Classifying Controversial Wikipedia Articles

Ephraim Kunz, Luke Dickinson, Alexandra Hurst, Peter Strein
Department of Computer Science
Brigham Young University

Abstract

(TODO: finish abstract. I think it will be easiest to finish the abstract once we have finished the paper) Your abstract should concisely answer the following three questions: 1) what problem are you addressing? 2) what approach are you taking to solve the problem? 3) what are your results?

1 Introduction

In human society, we have topics that are more sensitive than others. Some topics create an emotional response. Others create an intellectual response. Often people have differing views, and express these views through means of violence. These topics are considered controversial.

Wikipedia is a source of information publicly available to anyone. Anyone can view and even edit this information. Because anyone can change information, edit wars between opposing viewpoints are common for a controversial article. Marking an article as controversial allows readers to be more cautious and moderators to know which articles should be regulated.

We wanted to know if Wikipedia articles could be classified as controversial or non-controversial with machine learning. It is usually easy for a human to detect an article that is a controversial topic, because we are human, have human emotions, and know from past history what topics tend to be controversial. However as Wikipedia has 5,607,695 english articles [] and many more articles in other languages, it becomes impossible to read through all articles to classify them as controversial or not. Thus, a machine learning model that can classify Wikipedia articles could be generalized to work with many different kinds of articles.

2 Methodology

2.1 Data Sources

We began the project by collecting a large corpus of raw, labelled data. Wikipedia provides a web API to access their pages, as well as various metadata about a page (wikipedia api reference), such as image urls, links to other pages, inbound links to a page, and categories to which a page belongs.

We first scraped the 1300 articles listed on the Wikipedia page List of controversial issues (reference), labelling them as controversial. These are articles identified by Wikipedias editors as controversial due to edit warring or lack of a neutral point of view. We then scraped another 1300 articles at random, using a random article endpoint provided by Wikipedia. We labelled all of these articles non-controversial. We wanted a even split between the number controversial and non-controversial articles in order to avoid overfitting.

For this raw data scrape, we collected every raw feature available from the Wikipedia API so we could extract meaningful features later. We collected the entire text of each article, the timestamps of every edit ever made to the article and several lists of links (inbound from other Wikipedia pages, outbound to other Wikipedia pages, image urls, and all urls referenced by the article). The sheer amount of data collected made scraping difficult. We wrote a scraper that used rate-limiting and retries to avoid overwhelming Wikipedia and periodically dumped scraped data to files to checkpoint in case of failures. Total, we collected 668 MB of textual data from this first scrape.

TODO:Should this paragraph be near the end of the paper?

One of the limitations of our problem was the size of our collected data set. Because Wikipedia has only classified 1300 of its articles as controversial, that was the maximum number of controversial articles we could use to train on. There are over 5 million Wikipedia articles, but to make our training data unbiased towards controversial articles, we decided to keep the number of non-controversial articles around the same number of controversial articles.

2.2 Data Sets

We ended up creating three different datasets. The first two datasets use the features of Wikipedia as features and included the following: percentage of links that were .net, .org, .com, .gov, and other; number of edits; number of words in an article; number of links in an article; number of references; and number of positive/negative/controversial connotation words. Each instance included these features and an output class of whether it was controversial or not. An actual instance from this dataset is below.

Add Table

We created these features by parsing our raw, scraped data. We created a framework to allow the easy extraction of features from multiple checkpoint files of raw data. The framework allowed a user to supply a map function that would be applied over each piece of raw data, writing the results to a standard CSV format. Some features were more difficult to extract than others, such as avg-revision-per-day. The input to this feature extractor was just a long list of timestamps.

We also created a few features to try and capture word bias. These features were num-positive-words, num-negative-words, and num-controversial-words. To obtain these features, we searched for lists of words that had a positive connotation, a negative connotation, or that were controversial. Then, we read the body of the Wikipedia article, and simply made counts of how many times words from those respective lists were shown. For example, in the instance above, out of the 62,671 words, 149 were counted to be words with positive connotation based on the dictionary of words we found online, 73 were considered to be negative, and 1 word was considered to be controversial.

For our second dataset, we a total of 2750 articles, 1252 controversial and 1498 non-controversial articles. We collected the same features as used in our first dataset. In order to normalize our dataset for article length, we first created 10 bins splitting the controversial articles by article length. We then collected random articles through the Wikipedia API to fill matching bins to be used as our non-controversial data set. Because the only way we could collect articles were through the random article API, we modified our collection software to search for articles that fit into the bins previously mentioned. While the smaller article length bins were filled fairly quickly, the largest few bins took over 24 hours of scanning to fill.

For our third dataset, we decided to do a bag-of-words approach. Using the same data collected from the second dataset, we made a new feature set only analyzing the text of the article. Each article was originally loaded into python as one row in a matrix, which was then converted to a numeric array where each column represented the word count of a specific word in each different article. We also used the function TfidfTransform in our sklearn pipeline to make sure common words such as and, the, and or didnt carry too much weight. From there, our newly-formatted numeric data was ready for use with pythons sklearn algorithms.

2.3 Models

We trained two models for our final tests: a random forest and a Support Vector Machine.

We used WEKAs implementation of random forest to get good results in a tree-based approach to prediction. Trees are beneficial because they make it easier to understand which features are most important and how they are used. In other words, they give us insight into the generalization process of the model. Random forest is an ensemble method. A configurable number of trees are constructed. Each tree is given a random subset of the training set, and it picks a random feature from that subset for each split in the tree. Once all the trees are constructed, the test set is fed to each of them. For each instance in the test set, the vote of all the random trees is taken to determine the predicted output value.

There are many parameters to tune that can be used to improve the performance of random forest for a given dataset. Our initial accuracy with the default settings was 75.94%, up significantly from our previous best accuracy of 70.44% with a single tree (no ensemble). After significant trial and error, we were able to increase our accuracy to 77.38% with the random forest. This final model had 800 trees in the forest, broke ties randomly when several attributes looked equally good, and used a bag size of 100% of the training set. With eight cores being utilized, 10-fold cross validation took 16.2 seconds.

Support Vector Machines are classifiers suited well to high dimensional or nonlinear data. We used this method with our bag-of-words data. When we originally tried text classification using Naive Bayes classification, our accuracy was only around 65%, so SVMs offered a similar classification technique that might work better with what appeared to be nonlinear data. Our results were promising. Average classification accuracy using an 80-20 split yielded 90 accuracy on average. A histogram of accuracies over 20 iterations is shown below.

insert graph

To insure the legitimacy of our results, we created five standard test-train sets in our data to use both algorithms on and compare accuracy. The results are **fill in results**. Additionally, we experimented with Latent Dirichlet Allocation and Non-Negative Matrix Factorization, algorithms that find key words or ideas in articles, to try to highlight phrases that may lead to an algorithm being classified as controversial. While most of these efforts mostly produced a list of of, thus, and in, several controversial words like sex or rock showed up multiple times. Dates also seemed to show up more in controversial articles. Though the LDA and NMF efforts didnt produce any strong results, they highlighted some of the efforts of Support Vector Machines in trying to figure out what text elements lead to a controversial document.

3 Results

(TODO: Luke talk about PCA)

When we first started, we only worked with the first dataset. We used Weka to to measure the accuracy of our results. We used the J48 decision tree and the Multi-Layered Perceptron (MLP) with this dataset. We first ran it with all the features. With J48 we achieved an accuracy of 93.72% and with MLP we achieved an accuracy of 93.26%. We believed that this was because we were including the number of revisions, the major factor wikipedia uses to determine if something is controversial. So, we decided to remove the number of revisions, hoping to get a more reasonable answer. However, we still obtained surprisingly high results. With the J48 Decision tree we got 92.38% accuracy using 10-fold cross validation. With the MLP, we obtained an accuracy of 93.26% using 10-fold cross validation. These results seemed too high, especially as this was the first iteration. We decided to investigate further as to why we had such high accuracies.

As we analyzed the J48 decision tree and the data, we realized that our data was skewed. Because most articles on Wikipedia are very short, our random sample to collect non-controversial articles was mostly composed of short articles. This made it trivial for the model to detect controversial articles: just pick the long ones. Because we wanted to use inherent features of the article itself to determine controversiality (not just length), we decided to rescrape our non-controversial articles in order to get a length distribution matching the controversial articles.

Insert Graph

(TODO: I think the numbers on this graph are going to be too small, because it will need to be made smaller at some point. If so, should we remake it or call it good enough? Also, should I change the title?)

To do this, we still used the random article endpoint. This time, however, we created 14 bins based on article length. This allowed us to determine a maximum number of articles to collect for each length category. Scraping took much longer this time, as we had to try many times to randomly receive a longer article. Each longer article took longer to download and fetch metadata for, because it had more of everything. On average, it took [time] for this scrape to complete.

After collecting the data into bins, our datas distribution looked like this.

Insert Graph

(TODO: I know that this is too small, but I dont know how to make it bigger. Do we need the numbers? Should we try to make it bigger by hand or something?)

Using that data, we re-ran the J48 decision tree and the MLP. When we included the number edits, our accuracies were 75.96% and 76.04% respectively. When we excluded the number of revisions, our accuracies dropped to 70.44% and 70.25% respectively. We concluded that our word bias was removed and we then decided to proceed to see if we could boost our accuracy.

We tried a variety of methods, including using sklearn and multiple models. The model we achieved the highest accuracy was Random Forest using Weka. We achieved an accuracy of 76.73% using 10-fold cross validation. We tried improving some of the features, including doing better counts of positive, negative, and controversial words by using better filtering of punctuation and using regex. After improving the word counts, the accuracy using 10-fold cross validation using Wekas Random Forest was 77.38%. This was the highest we ever achieved using the 2nd dataset.

We decided that it would be best to begin analyzing the text of the articles instead of just the wikipedia features. So, we created another dataset to analyze the data using the bag of words approach. We first used Naive Bayes which achieved and accuracy of 64%. We then decided to use Support Vector Machine, and achieved an accuracy of 90%. This was the best results we ever achieved. (TODO: Alex, we will need more explanation of the results of SciKit and SVM here)

4 Conclusion

(TODO: I tried writing out a lot of this section, but it is poorly written. I am keeping the list of ideas we had at the bottom to make sure we keep everything. Feel free to change the order and wording of everything on here so that it makes more sense.)

In the case of our problem, the random forest and the Support Vector Machine (SVM) gave the best results, with the random forest getting an accuracy of 76.73% and the SVM an accuracy of 90%. (TODO: make it it is 90 percent and not something else). Compared to all the other models we used, those did the best. With random forest giving the best score, this shows that ensembles give the best results. However, they can only improve models so much. Thus, in the case of our problem, by thinking of the problem differently and using an SVM we got a much better answer. This is because we are analyzing the actual text of the articles instead of other features that appear to not give great results. (TODO: this paragraph seems meh, we should change it or do something else with it)

We have also discovered that there is no one feature that was indicative of controversy. It was a combination of higher order features that enabled us to get the best results. If we had more time, we would examine what articles are being misclassified, and why they are being misclassified. Were there any features in common? Were there similarities between which articles our random forest got wrong on the SVM got wrong? These are all questions we would like to explore.

A problem we ran into was the small dataset. Because Wikipedia already classified 1300 articles as classified, we could use those articles as a base. However, we could not expand our dataset to include more than 2600 articles because we did not want our program to be heavily biased towards non-controversial articles. We would like to explore bootstrapping our data so that we could get more data to train our models. We could potentially use our model to run against all Wikipedia articles. The articles that are classified as controversial we could have a human label as controversial or not, and then use that to append to the dataset. That way we could label articles as controversial but use our program to make a subset of all the articles that a human would have to look at to find the controversial articles.

Another realization is that we made is that our problem is essentially an outlier problem. There are over 5 million Wikipedia articles on Wikipedia, but only 1300 articles are classified as controversial. This means that finding controversial articles is the same as trying to find a few outliers in a giant dataset. If we had more time, we would analyze more the ways to detect outliers. We believe that there is a fundamental difference between controversial articles and non-controversial articles, but to find that difference requires us to find the subtle differences between the mainstream articles and the outlier controversial articles. We would look at different methods, like SciKit, that have methods of finding outliers.

We would also further analyze different output classes. We believe that there are more than just 2 clusters (controversial or non-controversial). There may be articles that are not controversial, are kind of controversial, that are very controversial, and that are completely controversial. There may also be other clusters that we are not aware of. We would like to know which clusters exist and if we can use those clusters to find controversial articles.

Ensembles work very well - random forest works very well SVM was very successful because it did text analysis No one feature was indicitive of controversy, it was higher order features Its hard to work more with this problem because we dont have any more labeled data, and we cant really make more How do you tell a controversial article from a long lived article? Controversial articles are outliers - many more articles out there by random than there are by controversial articles Further time: Combining the SVM data and the random forest by augmenting the dataset with the SVM accuracy What did misclassified articles have in common? Can we use clustering to find interesting results? Outlier detection - are there better ways - could we use all 5 million Wikipedia articles somehow and then use the controversial articles as outliers Maybe controversial/not controversial is the wrong split - maybe have probability or not controversial, kind of controversial, very controversial, compeletely controversial - relates to clustering Can we somehow bootstrap to do more? There is a chance that the articles that we chose as non-controversial are actually controversial - we need to validate our assumption - they may be controversial