ZOT! to Wikipedia Vandalism

Lab Report for PAN at CLEF 2010

James White and Rebecca Maessen

School of Information and Computer Science
University of California, Irvine
<jpwhite, rmaessen>@uci.edu

Abstract This vandalism detector uses features primarily derived from a word-preserving differencing of the text for each Wikipedia article from before and after the edit, along with a few metadata features and statistics on the before and after text. Features computed from the text difference are then a combination of statistics such as length, markup count, and blanking along with a selected number of TFIDF values for words and bigrams. Our training set was expanded from that supplied for the shared task to include the 5K vandalism edit corpus from West et al. Vandalism edits in the training set that were classified as "regular" by a classifier trained on all the data were removed from the training set used for the final classifier. Classification was performed using bagging of the Weka J48graft (C4.5) decision tree [3] which resulted in an evaluation score of 0.84340 AUC. It is unclear whether the expanded vandalism data improved or degraded performance because that changed the ratio of regular to vandalism edits in the training set and we did not make any adjustment for that when training the classifier.

1 Introduction

Although it has been shown by West et al [4] that metadata features alone can identify some vandalism, there are many edits that can only classified accurately by examining the text. The design of this Wikipedia vandalism detector largely follows that of Potthast et al [2] and Amit Belani [1] in that it combines a modest number of features derived from metadata and textual statistics with a bag-of-words approach to classification. Rather than use term counts in the article before and after edit as in Belani, we perform textual differencing and use characteristic stemmed terms and bigrams from that inserted text. We also compute statistics regarding edit size and changes to MediaWiki markup which prove to be useful.

2 Data Sources and Training Edits

The training set provided for the PAN @ CLEF vandalism detection task is a corpus of 15,000 training cases with 944 classified as vandalism, which comprise 6% of the total training set. This is comparable to the real world ratio of 'good' to 'bad' edits on Wikipedia. The small number of vandalism edits in this unbalanced data provide little information to automatically generalize over the entire Wikipedia. In order to find more patterns specific for vandalism we decided to include additional vandalism edits into our data set. The West et al data set (http://www.cis.upenn.edu/ westand/) contains 5.7million automatically determined offending edits along with 5,286 manually confirmed vandalism edits. We used these 5,286 manually classified vandalism edits to expand our training set. After expansion, our training dataset holds 14,076 regular edits and 6,207 vandalism edits. Compared to 6% ill-intended edits in the provided training set, this expanded set contains 31% vandalism edits (close to the ratio Potthast et al used in their work [2]).

3 Data Preparation

The data set as provided has few nominal or numerical features directly suitable for the regular vs. vandalism classification task. We have focused on deriving features using text analysis methods.

3.1 Text Differencing

The primary text for each edit consists of the entire article text (in WikiMedia "wiki-markup" format) from before the edit and after the edit. While we do use some features based on those whole text versions of the article (such as length and markup counts), much of our classification approach is based on trying to characterize just the new text added by the editor in this edit. The reason for trying to just deal with the edit's new text is that most (but certainly not all) vandalism consists of inserting vulgar or nonsense words and phrases into articles.

We searched for text differencing algorithms (with implementations) and tried several different methods in order to find one well suited to our task. A common problem with line-oriented text differencing algorithms is that they do not handle lines that get split into two (or more) lines very well. Usually in that case at least one of the "new" lines will be flagged as the insertion of text. But we want to recognize that all of the text in both lines is from the original, otherwise the text actually inserted by the vandal will be dilute by the (presumably) "regular" text from the old version of the article. The implementation we used is "google-diff-match-patch" by Neil Fraser (http://code.google.com/p/google-diff-match-patch/). It does a character-oriented diff with some line-oriented optimizations. There are also a couple of "clean up" methods and we are using the "semantic clean up" which employs useful heuristics. The results are not quite ideal for this application though, because (unlike sentences) we would like to get complete words when any part of a word is changed so that our word-oriented features will match properly. We implemented a character-for-word substitution codec in order to achieve that behavior.

3.2 Word Vectors

One set of features is derived from the text inserted by the edit (as determined by the text differencing routine described above) by using the Word Vector Tool for RapidMiner (http://wvtool.sf.net/). In order to put the text in the form required by the tool, the inserted text for each edit is put into a separate file (using the editid as the file name) and with the “vandalism" edits in one directory and the “regular" edits in another. The current settings are TFIDF (term frequency/inverse document frequency), prune below 3 (terms must appear in at least three documents), string tokenization, Snowball stemming, and N-grams up to length 2.

The Word Vecor tool resulted in a word vector of 29,106 attributes. We were able to run logistic regression on this data in a reasonable time. However, when we added the metadata features to the word vector our classifications took too long to run, even though this was an addition of only 20 attributes. To make our data more manageable, we used *CorpusBasedWeighting* and *AttributeWeightSelection* tools in RapidMiner to select the words from the word vector with the most significance to our classification by choosing 300 features from each of the two classes.

3.3 Historical Information

Some useful metadata is time sensitive, such as the length of time an editor has been using Wikipedia, the number of edits they’ve made, and whether they’ve vandalized Wikipedia before. In order to ensure that we don’t use “future" information in our classification, calculation of those features is performed in order based on the ’edittime’ feature. Due to the limited time span covered by the training set, we did not actually gain much information using these computed features.

3.4 Edit Features

The raw metadata features and article text files were processed to generate the following features:

Anonymous editor
A registered Wikipedia user is less likely to vandalize. If the *editor* displays an IP address the edit is made by an unregistered user. The value of this feature is 1 if the *editor* feature matches the pattern for an IP address, 0 otherwise.

Editor edit count
The number of edits the editor has made prior to this one.

Editor revert count
The number of edits the editor has made that have been reverted prior to this one.

Revert comment
1 if *editcomment* feature contains a pattern matching “revert" (ignoring case), 0 otherwise.

Vandal comment
1 if *editcomment* feature contains a pattern matching “vandal" (ignoring case), 0 otherwise.

Is a reversion
1 if this edit is a reversion to a prior version of the article, 0 otherwise. An edit is considered to be a reversion if the ’Revert comment’ or ’Vandal comment’ features are true, or if the MD5 hash of the new version of the article text matches that seen for a prior version of the article.

Comment length
The length of the ’Edit comment’ after removal of the automatically generated section label (“/* SECTION */"), if any. This is a valuable feature because many vandals leave no comment about their edit.

Old article size
Number of characters in the article text prior to the edit.

New article size
Number of characters in the article text after the edit.

Size difference
Difference in the number of characters in the article text (new – old). Vandalism tends to be short, but even very lengthy vandalism is rarely more than 1000 characters.

Size difference ratio
Difference in the number of characters as a ratio (if old != 0, (new – old)/old).

Old markup count
A measure of the amount of wikimarkup in the article text prior to the edit.

New markup count
A measure of the amount of wikimarkup in the article text after the edit.

Markup count difference
Difference in the amount of wikimarkup in the article text (new – old). Vandals usually do not use wikimarkup in their additions.

Markup count difference ratio
Difference in the amount of wikimarkup as a ratio (if old != 0, (new – old)/old).

Is article blanked
1 if the new article text is blank. Blanking an entire article is very often vandalism.

Is article replacement
1 if the new text is effectively a complete replacement for the old text, 0 otherwise. Note that the determination of what constitutes a “complete replacement" is based on the text differencing logic which tries to distinguish between coincidental similarity and actual changes.

Is section replacement
1 if the edit completely replaced an existing section with new text (including removing the previous section header), 0 otherwise. This is characteristic of many vandalizing edits because of the tendency for vandals to click the “edit" button for a section and then replace the entire section.

Is section content replacement
Section header is preserved in the new text. Another common pattern for vandals.

All CAP word count
The number of “words" (currently runs of at least 4 alphabetic characters) that are all capitals. Regular edits have few words longer than 3 letters that are all capitals.

Bad word count
The number of matches to a regular expression for various commonly used vulgar words.

4 Classification

We used a decision tree stucture to distinguish vandalism edits from regular edits. The datamining tool RapidMiner v4.6 functioned as a framework for our algorithm. We used the Weka J48-graft classifier to generate a C4. 5 decision tree [3]. Parameters that affect the results of the classification are the level of pruning and the minimum number of instances per leaf in the tree. The defaults were used for all settings (C=0.25, M=2, all option flags set to false). To improve our results we used bagging of that learner. The parameters are number of iterations (11) and the fraction of examples (0.9) selected for each iteration.

The classifier was trained two times. The first training pass used all the training data and was then used to classify the training data. Vandalism edits that were classified as "regular" by that classifier were removed from the data set used to train the final classifier. The rationale being that because our classifier would be unable to classify certain types of vandalism as vandalism (e.g. those that either have Neutral Point Of View issues or are carefully disguised pranks) that precision would improve by not using those cases. Cross-validation testing showed a substantial improvement in performance using this method.

Decision trees are interesting classifiers because they have the potential for being implemented in such a way that expensive-to-calculate features are only evaluated when they would be useful, which would be a consideration for a production implementation of the classifier for Wikipedia.

5 Evaluation

We evaluated our classifier by running a ten-fold cross validation on our training set. The 10-fold cross validation on our training data using the W-J48graft learner and bagging resulted in an f-measure of 82.03 and an AUC of 0.960. Using the trained learner on the test data gave an AUC of 0.84340.

6 Source Code

The source code for our implementation is a combination of Groovy and RapidMiner scripts and is available via Subversion under the GPL v2 license. Go to http://code.google.com/p/zot2wpv/ for access details.

7 Conclusion

The research that we performed shows that decision trees are a good approach to find ill-intended Wikipedia edits. Additionally, decision trees train a lot faster than other classifiers that we looked at, such as logistic regression. Efficient performance is an important consideration for a production implementation of the classifier for Wikipedia.

Our findings confirm that top down feature analysis on text difference as well as statistical information from the word vector is relevant to classifying vandalism edits.

We were able to demonstrate this by classifying using only the metadata features or the word vector features first, and subsequently run the same classifiers on the features combined. The fact that neither attribute set is as predictive as the attribute sets together indicates that both type of features are significant for predicting vandalism.

8 Acknowledgements

Most of this work was performed in the course of CS175 at UC Irvine taught by Arthur Asuncion during Spring 2010. We also thank Professor Bill Tomlinson for his support.

References

1. Belani, A.: Vandalism detection in Wikipedia: a bag-of-words classifier approach. Computing Research Repository (CoRR) abs/1001.0700 (2010)
2. Potthast, M., Stein, B., Gerling, R.: Automatic vandalism detection in Wikipedia. In: ECIR'08: Proceedings of the IR research, 30th European conference on Advances in information retrieval. pp. 663–668. Springer-Verlag, Berlin, Heidelberg (2008)
3. Webb, G.: Decision tree grafting from the all-tests-but-one partition. Morgan Kaufmann, San Francisco, CA (1999)
4. West, A.G., Kannan, S., Lee, I.: Detecting Wikipedia vandalism via spatio-temporal analysis of revision metadata. In: EUROSEC '10: Proceedings of the Third European Workshop on System Security. pp. 22–28. ACM, New York, NY, USA (2010)