311 Predictions on Kaggle – Austin Lee

Project Description

This project is an entry into the SeeClickFix contest on Kaggle. SeeClickFix is a system for reporting local civic issues on Open311. Each issue has an associated number of views, comments, and votes. The goal of the contest is to minimize the root mean square log error (RMSLE) of predictions of views, votes, and comments based on the provided data including latitude/longitude, summary/description text, source category, created time, and tag category.

For complete control of the algorithms, as well as self-education, I wrote many of the routines used in this project myself in Matlab instead of relying on libraries such as scikit-learn. The final code base was around 500 total lines in 18 functions. The most significant built-in function calls are to Matlab's left divide operator and kmeans algorithm.

Minimizing RMSLE

The contest is evaluated by taking the RMSLE of all predictions together. Since this metric isn't normalized, the "views" and "votes" predictions are given somewhat more importance than the "comments" prediction, which are often 0.

$$\mathrm{RMSLE} = \sqrt{\frac{1}{n}\sum_{\{i=1\}}^{n}\left(\log(p_i + 1) - \log(a_i + 1)\right)^2}$$

I use a simple linear least-squares model to minimize RMSLE by treating log(y+1) as the variable that we wish to predict, and then converting the predictions back to linear space by reporting exp(pred)-1. This gives the model the freedom to incorporate many different features at the top-level while still running very quickly. All of the features that are discussed in this report are combined in this linear model.

Validation

There are two data-sets for the competition. The training data set contains 223,129 issues and includes the actual number of views, votes, and comments. The testing data set contains 149,575 issues with the prediction variables stripped.

Kaggle judges entries using contestant’s predictions on 70% of the testing data. The remaining 30% is used to generate public leaderboard positions. Two entries can be submitted per day, and the leaderboard position for each entry provides valuable validation information. However, since the submission rate is limited, performing validation on the training data is also necessary.

I used simple 70:30 cross-validation to determine the generalization error of the model, and took the mean error of 10 CV runs to improve the stability of the calculated error. I found this method to give good agreement with the RMSLE values that Kaggle reported on my submissions. In addition, I found it useful to have a backward search algorithm to check for any over-fitting in my model.

Data Processing

I found Matlab’s data loading was slow and didn’t parse this data set well, so I wrote my own. My data loader performs a number of different parsing functions on the various fields. The data is sourced from 4 cities, so the loader runs kmeans with k=4 and an initial mean located in each city. This quickly creates the city categories, which the data does not provide. The text fields (“summary” and “description”) are parsed by converting to lower case, removing any non-alphanumeric characters, and tokenizing on spaces. Each token is run through a non-cryptographic hash function (sourced from an implementation of the djb2 function written by D. Kroon University of Twente), which greatly speeds up the vocabulary generation. Categorical variables (“source” and “tag”) are analyzed without hashing, since their vocabularies each only contain about 10 possible values. Dates are parsed with the `datenum` function in Matlab, which I found was one of the slower functions calls in my load routine.

This parsing process takes about 30 minutes for the Kaggle data, so it is loaded into a struct in Matlab so that it is easy to maintain and manipulate in memory from a single load operation.

Time Features

Plotting the mean of the data by day (Fig. 1), we see temporal patterns in the data that we can use as indicators in the linear model.

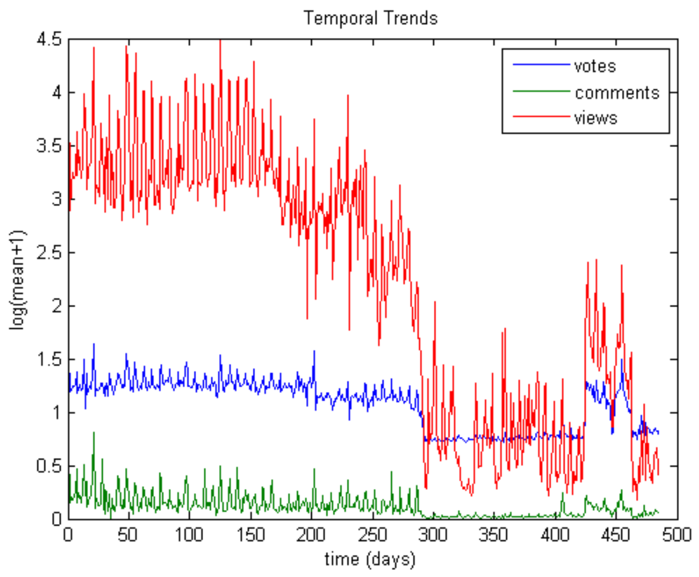

Figure 1: Mean values per day for all data

The most obvious pattern is a large decrease in the number of comments and votes around day 300 through day 425. Looking more closely, we see that this only affects Chicago (Fig. 2), and that the impact on the overall results is largely due to a marked increase in the number of submissions (Fig. 3), which skews the average. In this case, it is sufficient to simply make independent models for the 4 cities and break Chicago down into two categories so that we have 5 effective city categories.

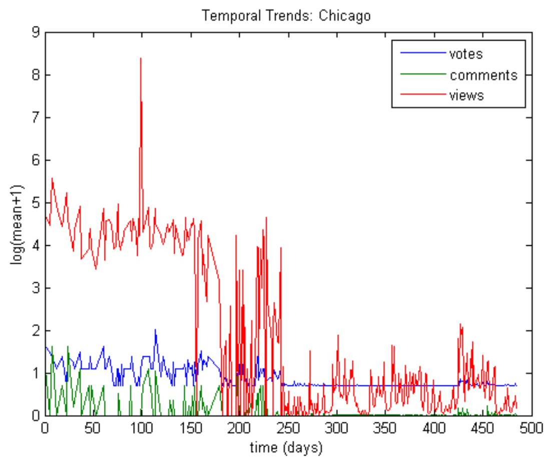

Figure 2: Mean values per day for Chicago

Once this is accounted for, we can properly categorize the temporal data and attempt to model it. In general, the cities have a linear overall trend (Fig. 4) as well as a clear weekly pattern (Fig. 5). To account for the linear decreasing trend, we can add the timestamp value itself as a feature for each category. For the cyclic effect, we can use an indicator that breaks down the data by day of the week. This gives us 8 features in each of 5 categories, for 40 total features to use in the linear model.

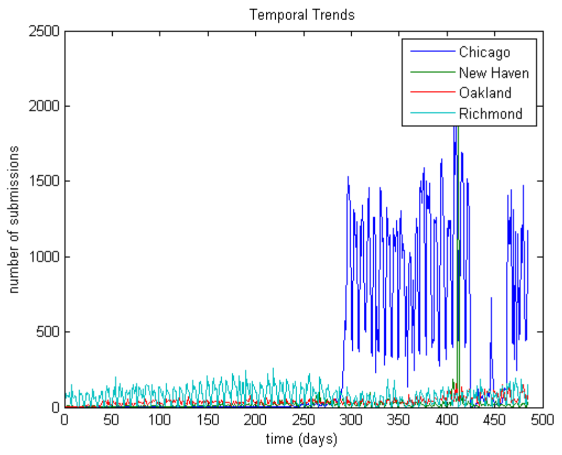

Figure 3: Number of issues per day for all data

I found that these features provide the greatest benefit in my model, going from about 0.62 RMSLE to 0.39 alone. This is hardly surprising given the large variation that is clearly explained by these features in the figures shown.

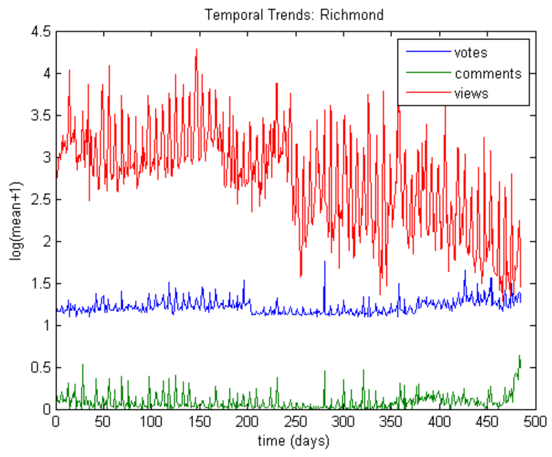

Figure 4: Mean values per day for Richmond, detail

Text Features

As mentioned earlier, the text fields are sanitized, tokenized, and hashed into numerical values. The vocabulary for the summary and the description are independently generated for the most common tokens.

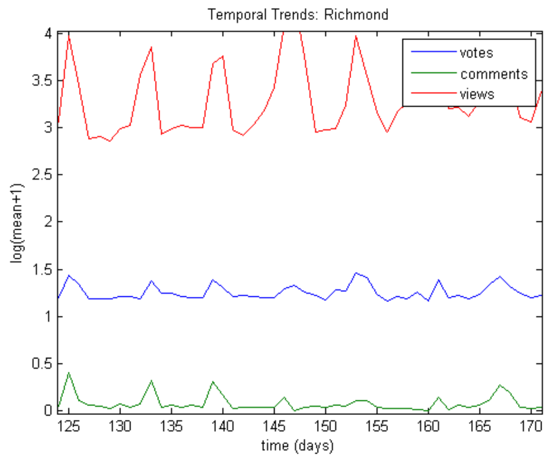

Figure 5: Mean values per day for Richmond

For each vocabulary set, we can then generate a feature for each vocab entry as an indicator of the presence or absence of that word in its associated field. I found a 100-word summary vocabulary and a 200-word description vocabulary provided good results. This includes tokens that are represented in about 10 issues at a minimum.

In addition, a length feature was generated for both the summary and the description fields based on the number of tokens for each. This is somewhat redundant with the token features, but does provide additional information when a token is used more than once. I found including this feature did improve generalization error slightly.

Categorical Features

The “source” and “tag” fields are trivial to implement as binary indicators in my model. They ended up only having a minor effect on the result compared to the time and text features described so far.

Location Features

City categories are captured in the time analysis discussed earlier, but that doesn’t make full use of the (potentially valuable) location information. I opted for a simple, somewhat brute-force model to try to capture these features:

I ran k-means on the entire data-set (the test set as well as the training set together) for a large k (over 100), and used those cluster assignments as indicators in my model. I found this worked best around k=500, although it only reduced my RMSLE from 0.321 to 0.318.

It may be possible to improve this part of the algorithm in particular. A soft-clustering algorithm may provide better results, or using an SVM to categorize high-view and low-view locations. Unfortunately, the short time-span of the project precluded investigating these options.

Attempted Tweaks

In the course of this project I tried a number of tweaks to the algorithm to improve it. None of them proved effective, likely because the algorithm I arrived at by the project milestone was already quite competitive (using all of the features described above except for the fine-grain location features).

A few notable attempted changes that increased RMSLE were: removing any of the features I had already generated, increasing the size of the vocabulary to 400 words, combining the “description” and “summary” vocabularies, and any attempt to use a different overall model (my linear model fit to the log of the data proved quite effective).

Results

I placed 88th out of 533 teams with a final RMSLE of 0.3176. The winner had an RMSLE of 0.2883. The median score was 0.4061. The baseline (all-zero) prediction had an RMSLE of 0.721.

Considering that the entire leaderboard is largely populated with PhD students and professionals, I am happy with this result and believe that it confirms that my algorithm works reasonably well. With more time to improve the algorithm I think the most promising features would be census data (population density and household income in particular), and figuring out whether any of my other features interact the same way I found city and time interact.

The most important result: I got to get my hands dirty with real-world machine learning and learned a lot in the process. Thanks for reading!