

Predicting Edit Wars on Wikipedia

Daniel Cohen

DS 4400

Fall 2017

Introduction

The goal of this project is to predict Wikipedia *edit wars*. More specifically, given the content of a Wikipedia article, we would like to predict whether or not that article will be the subject of an edit war.

An edit war is a situation where different Wikipedia editors have conflicting views on some topic. These editors want a Wikipedia page to reflect their view, and since their views conflict, they end up reverting each other's changes. One example of such an edit war was in the article about aluminum. Some editors wanted the title to use the American spelling, "aluminum," while others wanted the title to use the British spelling, "aluminium" (with an extra *i*). This caused a fairly large edit war, with the British side eventually winning out.

Data Collection

The data used was a random sample of 96,251 articles scraped from Wikipedia. This was obtained using the MediaWiki API, which has a built-in function for random sampling of articles without repeats. For each article, the following information was collected:

- Content of the article (i.e., the actual words that a reader sees)
- Word count
- Number of references (sources) per word

Word count was included in the data because I hypothesized that since longer articles have more content, the likelihood of something inflammatory appearing would be higher for longer articles. References per word was included because I hypothesized that more controversial articles would cite more sources to back up their claims.

Data Processing

To extract descriptive features from an article's contents, the first step was to generate a vocabulary. I performed the following steps to do so:

1. Convert the contents of each word to a bag-of-words in which word order is ignored.
2. Exclude stop-words.
3. Lemmatize all words (this means converting each word to its root form, e.g. *cacti* becomes *cactus*, *walking* becomes *walk*)
4. Exclude all words that appeared in fewer than 10 articles.

These steps were done programmatically with the help of Python's Natural Language Toolkit library. After taking these steps, the total vocabulary size for the data set was 59,637 words. For each word in the vocabulary, a column was added to the data set representing the percent frequency of that word in a given article. This means that, in addition to the two features mentioned above (word count and references per word), 59,637 features were added to the data set – one for each term in the vocabulary.

The next step was to find a way to programmatically determine whether an article constituted an "edit war" or not. For this purpose, I implemented a technique described in a 2014 paper, *The most controversial topics in Wikipedia: A multilingual and geographical analysis* [1].

The method described in the paper was to calculate a *controversy score* for each article based on its revision history as follows:

1. Determine which edits to an article are reversions to a previous version of the article. This is done by comparing the hash of an edit to previous edits. If two edits have the same hash, then the later edit is considered a reversion.
2. Define a *mutually reverting pair* to be a pair of editors, each of whom has reverted an edit from the other.
3. Weight each editor by the amount of edits they have made to the given article.

4. Define the controversy score as the sum of the weights of all the mutually reverting pairs (excluding the top weighted pair, in order to prevent a personal fight between two editors from skewing the results).

Creating a model

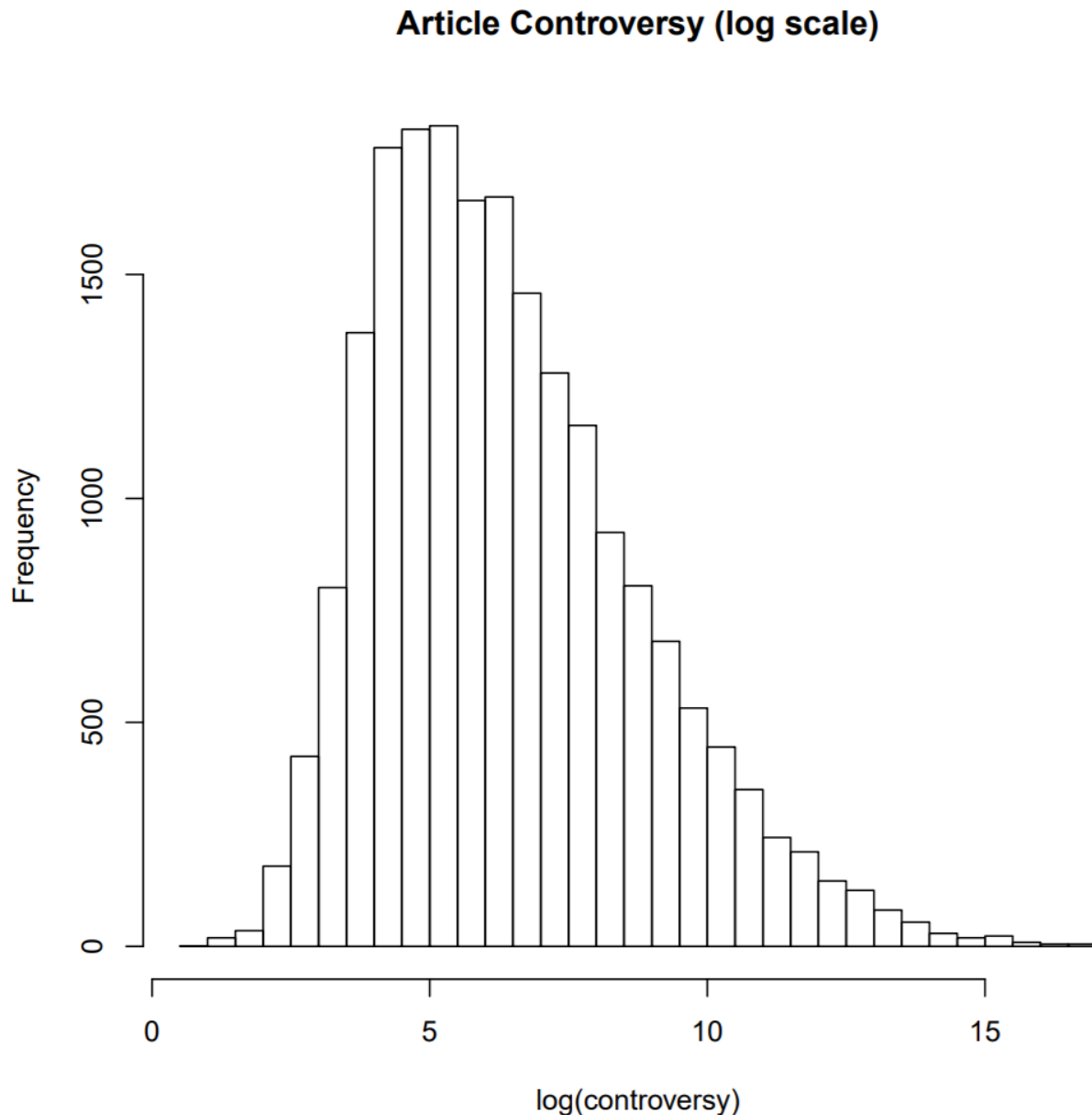
The model used for this project was a Naïve Bayes model. This was chosen because the data set had nearly 60,000 features, and Naïve Bayes' time complexity scales linearly with number of features, making Naïve Bayes the only model that I would be able to train in a feasible amount of time.

Naïve Bayes requires the target feature to be categorical, however the controversy scores in the dataset are continuous. To resolve this conflict, I used binning to convert the continuous scores into categorical values. This meant choosing a value to act as a cutoff point, such that an article is defined to have an edit war if and only if their controversy score is above the cutoff. The target feature of the data is then defined as whether or not an article has an edit war, according to the cutoff.

To choose a cutoff, I visualized the controversy scores as a histogram using R, excluding scores which were zero, to see if there was a natural way to split the data:

```
require(RMySQL)
require(dplyr) # for %>%

db <- dbConnect(MySQL(),
  user='root', password='<redacted>', dbname='edit_wars', host='localhost')
data <- db %>% dbSendQuery('select controversy from article') %>% fetch(n=-1)
data[data$controversy > 0,] %>% log() %>%
  hist(breaks=50, xlab="log(controversy)", main="Article Controversy (log scale)")
```



The visualization did not show any natural split in the data, so initially I arbitrarily chose the 90th percentile of scores as the cutoff point. This resulted in most models always predicting *False* no matter what, due to class imbalance. Next, I tried setting the cutoff point to 0, grouping together all articles with nonzero controversy. 21% of the articles had nonzero controversy, so this reduced (but did not completely eliminate) the class imbalance.

Scikit-Learn provides three types of Naïve Bayes models: Gaussian, Multinomial, and Bernoulli [2]. Here are their results when using a cutoff point of 0 and an 80/20 train/test split:

Gaussian Naïve Bayes

Actual/Predicted	True	False
True	0	4104
False	0	15110

Accuracy: **78.6%**
% True predicted correctly: **0%**
% False predicted correctly: **100%**

Multinomial Naïve Bayes

Actual/Predicted	True	False
True	1006	3098
False	534	14576

Accuracy: **81.1%**
% True predicted correctly: **24.5%**
% False predicted correctly: **96.5%**

Bernoulli Naïve Bayes

Actual/Predicted	True	False
True	2249	1855
False	2122	12988

Accuracy: **79.3%**
% True predicted correctly: **54.8%**
% False predicted correctly: **86.0%**

Note: for the Bernoulli model, all word frequencies were converted to booleans representing whether or not the word was present in the article.

Interpretation

We see that the Gaussian model always predicts *False* no matter what, since most of the input values are false. We wouldn't want to use this model in the real world if we cared about predicting which values were *True*, however this simple enough rule nets it an accuracy of 78.6%, making it a good baseline to compare the other models to.

The multinomial model, compared to the Gaussian model, sacrifices a tiny amount of accuracy when false for a large gain in accuracy when true, with slightly higher overall accuracy.

The Bernoulli, compared to the multinomial, sacrifices a little more accuracy in the false case for another even larger gain in accuracy in the true case. Its overall accuracy is in between the Gaussian and Multinomial models, however it is the only model where even after

separating out the true and false cases, both subsets have greater than fifty percent accuracy. This tells us that it is a definitively better predictor than random chance.

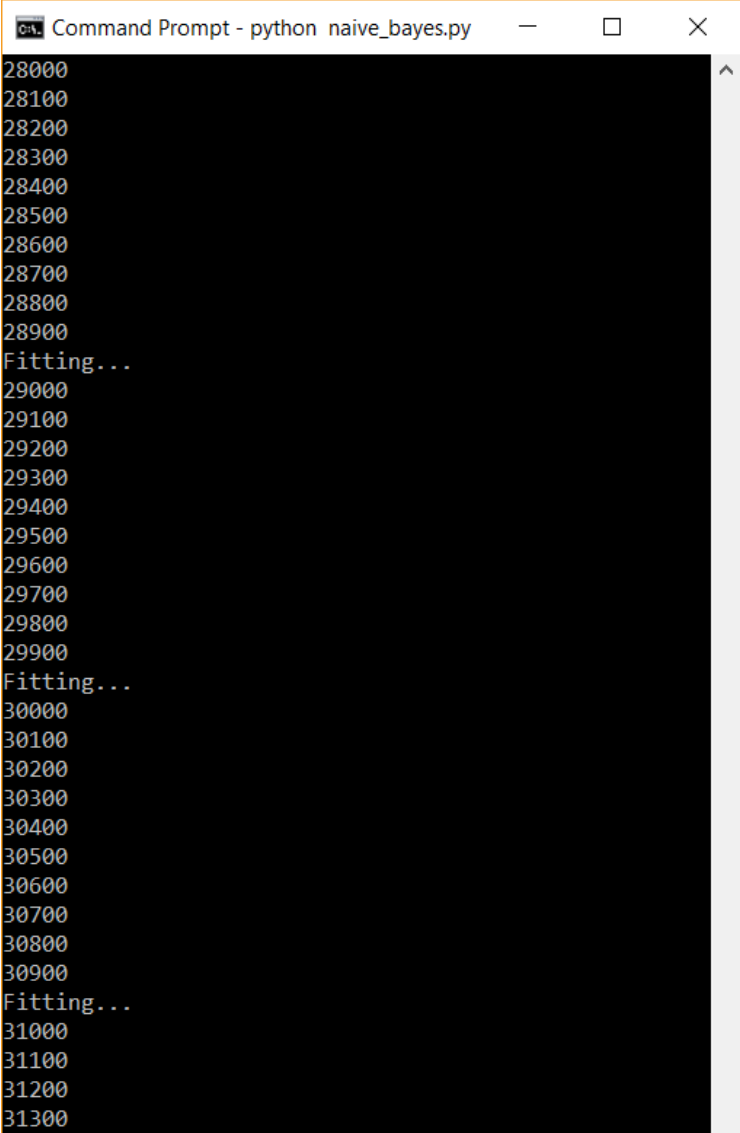
Another interesting thing to note is that the Bernoulli model, which has high overall accuracy across classes, does not look at how frequently words appear in a given article – it only takes into account the existence or nonexistence of a word. This suggests that the actual frequency of a word is relatively unimportant compared to whether or not it exists at all.

Difficulties

The primary difficulty for this project was the size of the data set. The data table was a CSV file that was 21 GB in size, which was larger than the amount of RAM I had available. Because of this, I was not able to run the algorithms on the entire data set at once – instead, I loaded batches of one thousand rows at a time into memory and ran the algorithms piecewise on each batch.

In the image on the right, you can see the process of training a model. The numbers represent how many articles have been loaded so far, and every thousand articles, it does another round of fitting.

Furthermore, because the dataset contains nearly sixty thousand features, training a model was a slow process, requiring over an hour to train each model, which limited the amount of different models I could try.



```
Command Prompt - python naive_bayes.py
28000
28100
28200
28300
28400
28500
28600
28700
28800
28900
Fitting...
29000
29100
29200
29300
29400
29500
29600
29700
29800
29900
Fitting...
30000
30100
30200
30300
30400
30500
30600
30700
30800
30900
Fitting...
31000
31100
31200
31300
```

Future work

One direction to take this project in the future would be the use of Principal Component Analysis (PCA). PCA is a technique used to reduce the dimensionality of data while losing as little information as possible. Since the primary limiting factor in what I was able to do was the amount of columns, using PCA to reduce the amount of columns in my data would open up many more avenues of investigation. Unfortunately, PCA runs in $O(p^3)$ time where p is the number of features, which meant that I had neither the time nor the computing power to run it for my project. In the future, however, given more time and computing power, PCA is something I would like to attempt.

Another improvement that could be made to this model would be to add article classification/genre as a feature to predict on. This seems like it would likely improve classification rate, because what inflames an expert in a given subject will heavily depend on what subject one is looking at. Wikipedia's API does not have a built-in way of getting the classification of an article, so I did not include that data in this project, however a future avenue of investigation may be to find an alternative way of getting an article's classification.

References

- [1] Yasseri T., Spoerri A., Graham M., and Kertész J., The most controversial topics in Wikipedia: A multilingual and geographical analysis. In: Fichman P., Hara N., editors, Global Wikipedia: International and cross-cultural issues in online collaboration. Scarecrow Press (2014).
- [2] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.