INTERNET ARTICLE COMMENT CLASSIFIER

Matt Jones, Eric Ma, Prasanna Vasudevan
Stanford CS 229 – Professor Andrew Ng
December 2008

1 INTRODUCTION

1.1 BACKGROUND

Part of the Web 2.0 revolution of the Internet in the past few years has been the explosion of user comments on articles, blogs, media, and other uploaded content on various websites (e.g. Slashdot, Digg). Many of these comments are positive, facilitating discussion and adding humor to the webpage; however, there are also a multitude of comments – including spam/advertisements, blatantly offensive posts, trolls, and boring posts – that detract from the ease of reading of a page and contribute nothing to the discussion around it.

To alleviate this issue, websites like Slashdot have implemented a comment-rating system where users can not only post comments, but rate other users' comments. This way, a user can look at a comment's rating and quality modifier (funny, insightful, etc.) and immediately guess whether it will be worth reading. The site can even filter out comments below a threshold so the user never has to see them (as Slashdot does).

1.2 GOAL

Despite the power of crowdsourcing, ideally a website should be able to "know" how interesting or useless a comment is as soon as it is posted, so it can be brought to users' attention (via placement at the top of the comments section) if it is interesting or it can be hidden otherwise.

So, our goal was to design a machine learning algorithm that trains itself on comments from multiple articles on Slashdot and then, given a sample test comment, predicts what the average score and modifier of that comment would have ended up being if rated by other users.

Specifically, we wanted to determine what makes online article comments a unique text classification problem and which features best capture their essence.

2 PRE-PROCESSING AND SETUP

2.1 SCRAPING

We acquired the data by crawling Slashdot daily index pages from June to present, noting each day's article URLs, and then downloading an AJAX-free version of each article page with all its comments displayed. We parsed each comment page with a combination of DOM manipulation and regular expression matching to extract the user, subject, score, modifier (if any), and actual body text. These data were stored into a MySQL database, and meta-features were calculated later for each comment, and then stored back in MySQL along with every other comment attribute. The score and modifier distributions of the comments are illustrated in Figure 1 and Figure 2, respectively. Figure 2 depicts the proportion of each modifier type as well as the average score.

2.2 FORMATTING OF INPUT DATA

We took the following steps to convert this raw data to a form suitable for machine learning algorithms:

a) Case-folded and removed "stop words" - these include 'a', 'the', and other words that have little relation to the meaning of a comment.
b) Applied Porter's stemmer - this algorithm converted each word to its stem, or root, reducing the feature space and collapsing words with similar semantic meaning into one term. For example, 'presidents' and 'presidency' would both be converted to 'presiden'.
c) Counted word frequencies - every stemmed word that existed in any of the comments in our data set was given an individual word ID. Then, we counted the frequencies of words in each comment.

The result of these steps was, effectively, a matrix of data with dimensions (number of comments) x (number of possible words), where the value of entry (i,j) was the number of occurrences of word j in comment i.

2.3 Formatting of Output Variables

The 2 output variables we had to work with were:

1) **Score** - possible values are integers -1 to 5 *[7 classes]*
2) **Modifier** - possible values are <none>, 'Insightful', 'Interesting', 'Informative', 'Funny', 'Redundant', 'Off-Topic', 'Troll', and 'Flamebait' *[9 classes]*

In addition to these, we created 4 artificial output variables which we thought would be useful dependent variables for our algorithms to predict:

3) 0 (Bad) for scores -1/0/1, 1 (Good) for scores 2/3/4/5 *[2 classes]*
4) 0 (Bad) for scores -1/0/1/2, 1 (Good) for scores 3/4/5 *[2 classes]*
5) 0 for negative modifiers ('Redundant', 'Off-Topic', 'Troll', and 'Flamebait'),
   1 for positive modifiers ('Insightful', 'Interesting', 'Informative', 'Funny'),
   2 for no modifier *[3 classes]*
6) 0 for negative or no modifier, 1 for positive modifier *[2 classes]*

Thus, we had 6 output classification types into which we wanted to classify comments.

3 Meta-Feature Selection

3.1 Thoughts About Comments

We made the following observations, based on our past experience, about the nature of different types of comments:

- Many spam messages are majority- or all-capital letters.
- Non-spam messages that are majority- or all-capital letters tend to be annoying.
- Long comments are probably better thought out than very short ones, and are more likely to contain insightful comments.
- Users that have more experience posting comments on a moderated environment such as Slashdot's are less likely to purposefully post irritating comments.
- Many spam messages contain URLs.
- Very few spam messages are very long. For the few that *are* long, it is usually because there are many paragraphs with one sentence per paragraph.

3.2 Meta-Features

Word frequencies alone are not enough to capture the above patterns. So, to better describe each comment, we added the following 5 meta-features to the word frequencies to form our new feature set:

- Percent of characters that are upper case
- Number of total characters
- Number of paragraphs
- Number of HTML tags
- Number of comments previously made by the commenter

Analyzing these meta-features and their effectiveness for classification were our main points of interest in this research. We wanted to determine what meta-features would be best at capturing the essence of online comments.

4 Algorithms

4.1 Naïve Bayes

We used our own Java implementation of Naïve Bayes (adapted from the library used in CS276). We experimented with different training sets, including just comments from the one article with the most comments (m = number of training samples = 2221), comments from the top 10 most-commented articles (m = 16780), and comments from the top 500 most-commented articles (m=329925). It ended up being infeasible given our implementation and resources to run on comments from the top 500 articles, and we got better (and more useful) results by training on the top 10 articles' comments. This is naturally a more useful training set than just the 1-article set. Since the end application of this classifier would be "given this comment, tell me if it's good or bad," having a classifier that only works for a given article wouldn't be very useful. Thus a classifier that's general enough to do well for comments from 10 articles should fulfill this end purpose more effectively that a classifier that only works on comments from a given article.

While we initially tested on our entire training set, once we had chosen features we switched to 10-fold cross-validation for more realistic accuracy. The partitioning into 10 blocks was done randomly (but in the same order across all runs). All numbers reported are the mean of 10-fold cross-validated accuracies.

4.2 OTHER

We adapted the SMO implementation for the SVM algorithm discussed in an earlier assignment to our data set. We intended to compare these results with those from Naïve Bayes, however the large sample size and high dimensionality of feature space made this algorithm too slow to return useable results.

We also implemented the Rocchio Classification Algorithm to test whether the centroid and variance of each class comprised a good model for that class. However, this algorithm produced extremely inaccurate results, and often predicted classes more poorly than random guessing. This indicated that our training examples were not oriented in the spherical clusters assumed by Rocchio Classification. We did not include our results from these tests in this paper and chose to focus on the Naïve Bayes data.

5 RESULTS

Our prediction accuracies from the 10-Fold Cross Validation tests using Naïve Bayes are reported in Figure 3. The results have been grouped by the 6 classification types described above, and illustrate the different accuracies achieved by using just word frequencies, just meta-features, and a combination of both word and meta-features.

For every classification type, we noticed a marked improvement when applying just the meta-features compared to using only word or a combination of features. The combined features led to inconsistent results, as they both improved and worsened accuracy depending on the classification type. Our graph clearly illustrates the improved accuracy achieved when trying to predict between fewer classes. When predicting a comment's modifier, all three of our feature subsets did worse than the 11% accuracy achievable by random guessing. For classifying raw score, all three of our tests beat random guessing accuracy of 14%, with meta-features leading at 29%, but were still much less accurate than the other types with 2 or 3 classes.

We also wanted to determine the isolated effects of the meta-features, and ran a series of tests with Naïve Bayes with each individual feature. Our results are displayed in Figure 4 – the graph represents the change in accuracy achieved by each feature for the various classification types. Rather than finding the difference from absolute randomness, we realized that many of the classification types had one class with an overwhelming majority of comments and calculated the difference from this class's percentage. When classifying by Good, Bad, and None modifiers, for example, 73% of the comments had no modifier. Thus, simply labeling all comments as None would have achieved 73% accuracy and simply measuring absolute accuracy would have reported a very deceptive, high value.

The graph shows that the number of comments made by the user had the biggest impact out of all the meta-features, but only helped when classifying raw score and good vs. bad with a score threshold of 2. When using just meta-features or a combination, the only improvements appeared for "Bad = {-1 0 1}". This classification had two classes of roughly even size, so our high accuracy numbers would be unachievable by simply classifying everything

as the same (see below table for the confusion matrix). The results highlight what types of classification our methods were effective in detecting.

CONFUSION MATRIX FOR GOOD VS. BAD = {-1 0 1}, USING NAÏVE BAYES / ONLY META-FEATURES

	ACTUALLY BAD	ACTUALLY GOOD
CLASSIFIED AS BAD = {-1 0 1}	5549	1586
CLASSIFIED AS GOOD	2208	7437

6 CONCLUSIONS

The results show that our chosen meta-features are extremely powerful, as they consistently performed better by themselves than word frequencies. This was a surprising discovery because we had originally expected the meta-features to be a useful supplement to word frequencies, but not be able to classify so accurately alone. This has led us to believe that so-called "good" comments across articles may not share that many words. Because each article discusses a different topic, it makes sense that generalizations about shared words across all comments are hard to make. Our meta-features probably performed so well because they capture fundamental differences between commenting styles. As discussed above, we believe the unique nature of comments as an un-edited forum for publishing text leads to very useful, though specific features.

Out of our tested meta-features, we have also concluded that the number of previous comments made by the user is the best indicator of comment quality. This could be attributed to the fact that returning users who consistently participate in discussions have the intention of constructively contributing. Users who infrequently use Slashdot have less incentive to post interesting comments because they won't care how their user rating is affected in the future. These users would be more likely to leave offensive or purposefully instigating remarks.

What surprised us were the results of combining word and meta-features. We had expected this to outperform both of the individual feature groups. We believe this was a result of how we combined the feature groups in our Naïve Bayes implementation. Our tests simply placed each meta-feature value alongside the word frequencies without accounting for weighting or normalization to make them more comparable. This would be useful to explore in future research for developing a more effective algorithm to combine meta and word features.

In applying these meta-features, we believe that the best method for classification is by labeling comments with scores of 2 and above as "Good" and those with scores less than 2 as "Bad." Intuitively, such a division makes sense as lower scores probably have more in common with each other as do higher scores. We were initially uncertain whether a threshold of 2 or 3 would be best, but the results indicate that comments with score 2 have more in common with higher scores than lower ones. We can apply this conclusion to Slashdot's automatic comment filter and recommend hiding scores of less than 2, but displaying everything else.

Our relative ineffectiveness at making specific score or modifier predictions emphasizes the importance of a comment's intrinsic meaning. Simple metrics such as word frequency or number of previous posts failed to fully capture this fact, and revealed that future research will have to attempt more sophisticated language processing.

7 FUTURE RESEARCH

The problem of classifying Internet article comments is ripe for further research beyond what we have investigated here. There are many possible avenues we can explore:

- *Don't make Naive Bayes assumption of word independence:* our Naive Bayes' assumption that word probabilities are independent is not completely true. To better capture the *meaning* of comments, we can look at joint frequencies of noun-adjective pairs; ignore nouns and look at only adjectives to extract mood;

split up nouns and adjectives into different categories and then look at pairings; or try another "smart" approach that tries to understand the meaning of a comment.

- *Investigate other meta-features*, such as a user's history (e.g. the average score of that user's previous comments, his usual type of response, his ratings of other users). We can also use different weights for different features.
- *Take into account threads of comments:* currently we are treating all comments as distinct, but some are responses to others.
- *Test additional algorithms:* algorithms besides Naive Bayes were not useful because of the extremely high dimensional feature space. We can look into reducing the dimensionality of the data or attempting to use only meta-features with other algorithms.
- *Analyze other metrics besides accuracy*: we can look at the entire confusion matrix for our results, instead of only focusing on the overall error across classes, and use this as a metric for how well our algorithms perform.
- *Test other sites:* many other sites besides Slashdot have comments below articles, and it would be interesting to see how our methods perform on those sites, especially the ones with different comment rating systems.

These steps may help us more accurately predict the score and genre of a new comment, and at the same time reveal some interesting patterns about crowdsourcing and comment-posting on the Internet.

FIGURES

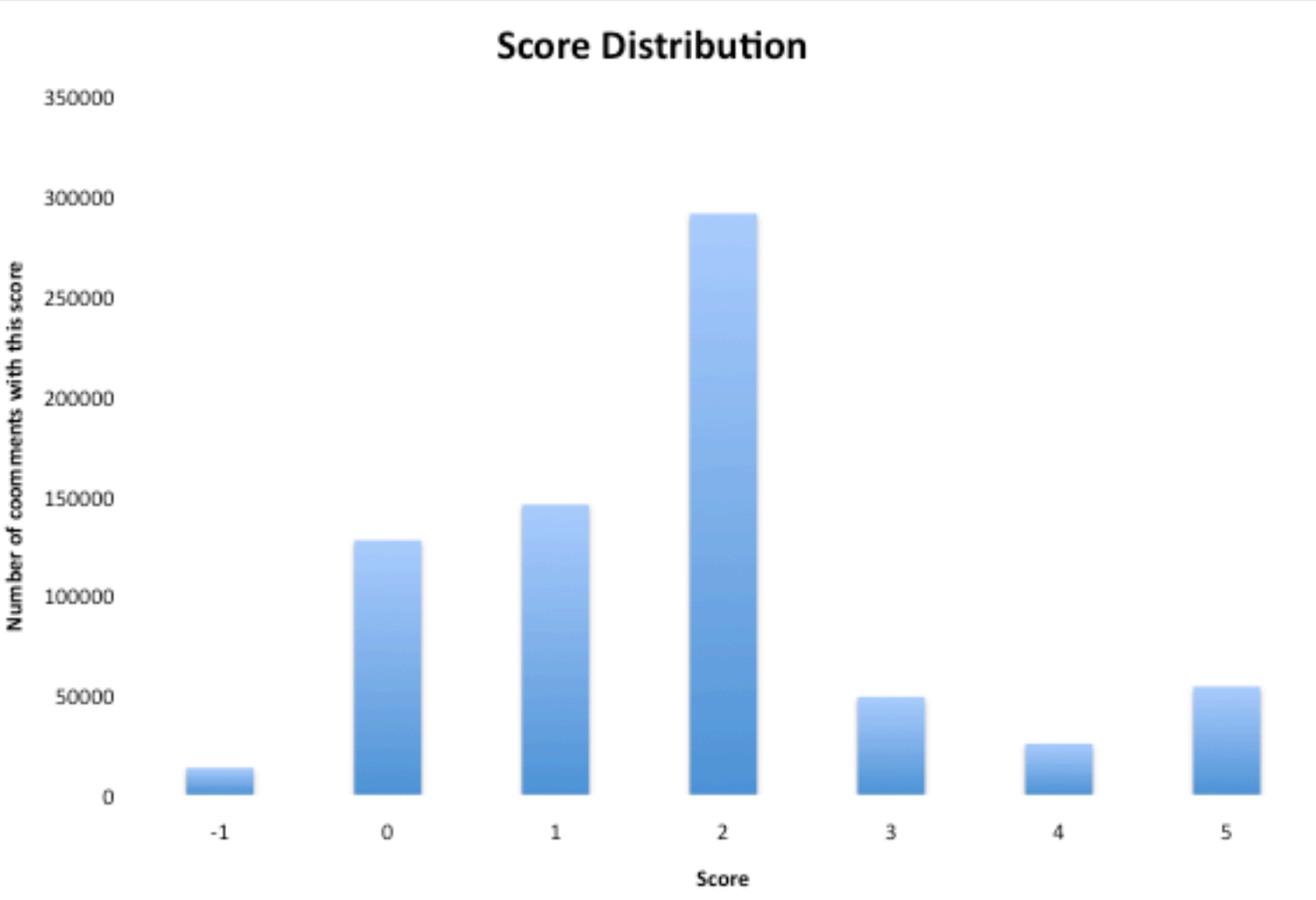

Figure 1

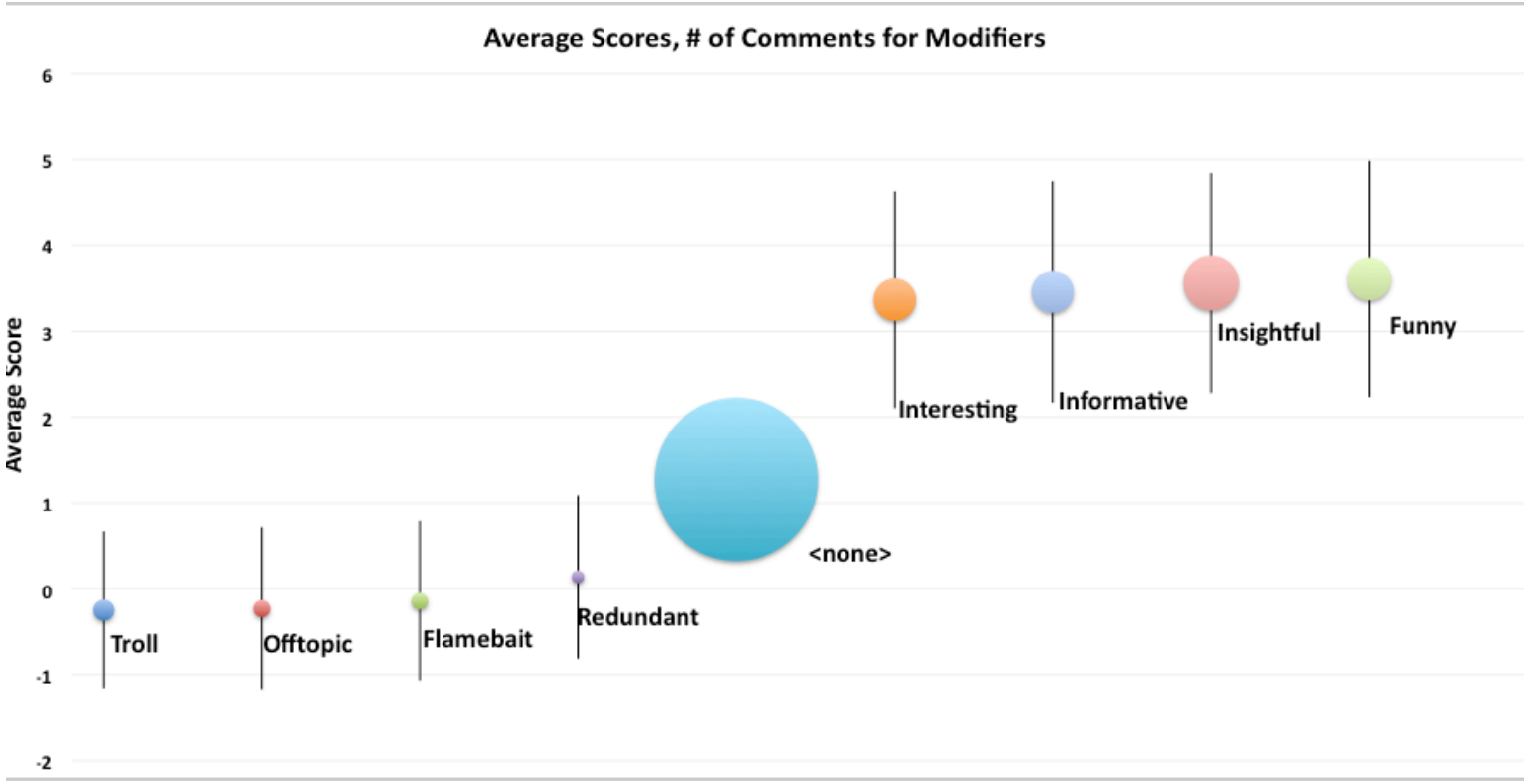

Figure 2

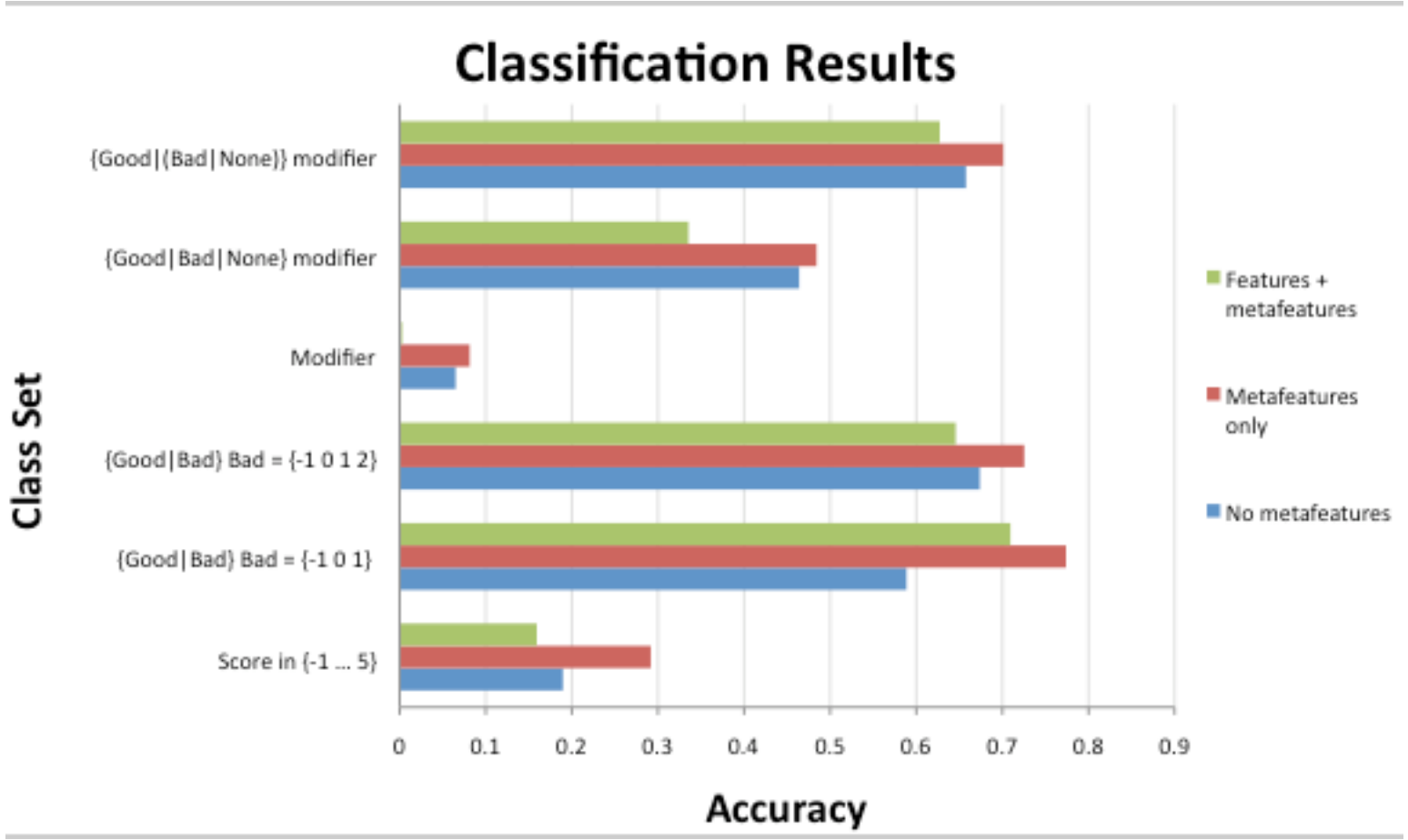

Figure 3

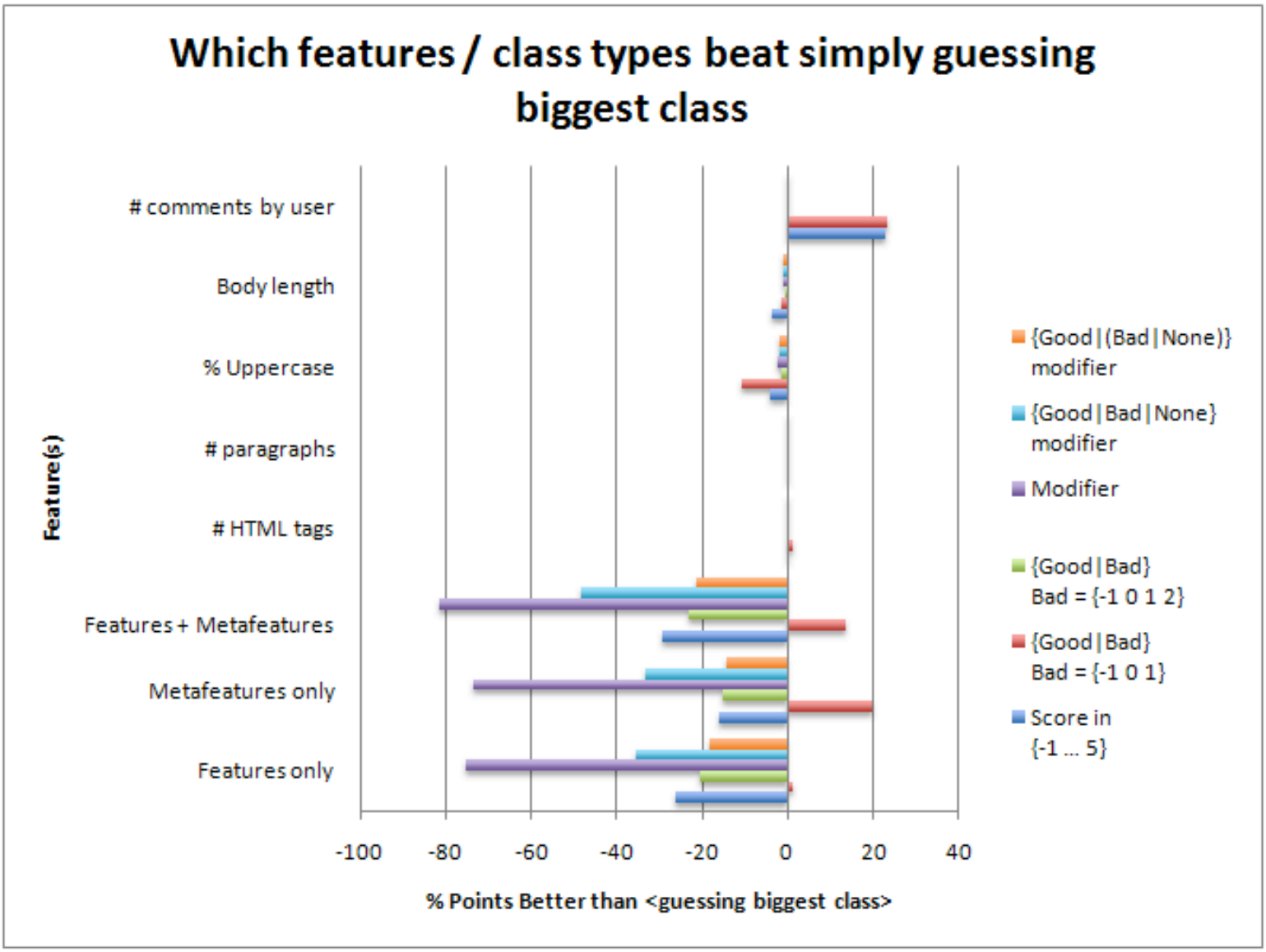
Which features / class types beat simply guessing biggest class
comments by user
Body length
% Uppercase
# paragraphs
# HTML tags
Features + Metafeatures
Metafeatures only
Features only
Feature(s)
-100
-80
-60
-40
-20
0
20
40
% Points Better than <guessing biggest class>
{Good|(Bad|None)} modifier
{Good|Bad|None} modifier
Modifier
{Good|Bad} Bad = {-1 0 1 2}
{Good|Bad} Bad = {-1 0 1}
Score in {-1 ... 5}

Figure 4