# SEAS 6401 Final Project - Predicting Excitement for DonorsChoose.org

Michael J. Salceda
*School of Engineering and Applied Science*
*The George Washington University*
Washington, D.C., USA
msalceda@gwu.edu

***Abstract*—DonorsChoose.org put up a challenge on Kaggle in May, 2014: predict "exciting" essays using their data in order to improve upon their user experience and drive more engagement. Many approaches to solving this issue involved using many features derived from the data files provided; this reliance on other features from other data files creates a burden when eventually bringing a successful model to production as there needs to be extensive work in storing, preprocessing, and using that data. In my project, I create a model that utilizes only the raw essay text and NLP methods to cut down on the overhead of additional files while still getting a high level of performance. My resulting model performance indicated that there may not be enough information contained in the essays alone to create an effective machine learning model.**

***Keywords—natural language processing, Kaggle, transfer learning, bag-of-words, word embeddings***

## I. Introduction

DonorsChoose.org is a charity organization where people from around the world can come together to help teachers and students in need. Teachers can use the DonorsChoose.org platform to put forward project ideas that they believe will help enhance their own students' education, and people can freely choose to donate money to one or several of these project ideas. Each project idea has a goal amount to be reached and if that goal amount is reached, DonorsChoose.org will take all the donations to that project and send the money/supplies in the project to the teacher.

With all these projects on the website, there is one issue: how can people choose which ones to donate to without having to read through each project page? According to DonorsChoose.org FAQ page, "over 70% of projects reach their funding goal", but of the 30% or less remaining projects, some of those could have been exceptionally important or unique.

There was already a Kaggle competition held with this very data, but many of those notebooks utilize non-text related features such as donations, teacher demographics, and project information. The question I wanted to answer was this: can I identify exciting projects solely from teacher essays?

My main data source was the KDD Cup 2014 Kaggle competition [1]. While there are multiple datasets provided, I focused my analysis on the teacher essay dataset since the bulk of relevant project information was in those narratives. This is a problem in the Natural Language Processing domain so I used NLP-focused Python libraries such as NLTK and spaCy. All the processing and modeling was done in a Google Colab environment with some additional packages installed. My goal for the end-product of this project was to have a machine learning model that would be able to identify which projects are "exciting" for DonorsChoose.org to promote to their users.

## II. Methodology

### *A. Business Case Evaluation*

With the aid of machine learning, DonorsChoose.org would be able to effectively drive more user participation towards projects that need it, and in turn, potentially bring more teachers to the platform which brings more exposure and value to DonorsChoose.org.

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

Right now, for a project to be considered "exciting", it must have fulfilled all following conditions [1]:

1. The project must be fully funded.
2. The project had to have at least one teacher-acquired donor.
3. The project had to have a greater-than-average comment percentage among donors.
4. The project had to have at least one "green" donation.
5. The project had to have one or more of the following:
    a. Donations from three or more non-teacher-acquired donors
    b. Had one non-teacher-acquired donor give more than $100
    c. Had at least one donation from a "thoughtful donor"

As evidenced from the conditions above, all these conditions can only be checked *after* the project has been completed, meaning a project can only really be promoted to the front page if other users have already found it.

If this model were to be brought into production, it would ideally be used as a model to drive the promotion of teacher projects on the front page. When a teacher submits a project, the essay would be submitted to the machine learning model and, based on the classification of whether it is exciting or not, it would be pushed to the front page for immediate exposure.

The main language for this would be Python and, with the mature machine learning ecosystem already existing in Python, it would be relatively easy to deploy the model for use on the site; however, more development effort may be needed on the backend to build out the necessary infrastructure for storing the model, keeping track of model performance, and model training.

Data-wise, the data is readily available since they are just essays from the various projects on the site. Depending on how the backend is currently built, it should ideally be easy to pick up the existing essay data from whatever storage solution is in place and send those essays to the model for training.

*B. Data Identification*

To create a model as posed previously, the data I need would have to be labeled narrative data. I would need to have the raw teacher-submitted essays along with a binary label showing whether DonorsChoose.org found the project to be exciting or not.

*C. Data Acquisition & Filtering*

Acquiring the data for this task was straightforward as the data was already provided by DonorsChoose.org for the Kaggle competition. Since this project's focus is on processing the essays, I only used two CSV files: `outcomes.csv` and `essays.csv`. There were other files provided by DonorsChoose.org that contained information on project details, donation details, etc. I found that data to have less relevance to the problem at hand since I wanted to see if using essays had enough information to determine an exciting project. In the end, the data I used consisted of 619,326 records from the `outcomes.csv` file and 664,098 essays from the `essays.csv` file.

*D. Data Extraction*

For data extraction, I used the Python `pandas` library to load in the CSV files. The man issue I had to keep in mind here was the data size. The `outcomes.csv` file was small so there was no issue in loading in that data. The `essays.csv` file, on the other hand, was about 1 GB in size. The reason for this was because the essays contained in the file were long and there were other fields in the `essays.csv` file that also contained strings. To combat this, I just loaded in the two columns I needed for joining and analysis: the project ID field and the raw essay text.

*E. Data Validation & Cleansing*

As part of the validation process, after loading in the file, I determined the shape (number of rows and columns) of the file to see if it matched with the raw file. If it did not match, then that indicated there was an issue loading it in.

I initially started out using PySpark because I wanted to utilize the parallel nature of Spark to process the data. This, however, brought to light an issue with loading in the `essays.csv` file with PySpark. Within the essays, there were strange newline characters (\r\\n). Because of this, when

PySpark would read in the CSV, it would actually break up the essays across multiple lines, inflating the number rows in the final Spark DataFrame – i.e., instead of 664,098 rows, I found over 900,000 rows in the Spark DataFrame.

When I switched to just using `pandas` to load in the data, I found that `pandas`, fortunately, had no issue loading in the essays properly. I stuck with using `pandas` throughout my work for handling the data instead of PySpark.

### *F. Data Aggregation & Representation*

Once the data from both files was loaded successfully, I created the final dataset by joining the outcomes and essays on the key common to both files: `projectid`. This ID uniquely identified projects across the site so it made it easy to determine which essay had which label. The final dataset had three columns: the project ID (`projectid`), the raw essay text (`essay`), and the "exciting" binary label (`is_exciting`) - 1 for "exciting", 0 otherwise. After joining, I had 619,326 records since not all `essays.csv` project IDs were found in the `outcomes.csv` file.

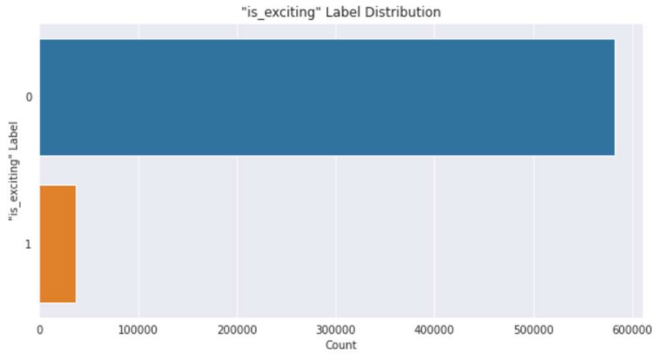


*Fig. 1 Label Distribution. The distribution of exciting vs. non-exciting essays is very imbalanced; almost all essays are not exciting.*

### *G. Data Analysis*

Analysis of the data utilized the `seaborn` visualization library and `pandas` for data manipulation. I focused my analysis on the two columns I had: the label column and the essays column.

### *H. Data Visualization*

As previously mentioned, visualization was done using `seaborn`. I also did use the `wordcloud` library to generate word clouds of the essays to see if there were any interesting words emphasized.

### *I. Utilization of Analysis Results*

After doing analysis and visualization of the results, I was able to form a strategy for processing the data and modeling. Because of what I found, I settled upon a preprocessing scheme which is explained in more detail below.

## III. Exploratory Data Analysis

### *A. `is_exciting`*

One of the first things I wanted to look at was the label distribution of `is_exciting`. I needed to understand what I was targeting for my machine learning model. When I plotted the label distribution (Fig. 1), I found an interesting insight: this is was a highly imbalanced problem.

Because of the class imbalance, when assessing model performance, I cannot just use accuracy. Using accuracy as a performance measure with imbalanced data, you can get a misleading picture of model performance. For example, let's say you have some imbalanced data, and you ran it through a machine learning model. The results you got are as follows:

| ***Features*** | ***Actual Label*** | ***Predicted Label*** |
|---|---|---|
| … | 1 | 0 |
| … | 0 | 0 |
| … | 0 | 0 |
| … | 0 | 0 |
| … | 0 | 0 |

Using accuracy as a performance metric, we would find that the model was 80% accurate since it correctly predicted "0" four out of five times! This is extremely misleading since we can see that the model only predicted "0" across the board. To rectify this issue, I used the F1 score as a measure of model performance.

The F1 score is the harmonic mean between the precision and recall scores of the model. Think of precision as: "of the predicted class, how many were actually correct" and recall as: "of the true class, how many were found".

In my example above, the precision of label 0 would be 80% - four of the five rows predicted to label 0 actually belonged to label 0, and recall of label 0 would be 100% - four of the rows that actually belonged to label 0 were predicted to label 0. For label 1, precision would be 0% - no labels were predicted to label 1, and recall would be 0% - the one row that belonged to label 1 was not predicted to label 1.

We can calculate the F1 score for each class and then average them for the final model performance by finding the harmonic mean between the two scores. For label 0, the F1 score turns out to be approximately 89% and the F1 score for label 1 is 0%, so the final F1 score for the model would be approximately 44%. Now, we see that the model did not perform well.

*B.* `essay`

The `essay` column contains the raw essays that teachers submitted as part of their projects. The first step I took here was to see if there were any empty essays. Fortunately, there were only three essays that were empty, either due to some data exporting issue from DonorsChoose.org or something else. I just dropped those essays from the dataset and had 619,323 records left.

The next step I took was to try to visualize the word frequency in both exciting and non-exciting essays. To do this, I utilized word clouds. The larger the word in the word cloud, the more it shows up in all the essays.

Based on the word clouds for both exciting (Fig. 2) and non-exciting (Fig. 3) essays, I did not notice any visible difference in frequent words.

What did jump out to me was the word "will" being emphasized due to its high frequency. Also, there are some words in the word cloud like "nMy" and "nI". This indicates to me that there needs to be some text processing to take care of stop words like "will" and cleaning up of weird things like the "nMy" words.

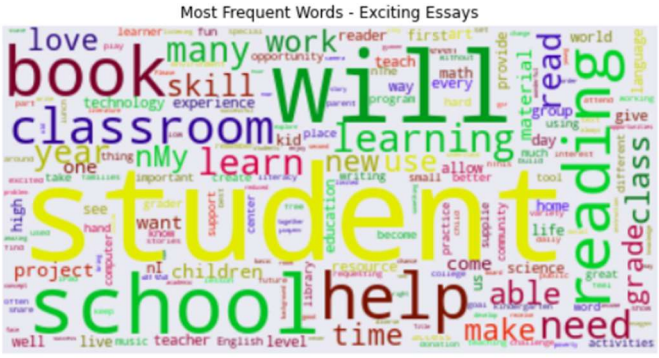


*Fig. 2 Exciting Essay Word Cloud. This word cloud shows the frequency of words relative to other words based on the word size. Interesting words to note that show up are "will" and "nMy".*

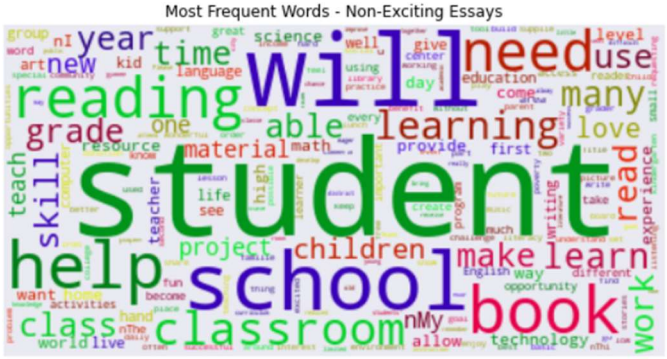


*Fig. 3 Non-Exciting Essay Word Cloud. This word cloud shows the frequency of words relative to other words based on the size of the word. Interesting words to note that show up are "will", "nMy", and "nI". This shows no real difference in word frequency with the exciting essay word cloud (Fig. 2).*

After exploring the word clouds, I wanted to see if there was any relationship between the `is_exciting` labels and the character/word counts – i.e., do longer essays tend to be more exciting?

Based on essay word counts, there does not seem to be any relationship between the `is_exciting` labels and the word count (Fig. 4). The only major difference between the two is that the non-exciting essays show a longer tail and wider range of values.

When looking at the character counts, I see the same lack of relationship between the `is_exciting` label and the character count (Fig. 5). Again, the only difference between the two types of essays are that that distribution has a longer tail and wider range of character counts.

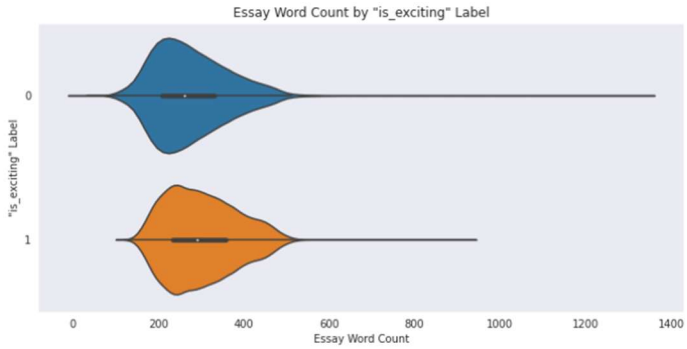


*Fig. 4 Word Count Distribution by Label. The distribution of word counts is approximately the same between exciting and non-exciting essays.*

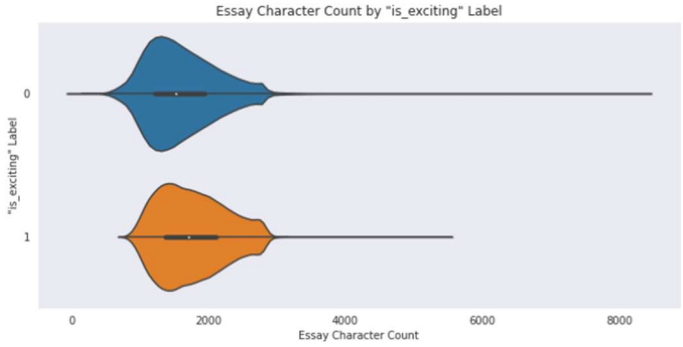


*Fig. 5 Character Count Distribution by Label. The distribution of character counts is roughly the same between exciting and non-exciting essays, like word counts (Fig. 4).*

## IV. Data Processing

Once the exploratory data analysis (EDA) was finished, I processed the essay data. Based on the word clouds from the EDA, I knew that I had to do some processing to remove stop words. I also knew there was some strange issue with "n" appearing in front of words.

When I looked at some data, I found that interspersed within the essays were corrupted carriage return characters.

Normally, carriage returns are indicated in text by the "\r\n" pattern. For some reason, when DonorsChoose.org extracted their essay data, those carriage returns turned into "\r\\n". When `pandas` parsed those characters, it ignored the "\r" like it should but read in "\\n" as "n". Therefore, additional processing is needed to find and treat those corrupted carriage returns.

Overall, the steps I took for processing the data went as follows:

1. Replace "\r\\n" in the text.
2. Lowercase the text.
3. Remove extra spaces.
4. Tokenize the text.
5. Remove punctuation.
6. Remove stop words.
7. Lemmatize the text.
8. Remove punctuation again to check for leftovers.

I used the spaCy library to help with processing the data. The biggest use of spaCy was for lemmatizing the text.

Lemmatizing is the process of reducing words to a "root" form so that they are consistent as features. For example, if there are two words like "producing" and "produced" in the text, without lemmatization, the model would treat these two words as separate words even though they are the same word in a different tense. Lemmatization would transform these two words into "produce", the "lemma" or base form of the word. There is another similar process called "stemming" in which the tense endings are removed so "producing" and "produced" would turn in to "produc", the "stem". The reason I chose lemmatizing is because it tends to lead to a better representation of the word, and it does not lead to nonsense words like "produc".

To show what the text processing has done to the essays, Fig. 6 and Fig. 7 show a couple examples of the part of the original text and the processed text.

Original Text

*Over the many years of teaching I have learned that most of my students are visual learners, especially in math.\r\\n\r\\nOne of the most difficult concepts to teach is the adding and subtracting of two digit numbers. My students sometimes do not understand what to do with "too many ones!*

Processed Text

*year teaching learn student visual learner especially math difficult concept teach add subtract digit number student understand one*

*Fig. 6 Original Essay Excerpt and Processed Essay Excerpt.*

Original Text

*I teach all subjects in kindergarten, with an emphasis being placed on learning to read. Our school incudes children that come from a broad variety of backgrounds and situations. My class consists of students at all levels; including some high needs students.*

Processed Text

*teach subject kindergarten emphasis place learn read school incude child come broad variety background situation class consist student level include high need student*

*Fig. 7 Original Essay Excerpt and Processed Essay Excerpt.*

Once the text was processed, I generated the word clouds again as shown in Fig. 8 and Fig. 9. After processing, "will" does not show up as a high-frequency word and there don't appear to be any incorrect "nMy" or similar words showing up.

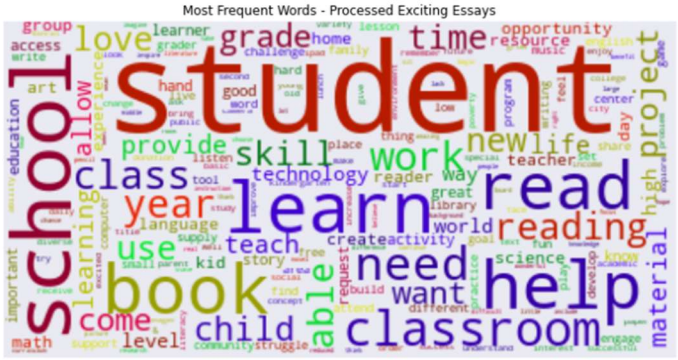


*Fig. 8 Processed Exciting Essay Word Cloud. "will" has been removed and the words are more cleaned up compared to the original word cloud.*

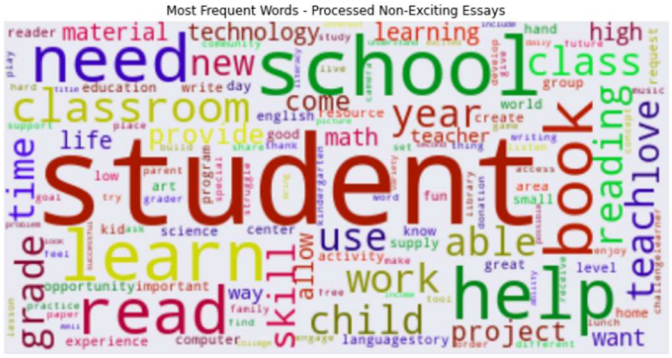


*Fig. 9 Processed Non-Exciting Essay Word Cloud. "will" has been removed and the words are more cleaned up compared to the original word cloud.*

## V. Modeling

For the modeling part of this project, I took an iterative approach over three different methods of dealing with NLP-domain problems. For the first method, I used bag-of-words for creating features. My second method explored the use of word embeddings for creating model features, and my third method utilized transfer learning for the modeling. For all the iterations, I ran hyperparameter tuning trials in order to find the best model and get the best model metrics. Within each trial, I performed three-fold cross-validation on it to ensure that I would get a more accurate picture of model performance.

### *A. Bag-of-Words*

The first method I experimented with for creating a model was doing a ***bag-of-words*** transformation of the features. The bag-of-words approach does not care about context or syntax. All the basic form of bag-of-words does is takes in a text and count how many times the word comes up. This is the term-frequency (TF) method for bag-of-words (Fig. 10).

There is another method called TF-IDF or "term frequency-inverse document frequency" for bag-of-words that does some additional calculations on top of the term frequency counts. What TF-IDF does differently is that it also calculates what is called the "inverse document frequency", i.e., the number of times a term comes up across documents. This effectively upweights words that show up often in few documents (like keywords) and reduces weights of words that show up often in a lot of documents (like stop words).

The first model I created was a baseline model using TF bag-of-words to process the text and a logistic regression model. The vectorizing process (bag-of-words transformation) and the modeling was done using `scikit-learn`. Once that baseline model was created, as previously mentioned, I ran hyperparameter tuning trials using `hyperopt` and Spark to run trials in parallel. I tested out varying parameters of logistic regression model parameters and TF bag-of-words parameters.

After hyperparameter tuning the baseline, I switched out the TF bag-of-words vectorizer to a TF-IDF bag-of-words vectorizer to see if there was any improvement over the baseline. I did the same process with this model set by running hyperparameter tuning trials across the TF-IDF bag-

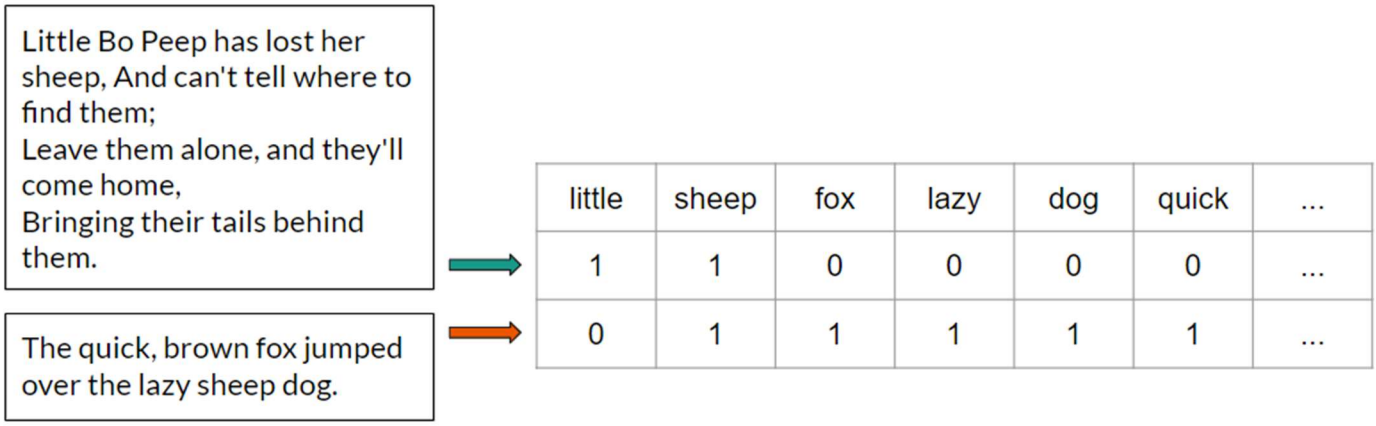


*Fig. 10 Bag-of-Words Example.*

of-words parameters as well as logistic regression parameters.

The next iteration involved switching out the logistic regression model for a random forest model. I still used a TF-IDF bag-of-words approach for vectorizing since the previous iteration (discussed above) showed that TF-IDF did bring some improvement over just a TF bag-of-words.

The final iteration involved upsampling. Due to the imbalanced nature of our labels, my hypothesis was that upsampling will help in the performance of the model. To perform upsampling, I used the `imbalanced-learn` package. I stuck with a logistic regression model along with the TF-IDF vectorizer since, based on the previous model iterations, those are what performed the best thus far. I did not upsample the test set; this was on purpose. When upsampling, downsampling, or augmenting data for the purposes of training, the augmentations should only be applied to the training data.

### *B. Word Embeddings*

For the word embedding approach of text processing, I used Word2Vec. What word embeddings do is turn texts into vectors of numbers,

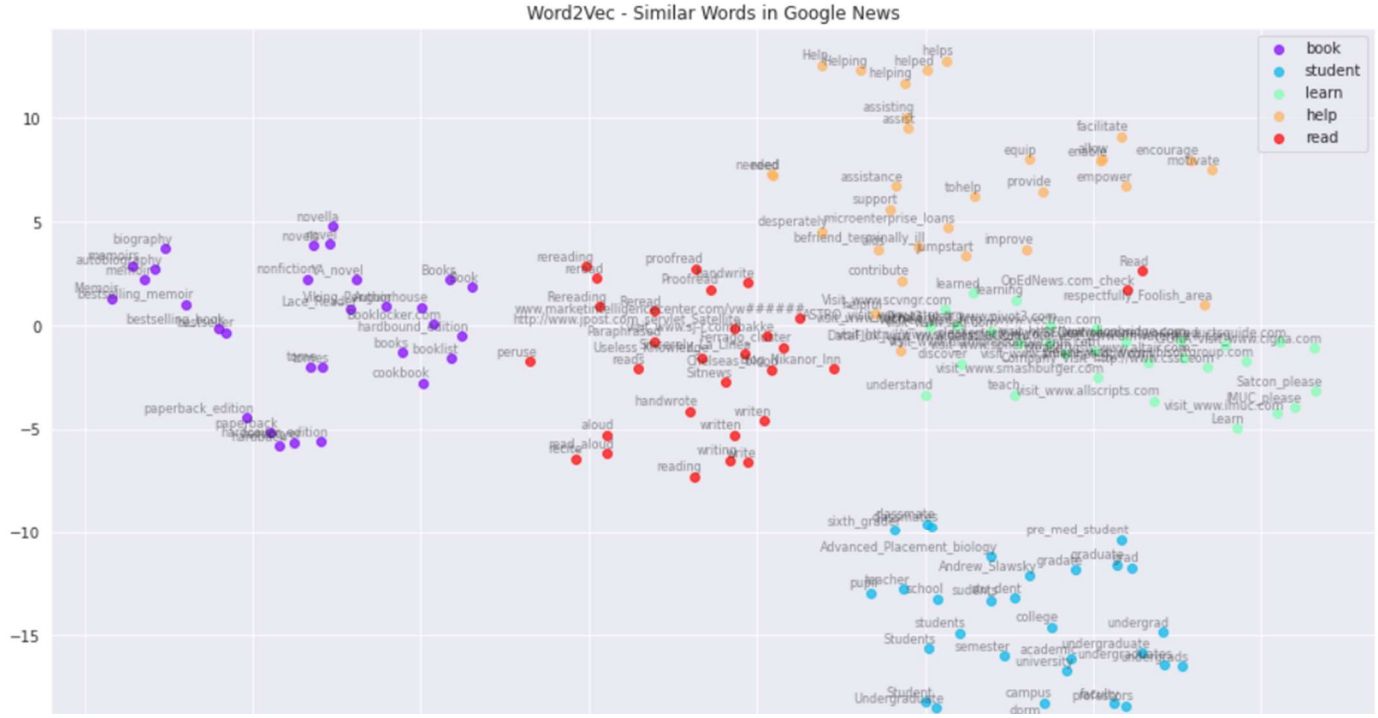


*Fig. 11 Word2Vec Text Embedding Representation. This shows the related words by color to the query terms in the legend. For example, all the words similar in meaning/context to "book" are in purple.*

similar to the bag-of-words. The difference is that the vectors are attempting to capture the context and meaning of the actual words. This means that closely related words should ideally be close together in the vector space. This can be seen in Fig. 11 where I graph the words that are closely related to the terms in the legend.

Instead of training a Word2Vec model from scratch, I utilized a pretrained Word2Vec model trained on Google News.

The first iteration of this model involved using the Word2Vec word embeddings with a logistic regression model. I followed the same training paradigm as the previous bag-of-words approach and ran hyperparameter tuning trials over the logistic regression model parameters. The second iteration of this model switched out the logistic regression model with a random forest model.

I did not perform an upsampling iteration for this approach because I found that the upsampling did not seem to help much for performance and it added a lot more time to the processing.

### *C. Transfer Learning*

The final model I tested out involved transfer learning. Transfer learning is the process of taking a large pretrained model that has been trained on a massive corpus of text and applying it to your problem. To make the transfer learning model work for your problem, instead of having to train an entire complex model from scratch, you just need to train the last couple layers. This, theoretically, allows us to leverage really good models for our problems without having to spend a lot of resources to train it.

The library I used for this was `fastai`. I utilized the built-in pretrained model which is the ASGD Weight-Dropped LSTM model trained on WikiText-103 dataset, a collection of Wikipedia articles.

Since I had an imbalanced label distribution, I needed to change the normal cross-entropy loss function to a weighted cross-entropy loss function in order to take into account the difference in distribution. I also did not do any hyperparameter tuning for this model since the run time would have been too long for the Google Colab environment to stay active.

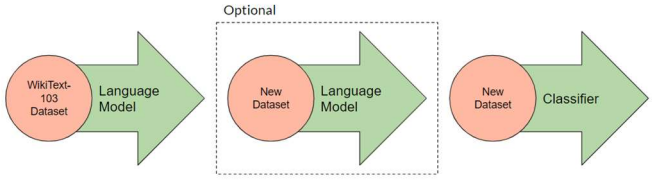


*Fig. 13 Transfer Learning Flow. For my transfer learning model, I did not train a language model (middle) on the essays to reduce on time.*

## VI. Results

The table (Fig. 12) shows the results of the various approaches and hyperparameter tuning trials. From

| | Description | Accuracy (%) | F1 (%) | Avg. Runtime (min) |
|---|---|---|---|---|
| Baseline | LR + TF | 90 | 54 | 9 |
| Bag-of-Words | LR + TF-IDF | 86 | 55 | 9 |
| | RF + TF-IDF | 82 | 53 | 7 |
| | Up + LR + TF-IDF | 92 | 53 | 10 |
| Embeddings | LR | 59 | 44 | 1 |
| | RF | 91 | 52 | 7 |
| Transfer Learning | AWD-LSTM + WCE Loss | 70 | 50 | 36* |

Acronyms: LR = Logistic Regression, TF = Term Frequency, RF = Random Forest, Up = Upsampling, AWD-LSTM = ASGD Weight-Dropped LSTM, WCE = Weighted Cross-Entropy

*This is an average per epoch. The model was trained for 5 epochs total.

*Fig. 12 Final Results. The best performing model was a logistic regression with TF-IDF vectorization.*

the results, my best model in terms of F1 was the bag-of-words model using logistic regression and TF-IDF vectorization. The highlighted fields show where those models beat the baseline.

The hyperparameter tuning results of the best model for all the iterations are show in Fig. 14 and Fig. 15. About half of the trials were sub-50 F1 scores. The other half were competitive with the baseline model. The main hyperparameter that was tuned for the logistic regression model was the C parameter. The C parameter controls the amount of regularization that is applied to the coefficients. The other hyperparameters that were tuned were for the TF-IDF vectorizer, namely the n-gram range and the analyzer.

The n-gram range determines what types of n-grams to create when vectorizing the words. I iterated through creating unigrams, bigrams, and a combination of both.

The analyzer is simply how to break up the essays: by words or by characters.

The best hyperameters turned out to be a C value of 1, the word analyzer, and an n-gram range of (2, 2) or just using bigrams.

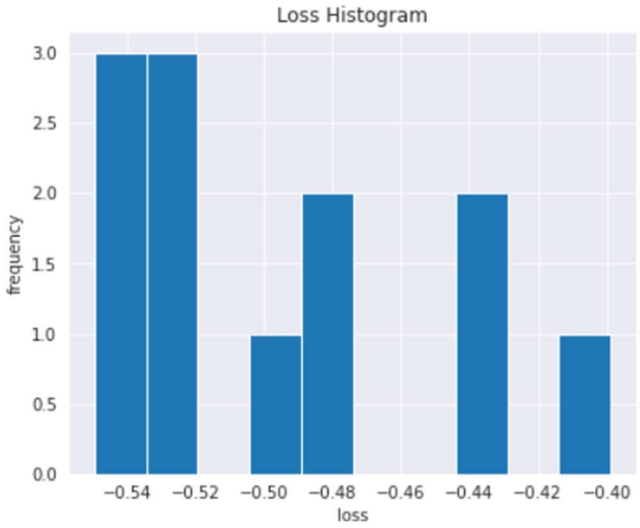


*Fig. 14 Loss Histogram of Logistic Regression with TF-IDF. The losses are negative because the `hyperopt` library minimizes an objective function. Since we wanted to maximize F1, I had to convert the scores to negative.*

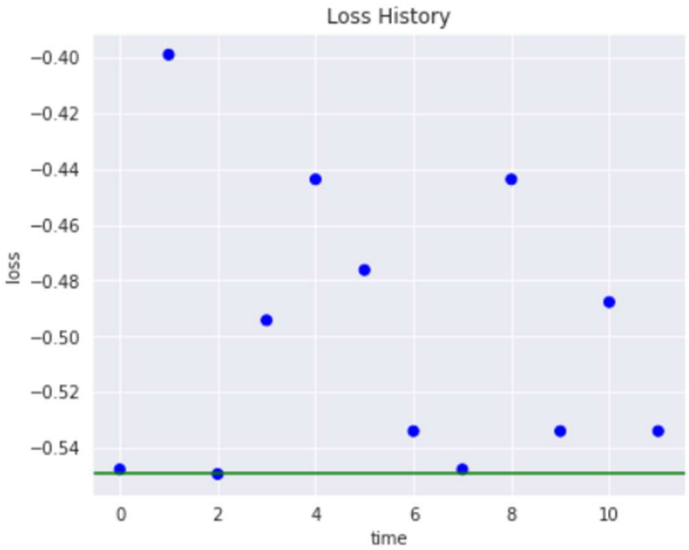


*Fig. 15 Loss History of Logistic Regression with TF-IDF. We can see the various F1 scores of the different trials as `hyperopt` went through different hyperparameters for the logistic regression model and TF-IDF vectorizer.*

## VII. Conclusion & Future Work

Because the F1 was around 50% - even with upsampling - this tells me that **there is not enough information in the essays to determine if it is an "exciting" project**. I would have to use other features from the other files available (teacher donations, project details, resource information, etc.) to potentially create more informative features for modeling; however, there are some improvements and/or changes that can be made to this project that could provide more robust and conclusive results.

### *A. Upsampling Strategy*

When upsampling, I used random upsampling on the minority class – i.e., I randomly chose records in the data belonging to the minority class and replicated it exactly, effectively creating many copies of records in the minority class.

The drawback to doing it this way is that I am not creating any "new" data points, so the model learns just from seeing repeated versions of the minority class. There are other upsampling methods such as SMOTE (Synthetic Minority Oversampling Technique) and ADASYN (Adaptive Synthetic Sampling) that create "new" data points based on the surrounding data. This would help the model by allowing it to see other potential values besides the values already present in the minority class.

### *B. Additional Machine Learning Models*

There are several other machine learning models that I could have tested out besides logistic regression and random forest. For example, XGBoost is another decision tree algorithm like random forest, but it uses gradient boosting to help reduce bias and improve model performance.

### *C. Transfer Learning Improvements*

The transfer learning model did not perform as well as I expected, but that may be because training a deep learning model can be considered an art. There are many things that go into how well a deep learning model performs and not all those things are transparent.

As a start, allowing the transfer learning model to train for more epochs might help lead to better results. With early stopping, I was strict in my "patience" parameter, i.e., when to stop training. If I increased the patience of the early stopping code, it would have not stopped after 5 epochs – the number of epochs the model trained before the early stopping code kicked in.

Another parameter that is very important for deep learning models is the learning rate. I set the learning rate to a standard value, but the learning rate is a parameter that is worth taking the time the tune. The main reason I did not do that for this particular project was the fact that training took such a long time and Google Colab enforces time limits on how long the environment can run before it automatically shuts down.

There are some other aspects that I did not do such as training an essay-specific language model before training a classifier, tuning the batch size, etc. In other words, there are still many other ways to potentially optimize and improve the performance of the transfer learning model which I was not able to implement in this project.

## VIII. REFERENCES

[1] SigKDD. (2014, May). KDD Cup 2014 – Predicting Excitement at DonorsChoose.org. Retrieved November 21, 2020 from https://www.kaggle.com/c/kdd-cup-2014-predicting-excitement-at-donors-choose/data.