Predicting SeeClickFix Issue Importance
Kyle Reinke

Abstract
The objective of this project is to predict the number of views, votes, and comments that future reported issues will receive based on the data provided by SeeClickFix[(1)] of past reported issues. Through the use of Gradient Boosted Regression Trees[(2)] and weighted model averaging, the final root mean squared log error of the predictions is 0.350.

Introduction
SeeClickFix is a social media web site for identifying issues which need to be addressed by local municipalities. One feature of this platform is crowdsourcing of the identification of how important an issue is to the community. SeeClickFix defines importance as a function of the number of views, votes, and comments that a reported issue receives. By determining which issues are identified as being most important to the community can then be quickly addressed. SeeClickFix is interested in predicting how important a newly reported issue will be to the community at the time it is posted. This problem has been posed as a contest on Kaggle,[(3)] an online statistics, data mining, and machine learning community.

Data
SeeClickFix has provided past data from 223,129 reported issues on the website, including the count of how many views, comments, and votes each reported issue received, over a span from January 1, 2012 to April 30, 2013. In addition to this training data, a test data set of 149,575 reported issues immediately following the training set from May 1, 2013 to September 17, 2013.

Each example from the data contains 7 fields. The first is a unique numeric identifier for each reported issue. The second field is a formatted string for the date and time that the issue was reported. The next field is a short summary string of the reported issue. The fourth field is a longer string that gives a detailed description of the reported issue. The fifth and sixth fields are the geographical latitude and longitude, respectively, of the location of the reported issue. The final field in the data is the single-word string tag that was given to categorize the issue by the user. Many examples were not categorized, and have a default tag.

Methods
The process of generating the final model was completed in four phases. In the first phase, a set of relevant features were extracted from the raw data. Second, a preliminary model was fit to the data to determine a baseline prediction performance. Next, a new modeling algorithm was applied to generate multiple submodels, and a weighted average was applied to the submodel results to generate the final result. Finally, the accuracy of the final model was compared to the preliminary model.

In the feature extraction and selection phase, 7 relevant features were chosen. The first feature selected is the geographical location of the problem as derived from provided latitude and longitude data. Figure 1 shows the locations of all reported issues. Four major clusters are easily seen within the dataset, which correspond to a city.

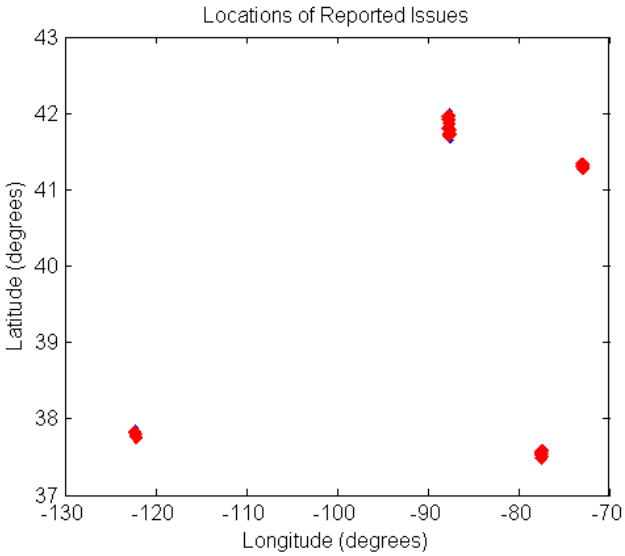

Figure 1. Plot of latitude and longitude of all reported issues in training examples

A k-means algorithm was used to identify 25 centroids in the lat-long data to further refine the apparent city clusters into a location feature representing a neighborhood within a city. The resulting clusters in each city are shown in Figure 2 and Figure 3. Each example was then assigned a location feature based on its nearest centroid.

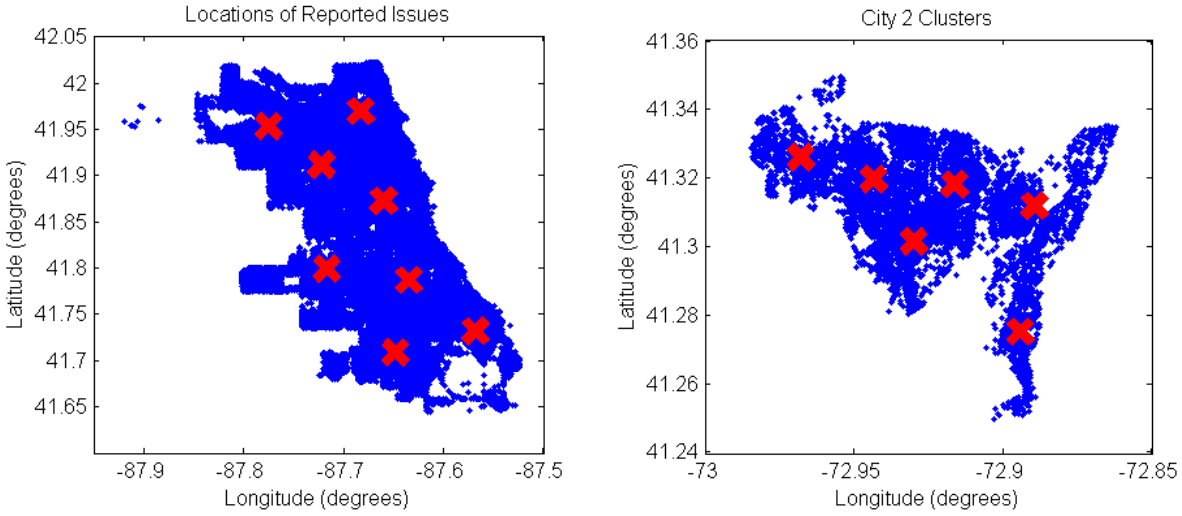

Figure 2. Location feature centroids (red) plotted with reported issues (blue) within City 1 and City 2.

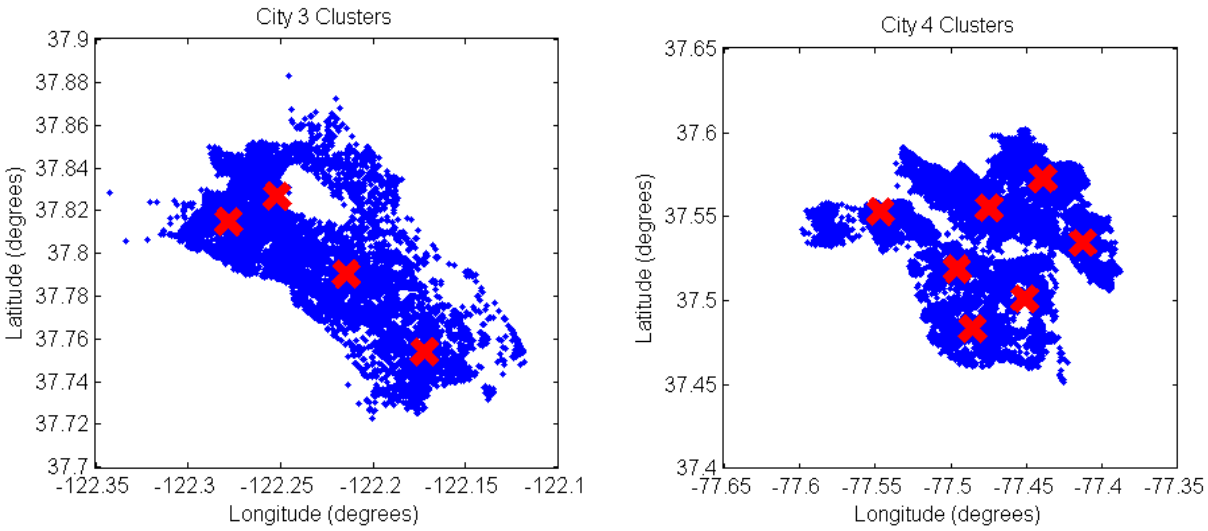

Figure 2. Location feature centroids (red) plotted with reported issues (blue) within City 3 and City 4.

The second feature extracted is a learned subject feature, which was extracted from the Summary, and Description text fields in the raw data. The methodology was to use a bag-of-words model along with a supervised Naive Bayes model. The bag-of-words was chosen to be the 6000 most frequently used words, excepting ubiquitous and generic words (e.g. “it,” “the,” “and,” “is,” etc.) The Naive Bayes model was trained using examples from all data available that were given a tag value in the raw data. The subject for all examples without tag values was then predicted, along with the probability of being the correct subject based on the model. All predictions for subject that exceeded 30% probability were assigned the predicted subject. All others were assigned a subject of “uncertain.”

The third feature extracted is the local time of day that the problem was submitted. This was determined from the timestamp in conjunction with the location feature. Each location was assigned an approximate time zone based on the centroid longitude. The time from the timestamp of each example was adjusted according to the local time zone. The feature value is the resulting time converted to seconds, considered to be a continuous feature.

The fourth feature is the day of the week that the problem was submitted. This was derived from the provided submission timestamp. Each example was assigned a value of 1 through 7 based on which weekday corresponded to the timestamp, using the built-in Matlab function *weekday()*.

The final three features selected are the category tag from raw data, the sequential month count of the examples, and the log of the number of words in the detailed Description field of the raw data.

After all features were extracted from the data, the preliminary baseline model was computed as a simple regression tree algorithm to return continuous prediction values. The Matlab Statistics Toolbox RegressionTree class[4] was used with the extracted features and default model

parameters to formulate three models; one for each prediction class of views, votes, and comments.

Once the baseline prediction accuracy was determined, a new model was fit to the data to improve prediction accuracy. In this model, a submodel was fit to data from each month, producing 16 submodels for each prediction class for a total of 48 submodels. Each submodel is a best-fit to a Gradient Boosted Decision Tree Regression class using Python Scikit-Learn ensemble tools class: sklearn.ensemble.GradientBoostingRegressor.[2] The class parameters were fit to a local optimum through experimentation. To obtain a single prediction from the 16 submodels, a weighted average was applied based on a gaussian distribution as in Equation 1.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (1)$$

where f is submodel weight, x is submodel month, μ is example month, and σ is a free parameter to be determined experimentally for best fit.

The modeling parameters refined during the project were the learning rate, number of estimators, and max tree depth for the submodel; and standard deviation for the weighted average. This process was performed by manually performing a gradient descent for these parameters, where each iteration adjusted a parameter by a small amount to determine if accuracy was improved until a local optimum was found.

Results

To measure prediction accuracy, a root mean squares log error (RMSLE) method was used across all three prediction values (views, comments,votes) as summarized in Equation 2.

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

where p_i is prediction for tuple (example, *field*), i; a_i is actual value for tuple (example, *field*), i; and *field* takes on values of views, votes, and comments.

The results of the preliminary baseline model yielded a training set RMSLE of 0.322. The same model was then used to predict the test set of data with a RMSLE of 0.556. The learning curve of RMSLE versus number of training examples is shown in Figure 4. The plot shows that error has reached a steady value for the full dataset. Therefore bias error is expected to be the main source of error as opposed to variance error, and further reduction in test error will come from an improved model.

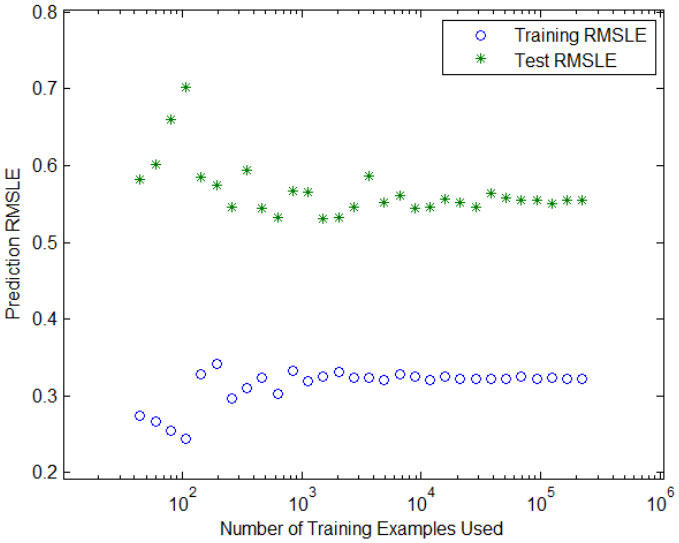

Figure 4. Learning curve for preliminary model on semi-log axes

Using the process described in the Methods section, the final model parameter values for the local optimum converged are learning rate of 0.01, number of estimators of 1000, max tree depth of 10, and standard deviation of 1.4. The prediction RMSLE from this model is 0.495 for the training set, and 0.350 for the test set. This represents a 37.1% reduction of test error from the preliminary model to the final model.

Conclusion

This project was successful in reducing the prediction error of the future reported issue importance based on past data. The final RMSLE of the predictions is 0.350, which represents a 37.1% reduction from the preliminary model. Based on the method used to arrive at this model, it is highly likely that the result represents only a local optimum model, and that further gains are possible through model refinement if higher accuracy is needed.

References

1. en.seeclickfix.com Dec 1, 2013
2. "3.2.3.3.6. sklearn.ensemble.GradientBoostingRegressor" scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor Dec 1, 2013
3. "See Click Predict Fix." www.kaggle.com/c/see-click-predict-fix Kaggle Inc. Dec 1, 2013
4. "RegressionTree Class." www.mathworks.com/help/stats/regressiontreeclass.html The Mathworks Inc. Dec 1, 2013
5. Ng, Andrew. "CS 229: Machine Learning," Course Notes. Stanford University. Autumn 2013