Bill Murphy
7_18_2018

Capstone 2 Milestone Report

1. Define the problem

Donor's Choice Kaggle Challenge:

There is a complete overview here, but in a nutshell, Donor's Choice is a charitable organization that matches classroom projects with potential donors. The needs and numbers of proposed projects are growing rapidly, and today's manual review process is unsustainable in the future.

The challenge is to effectively screen the proposals prior to DC posting on its website.The challenge is to identify as many of the proposals as possible that can be posted to the website without manual review. For the purposes of this exercise, the roc_score will be used.

From a pragmatic standpoint, approximately 85% of the proposals are approved, so the data set is somewhat unbalanced, and this will need to be accounted for in the analysis.

2. Identify your client

In this case, the "client" is the Donor's Choice organization, with two distinct groups:

a) Those responsible for reviewing the proposals
b) Those responsible for assuring the overall quality of the website.

3. Describe your data set, and how you cleaned/wrangled it.

The dataset was provided by Kaggle, and as such, was relatively clean. It consists of some numerics (number of proposals previously submitted, cost of proposal, etc.), some categorical data (state, subject category, etc.) and some textual data that includes up to four essays (two of which are present across almost all proposals and two of which are fairly empty), a project resource description, and some smaller fields (title, main subject, etc.)

The amount of cleaning and wrangling was driven by the data type, with one exception: one column was named id and for the benefit of some of the libraries, I renamed that to proj_id.

Numeric: This was mostly straight forward, normalization with the one exception being the cost features. This required merging a resource file with the training set and calculating overall proposal cost. Once that dataframe was established, other useful features were easily derived.

For example, it was straightforward to build a feature that had the state (local) approval rate by category, which provided more resolution than an overall average approval rate.

Once the state data was available, it was possible to develop secondary features (ie: relative cost differences to state norms for subject categories) which added more resolution than the basic numerics.

Categorical:

The categorical variables state, title (Ms., Mrs, Mr, etc), primary subject category, etc. were treated in a straightforward way: dummy encoding for all categorical values except teacher_id. Teacher_id was encoded as a one-hot variable with the label_encoder method. This nets a feature that relates specific teacher experience to approval rate.

Text Variables: The text variables required the most cleaning. First and foremost, "text" was really unicode and included everything from emoji's, non-printing characters, missing values and foreign words/punctuation to basic ascii characters.

I tried several standard conversions, none of which was completely successful. To get the data into a useable state, I made the following up-front decisions:

1) Focus on English only – this has some limitations as there was some Spanish involved, but the majority of proposals were in English.
2) Ascii would be the "language" of choice, as some of the function in the Textblob package require ascii only.
3) High frequency words would be ignored. I used NLTK's stopwords to approximate all hf words. This was a set of 178 words removed from the corpora
4) Missing values would default to a known state of ' '

To implement these decisions, I used several built-in functions, and then added custom screens for the exceptions. However, even with these simplifications, several of the packages were simply too slow to be usable (ie: spell checking through TextBlob.) To overcome that problem, I built a simplified spell checker by loading the Brown and Wordnet corpora form the NTLK package and creating a custom dictionary.

Using simple set math, it was fast to count the number of misspellings in a given sample. Set math also proved to be much quicker than the list comprehension methods of finding high frequency stop words. On my laptop, this cut the processing time from over ten hours when using Textblob to less than two minutes for spell checking and stop word removal. Code for this is included on the Github site.

After making the above adjustments, it was simply a matter of converting from strings to lists and vice versa to use appropriate built in functions. For example, polarity and subjectivity were easily derived using TB. Other features were straightforward: percentage capitalization, overall length of essays in both words and letters, etc.

Finally, when I added ngrams and tdidf, the amount of memory required became unmanageable for my laptop. Though not "cleaning" in the traditional sense, I needed to move the analysis platform from my laptop to an ec2 instance on AWS.

The following is an output of a simple memory monitor I built to determine where the memory issues were coming from.

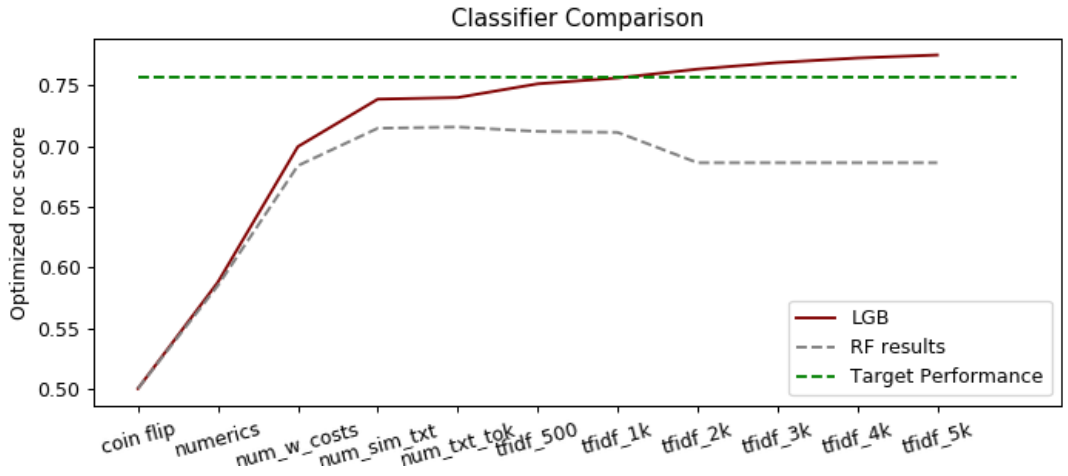
Insert Memory Graph

4. List other potential data sets you could use

The nature of this effort preludes additional data sets (Kaggle competition.) However, in an actual customer situation, individual teachers would be identifiable to the school and zip code level. That additional geographical resolution would be useful. At a minimum, this would eliminate the single application point from some schools, and provide a much greater resolution at the proposing teacher level.

5. Explain your initial findings

The following graph while simple in construction, contains a great deal of insight.



The green line represents the 50th percentile of Kaggle scores. When I started this project, that was the goal, get in the top half of competitive models. What was interesting about this project was that based on previous experience, originally, I focused on the RandomForrest Classifier. But after adding simple text features, that classifier plateaued. It despite optimizing hyper parameters at each feature level, RF did not meet the goal.

Light Gradient Boost however, performed much better. Without any complicated text features (ie; tfidf) it almost met the goal. After combining all text fields and applying tdidf to that corpus, performance continues to improve.

In the final report, I will look at least one more classifier, and itemize some areas where performance may be improved. At this interim point however, the original target has been met.