An NLP-based Approach for Improving Prediction of Twitter Follower Count

Brandon Carter and Brandon Axe
Massachusetts Institute of Technology

6.864 Advanced Natural Language Processing
December 14, 2015

The goal of this project was to use NLP-based methods to improve upon traditional machine learning approaches to predict the follower count for a user on Twitter given tweet text and simple metadata. The Twitter corpus presents a variety of interesting and difficult ML and NLP problems partly because tweets are limited to 140 characters and often lack syntactic correctness. They also contain hashtags, mentions, emoticons, and links, as well as conversational abbreviations which can be used to improve NLP models. We train various regression models to predict the number of Twitter followers from tweets and find that NLP augmentation improves prediction accuracy over purely ML approaches, though the significance of the improvement is dependent on the type of regression. We show that the most accurate predictions follow training with hybrid NLP and ML methods.

The tools from this project are available at: https://github.mit.edu/bcarter/6864FinalProject

---

INTRODUCTION

The Twitter corpus presents a variety of very interesting natural language processing (NLP) and machine learning (ML) problems due to its size (over 300 million monthly active users[1]) as well as lack of formal syntactic structure, spelling, grammar, etc. in tweet text, which follows from a 140 character limit per tweet (Einstein 2013).

Primary issues in text from microblogging services like Twitter include unintentional typographical errors, variation in dialect, range of content, and informal use of language and orthography (Einstein 2013). Due to their noisy and informal nature, tweets present challenges

[1] About Twitter: https://about.twitter.com/company.

for language technology (Ritter et al. 2011), and so traditional machine learning approaches have difficulty dealing with the Twitter corpus.

Moreover, because most NLP tools are developed for use with highly edited genres such as news corpora, they need to be modified for use with the Twitter corpus because conversational text like that written on Twitter contains main non-standard lexical items and syntactic patterns (Owoputi et al. 2013). Despite these issues, however, tweets also include hashtags, mentions, emoticons, and links, which create some noise in the tweet data but at the same time can aid the learning tasks if used intelligently.

Previous work has suggested that combined NLP and ML approaches can improve accuracies in the task of tweet classification (Stavrianou et al. 2014). Our goal was to combine traditional ML and NLP techniques in order to predict the number of followers for a Twitter user based on a given tweet. Number of followers is a valuable metric for a prediction task because it is an easily observable characteristic for a given Twitter account. It is also useful in approximating a Twitter user's influence in the Twitter social network without requiring the entire graph of users.

We show that augmenting traditional machine learning approaches with NLP techniques such as word vector embeddings and part-of-speech (POS) tagging can improve the accuracy of predicting Twitter follower count in various regression models.

METHODS

Scraping and Tokenization

We obtained a corpus of 1,364,728 raw tweets and associated metadata by streaming all English tweets from the Twitter API[2] between November 27-28, 2015. It is critical that tweets are from a similar time period because tweet content exhibits strong statistical dependence over time, which would otherwise present challenges during evaluation (Karimi et al. 2015).

Unfortunately, our corpus of tweets cannot be shared or re-uploaded in accordance with the Twitter API policies. However, it is easy to obtain a Twitter API key and re-extract similar corpora using the provided code and instructions in the readme documentation.

The corpus was cleaned by first removing any erroneous tweets with missing data. Raw tweet text was first tokenized using Twokenizer, a tool from CMU built for unigram tokenization of Twitter text[3]. This tokenizer works well because it can recognize hashtag patterns, "@"-mention patterns, emoji patterns, and more. Tokens were then stripped of capitalization. It has been found by Ritter et al. that capitalization in tweets is much less reliable for NLP tasks than in other edited texts (Ritter et al. 2011). Some tweets are written in all caps, while others are all lowercase, and removing capitalization improves simplicity of other NLP tasks. All

[2] Twitter API: https://dev.twitter.com.
[3] Twitter Natural Language Processing: http://www.cs.cmu.edu/~ark/TweetNLP/index.html.

hyperlinks in tweets were then replaced with a static 'URL' string, which was done because we do not extract any information from the hyperlink or corresponding webpage. Rather, it is useful to know if and where links appear within tweets. Designing methods to extract useful natural language and contextual information from linked content in tweets is left as future work. Finally, stopword tokens were removed based on the published set of English stopwords from NLTK[4]. Tweets with no valid tokens were discarded.

Post-tokenization, the corpus consisted of 1,362,482 valid tweets (99.8% of raw tweets) with distributions for tweets length, tokenization, and number of followers as shown in Table 1 and Figure 1.

	Num Followers	**Raw Tweet Length**	**Num Tokens**
Min	0	4	1
Max	13,477,840	473 (due to ascii conversions)	56
Mean	2124.3	64.36	7.01
Median	507	55	6
STDEV	35,897.15	37.09	4.54

Table 1: Distribution statistics for tweet corpus

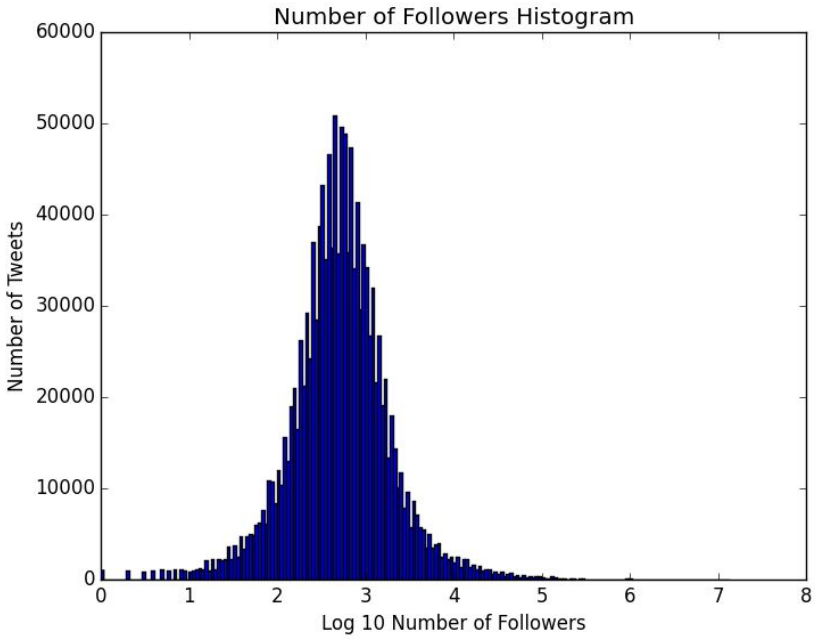

Figure 1: Histogram of number of followers for tweets in the corpus

[4] Natural Language Toolkit: http://www.nltk.org.

Word Embeddings and POS Tagging

Word embeddings were computed using the Gensim port of word2vec[5,6]. The word2vec embeddings model was trained on the entire corpus of tweets with dimension 100 and default parameters, and each individual tweet's embedding is modeled as the mean of the embedding vectors for each of its tokens, using the average word2vec vector for the entire corpus in place of any unseen tokens.

Part-of-speech tagging was performed using a Java-based tagger from the CMU Twitter NLP tools[7] (Owoputi et al. 2013). This tagger was trained on the Twitter corpus to tag conversational speech that occurs frequently in tweet text, recognizing words like "idk" or "smh" and achieving 93% tagging accuracy on tweets as published (Owoputi et al. 2013). The POS tagger recognizes 25 unique parts of speech.

Feature Vectors

Feature vectors were engineered by concatenation of four components: (a) machine learning features, (b) NLP features, (c) word embeddings, and (d) POS tags. Regressions as described below were run using various subset concatenations as well as the total concatenation of these four groups of vectors in order to evaluate how resulting performance could be attributed to each component.

(a) The machine learning features include simple non-natural language based tweet metadata that classical machine learning algorithms might use to predict twitter number of followers based on the user who tweeted. These features include the age of the user's Twitter account (which we converted into the number of epoch-seconds since Twitter was launched), the total number of tweets from that user's account, the number of other Twitter users that the user is following (the "friends count"), the total number of tweets from other users that the user has marked as "favorites," and whether Twitter recognizes the user as "verified" (as a boolean value) to identify authentic accounts for key individuals and brands. The raw Twitter API tweet data provides all of these features along with the tweet text off the tweet stream, so no additional API requests were needed to obtain this metadata.

(b) For basic NLP features, we extracted some syntactic information from the tweet text itself. All of these features were easily observable based on the tweet. These features included: the length of the untokenized tweet text, the number of tokens in the tokenized text, the number of hashtags (user-provided categorizations for their tweet identified with '#' before a word or phrase), the number of emoticons (based on a large Western emoticon set from Wikipedia[8]), the number of URLs in the tweet, and the number of mentions made in the tweet (mentions notify

[5] Gensim Word2vec: https://radimrehurek.com/gensim/models/word2vec.html.
[6] Word2vec: https://code.google.com/p/word2vec/.
[7] Twitter Natural Language Processing: http://www.cs.cmu.edu/~ark/TweetNLP/index.html.
[8] List of Emoticons: https://en.wikipedia.org/wiki/List_of_emoticons#Western.

other Twitter users directly of the tweet and are identified with an '@' before the desired user's account name). In particular, emoticons can improve the NLP task because they mirror the role played by facial expressions in speech and also provide some pragmatic role of marking utterances as facetious, for example (Einstein 2013).

(c) Word embedding vectors of length 100 were used to capture capture deep semantic structural information from tweet text. As described above, these vectors were produced as means over individual tokens in the tweet.

(d) POS tag vectors of length 25 were used to capture POS information from the tweets. The vector represents the frequency of each of the 25 parts of speech recognized by the tagger.

Full feature vectors include concatenation of (a), (b), (c), and (d) giving total dimension 136.

Baseline Vectors

As a baseline, "random" feature vectors were produced as 200-dimensional random vectors from [0, 1). Moreover, the "ML-only" vectors consisting of only the machine learning feature vectors (listed as (a) above) were used as a baseline for evaluating how NLP-based features from (b), (c), and (d) can augment the machine learning features to improve prediction accuracy. "NLP-only" vectors refer to vectors using just (b) above (no word embeddings or part of speech tags) to provide a baseline for comparing the simplistic NLP features to the word embeddings and POS tags.

Regression Methods

Using feature vectors as generated above, we trained various regression algorithms using a sample of 20,000 tweets (the training set) and then tested on another sample of 20,000 different tweets (the "dev"/testing set).

Our training prediction task uses the base-10 logarithm of the number of Twitter followers, rather than the number of followers itself. The log number of followers is a better count for this task for several reasons. Mainly, the same prediction difference for a user with few followers versus a user with many followers should be treated differently. As an arbitrary concrete example of this, predicting incorrectly by 1000 followers for a user with 100 followers is a much worse prediction than predicting incorrectly by 1000 followers for a user with 10000 followers. The difference between 1100 and 100 is the same as the difference between 11000 and 10000, but the difference between $\log_{10}(1100)$ and $\log_{10}(100)$ is 3.041-2=1.041 while the difference between $\log_{10}(11000)$ and $\log_{10}(10000)$ is 4.041-4=0.041. As desired, a deviation of 1000 from 100 is seen as much worse than the same deviation from 10000. Furthermore, the histogram distribution for the log 10 number of followers takes the shape of a normal distribution, as shown in a histogram of our corpus of tweets (Figure 1), which makes the log 10 value better suited for fitting a regression curve.

The most basic regression algorithm used was linear regression[9]. Linear regression is convenient because of its speed and simplicity compared to the other algorithms, as well as being a helpful baseline to compare the results of other, more involved regression algorithms. The error function used by linear regression was the standard least-squares, fitting our feature vector matrix X to log 10 follower counts y by finding the parameter vector w which minimized the norm $||Xw-y||^2$.

Another regression algorithm used was support vector regression (SVR)[10]. SVR was chosen because it allowed for the radial basis function, $exp(-||x-x'||^2)$, as a kernel, which is related to Gaussian curves. SVR works similarly to support vector machine classification, building a model that predicts a value based off of the separation from a maximal-margin hyperplane using the kernel function as a dot product. The difficulty with using this regression algorithm is performance, as computing the kernel function for every pair of vectors is much slower than taking a dot product.

Lastly, we used a random forest regression (RFR)[11]. This algorithm uses an ensemble classifier of different decision trees, the randomness coming from each decision tree in the ensemble using only a portion of the features provided. An ensemble classifier is appropriate because of the large size of our data sets, and allows for a non-linear relationship. At the same time, it has a parallelized training algorithm which allows for more training.

All regression models were implemented using the scikit-learn Python library.

Evaluation

To evaluate prediction accuracy of the different regression models and choices of feature vectors, we use root mean square error (RMSE) on the predicted and actual log follower count. RMSE has been shown to be among the best evaluation methods in comparing regression models (Nau 2015). RMSE is calculated by first squaring the difference between the actual and predicted count for each of the data points, taking the mean, and then taking the square root. Squaring the difference penalizes more strongly when predictions are further from the actual value, which is desired, because being off by a log count of 2 is more than twice as bad as being off by a log count of 1. Taking the mean and then a square root causes the resulting value to be equally influenced by each data point and have the same units as the original predictions. We utilized the mean square error implementation provided by scikit-learn.

[9] Sklearn Linear Regression: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.
[10] Support Vector Machines: http://scikit-learn.org/stable/modules/svm.html#regression.
[11] Random Forest Regressor: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

RESULTS AND DISCUSSION

Figures 2, 3, and 4 show the results of each of the three regression models using all features. In the figures, the blue curve shows actual log follower count per tweet (in sorted order), while the green curve shows the regression algorithm's prediction for the same tweet.

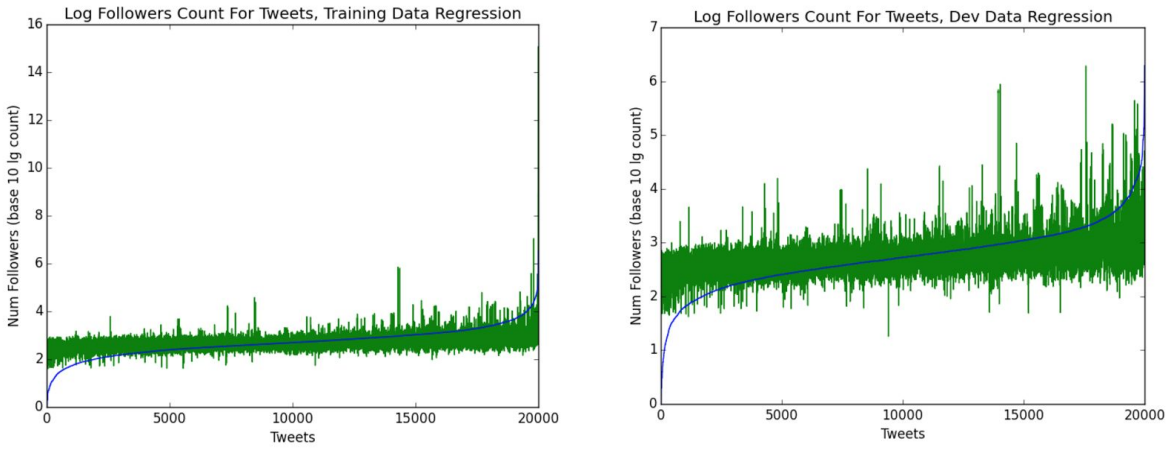

Figure 2: Actual vs Predicted Number of Followers per Tweet, Linear Regression

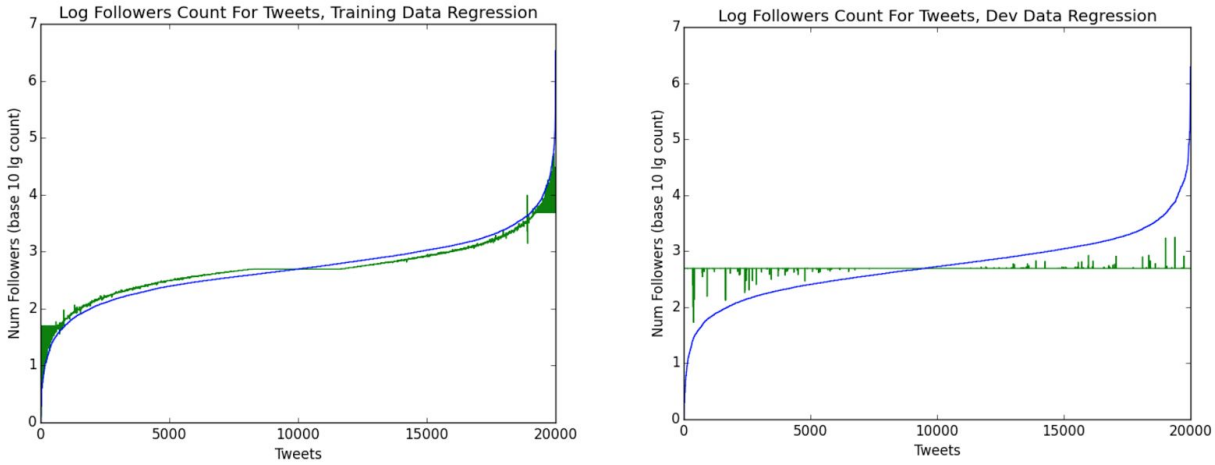

Figure 3: Actual vs Predicted Number of Followers per Tweet, SVR Regression (RBF kernel)

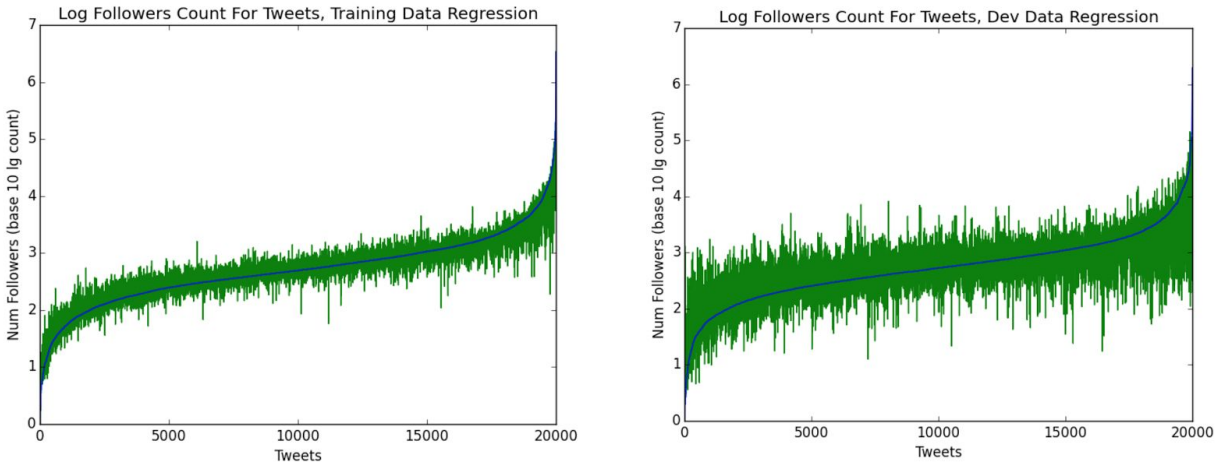

Figure 4: Actual vs Predicted Number of Followers per Tweet, Random Forest Regression

We see that the linear regression has significantly more noise than the other regression models and was not able to accurately predict follower count on either the training or the dev sets (Figure 2). The support vector regression has very good performance on the training set, but seems to overfit, as it predicts very close to the mean log number of followers for most the unseen data in the dev set (Figure 3). The random forest regression has the best fit, as we can see a strong correlation on both the training and dev data sets, with slightly more nose on the dev set (Figure 4).

We used random feature vectors and ML-only feature vecturs as baselines and compared the results to using NLP-only features (the concatenation of NLP features, word embeddings, and POS tagging), as well as both ML and NLP features (all features) for each of the three regression algorithms. We provide four different RMSE values for all of these models in Table 2. In addition to the overall RMSE of the predictions, we report performance on subsets of the data with follower count that was low (<100, or log count <2), medium (101-10000, or log count 2-4), or high (>10000, or log count >4). Ideally, the regression algorithm performs well on the whole dataset as well as on each of these three “bins” of Twitter users.

Regression Type	Feature Vector	Training RMSE	Training Low RMSE	Training Mid RMSE	Training High RMSE	Dev RMSE	Dev Low RMSE	Dev Med RMSE	Dev High RMSE
Linear	Random	0.6029	1.1770	0.4204	1.7395	0.6896	1.2536	0.4982	1.7539
SVR	Random	0.6017	1.1715	0.4196	1.7448	0.6388	1.1876	0.4779	1.7587
Random Forest	Random	0.6308	1.1835	0.4606	1.7616	0.6230	1.1772	0.4602	1.7877
Linear	ML Only	0.5274	0.9933	0.3828	1.4837	0.5201	0.9732	0.3871	1.4879
SVR	ML Only	0.2398	0.4273	0.1683	0.7863	0.5867	1.1334	0.4170	1.7715
Random Forest	ML Only	0.1252	0.1685	0.1061	0.3547	0.3618	0.5054	0.2920	1.1547
Linear	All NLP	0.5867	1.1282	0.4142	1.6938	0.5809	1.1260	0.4147	1.7290
SVR	All NLP	0.5630	1.0813	0.3925	1.6738	0.5834	1.1205	0.4186	1.7423
Random Forest	All NLP	0.2562	0.4415	0.2024	0.6638	0.6112	1.1467	0.4604	1.6889
Linear	All	0.5136	0.9385	0.3810	1.4445	0.5114	0.9231	0.3870	1.4801
SVR	All	0.1718	0.3067	0.0952	0.7195	0.5861	1.1411	0.4150	1.7632
Random Forest	All	0.1419	0.1732	0.1213	0.4228	0.3604	0.4886	0.2910	1.1715

Table 2: RMSE data for each of regression algorithm and selection of features

Figures 5 and 6 visualize the RMSE across the various regression models. Figure 5 shows the binned RMSE results for each of the tests on the dev data. Figure 6 shows the training and dev RMSE values for the entire dataset (unbinned) for each of the regressions, and for each of the feature vector sets (random, ML-only, NLP-only, NLP+ML).

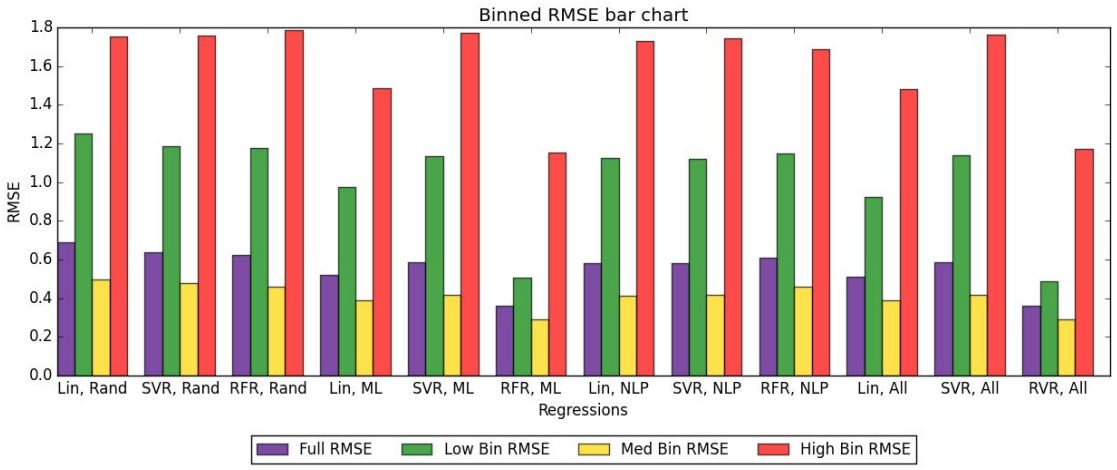

Figure 5: Binned RMSE on Dev Set Prediction Results vs Regression Algorithm

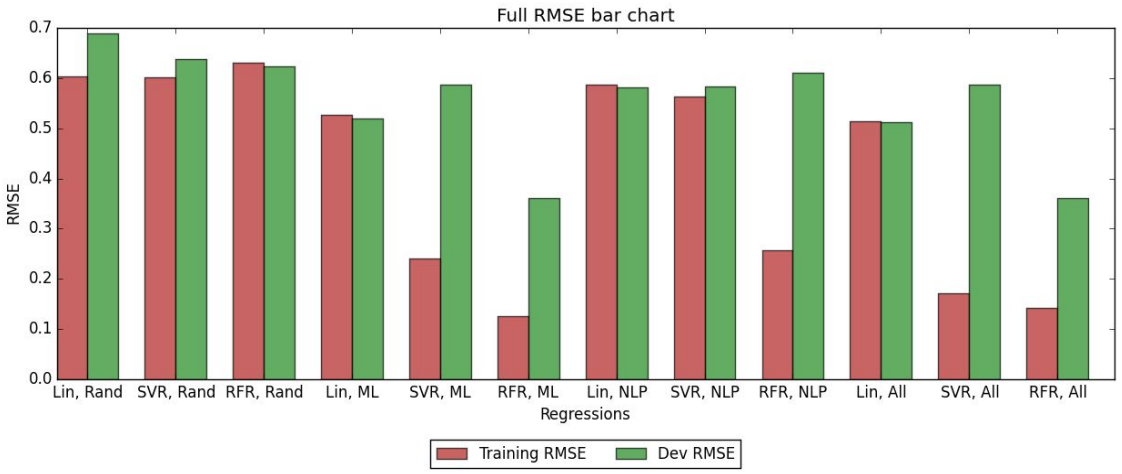

Figure 6: Training and Dev Data Prediction RMSE vs Regression Algorithm

We can see that compared to the random baseline across all tweets, RFR with ML-only features performs almost equally well to RFR with all features (Figure 6). Considering the binned results, we see that most of the error in RFR is due to error in the high bin, while it has the lowest RMSE for both the full, low, and medium bins. In general, the medium bin has lower RMSE than the low or high bins, because there are significantly more users in this bin, so predicting poorly in this bin is penalized more harshly by regression algorithms than poor performance in the low or high bins.

From Table 2, we see that across all models, NLP+ML features has a lower RMSE on the dev tweets than the respective models with ML-only features, though these results are not very significant. Such little difference between the results of ML-only features and NLP+ML features is unexpected, and may suggest that the metadata features are much more relevant than the NLP features. However, it seems that using NLP features should have more than a negligible impact on our results, so this result may be a consequence of not scaling the various components of our feature vectors. The expected values for the ML features is larger than that of the other features; for instance, for most vectors the greatest value in any dimension is the number of followers. If we were to preprocess the data by scaling each of the vector dimension so that all values were in the range [0,1], we might find better results, especially with SVR using an RBF kernel[12], considering it is a scale-invariant regression algorithm.

Future work will involve investigating how these NLP methods can be extended to continue to improve the ML methods. As discussed previously, we believe relevant natural language information can be extracted from links in tweets to provide more information about the tweet and user. We also hope to augment the NLP vectors with top n-grams as these provide insight into tweet content. Further, as discussed above, we hope to provide a better normalization between the ML and NLP vectors. Another interesting possibility for future work involves

[12] Preprocessing Data: http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

training a prediction model using a deep neural net that can minimize RMSE loss and then comparing the results of this model to the classical regression models we have discussed.

We have shown how NLP methods can improve accuracy of traditional ML methods in the case of predicting number of followers on tweets. Adopting NLP techniques on these microblog corpora allow the learning tasks to accommodate short text, lack of syntactic structure, and orthography, and we hope that this work provides a springboard for future hybrid NLP and ML approaches that can solve problems involving short conversational text like that on Twitter.

REFERENCES

J. Einstein. “What to do about bad language on the internet.” In: Proc. of NAACL (2013).

S. Karimi, J. Yin, J. Baum. “Evaluation Methods for Statistically Dependent Text.” Computational Linguistics: 539-48 (2015).

R. Nau. “How to Compare Regression Models.” Accessed December 11, 2015. http://people.duke.edu/~rnau/compare.htm.

O. Owoputi, B. O’Connor, C. Dyer, K. Gimpel, Nathan Schneider, Noah A. Smith. “Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters.” HLT-NAACL: 380-390 (2013)

A. Ritter, S. Clark, O. Etzioni. “Named Entity Recognition in Tweets: An Experimental Study.” In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (2011).

A. Stavrianou, C. Brun, T. Silander, C. Roux. “NLP-based Feature Extraction for Automated Tweet Classification.” In: Proceedings of DMNLP (2014).