Hacking the Hivemind: Predicting Comment Karma on Internet Forums

Daria Lamberson (darial), Leo Martel (lmartel), Simon Zheng (szheng0)

1 Introduction

Virality on online platforms has a number of social and monetary implications. To measure virality, many websites such as Hacker News and Reddit allow users to "upvote" or "downvote" others' comments, and these votes are aggregated into a single numeric "karma" score for each comment. While these karma scores are a numeric proxy for the popularity and virality of a user's views, they are affected by a multitude of factors, and as a result, virality is notoriously difficult to predict. In this project, we apply machine learning techniques to build systems that can accurately predict comments' success, using karma as a proxy. We further analyze the content-based and metadata features that are most weighted in determining a comments' popularity in order to provide insight into the role of such features in comment virality.

2 Approach

2.1 Data Collection

We scraped 37,391 comments from Hacker News using a custom Ruby script that pulls from both the official Hacker News API [2] and the scores from a prominent unofficial API called Algolia [3] (because the former does not publish comment scores). For the Reddit data, we parsed an existing SNAP dataset of several million Reddit comments [4]. Our database of comments includes the text as well as all available metadata: author, post time, parent comment/post, and score. The score is our ground truth; posts with positive scores are labeled GOOD.

We aimed to scrape recent comments to maximize the applicability of our algorithm to comments posted today. Too-recent comments, however, do not yet have stable scores because users may still vote on them, so we scraped a block of comments from late September 2014. An alternative approach would be to scrape a random subset of all accessible comments throughout the history of the website, but we opted to favor recent trends and practical usefulness over historical fairness.

In addition to the full text and score of each comment, our script collected various metadata:

- Text of comment
- Author's username
- User the comment replies to (if any)
- Depth within comment thread
- Time of day posted
- Time elapsed since article posting
- Headline of article
- URL of article

2.2 Data Cleaning

We wrote a script in Ruby to clean the comment text and store important data. We removed HTML entities and tags, all capitalization, punctuation, and stop words (content-less words like "the"). Using this format of comment text, we were able to ignore much of the insignificant parts of the original text for content-based features (such as our unigrams for Naive Bayes) and thus improve the accuracy of our content-based predictions. We kept a copy of the raw text for the stripped-out elements for some of our metadata features, such as the number of hyperlinks.

For example, a raw comment:

```
I&#x27;ve been looking into these issues for over
a decade. Sarno&#x27;s theory is psychobabble: <a
href=\"http://en.wikipedia.org/wiki/Tension_myosi
tis_syndrome\"rel=\"nofollow\">Tension_myositis_
syndrome</a>.<p><i>&quot;The treatment protocol
for TMS includes education, writing about
emotional issues, resumption of a normal life-
style and, for some patients, support
meetings and...
```

Afterwards, the links are saved:

```
["http://en.wikipedia.org/wiki/
Tension_myositis_syndrome"]
```

And the final clean comment becomes:

```
issues decade sarnos theory psychobabble
tensionmyositissyndrome treatment
protocol tms includes education writing
emotional issues resumption normal
lifestyle patients support meetings...
```

2.3 Features

After collecting and cleaning the data, we created a number of heuristics to create roughly two dozen features that corresponded to popularity. These primarily fall into four categories:

- Raw timing features such as 'posted in the afternoon' and 'time since article posted'
- Raw relevance features such as 'popularity (score) of article' and 'depth within comment thread'
- Raw sophistication features such as 'average word length' and 'profanity count'
- Derived content features such as 'sentiment analysis score' and 'Naive Bayes classification'

See Figure 6.1 in the Results section for the complete list of features. Most are self-explanatory, and the rest we explain below:

- 'Naive Bayes Prediction' is the class predicted by our Naive Bayes classifier, trained beforehand on the same data set.
- 'Sentiment' is the emotional positivity/negativity score generated by a sentiment analysis library [7].
- 'Comparative Sentiment' is the Sentiment score normalized by comment length.
- 'Depth' is the thread depth, the size of the reply chain between the top-level article and comment.

3 Model

We considered many types of models for our task but ultimately decided to focus on binary classification and regression.

3.1 Binary Classification

In our binary classification model, we established a karma threshold to distinguish between "popular" and "unpopular" comments, where "popular" comments have received more upvotes than downvotes. The advantage of binary classification is that it maps directly to the expected use case of our system: a user asking "should I post this comment?" We tested three different binary classification algorithms:

- Naive Bayes (NB) classification using unigram features (in the bag-of-words, order-agnostic model) pulled from the text of each comment. We process comments before extracting unigrams by removing HTML, punctuation, and stop words. We used a python implementation of NB from the machine learning package scikit-learn [6].
- Linear Support Vector Machine (SVM) with a Sequential Minimal Optimization (SMO) for binary classification of "popular" vs. "unpopular" using (only) the metadata features discussed above. For this SVM with SMO, we used the Weka SVM Java package [5].
- Linear SVM with SMO with NB classification output as an extra feature. We implemented this by chaining the previous two implementations.

3.2 Regression

We also took a regression approach to the problem, in which the output of our system for each comment is a number– a prediction of the comment's score rather than a simple classification of popular/unpopular. We implemented two different regression algorithms and ran three experiments:

- n-class classification (integer score prediction) using Naive Bayes with unigram features. Each class corresponds to a popularity level (e.g. all one-upvote comments, two-upvote comments, etc. up to the highest popularity level).
- Linear Regression using unigram features.
- Linear Regression using only the metadata features.

4 Evaluation Metrics

4.1 Binary Classification Metrics

To measure our performance in binary classification, we used the following metrics:

- Precision = TP/(TP+FP)
- Recall = TP/(TP+FN)
- Accuracy = (TP+TN)/(TP+TN+FP+FN)

where TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives.

By using all three of these metrics, we can determine how accurate our system is and what types of errors each of our models is making.

4.2 Regression Evaluation Metrics

To measure our performance in regression, we used root mean squared error (RMSE), which is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m} (y_i - \hat{y}_i)^2}{m}}$$

For regression, the goal is to minimizing root squared error to increase accuracy of predicted karma scores.

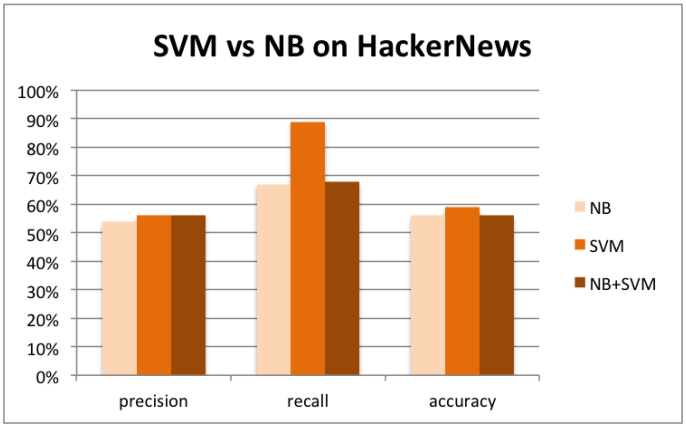
5 Results

For each approach, we trained on a randomly-chosen 70% of the data and tested using the remaining 30%. We used

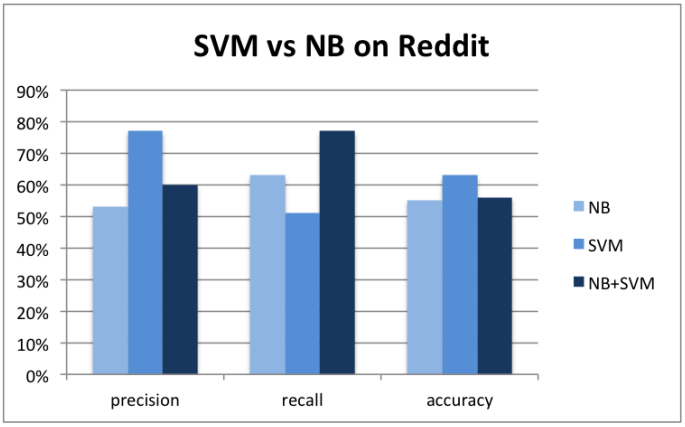
all 37,391 available comments in our data set for Hacker News classification, but used the most recent 100,000 Reddit comments rather than the entire 3 million to speed up training.

5.1 Binary Classification

When comparing our three binary classifiers across both Reddit and Hacker News, we found that the SVM with metadata features and SMO had the highest recall for predicting Hacker News comments and the highest precision on Reddit comments. Further, our SVM with Naive Bayes and metadata features and SMO had the highest recall on Reddit comments. Naive Bayes solely with unigram features never outperformed the other two in any of the categories.



For binary classification on Hacker News data, we find that we have comparable performance in precision and overall accuracy for all three models but the SVM with SMO with no unigrams far outperforms the other two in recall (89% as opposed to the NB classifier's 67% and SVM+NB's 68%).



For Reddit data, we find that the SVM model without NB output far outperforms the other two in terms of precision (77% versus 53% for NB and 60% for SVM+NB. The SVM model also has the highest overall accuracy but the difference is not as large (63% as opposed to NB's 55% and SVM+NB's 56%). However, in terms of recall, the SVM model with the NB output far outperforms the other two (77% vs 63% for NB and 51% for SVM without NB).

5.2 Regression

	Train RMSE	Test RMSE
LR (unigrams)	0.2	40.3
LR (metadata)	3.5	3.6
NB (n-class)	4.1	4.4

These results show us that we achieve the best performance on our regression when predicting using linear regression based on our metadata features mentioned above (with $RMSE = 3.6$, meaning that on average we can make predictions within 3.6 points of the true popularity).

We also see that using linear regression based on unigram features has extraordinarily high overfitting as demonstrated by having a training RMSE of 0.2 and a testing RMSE of 40.3. This is discussed in the error analysis.

6 Discussion

6.1 Feature Analysis

We extracted feature coefficients from our SVM model and found that there can be a large discrepancy in the features that are important in determining comment popularity across different platforms. In many ways, we can see that the different weighting given to different features reflects the dynamics of each of the networks.

Note that the strongest predictor in determining popularity of both the Hacker News and Reddit comments is the Naive Bayes Prediction. This makes sense since the NB output is based solely on unigram counts whereas most of the other features are based on comment metadata. By incorporating the NB output, we allow our SVM+NB model to take into account a vague sense of the comment content. The weighting indicates that it gives it relative importance, which causes our SVM+NB model to take on some of the strengths and weaknesses of our NB classifier that the SVM alone does not.

We use our feature weights to make some recommendations regarding how to structure comments on each of these platforms to maximize popularity. According to the coefficients for each feature, we recommend that on Hacker News, one could optimize karma by posting a **long, pessimistic reply to a brand-new article**. On Reddit, one should focus on producing **short, profane responses to popular, visible posts**.

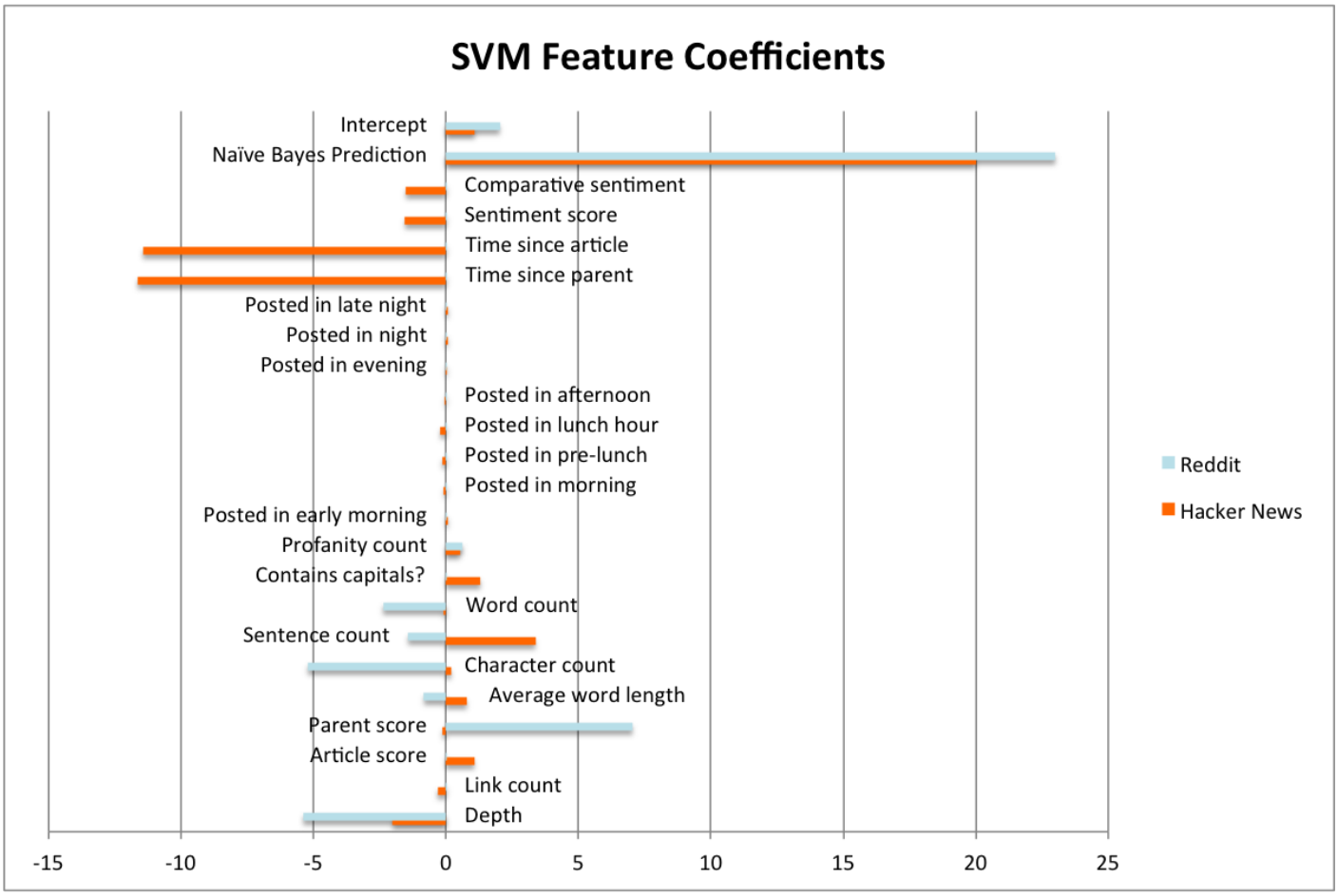


Figure 6.1: the weights that our SVM model gives to each of our features.

Similarly, examining Naive Bayes coefficients produced a plausible list of best unigrams for each corpus:

For Hacker News:

1. -4.9367 people
2. -5.4366 time
3. -5.6840 make
4. -5.6912 good
5. -5.7318 work
6. -5.8688 data
7. -5.9093 things
8. -5.9348 lot
9. -6.0286 code
10. -6.0721 years

For Reddit:

1. -4.8304 pts
2. -4.8961 coms
3. -5.0189 people
4. -5.2623 time
5. -5.3674 funny
6. -5.5697 mos
7. -5.7069 good
8. -5.7796 make
9. -5.7940 fuck
10. -5.8034 shit

6.2 Error Analysis

Our classification results are good but not spectacular. Our features, which include post timing and average word length, correlate with comment popularity, but we suspect that the most direct triggers to upvoting and downvoting behavior (such as agreement with a controversial opinion or disapproval of an off-topic tangent) would require a high level of natural language understanding and therefore be extraordinarily hard to detect with broad machine learning algorithms. Many of our misclassifications seemed to fall victim to this; in real life, a quick post time and high word count don't salvage a completely irrelevant comment.

That said, we find that our Hacker News SVM's recall of 89% based only on metadata is high enough to still be very useful. If our classifier tells you not to post something, you can be confident that your comment is bad, because such high recall implies a low false negative rate. On the other hand, the somewhat lower precision of 56% means that the algorithm will endorse some bad posts as good. Thus, our system will filter out many (but not all) of your bad posts and very few of your good ones; clearly a useful tool compared to the baseline behavior of posting every comment you compose.

Furthermore, for Reddit data, our SVM without NB prediction has relatively high precision whereas our SVM with NB prediction has relatively high recall (77% in both cases). These are both useful models to have based on whether we want to minimize false negatives (to avoid wasting good posts) or minimize false positives (to avoid the negative attention drawn by a bad post).

In general, Reddit data led to higher precision and Hacker News data led to higher recall. We hypothesize that bad Reddit comments are often more blatantly offensive, lead-

ing to fewer false positives, while the Hacker News community is less aggressive and arbitrary with downvotes, leading to fewer false negatives.

Our worst performing regression model was linear regression based on unigram features, which had high overfitting as demontrated by having a training RMSE of 0.2 and a testing RMSE of 40.3. This severe overfitting was expected, because unigram features produced an extremely high-dimensional feature space. We addressed this by running linear regression on the metafeatures instead, with much better results.

6.3 Data Collection Challenges

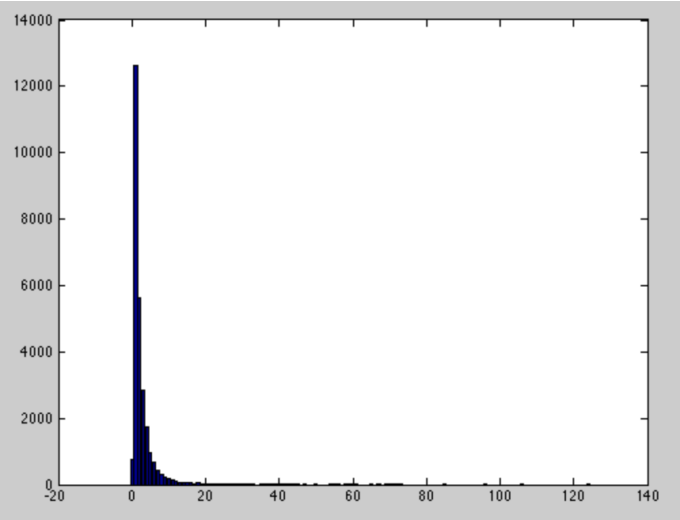


Fig: The distribution of API-accessible HN comment scores

The most significant complicating factor in data collection was that both Reddit and Hacker News (and most internet forums) hide and delete unpopular comments, so finding a large training set of unpopular comments proved difficult. As such, our data set was mostly limited to "mediocre and better" (but lacking in very inflammatory comments), which makes decisive classification harder and can lead to class imbalance. For example, while Hacker News comments can become negative, we only saw those that were only mildly disliked (karma score of 0, meaning one more downvote than upvote) or neutral (karma score of 1). To see this, we show the distribution of karma scores for our Hacker News comments below:

The following figure demonstrates that a plurality of our Hacker News cmments have scores of exactly 1, and we have relatively few (around 750) examples of truly "bad" comments (score 0). However, we posit that this data still holds value as a training set. In our experience, strongly negative comments tend to be purposefully inflammatory ("trolling," racial slurs, etc), so an earnest user of our tool is likely trying to differentiate between whether her comment will be popular or ignored, rather than popular or reviled.

6.4 Data Preprocessing Challenges

Most existing sentiment analysis libraries do not work well with forum comments because of their high concentrations of jargon, slang, and "internet-speak." Our sentiment features were moderately helpful, but many comments were given a sentiment score of 0 (neutral/unknown) despite clear signs of enthusiasm or vitriol upon manual inspection. Similarly, we tried to use a few topic modeling libraries but they were completely flummoxed by the vocabulary used on these websites.

7 Future Work

In our next steps we would hope to:

1. Expand our content-based features to try to attain a higher level of natural language understanding. In particular, we would like to measure the relevance of comments to their parents.
2. Explore more complex multi-class classifications than binary, for example: unpopular, ignored, fairly popular, and über popular.
3. Spend more time on regression and build more complex continuous-output models. Binary classification provides the core use case (whether or not to post), but finer-grained outputs would allow optimization for higher scores.
4. Explore less general – and potentially more useful – data sets, such as restricting the Reddit classifier to one "subreddit" (subforum).

References

[1] F. Saif and A. He. "Stopwords". *Proceedings of the International Conference on Language Resources and Evaluation*, 2014.

[2] "Hacker News API". https://github.com/HackerNews/API. Visited on October 14, 2014.

[3] "HN Search API". http://hn.algolia.com/api. Visited on October 14, 2014.

[4] J. Leskovec and A. Krevl. "SNAP Datasets: Stanford Large Network Dataset Collection", 2014. (http://snap.stanford.edu/data).

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten. *The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1*, 2009.

[6] M. Pedregosa, et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research 12*, 2011.

[7] "Sentiment". https://github.com/thisandagain/sentiment