Machine Learning Capstone Project
Machine Learning Engineer Nanodegree

Mark Alexander
May 11, 2018

DonorsChoose.org Application Screening

Definition

Project Overview

Due to the perpetual underfunding of education budgets, teachers across the U.S. often find themselves paying for school supplies out of pocket. According to the Education Market Association, teachers spend approximately $500 on average, and one in 10 spend $1,000 or more [1].

To help pay for the expenses of desired materials, teachers often submit proposals for educational grants. One great place to do this is the crowdfunding website DonorsChoose.org. Educators write a proposal for a specific project, including essays describing their situation and objectives, giving details of the classroom materials requested. Donors this site can then browse through all the posted proposals, and donate to the project(s) that they like.

Before the projects get posted to the website, volunteers at DonorsChoose.org must painstakingly evaluate every proposal submitted, approving some to be posted on the site, while rejecting others. In this application screening project, I have created machine learning models to assess proposals, assigning each a probability of approval. This probability can then be used by DonorsChoose.org to automatically approve proposals that meet a predetermined threshold.

This paper describes my submission to the Kaggle "DonorsChoose.org Application Screening" competition (https://www.kaggle.com/c/donorschoose-application-screening). Kaggle has teamed with DonorsChoose.org to run a competition to predict if project proposals are accepted. DonorsChoose.org has provided datasets which include the following types of data:

1. Numeric data – Individual materials costs and quantities and the number of previously submitted proposals per teacher.
2. Categorical data – Each proposal specifies several categorical selections such as state, teacher prefix, and the related educational categories (i.e. Math & Science, Literacy & Language, etc.)
3. Text data – The text data is the most important input for making predictions. The text data includes materials descriptions, project title, resource summary, and essays which fully describe the students, their needs, and the purpose for which materials are being requested.

Problem Statement

DonorsChoose.org expects to receive close to 500,000 proposals next year, straining the small number of volunteers who must manually read and review each proposal. The goal of this project is to assist DonorsChoose.org in assessing which educator proposals can be approved automatically for posting to the site. Those proposals which are not automatically approved would then be reviewed manually, allowing the volunteers to assist those educators in improving their proposals.

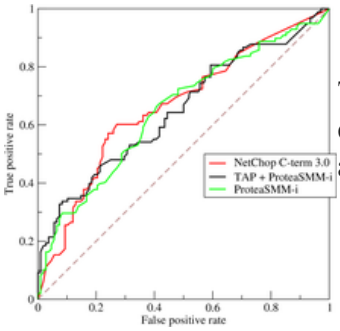
The tasks needed to solve the problem of assigning proposal approval probabilities are:

1. Data preparation and preprocessing – The dataset includes numeric, categorical, and text data. The numeric data must be scaled. The categorical data must be converted to hash data and converted to an embedding. The text data must be concatenated, cleaned, and converted to an embedding. Additionally, as it was found in EDA that there is an imbalance between approved and rejected proposals (85% of proposals were approved), we want to introduce a method to balance this data so that this bias doesn't influence our approval probabilities.
2. Exploratory Data Analysis (EDA) – explore the data to understand its characteristics and understand any problems that the machine learning model may need to overcome.
3. Train machine learning models – We will train deep learning neural networks as a classifier using the training dataset. Each data type (numeric, categorical, text) will have its own branch neural network model, and the outputs of these models will be combined with a fully connected neural network layer.
4. Prediction and evaluation on a test set – Using the trained models, predictions will be made against validation and test sets. Evaluation of these predictions vs. the known results of the test set will be made using the AUROC metric.

Metrics

The performance of our predictions is quantified using the AUROC (Area Under the Receiver Operating Curve) metric, as required by the Kaggle competition. The Receiver Operating Curve (ROC) metric is the probability that a binary classifier will be more confident that a randomly chosen positive example is positive than a randomly chosen negative example is positive. The metric computes True Positives (Sensitivity) vs. False Positives (Specificity) over a range of threshold settings (threshold at which a predicted probability is considered positive or negative).

See this example figure from wikipedia:
https://en.wikipedia.org/wiki/Receiver_operating_characteristic



The area under this ROC curve is computed, and values greater than 0.5 are considered better than random guess, becoming more accurate as the value approaches 1.

Analysis

Data Exploration and Visualization

The dataset consists of train.csv, test.csv, and resources.csv files. Each row of the train and test sets represents a single proposal. The resources file contains one or more rows for each proposal which detail materials requested, their quantities, and costs. The train and test datasets contain numeric, categorical, and text data. The resources dataset contains numeric and text data.

A brief overview of the data types:

1. Categorical features
   These features categorize some of the areas that the proposals fall into. An example from the training set:

	project_grade_category	project_subject_categories	project_subject_subcategories	teacher_prefix	school_state
0	Grades PreK-2	Literacy & Language	Literacy	Ms.	NV
1	Grades 3-5	Music & The Arts, Health & Sports	Performing Arts, Team Sports	Mrs.	GA

2. Numeric features
   The numeric features in the resources dataset detail quantity and price per type of requested resource. An example:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

3. Text features
   The text features can be found in both the resources dataset (see image above) and the test/train datasets, as shown here:

	project_title	project_essay_1	project_essay_2
0	Super Sight Word Centers	Most of my kindergarten students come from low...	I currently have a differentiated sight word c...
1	Keep Calm and Dance On	Our elementary school is a culturally rich sch...	We strive to provide our diverse population of...

	project_essay_3	project_essay_4	project_resource_summary
0	NaN	NaN	My students need 6 Ipod Nano's to create and d...
1	NaN	NaN	My students need matching shirts to wear for d...

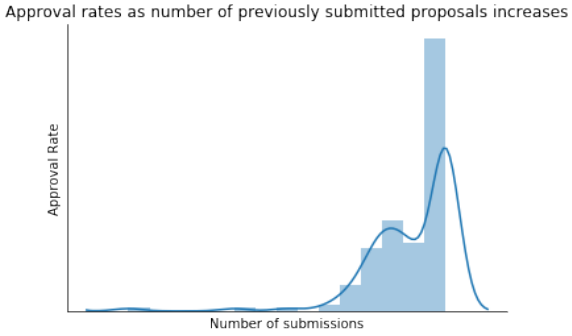
4. Engineered features
   Some new features have been created. From the resources dataset, the quantities and costs have been multiplied to represent a total cost. From this data, statistical features (count, sum, min, max, mean, median, stddev) have been created to use as additional numeric data. Also, month and day have been extracted as categorical features from a date of submission timestamp for each proposal.
5. Other
   A few other columns that don't fit into the above:
   1. id – This is a unique id for each proposal. These can be found in all of the files, and will be used to merge resources data together with the train/test datasets.
   2. teacher_id – An id unique to each teacher, which might be seen across multiple proposals. This feature was only used for EDA in this project.
   3. project_is_approved – This is the target label in the training dataset which indicates if a proposal was approved or not.

The most glaring characteristic of the training dataset is the large imbalance in approved vs. unapproved proposals. There are 182,080 proposals in the training dataset, of which 85% are approvals. This high approval rate makes sense for the DonorsChoose organization as they want to get good proposals out in front of donors. However, this dataset bias will be learned by the model, resulting in inaccurate probabilities of approval on new data. An effort to mitigate this issue by oversampling with synthetic samples will be discussed later in this paper.

Numeric data:

Using the original numeric feature of *price* and *quantity*, a new feature *sum* is created for each proposal. Comparing these sums for rejected vs. approved proposals, we see that the maximum (15,299.69 vs. 13,543.82) and average (588.45 vs. 538.07) *sum* values are higher for rejected proposals. This could indicate that proposals are more likely to be rejected if project sums are too high. Therefore, this *sum* feature is expected to be useful in the learning model. Additional feature engineered statistical features include *count, min, max, mean, median, and stddev*.

For the feature *teacher_number_of_previously_posted_projects*, the maximum (379 vs. 451) and average (6.9 vs. 12.02) is lower for those proposals that were rejected. This reinforces the hypothesis, and is supported by the benchmark results, that proposals are more likely to be approved as a teacher has gained more experience in writing DonorsChoose proposals. This histogram/KDE plot shows approval rates increasing as the number of previous submissions increases.

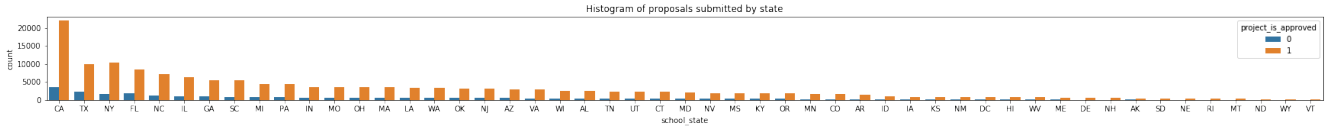


After the creation of the engineered statistic features, some of the STDDEV values were NAN. This happened when only one type of resource was requested, and its quantity was 1. We will fill these NAN values with 0.0.

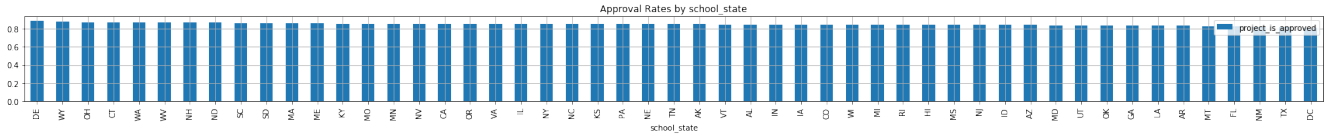
Categorical data:

Categorial features include *teacher_prefix*, *school_state*, *project_grade_category*, *project_subject_categories*, and *project_subject_subcategories*. The *teacher_prefix*, *school_state*, and *project_grade_category* features contain a relatively small number of possible selections, whereas the *project_subject_categories*, and *project_subject_subcategories* features contain multiple possible selections, which can be combined if more than one category applies, creating a large number of unique selections (high cardinality).

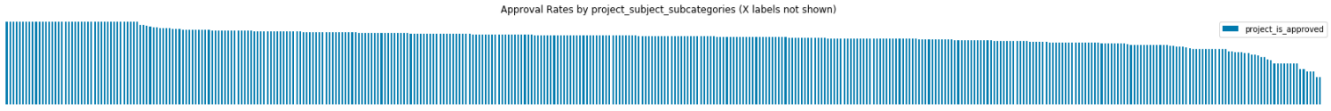
For all of the categorical features, the submission rates have a large variance amongst the selections, as in the following for *school_state*:



However, for the *teacher_prefix*, *school_state*, and *project_grade_category* features, approval rates are largely similar, as seen again for *school_state*:



On the other hand, approval rates for the *project_subject_categories* and *project_subject_subcategories* features have considerable variance:



The middle section of this plot shows approval rates in the *project_subject_categories* feature which are similar. However, a subset of the approval rates are very high (100%) or very low due to the small number of samples which contain this selection, often only 1 sample in the training dataset of 182,080 samples. This pronounced variance is due to the fact that the basic selections like Literacy, Mathematics, or Special Needs can be combined to make unique categories. No attempts were made in this project to mitigate this extreme variance.

In the test and training datasets, most of the categorical data was clean. However, for the *teacher_prefix* feature, there were some missing values (4 in the training set, 1 in the test set). This was resolved in the training dataset by simply dropping these small numbers of rows, instead of trying to guess a correct value. For the test set, since we are required to make a prediction for this entry, the value was set to "Teacher".

Text data:

The text features in the datasets include *project_title*, *project_essay_1*, *project_essay_2*, *project_essay_3*, *project_essay_4*, *project_resource_summary*, and *description*. The *project_title* and *project_resource_summary* features are relatively short statements about the project. The *description* feature is a small description of the items being requested. The essays, however, are a more involved expansion, describing the students, their environment, and the reason that these materials are being requested.

The first thing to note is that approximately 97% of essays 3 & 4 are missing. This is because before May 17, 2016, four essays were required. After this date, the essay prompts were changed so that only 2 essays were required. It has been noted by others in the Kaggle competition that the old essays can be combined such that they are similar to the intent of the new essays. Therefore, for those rows containing 4 essays, essays 1 & 2 will be concatenated as essay 1, and essays 3 & 4 will be concatenated as essay 2.

In the resources dataset, there are 292 *description* fields that are NAN. In spite of these missing item descriptions, there are still *quantity* and *price* entries for these rows. A sampling of the associated

proposals show that they are still getting approved. As this appears to be allowed by the reviewers, we will make the assumption that the required information is still contained in the other text fields. We will simply fill in these NANs with empty strings.

Some statistics on the text features were collected for approved and rejected proposals. It was noted that in other models, these numeric features were useful in improving AUROC scores. They have been explored in the EDA for this project, but not used in my model. Broadly, these stats show that text fields were a bit longer for approved projects. The assumption is that these text fields were generally more involved and descriptive. The mean for the *description* feature is an exception to this rule. It could be that rejected proposals are asking for more (too many?) resources.

Statistics for approved proposals:

	project_title	project_essay_1	project_essay_2	project_resource_summary	description
min	5.0000	311.000000	435.00000	25.000000	2.000000
max	141.0000	2760.000000	5224.00000	902.000000	7215.000000
mean	32.5191	682.731415	857.82003	121.400254	299.922685

Statistics for rejected proposals:

	project_title	project_essay_1	project_essay_2	project_resource_summary	description
min	4.000000	169.000000	248.000000	25.000000	3.000000
max	102.000000	1523.000000	2262.000000	301.000000	5544.000000
mean	32.171775	675.563316	792.691209	127.300606	450.208805

Algorithms and Techniques

`Learning model:`

The machine learning model used in this project is comprised of 3 branch neural networks, one for each of the data types, tied together by a fully connected layer which outputs a probability of approval. The two branch models for the numeric and categorical data are fully connected neural networks. The third model for text data is a bidirectional GRU (biGRU) Recurrent Neural Network (RNN). The training parameters used for this classifier model are:

- Optimizer (Mini-batch Gradient Descent algorithm, Adam was used)
- Epochs (number of times the complete dataset is presented to learn, 4 was used)
- Batch size (Number of subsets dataset is divided into, each batch subset used per training iteration, 256 was used)
- Learning rate (lr, how fast the learning steps should be, 0.0005 was used)
- Decay (amount that the learning rate is reduced over each update, 1e-6 was used)
- Loss function (algorithm used during training to measure loss, used binary_crossentropy)
- Validation metric (metric used against validation set to control and report learning, accuracy was used)
- Validation split (validation dataset used to control and report learning, 0.10 (10%) was used)
- All other parameters were left at default

Additionally, some checkpoints were used for this model to save learned weights and control overfit:

- ModelCheckpoint (when new learned weights result in an improved accuracy, the model weights are saved to file to be loaded into the model later)
- EarlyStopping (algorithm used to stop the learning if a certain number of epochs has passed without metric improvements on the validation set, 4 was used)

The basic fully connected neural network models are chosen for the categorical and numeric data as they will fit nicely into the rest of our model and the whole dataset can be learned together.

Recurrent Neural Networks are considered state of the art for Natural Language Processing problems, because they add the ability to capture the inherent sequential nature present in language [2]. They are different than standard neural network models and Convolutional Neural Networks (CNN) in that they are useful for sequences of inputs, because they add feedback and memory. Specifically, the bidirectional GRU (biGRU) model is powerful as it provides sequences both forward and backward as input. These bidirectional biGRU models are commonly seen and used in winning NLP problems on Kaggle [3].

`Pre-trained word embedding:`
Instead of training our own embedding, a pre-trained word embedding is used to improve the model. In this case, Fasttext Common Crawl (crawl-300d-2M.vec) is used. The Fasttext Common Crawl is a commonly used word embedding, and was used by the 1st place team in the Kaggle "Toxic Comment Classification Challenge". [3] This word embedding was chosen over the Glove pre-trained word embedding as testing showed a 0.04 increase in the AUROC metric.

`SMOTE:`
As mentioned in previous sections, there is an imbalance in the number of approved (85%) vs. rejected (15%) proposals in our dataset. To alleviate this problem we will use SMOTE (Synthetic Minority Over-Sampling TEchnique) to oversample the rejected data. Instead of simply reusing copies of the minority class data, the SMOTE algorithm creates "synthetic" data points based off the existing data. We will use the default parameters from the imblearn.over_sampling.SMOTE package.

Benchmark

A benchmark model was supplied by Google's engineering education team at https://www.kaggle.com/skleinfeld/getting-started-with-the-donorschoose-data-set. This kernel trains a linear classifier model on the numeric feature *teacher_number_of_previously_posted_projects*. This feature was hypothesized to result in more approvals as a teacher submits additional projects, becoming more familiar with the process. This benchmark model results in an AUROC score of 0.56522, which is better than random guessing. Any models trained in this project should outperform this benchmark to be considered useful.

Methodology

Data Preprocessing

The preprocessing steps taken before data is used in the model are:

1. NAN values cleanup
2. Feature engineering
3. Scaling of numeric data: StandardScalar
4. Hashing of categorical features
5. Text cleaning and Tokenization
6. Splitting datasets
7. SMOTE

Each of these steps is described below.

`NAN values cleanup:`

As mentioned previously in the Data Exploration and Visualization section, each of the data types had some NAN values that needed to be cleaned. Briefly summarized, some item descriptions in the resources dataset were NAN. It was determined that this was OK, so they were replaced with empty strings. After numeric feature engineering, some STDDEV values were NAN, and these were replaced with 0.0. Finally, some teacher prefixes were missing. The ones found in the training dataset were simply dropped, and the single case in the test dataset was randomly changed to "Teacher".

`Feature engineering:`

A few new features have been engineered to parse out relevant information to be learned by our model. For categorical data, *year*, *month*, and *day* have been engineered from the *project_submitted_datetime* timestamp feature.

For numeric data, multiple rows in the resource dataset containing quantities and price are combined into a single entry, *total*, for each proposal. To retain some information about the characteristics of this data, the new features *count*, *sum*, *min*, *max*, *mean*, *median*, and *std* have been created.

Though not technically feature engineering, it should be mentioned that for each proposal, all of the text fields are combined into a single *text* feature, which will be learned together as a whole using the NLP model.

`StandardScalar:`

For machine learning, it is beneficial to normalize and scale data to speed training and avoid getting stuck in local minima. To do this, the sklearn.preprocessing.StandardScalar package is used with default settings. StandardScalar will scale each numeric feature independently to remove the mean and scale to unit variance.

`Feature hashing:`

The categorical data cannot be used in the model in its current text form, as it is intended to be used in an embedding. To prepare for this, the data in each categorical feature will be hashed, which will transform the features into a vector. The context of what these vectors represent will be lost, but this information is not needed when fed into our neural network model.

Text cleaning and Tokenization:
To prepare the text for use in an embedding, this data is vectorized using the keras.preprocessing.text.Tokenizer package. As this package has a default list of punctuation that will be filtered out, no additional text cleaning was performed. Some non-alphanumeric characters still remained in the text, but these didn't appear to affect AUROC scores.

Dataset splitting:
A validation set was reserved so that AUROC scores from the Kaggle test set were not overused. If a test set is utilized to tune parameters to increase scores, the model will be made to overfit on that test data, and the model may not be general enough to work well on new data. To avoid this, 10% of the training data is split off to be used for validation.

SMOTE:
As mentioned in the Algorithms and Techniques section, SMOTE is being used to oversample our minority data (rejected proposals). Before SMOTE, 84.7% of the training data are approved proposals. After SMOTE, the data is now evenly divided between approved and rejected proposals.

Implementation

The implementation of this model follows this outline:

1. Load the training, testing, and resource datasets into memory, create engineered features, clean up NAN values, and merge datasets. Only training and testing datasets will remain, with one row per proposal.
2. Combine the text fields, scale the numeric data, and perform feature hashing on the categorical data.
3. Tokenize the text data and create sequences of index values.
4. Split the training data to reserve 10% for final model validation.
5. Use SMOTE to oversample the minority data.
6. Load pre-trained word embedding data into memory and create an embedding matrix based on words learned in the text data tokenization.
7. Define the full model with training parameters and checkpoints, instantiate the model, and perform the training.
8. Load the final model weights into the instantiated model and run predictions on the validation data which was split earlier from the training data. Report the AUROC score.
9. If the AUROC score on the validation data is acceptable, run final predictions on the Kaggle test set for submission to the competition.

The machine language model used for this project is a neural network model comprised of a separate branch model for each of the numeric, categorical, and text data types, combined by a fully connected neural network layer.

The model for the numeric data is the simplest as the input is fed through two fully connect layers with dropouts.

The categorical hash inputs are fed into an embedding layer with spatial dropout. A flatten layer then feeds into two fully connected layers with dropouts.

The most complicated branch of the model is that for the text data. To prepare the text data for input, it is tokenized. This tokenizer is fit on all of the text in our training and test sets. The training data is then turned into sequences of index values from the tokenized word index. The pre-trained Fasttext Common Crawl (crawl-300d-2M.vec) word embedding is used with the tokenized words to create an index of weights to be used by an input embedding layer. This input embedding layer feeds through a SpatialDropout layer to bidirectional GRU and Convolutional layers, then a global max pooling layer. This model is completed with two fully connect layers each with dropouts.

These three branch models are tied together with a fully connected layer with dropout and a fully connected output layer with a sigmoid activation. This model is compiled using an Adam optimizer, and a binary_crossentropy loss function.

To train this model, the numeric, categorical, and text data which were oversampled using SMOTE, are fit to the model over 4 epochs with a batch_size of 256. A callback is used to save the model weights for the best accuracy results.

Once the model is trained and best weights are loaded, predictions are made on our validation set which was split from the original training data, and an AUROC score is presented.

Refinement

The model change which had the largest effect on the AUROC validation scores was the choice of our pre-trained word embedding. Initially, 100 and then 200 dimension GloVe models were used with good results. The 300 dimension Fasttext model was finally used once CPU resource issues were resolved. As the word embedding dimensions are increased, training of the model becomes longer. Therefore, only 4 epochs are trained. Validation AUROC scores reported with the pre-trained embeddings were:

- Glove 100d: 0.76961
- Glove 200d: 0.79315
- Fasttext 300d: 0.79488

Another refinement which had a large effect on validation scores was the choice of the neural network model used for the text data. An initial test model, used as a baseline, essentially excluded the bidirectional GRU layer used in the final model. This early model still used the Glove 100d word embedding, and obtained an AUROC score of 0.68405, compared to the biGRU score of 0.76961.

Minority class oversampling is another change not used in earlier implementations. Testing on these original models showed that using SMOTE produced a modest gain in the AUROC validation scores, increasing them from 0.74966 to 0.75105.

Finally, early on there was a problem performing oversampling on the categorical data, causing validation scores to be lower with SMOTE. The numeric and text branch models used SMOTE data, whereas the categorical model did not. Because of this, the number of training data inputs was different between models, and therefore they could not be combined with fully connected layers in the same model. So these models were stacked using Linear Regression. The branch models were trained and predictions were made on the validation data. These validation predictions were used to train the Linear Regression model, which was then used to make predictions on the Kaggle test data. The stacking of these branch models resulted in an AUROC score of 0.73557. This score can be contrasted with the similar full model score of 0.76961.

Results

Model Evaluation and Validation

The final model configuration was chosen as it reported the best results on our validation and test sets. The choice of a fully connected neural network layer to concatenate the branch models was chosen because of the effectiveness of neural networks, and the fact that it will learn with the branch models, as opposed to ensembling or blending the results of the branch models with an independent model. The specific choice of a layer with 50 outputs, and a dropout of 0.25 were initially chosen, and modifications didn't seem to affect the AUROC scores, so initial values were kept.

For the branch models, the variations of models used for the categorical data seemed to have little effect. Independent testing on categorical data only seemed to result in AUROC scores of between 0.56 and 0.57, which isn't much higher than the benchmark value of 0.56522. Therefore, the initial model was used, having a SpatialDropout of 0.30, and fully connected layers of 50 and 100 outputs, with a 0.25 dropout in between.

The numeric branch model was chosen to have two fully connected layers of 100 outputs, each with dropouts of 0.30.

The bidirectional GRU model for the text data was specifically chosen because of its proven results on previous Kaggle competitions, specifically the "Toxic Comment Classification Challenge". The final metrics were affected most, not by hyperparameter changes, but on the choice of pre-trained word embeddings, in which the 300 dimension Fasttext Common Crawl was used. After the embedding layer, a SpatialDropout of 0.30 was used, leading to the bidirectional GRU with 50 outputs, a 100 filter convolutional layer with kernel size 3, and a global max pooling layer.

Robustness of a model means that it will continue to perform well when exposed to new inputs. A model that sees test data and is over tuned to perform well on that data will not generalize well to new data. To avoid data leakage and overfitting to the public test data in this Kaggle competition, predictions made on a public test set and submitted to have AUROC reported on a leader board were restricted to 5 submissions per day. Submissions were limited to avoid data leakage by the modification of hyperparameters to increase the score on the test set, overfitting to that data. The bulk of testing was performed against a validation set which was split from the original training data.

The robustness of this model is exercised by using new unseen data. The public test set was only 30% of the test data available. A final 70% of test data, a private test set, was withheld until the end of the competition. This new data shows that the model continues to perform well on new data, reporting an AUROC score of 0.79336 vs. 0.79488 on the public test set.

Justification

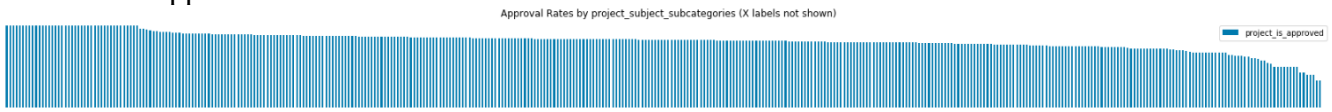
A final AUROC score of 0.79336 was reported on the private test set. This is in comparison to the benchmark metric of 0.56522, a 40% increase. These results are, as expected, much stronger than the benchmark as there was much information to be learned, especially from the text data, to help classify the proposals.

The highest AUROC score reported on the private leaderboard was 0.82812, 4.4% higher than my model. It should be noted that the top scores on the leaderboard were not single models, but were instead ensembles of multiple models. In these ensembles, they are combining the predictions of individual models, weighted by their AUROC score performance on the public test set. In light of this difference, the performance of this single model with minimum feature engineering looks to be effective for this problem.

Conclusion

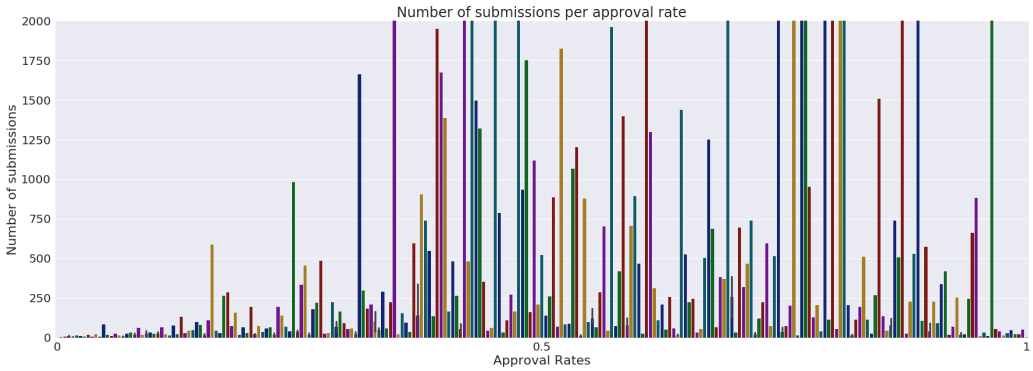
Free-form Visualization

In the discussion of categorical data in the Data Exploration and Visualization section, a graph of approval rates for the feature *project_subject_subcategories,* repeated below, shows pronounced variation in approval rates.

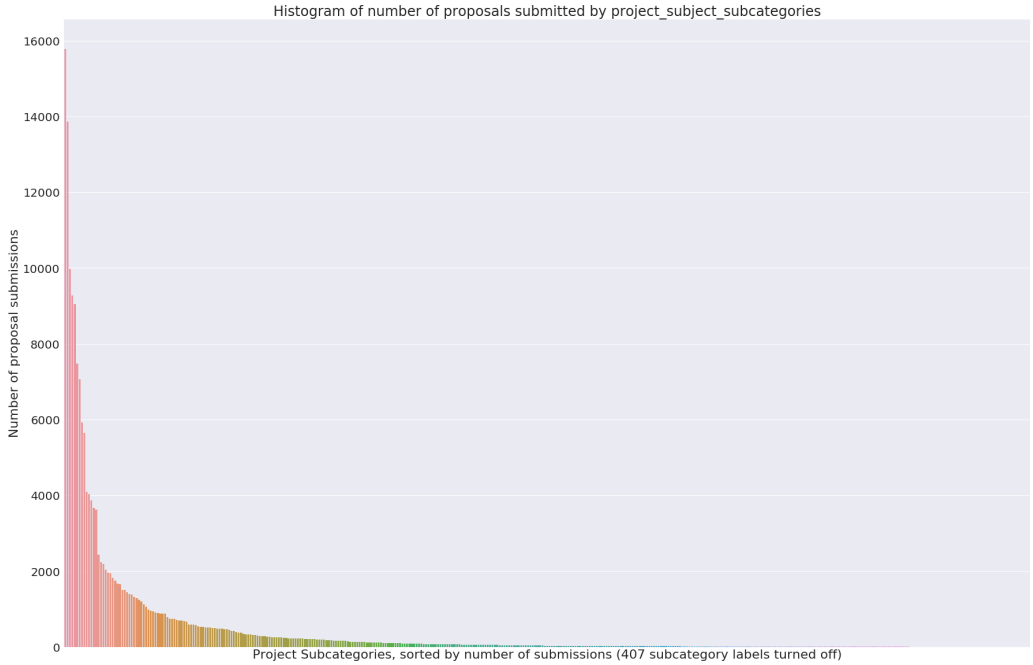


A hypothesis is that these unusual approval rates are due to the low number of samples. For this feature, there are 28 base subcategories such as *Economics*, *Literacy*, or *Mathematics*. When creating a proposal, an educator can select more than one of these base labels. Therefore, an entry for this feature might be something like "*Parent Involvement, Visual Arts*". In this project, any new combination like this is considered to be its own label, and therefore we end up with 407 unique subcategories. As there are so many combinations, it is therefore not surprising that some subcategories may not be often used, and so not many samples exist for that rare subcategory. This low number of samples reduces the resolution, making it more likely that corresponding approval rates will fall outside the median rates.

To explore this hypothesis, we first look at approval rates for the subcategory combinations vs. the corresponding number of submissions. The graph below confirms that there is a very large variation in the number of submissions.

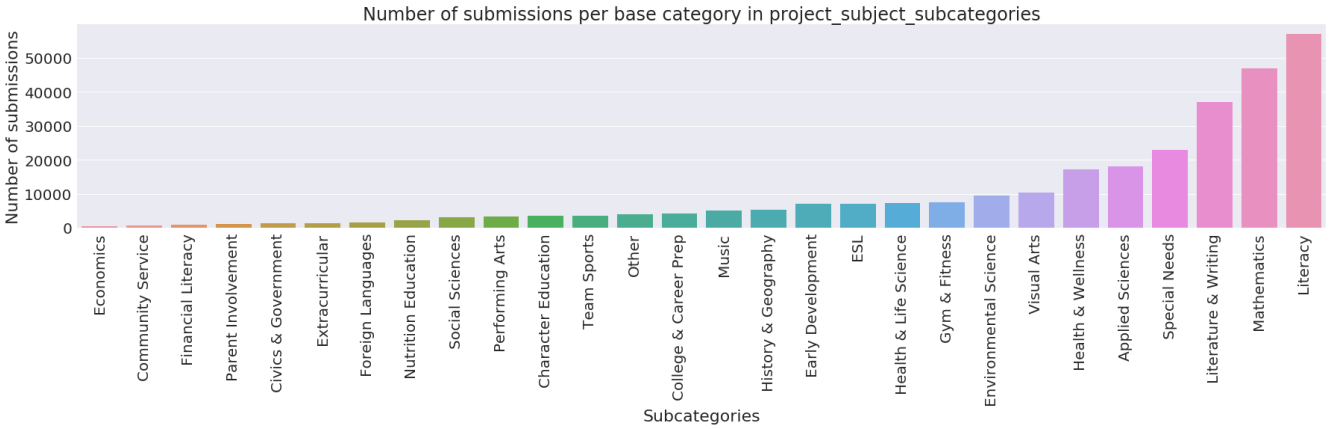


Note that the y-axis above for the number of submissions was limited to 2000 so that we can make out the lower values. The next graphic, which shows the number of submissions per unique subcategory, emphasizes this variation and shows that the bulk of subcategories had low submission rates, whereas some popular combinations had as high as 16,000 submissions.

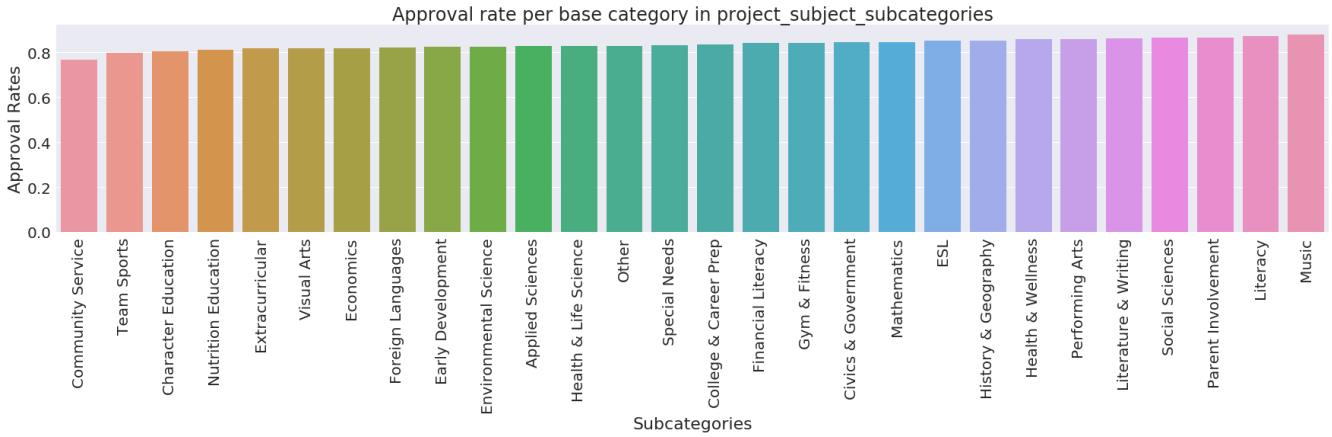


This was likely the wrong approach to dealing with this categorical feature. Perhaps this feature should have been handled by feature engineering, possibly one-hot encoding, so that the number of unique possibilities is reduced.

Instead, if we look at the number of submissions filtered by the base subcategory, there is still variation seen, but not as pronounced as before.



Importantly, much more stability is seen in the approval rates when filtered by these base subcategories, even for *Economics* which had the lowest number of submissions.



The takeaway here is that the model may be learning, falsely, that some subcategories should be approved at unusually high or low rates. There is a bias in our data when looking at the subcategories as combinations. By concentrating the samples into fewer subcategory selections, the approval rates are smoothed out, reducing the variation and allowing our model to more accurately learn from this feature.

Reflection

To ultimately predict the probabilities of a DonorsChoose.org proposal approval, the following steps were taken:

1. Load training and testing proposal data which include essays, item descriptions, quantities, prices, etc.
2. Perform Exploratory Data Analysis (EDA) to thoroughly understand characteristics of the data.
3. Use some of the data to create a quick benchmark which performs better than guessing, in which we can compare later solutions.
4. Preprocess the data to prepare it for training our model, splitting off a subset for validation testing.
5. Train the chosen model, test against the validation set, and modify the models configuration and hyperparameters, as needed.
6. Make predictions on a test set, submit to the Kaggle competition, and compare against benchmark metrics.

As discussed in the Justification section, the final model obtains good results predicting the probability that a DonorsChoose proposal will be approved. This model would definitely be useful to the DonorsChoose.org team in automating some of the approval process and lightening the load for their volunteers.

One aspect of the project that I found interesting was exploring and understanding the data. EDA, the approach of first analyzing the data, is important to building a correct and effective model. As the DonorsChoose data sets were investigated, I found myself with many questions that I wanted to explore deeper. For example, the approval rates for subcategories were found to vary widely. Was this because there is a solid reason for a reviewer to approve or reject these? Or was this an artifact of the number of samples for those subcategories? If there is some variance in the samples, is that something that needs to be balanced out in our model? The combinations of ways to look at the data are numerous and I wish I had had more time to explore this in greater depth.

The most interesting part of this project was utilizing Natural Language Processing (NLP). NLP neural network models were vital to the success of this project as text data (essays, descriptions) are the deciding factor in approving a proposal, whether by a reviewer or machine learning model. When testing branch models independently, the text models were the most important component in achieving a good AUROC metric. On the subject of NLP, there are many more papers, website write-ups, and Kaggle competitions that I hope to explore.

Improvement

If this final model were chosen to be the benchmark, could another model obtain a better AUROC score? The answer to this is yes. This solution was kept simple, and as discussed in the Justification section, it performs well. However, many additional approaches could have been taken to improve it.

I believe the single most important thing to improve the model is more feature engineering. Specifically, as it was found that the text data is most important to our model, more features could have been created based on this. For instance, the number of words per individual text feature, the number of misspellings, the number of unique words, the percentage of repeated words between essays, count of capital letters in each text feature, and so on. As mentioned by the winner of this competition, the approach of “create as many new features as you can without thinking too much about whether they will help or not.”[5] is very important to improving the solution.

There are additional measures that can strengthen NLP, like using lemmatization and additional stop words. Lemmatization reduces the inflection forms of words like *organize*, *organizing*, and *organization* to a single common base form. Additional stop words can include high frequency words seen in our training data. The advantage to both of these measures is in reducing the word space allowing the model to focus on the more important, lower frequency words.

Finally, the code was run as a kernel on Kaggle. Due to resource issues, the model couldn’t be run on GPU. There is a 6 hour time limit on kernels, so only 4 epochs could be run on CPU. Running the model on a machine with additional sources for more epochs should have improved the prediction model.

References

[1] White, Martha C. (August 3, 2016). Here's How Much Your Kid's Teacher Is Shelling Out for School Supplies. *Money*. http://time.com/money/4392319/teachers-buying-school-supplies/
[2] Young, Tom. Hazarika, Devamanyu. Poria, Soujanya. Cambria, Erik. (Feb 20, 2018). Recent Trends in Deep Learning Based Natural Language Processing. *ArXiv:1708.02709*. https://arxiv.org/pdf/1708.02709.pdf
[3] Chun Ming Lee. (March 20, 2018). 1st place solution overview. [discussion board posting]. Retrieved from https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52557
[4] https://www.kaggle.com/skleinfeld/getting-started-with-the-donorschoose-data-set
[5] Seymour, Harlan. (April 25, 2018). 1st Place Solution. [Kaggle notebook]. Retrieved from https://www.kaggle.com/shadowwarrior/1st-place-solution