Predicting the Popularity of Social News Posts

Joe Maguire
Department of Electrical Engineering
Stanford University
Stanford, California
jmaguire@stanford.edu

Scott Michelson
Department of Electrical Engineering
Stanford University
Stanford, California
smich@stanford.edu

***Abstract* — This project demonstrates that machine learning can be used to accurately predict a post's popularity. After collecting several thousand posts from HackerNews over several weeks, basic machine learning techniques were applied to a generic set of features. After analyzing trends in the data and refining the learning processes, our model predicted a post's popularity with 85% accuracy. These results demonstrate that basic machine learning models can be used to predict the popularity of social media posts**

Keywords—Social News; Machine Learning

I. Introduction

Over the past several years, social media has gained a wide and influential presence online. The emergence of social media news sites is reflective of this trend. Sites like HackerNews and Reddit drive enormous amounts of traffic and draw on impressive user bases. As these sites become more and more popular, the capability they provide to influence or connect to large segments of the population will become increasingly desirable. Determining if there are well-defined features in posts that accurately predict a post's popularity is both an interesting and a useful endeavor.

For the purposes of this paper, we define a popular post to be one that has received over 100 'upvotes', and an unpopular one to be one that has received less than 50.

II. Training Data

A. Data Collection

We chose to collect training data from Hacker News (http://news.ycombinator.com/). Hacker News provides a slightly more restricted set of post options when compared to social news sites like Reddit, such as restricting the use of pictures. This makes for a more focused analysis.

We collected the training data by utilizing a Hacker News API called Rewind Hacker News[3]. This resource samples the top several hundred posts at different times of day and stores them in a json format. Using this API, we acquired around 4000 posts from about a two month period. The time of these posts range across all hours of the day.

B. Features

The feature set we used included the following features

- The words in the post
- The domain the post links to
- The time of day the post was created, in units of half hours

Before creating test and training data, we used posts to populate dictionaries with words in the posts and domain names from post links. Unknown fields were added to each field to account for keys found in the testing data but not the sample data.

Time of day was recorded by discretizing time into half hour periods, and marking a bit for whichever period the post fell into.

Some features are currently left out: For example, poster karma (a reflection of how popular the poster is) is naturally a useful indicator of a popular post; However, we thought it would be useful to initially avoid features intrinsic to the poster. Since this project is about what makes a popular post, an answer such as 'a popular poster' is not useful for someone hoping to penetrate the social news community.

III. Machine Learning Approaches

We used multiple machine learning techniques to gauge the feasibility of our problems. Our experiences with them are cataloged in this section.

A. Naïve Bayes

Our initial approach used an extremely simple Multivariate Naive Bayes model, run with our full feature set. Since the contents of the posts are relatively short, representing the words as binary flags is not necessarily a huge detriment, the binary nature of the model still failed to take into account all of the information present in posts, and this approach gave us about 70% accuracy. With a multinomial event model we saw the chance to leverage more context than simply the presence or absence of a word. The simple move to a multinomial model with no other changes caused an increase of 4% to our initial accuracy.

Our initial feature set contained roughly 10,000 distinct features, and many of the features fell into two not particularly useful camps. Many words were domain specific or otherwise extremely rare, and showed up in very few posts. Another set consisted mainly of articles and prepositions, and showed up in

an extremely large number of posts. We surmised that the presence of large numbers of features of this sort contributed to a significant amount of noise in the data. We decided to group each of these sets into single categories, either 'uncommon word' or 'stopword', to cut down on the influence, without completely discarding the information. After this trimming, we were able to improve our accuracy by another 4% to 78%.

Our final optimization was to use a more involved method to determine the most useful features to the learning process. We decided to use the Kullback-Leibler (K-L) Measure to assess the usefulness of our various features. Our approach to this is discussed in detail in Section IV. Sweeping the number of features used, we found that around 200 features gave us the best accuracy while still retaining as many features as possible. With this optimization, Naïve Bayes achieved 81% accuracy.

B. *Perceptron*

Our data was collected over a period of about 8 weeks. While a model we generate now may be able to accurately predict popularity for a short time, internet trends tend to be very short lived, and an unadaptive algorithm may quickly become obsolete. This makes highly adaptive algorithms like the Perceptron appealing for their built in ability to learn continuously, despite their simplicity

The Perceptron achieved 73% accuracy on our test set after running through our entire training set. The error on the second half of our training set was about 71%, suggesting it's unlikely more training examples would significantly improve accuracy. We surmise that while it doesn't do anything too complicated, the Perceptron does well to capture easy patterns that lead to popular posts, such as very popular buzzwords, websites, and times, and manages to catch the low-hanging fruit.

As a continuation to the experiment, we did some investigation into a kernelized Perceptron. The kernelized version could measure cross-correlation between various features, which is important in text analysis since words in a post are not independent. In addition, a kernelized perceptron stores all of the examples where it mispredicted; this can be a downside, as the storage requirements may not be trivial with a high error rate, however by also storing the dates we can gradually lower the influence of older examples when calculating a new prediction.

We implemented a simple quadratic kernel perceptron, and found that it improved accuracy to 77%. This is an interesting result, but given that it didn't even perform as well as Naive Bayes, we decided not to pursue the avenue any further.

C. *Least Squares*

With previous approaches we treated popularity as binary: a post is either popular, or it is not popular. However, this is not a limitation of our data. Rather, we have non binary metrics for popularity in the form of comments and upvotes. As such, it seemed useful to try and perform a regression on the data in order to try to predict the precise value of those quantities. To this end, we briefly investigated linear regression models.

Our initial attempt tried to create a simple least squares fit model. This had the effect of projecting essentially every test input to about the same popularity, which was undesirable. To curb this, we attempted a locally weighted least squares model; however, regardless of bandwidth parameter we saw results virtually indistinguishable from random.

Our initial guess as to our poor success was that the large feature set was introducing too much noise into our fit. In attempt to address this, we decided to try projecting the features into a lower-dimensional feature space in an attempt to digest the data and reduce noise. We retried our fit using Principal Component Analysis to project into spaces ranging from 1 to 20 dimensions. The results were poor; our best prediction occurred with 10 dimensions, where we got 61% accuracy. Even this result may simply be the result of noise. Given how poorly this approach performed, we did not investigate it further, or otherwise examine how to reduce the RMS error of predictions.

D. *Support Vector Machine*

After investigation into more unconventional methods, we decided to return to the more standard models. As a follow up to Naive Bayes, we ran a Linear SVM on the data, with the features optimized in the same way. We had previously seen that when run on a large amount of text data, SVMs tended to have better asymptotic performance than Naive Bayes. This was corroborated by the results; the SVM achieved 85% accuracy on the data.

Encouraged by these results, we ran kernelized versions of the SVM over the data using the LIBSVM library[4]. We tried a sigmoid kernel, a radial basis function kernel, and a variety of polynomial kernels. Unfortunately, none of these performed as well as the linear kernel. This was somewhat surprising, given the improvement kernelization provided to the Perceptron. Our current theory is that while kernelized SVMs can fit relatively complex functions to the data, they overfit the training set in our high dimensional feature space, making the resulting models inextensible to other data. Because the kernelized perceptron is continuously learning as it classifies data, it was not susceptible to this overfitting problem.

IV. K-L Divergence Scoring

The Kullback-Leibler Measure is commonly a used technique for feature analysis, providing a measure of how much information a specific feature provides to the learning process. We hoped that scoring would allow us to reduce the size of the feature set while maintaining or improving classification accuracy. Furthermore, the KL-Measure is useful to elucidate underlying trends in the data. These trends can be used to better understand the data set.

We implemented K-L divergence as outlined in [1]. The endeavor was successful; in addition to actually improving our accuracy for Naïve Bayes, scoring provided additional insight into our data, discussed in part B of the next section.

V. Results

A. Machine Learning Classifications

Below is a summary of our achieved accuracy with each technique

TABLE I. Classification Accuracies

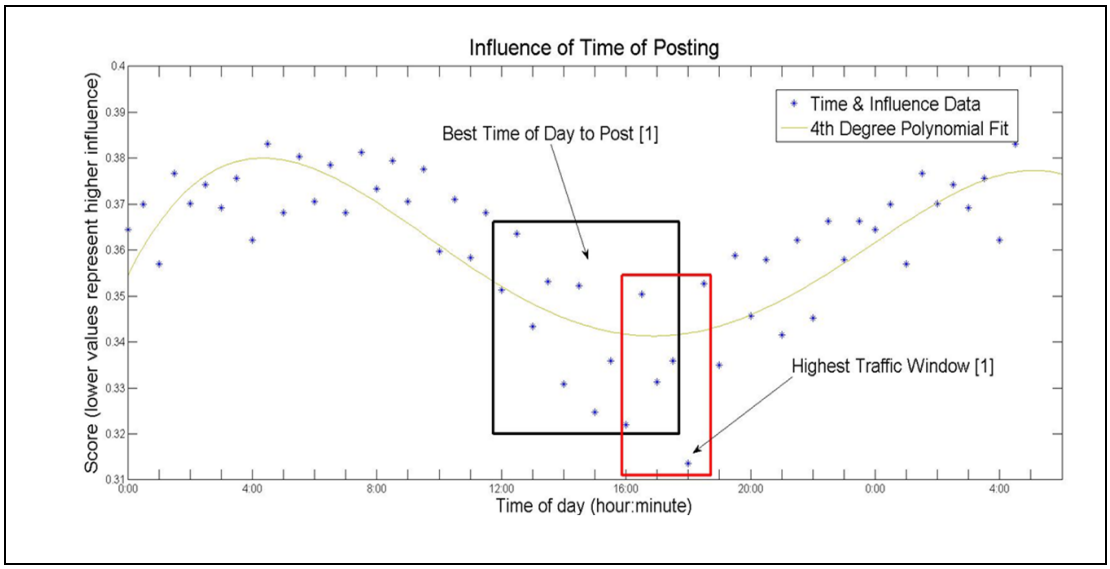
Algorithm	*Accuracy*
Multinomial Naïve Bayes, features narrowed by K-L divergence	81%
Quadratic Kernelized Perceptron	77%
Locally Weighted Least Squares, features projected to 10 dimensions	61%
Linear Support Vector Machine	85%

B. Feature Analysis

Besides using our resulting scores to improve our model, we also used our scores to better understand the dataset. Our scoring revealed several interesting if not surprising trends.

Time

The scores for the time related features showed that the time of posting was highly influential. A graph of the influence at a given time is shown below. It should be noted that the lower the score, the higher the influence on the learning algorithm.



We see a clear sinusoidal trend in this data. Interestingly, the periods of highest influence map well to a previous study of the best times to post on HackerNews[2]. While the match between our influence scores and the previously determined best time windows is good, it is not perfect. We are slightly concerned that the period of highest influence for our data occurs directly inside the window of highest traffic. One would assume that the period of highest influence would precede the window of highest traffic. A post posted right before the website is at peak traffic is more likely to be in a better position to be driven up or down in popularity by greater number of users. We theorize that this may be related to how we discretize time. A post that occurs at 12am is more closely related to a post that occurs at 11pm then a post that occurs at 6am. However we do not model this relationship, and instead treat time periods as orthogonal.

Domains

The table below shows the best scoring websites, divided into news sites and all sites. These are not particularly surprising, as they are all well known and respected websites. Note, however, that having a good K-L score does not necessarily imply how indicative that feature is of a popular post. Rather, it simply means the feature is useful for distinguishing between popular and unpopular. For example, Google tends to have relatively generic links associated with it, and may in fact be an indicator of unpopular posts.

TABLE II. Influential Domains

Rank	*News Sites*	*All Sites*
1	Tech Crunch	GitHub
2	The New York Times	Google
3	Ars Technica	Tech Crunch
4	Wired	The New York Times
5	BBC	Ars Technica

Words

The leading words in our analysis tended to be common tech-oriented words. In addition, the names of three major technology companies appeared as highly influential. It should be noted as before that high influence does not mean high popularity. A highly influential feature may be a great indicator of unpopularity.

TABLE III. Influential Words

Rank	*Word*	*Rank*	*Word*
1	Show	5	Web
2	Startup	6	Windows
3	Apple	7	Data
4	Google	8	Free

VI. Future Work

As of now, we only store the domain of the link. But, this is insufficient to truly understand what HackerNews readers find interesting about a link. Just because TechCrunch is highly influential doesn't mean that any link to TechCrunch will lead to a popular post. For example, a post linking to TechCrunch's contact information will not become popular. Therefore, it would be interesting to store features relating to the specific article on that domain. For example, we could use the title of the linked article as well as words from that article. An analysis of those new features would give a more complete understand of why certain URLs are more influential than others.

Because titles are relatively short, bigrams should be analyzed in a future study. This will make the feature analysis of words more powerful by providing the learning algorithm with some context. For example, it may be that all popular Google related posts contain the words "Google News" while all unpopular Google posts contain the words "Google Wave". Without bigrams, consecutive words like "Google" and "News" are treated as independent, which is likely not the case.

An ultimate extension of the research would be to use it to craft a popular post from scratch. Creating a bot to do this would be an interesting exercise.

VII. Conclusion

The rise of social news will not only continue to change how media is disseminated, but also force individuals as well as corporation to rethink how they communicate with their audience. This study has shown that machine learning can provide valuable insights into how to connect with users of social news sites by revealing links between post content and post popularity. This success should motivate a more in depth study of this domain. Ultimately the aim should be to demonstrate that the knowledge gained through these techniques can actually be applied to create popular posts.

Acknowledgment

We would like to thank Andrew Ng and the entirety of the CS229 Course staff for providing us with the tools and necessary machine learning background to conduct this research.

References

[1] G. Chang-Hwan Lee; Gutierrez, F.; Dejing Dou, "Calculating Feature Weights in Naive Bayes with Kullback-Leibler Measure," *Data Mining (ICDM), 2011 IEEE 11th International Conference on* , vol., no., pp.1146-1151, 11-14 Dec. 2011.

[2] "The Best Time to Post on Hacker News: A Comprehensive Answer". Retrieved December 2012, from: http://nathanael.hevenet.com/the-best-time-to-post-on-hacker-news-a-comprehensive-answer/

[3] Dabrowski, Dominik. Rewind HackerNews. Retrieved November 2012 from http://rewindhn.com/

[4] Chang, Chih-Chung; Lin, Chih-Jen, LIBSVM – A Library for Support Vector Machines. Retrieved November 2012 from http://www.csie.ntu.edu.tw/~cjlin/libsvm/