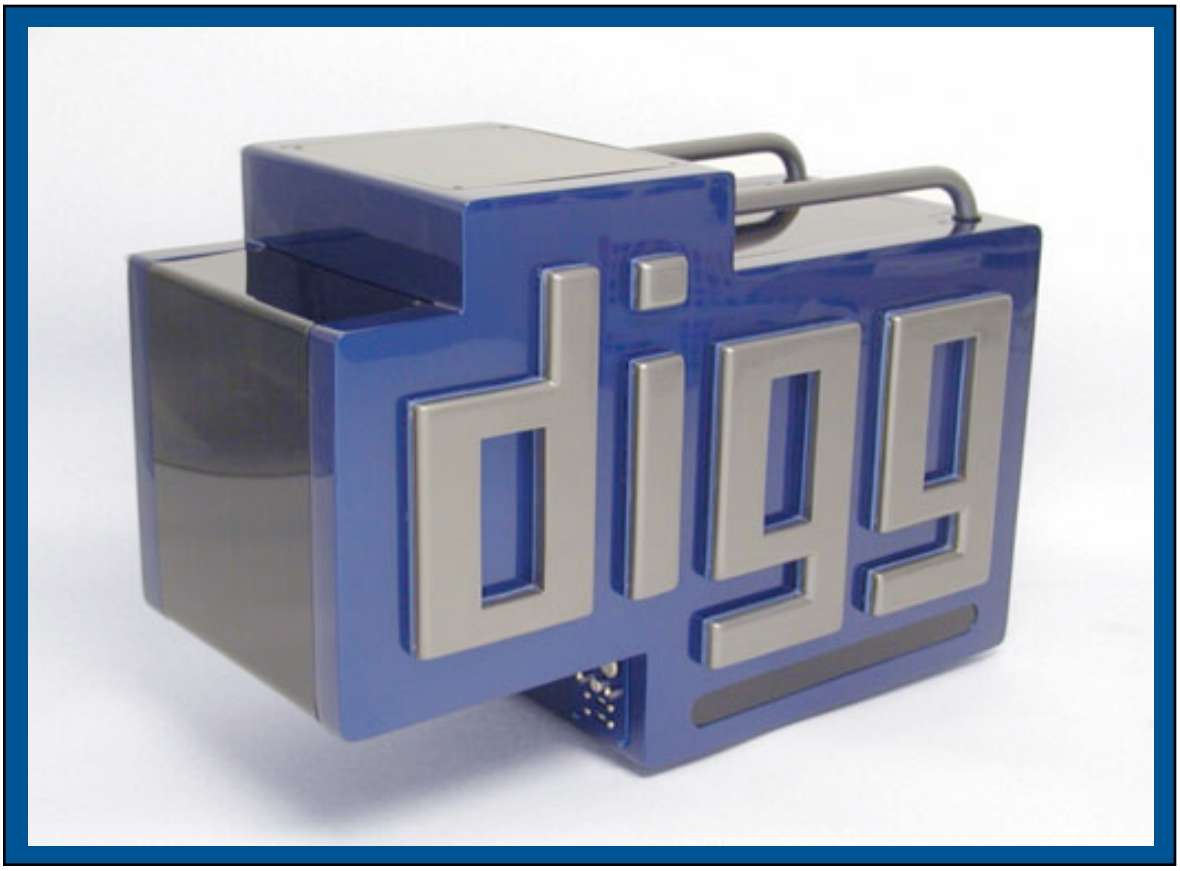
PROJECT REPORT

Analyzing Digg.com



Karn Seth
CS 34 Spring 2009

PROJECT REPORT

Analyzing Digg.com

Karn Seth
Cs34 Spring 2009

Introduction

In this project, I tried to tackle the problem of classifying stories submitted by users to the social news website digg.com.

Stories submitted to Digg are initially placed in an upcoming section, where they are voted on ("dugg") by the site's users. Stories that receive enough diggs are promoted to the Digg front page, after which they usually receive massive amounts of traffic.

Using information obtained using Digg's Public API, I tried to classify stories as "popular" or "unpopular", that is, whether they would eventually reach the front page or not.

I picked this particular topic because of the large, feature-rich dataset involved, and also in the hope of learning something interesting about an unusual website.

Dataset

The dataset consisted of about 3 million stories downloaded using the Digg API, with stories from over 400,000 users. Each story contained the following information:

- Story Title and Description

- Story Link

- Submitting User

- Current status (popular/upcoming)

In addition to this, I also obtained data on each of the 400,000 users, including

- List of friends

- List of fans

Other features, like total number of submissions from a user, or total number of popular stories from a user, were computed using the training set itself.

Methods

My classification approaches were as follows:

Logistic Regression:

The results of logistic regression were quite weak:

- Only 11.1% of popular stories were correctly classified as such
- Of the stories classified as popular, only 28.6 percent of them were actually popular. The rest were unpopular stories that had been misclassified.

Fuller details on this approach can be found in the milestone report. This approach was not used further.

Naive Bayes

After initial attempts at classification using logistic regression fared poorly, I decided to switch to a Naive Bayes approach with Laplacian Smoothing at the suggestion of the professor. This approach had two main advantages over logistic regression:

1) Parameters for the model could be computed without expensive iterative optimizations.
2) The model allowed us to explicitly include the fact that the distribution of popular stories is extremely skewed (~0.6%), through the inclusion of class priors.

There are also, of course, some disadvantages, chiefly that the model assumes that individual features in the input vector are conditionally independent. While this is a very strong assumption, and does not often hold in practice, in general the results obtained using Naive Bayes are not bad.

Naive Bayes was the main approach used, and will be discussed in the rest of the paper

Choosing the feature vector for each story:

I tried a variety of different feature vectors, at first using the following combination of features for each story:

1) User Features: Identity of the user submitting the story
2) Link Features: Identity of the link of the story
3) Title and Description features: A bit vector for each, title and description, with as many entries as there were distinct title or description words in the training set (excluding extremely common words). An entry in a particular story's vector was 1 if the corresponding word appeared in that story's title or description, and 0 otherwise.

4) User Friend features: A bit vector with one entry for each possible friend of the submitting user. An entry corresponding to a particular friend was 1 if he was friends with the submitting user. There were as many features as there were distinct friends in the training set.

My first attempt was to assume that the feature values all came from multinomial distributions.

Feature Vector with features drawn from multinomial distributions

In this case, for the User, I had a single feature that could take on a different value for each different user who could have submitted the story(~400,000 different values, varies based on training set size), and similarly for the Link, a single feature that could take on a different value for each possible link the story could go to (~500,000 different values, varies based on training set size). This did not work very well at all, since, among other things, the smoothing term for these features was extremely large, and appeared to drown out the significance of the training data. In fact, removing the features entirely had no effect on the results, suggesting that the smoothing made the features meaningless in their current form.

I decided to alter the feature vector so that the user and link features were now had 3 real valued features each instead of a single discrete-valued feature, and further assumed that these real-valued features were drawn from gaussians.

Resvised feature vector, with real-valued User and Link features.

The three real-valued features that I chose to use were:

- Total number of stories from that user/link (popular + unpopular)

- Total number of popular stories from that user/link

- Percentage of popular stories from that user/link

The "words" and "friends" sections of the vector remained unchanged.

(I experimented with dropping some of these features later on, first the friends features, and then both the friends and words features. The motivation and results for this are discussed below.)

Based on this feature choice, I implemented Naive Bayes in Python (Matlab was not used, since large matrices did not fit in system memory). The implementation took advantage of the fact that the "words" and "friends" parts of the feature vectors were extremely sparse by precomputing the output assuming all the entries were 0, and then adding correcting terms for those entries which were actually 1. This greatly reduced space requirements and running time, because only the entries with 1 needed to be remembered and computed on.

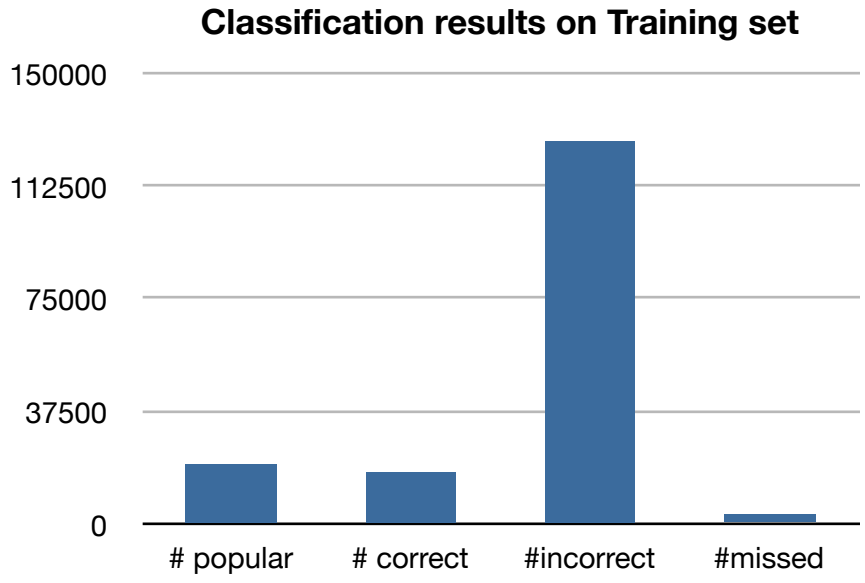
Results

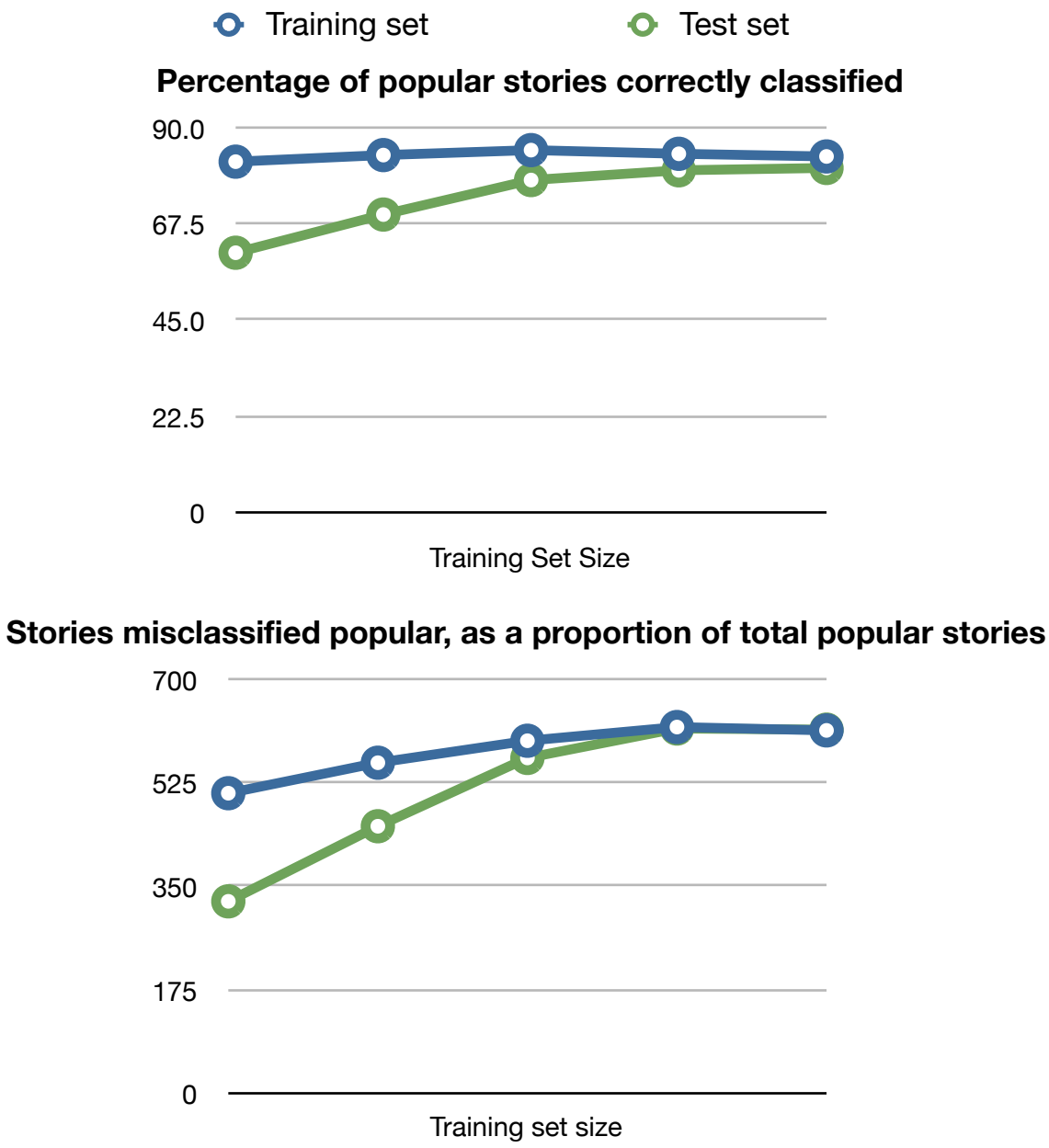
For the line- graphs below, the five plotted points correspond to 5 different sizes of the training set, ranging from 500,000 stories to 2.5 million stories, in increments of 500,000 each. The test set line that appears in these graphs is based on the results from a disjoint set of 250,000 stories. The natural distribution of the stories was not tampered with: each of the training and testing set had only ~0.6% popular stories.

For the bar graphs, the classifier trained using the set of size 2.5 million was applied to the training set itself. The bars show how many popular stories there actually were in the training set, how many were correctly identified, how many incorrect stories were incorrectly classified as popular, and how many popular stories were missed by the classifier.

1) Results using all the features mentioned:

The feature vector mentioned gave some interesting results:





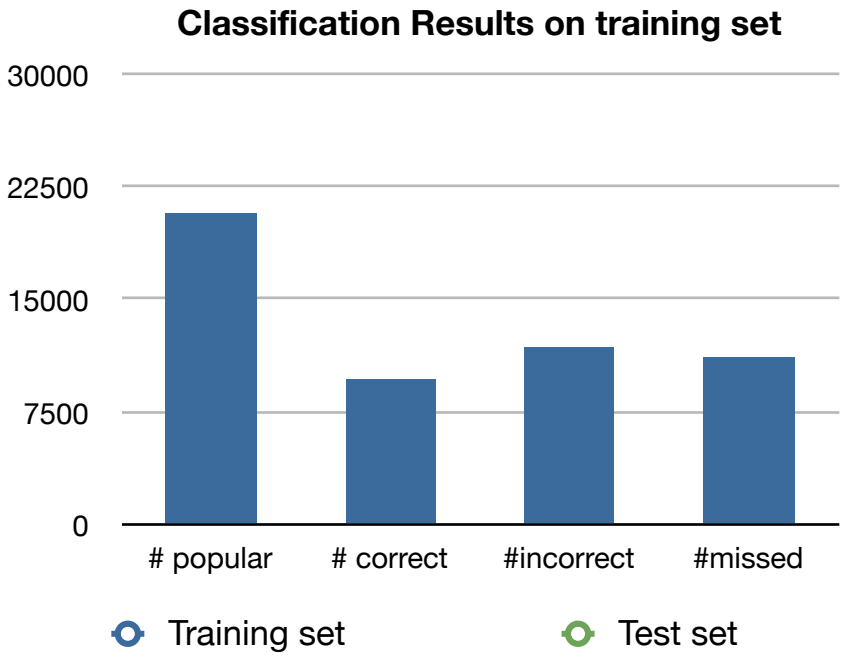
- The proportion of popular stories correctly classified as popular was high (~85%). This was an improvement over logistic regression.

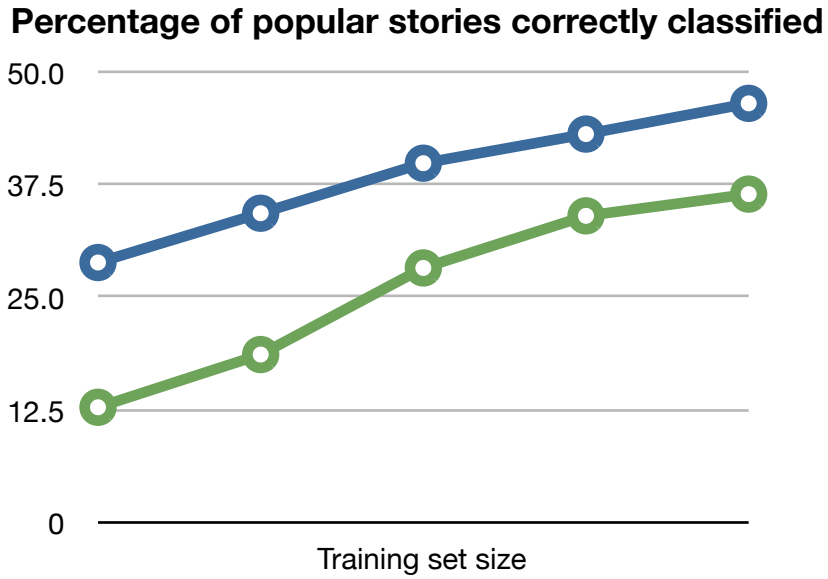
- For every story correctly classified as popular, there were about 10 unpopular stories which had been misclassified as popular. That is, if a story had been classified as popular, there was a <10% chance that it was actually popular. This was a deterioration from logistic regression

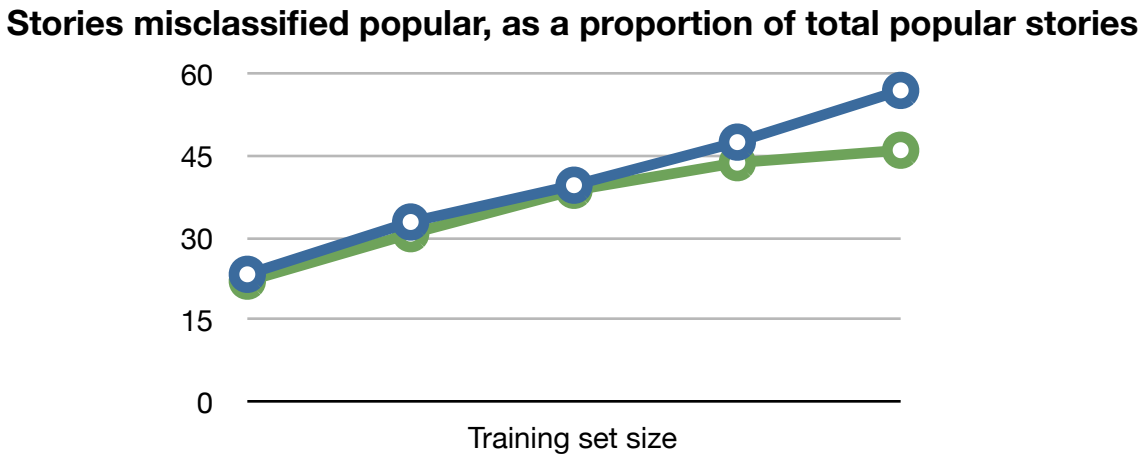
There was a large number of misclassified stories from users who only submitted stories form one link (spam users). Although these users had absolutely no success at reaching the front page, they had made friends with lots of power users. It suggested that the friend features were biasing the results. So, in the next test, I tried dropping them.

2)Results dropping friends features:

After dropping the friends features to get rid of the undesirable, spam-like websites, the results were as follows:







- The number of stories miscategorized as popular drops drastically, and the spam-like websites completely disappear.

- The proportion of correctly classified stories drops as well, from 85% to below 50%. However, if a story is classified as popular, there is now a ~50% chance that it is in fact popular. This is up from <10% in the previous attempt.

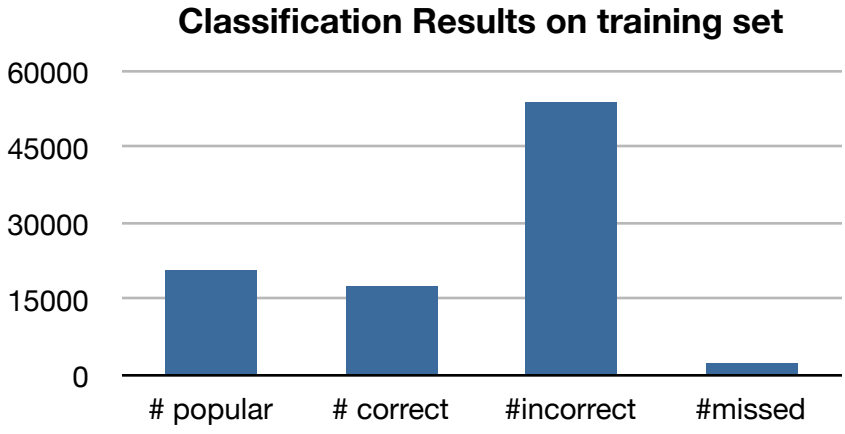
- The proportion of misclassified stories appears to increase with training set size as well, but the fact that it remains low is impressive.

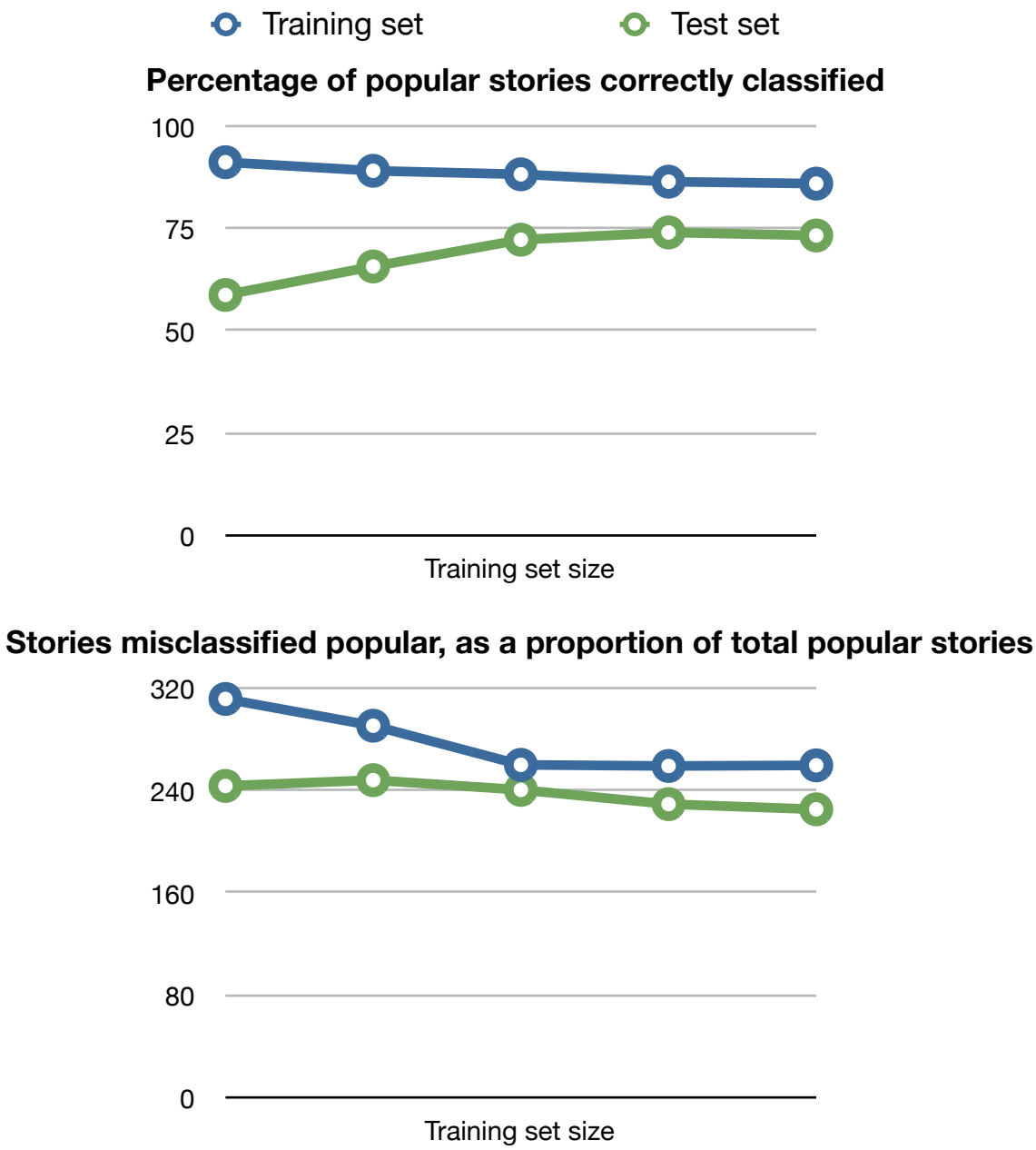
The classifier trained on 2.5 million stories yields a test for popular stories that is 50% sensitive and 99.4% specific.

This classifier seemed to be fairly discriminative in choosing popular stories. For example, one user had 200 submitted stories, out of which 23 were popular. The classifier picked 24 stories to mark as popular, with 12 of them actually being popular.

3) Dropping Friends and Words

For purposes of comparison, word features were dropped as well. The results are as follows.





- If a user has more than a certain threshold number of popular submissions, all stories from that user are classified as popular. Most such users have front page rates of 10-30%, and the classifier's correct classification rate tracks this almost exactly. Users who have only a few front page submissions had all their stories classified as unpopular, but they formed only a small proportion.

- The error on the training set increases with the size of the training set, indicating that the model has high bias. This is not surprising, since it only uses 6 features.

The classifier trained on 2.5 million stories yields a test for popular stories that is 86% sensitive and 98% specific.

Conclusion

These results barely scratch the surface of what could be done with this dataset, but there are still several interesting conclusions that can be drawn from the classification results:

- The submitting user is by far the most important factor in getting to the front page
- The words in the title and description seem to be highly relevant in distinguishing popular stories from unpopular ones (as seen from the second set of results).

There are several possible improvements as well:

- Weight features instead of dropping them entirely
- Raise the threshold to classify a story as popular, in order to decrease the misclassification rate.
- And of course, there remain many more classification methods to try.

In either case, this project was a rewarding investigation into a subject that interests me. I hope to be able to continue to improve my results.

References and Acknowledgements:

1) Digg API Documentation: http://apidoc.digg.com

2) PyDigg, a Python toolkit for the Digg API : http://neothoughts.com/pydigg

3) Mitchell, T (1997). Machine Learning, McGraw Hill.

4) Kurt Wilms, Digg Staff

5) Professor Torresani and TA Wei Pan