having a slightly better (by .2%) F1-score of 51%. Both ensembles outperformed each model taken alone. The final ensemble still struggled to distinguish between levels of success and failure, but improved upon the other models with a 57% accuracy. The ensemble did, however, achieve a combined accuracy score of around 68%, which confirms our hypothesis that we would be able to create accurate predictions of success or failure for projects and that we could also, less accurately, characterize the degree of success or failure. We were, however, unable to make conclusions on the components of our hypothesis regarding how much of a factor specific attributes contributed to the success or failure of a project. To reach such conclusions, more experiments would be necessary, including creating sample projects, running them through the model, and analyzing differences in results.

# 5 CONCLUSIONS

## 5.1 Lessons Learned

One of the major contributing factors to the results of the experiment was the unbalanced distribution of the training and testing datasets, which both contained few instances of "Moderate Failure" and "Nearly Successful", and an overwhelming number of "Poor" instances present in both. Even though the training dataset had a similar distribution of data (across the 5 class labels) to the test dataset, the multi-layer perceptron rarely even labelled an instance as "Nearly Successful" or "Moderate Failure", and most of the time it did predict either, it was incorrect. The most likely explanation is that there was not enough instances of certain categories in the training data for the model to learn what differentiated them from others. However, there is also the fact that, since the test dataset was similar to the training dataset, there was not enough tests performed on the model to see if it was capable of correctly classifying an instance as "Nearly Successful" or "Moderate Failure", compared to the plethora of test cases for the other categories. Even though stratified sampling was employed, and that the testing dataset was purposefully curated to put less emphasis on "Poor" instances (the most common), the imbalance of the dataset still impacted our model's performance and evaluation.

The dataset itself tracked many relevant attributes for (most of) the instances, however there are a variety of other factors that very likely contributed to the success or failure of a project that were not recorded. After preprocessing, some of the original given attributes were combined (i.e. "Date launched" and "Deadline" were combined into a single "Runtime (days)" attribute), some were removed for being unnecessary (for example, "Goal" was removed, in favor of using "US Dollars Goal (Real)" for the sake of using the same unit of currency), and any attribute that a new project would not have at start ("Backers" and "Amount Raised") was removed so that predictions could not be made based off data that was not available to the model at start of campaign, which left us with only 6 usable, unique attributes for the perceptron (not counting "Name" which was used solely by the regression model). Some other attributes that would have been of note include the number of updates a project underwent while available for backing, the number of prior projects the creator of a project had pitched (and if those other projects succeeded or not, in order to factor in if some project had "credible" names attached to it), the amount individual backers contributed to a project's donations (to determine if the majority of contributions came from a few wealthy donors versus a large group of small donations), and if there were any incentives or rewards offered to donors in exchange for contributing a certain amount (such as early copies of a product, promotional material, and so on). Additionally, the Doc2Vec-trained logistic regression classifier that was trained on names could also have worked with the "Description" section of a project, since it would have larger document vectors (in terms of relevant information) for each instance, so there would be more data to draw conclusions from.

Given more time for analysis, it could be possible to determine which (if any) of the attributes (Name, Runtime, Goal, etc) are the dominant factors in deciding if a project will fail or succeed (and to what extent). While it is possible that no attribute in isolation plays a key part in deciding success, the existence of many "Successful/Very Successful"-performing projects and many "Poor"-performing projects, with few in between, could indicate that there is some commonality in the projects at either extreme (instead of random chance deciding if a project makes its goal).

# 6 LINK TO PROJECT CODE

The project code and results can be found in the form of a Jupyter Notebook in the repository at this address: https://github.com/gavinhardin/kickstarter-ensemble.

# 7 MEETINGS

- April 6, 2021 - 1:00 PM
  - Attendees: gchardin, jbroddy, astewar6, ndeggens
- April 7, 2021 - 1:00 PM
  - Attendees: gchardin, jbroddy, astewar6, ndeggens
- April 8, 2021 - 2:45 PM
  - Attendees: gchardin, astewar6, ndeggens
- April 8, 2021 - 7:30 PM
  - Attendees: gchardin, jbroddy, astewar6, ndeggens
- April 13, 2021 - 4:50 PM
  - Attendees: gchardin, jbroddy, astewar6, ndeggens
- Recurring Tuesdays and Thursdays Before Class until April 29, 2021 - 2:00 PM
  - All in Attendance

# REFERENCES


[1] Rania Albalawi, Tet Hin Yeap, and Morad Benyoucef. 2020. Using Topic Modeling Methods for Short-Text Data: A Comparative Analysis. *Frontiers in Artificial Intelligence* 3 (2020), 42. https://doi.org/10.3389/frai.2020.00042

[2] Fernandez-Blanco, Joaquín Balsera, Vicente Montequín, and Henar Morán Palacios. 2020. Key Factors for Project Crowdfunding Success: An Empirical Study. *Sustainability* 12 (01 2020), 599. https://doi.org/10.3390/su12020599

[3] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. arXiv:1405.4053 [cs.CL]

[4] Susan Li. 2018. https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4

[5] Mickael Mouille. 2018. *Kickstarter Projects Dataset.* Retrieved March 23, 2021 from https://www.kaggle.com/kemical/kickstarter-projects?select=ks-projects-201801.csv

[6] David Opitz and Richard Maclin. 1999. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research* 11 (aug 1999), 169–198. https://doi.org/10.1613/jair.614

[7] Suppawong Tuarob, Conrad S. Tucker, Marcel Salathe, and Nilam Ram. 2014. An ensemble heterogeneous classification methodology for discovering health-related knowledge in social media messages. *Journal of Biomedical Informatics* 49 (2014), 255–268. https://doi.org/10.1016/j.jbi.2014.03.005

[8] Ye Wang, Zhi Zhou, Shan Jin, Debin Liu, and Mi Lu. 2017. Comparisons and Selections of Features and Classifiers for Short Text Classification. *IOP Conference Series: Materials Science and Engineering* 261 (oct 2017), 012018. https://doi.org/10.1088/1757-899x/261/1/012018