Practical Machine-Learning Vandalism Detection on Wikipedia

Introduction

It is assumed that all reading this have some familiarity with Wikipedia and its operation, and are aware of the growing problem of vandalism on its pages. Presently, there are a number of anti-vandal bots which analyze edits in real-time and revert edits they detect as vandalism, however, these bots both miss a majority of vandalism, and generate a number of false positives.

Currently, the ant-vandal bots running on Wikipedia, including ClueBot, use a set of simple heuristics to determine whether or not an edit is vandalism. To avoid an unacceptable false positive rate, these heuristics are scaled down to the point where they are only effective against the most obvious and easy-to-detect vandalism.

There have been a number of experiments relating to machine-learning vandalism detection on Wikipedia, but none have been put into production. After examining several such reports, I can see several critical flaws in most or all of the proposed systems, that may allow them to work well in theory, but not when put into practice.

The most common statistic given when analyzing a new method of vandalism detection is "percentage of correctly identified edits". While 90% of edits correctly identified is good, in theory, think about what this could mean. If 10% of edits are misclassified, that is probably about equal parts false positives and false negatives. Even if a bot correctly detects and reverts 90% of vandalism, a 5% false positive rate is generally considered completely unacceptable. This would mean that one out of every twenty legitimate edits on Wikipedia is incorrectly identified as vandalism and reverted.

Another often-ignored practical consideration is speed. This is partly an algorithmic concern, and partly an implementation concern. Many of the proposed machine-learning algorithms relating to vandalism are inherently slow while running, and would be beaten by human vandal-fighters, defeating the purpose. Also, although I don't have details on the performance of specific experimental implementations, I'd imagine that hacked-together proof-of-concept code leaves a lot to be desired in terms of speed and reliability.

I am attempting to solve both of these problems, and more, developing an effective machine-learning anti-vandalism bot for Wikipedia, intended to be put into production, and not reach the end of its life in a research paper.

A Word About The Name

This new bot is called Cluebot-NG. Although it shares the majority of its name with the original Cluebot, it shares none of the logic or code. The reason these names are the same is that they both came out of the same community and software foundry – Cluenet – of which the author of Cluebot-NG and the original author of Cluebot are both co-founders.

Machine Learning Algorithms

There are many different machine-learning algorithms that can be used to detect vandalism, each with its own set of advantages and disadvantages. Instead of picking a single algorithm, it may be beneficial to apply multiple to the same problem.

Cluebot-NG is designed to be modular so it is easy to add new algorithms. Currently, two are in use – a naïve Bayes classifier, and an artificial neural network.

Naïve Bayes Classifier

The naïve Bayes classifier portion of the bot is fairly traditional, using standard algorithms. With a sufficiently-sized dataset, it learns with fair certainty probabilities of words used in both vandalism and non-vandalism. This alone is a substantial improvement over existing bots in production, as simple heuristics rely on estimated weights. A naïve Bayes classifier can precisely calculate probabilities – both for vandalism and non-vandalism.

The classifier in practice generates few to none false positives, as the most strongly weighted vandal-words are almost never used in legitimate edits, and in the few instances they are used, they are counter-balanced by the words that are not commonly used in vandalism in the same edit. However, a naïve Bayes classifier misses a lost of vandalism – specifically, vandalism that does not contain words previously seen to be related to vandalism. In itself, it is an improvement on existing bots, but is not sufficient to achieve over 50% vandalism detection rate.

Artificial Neural Network

The artificial neural network portion of the bot works with the statistical aspect of vandalism. Various statistics are performed on the edit and fed into the input layer of the neural network. The accuracy of this method depends chiefly on the selection of statistics to use, the precision in calculating the statistics, and the scaling function used to convert the statistic into a value usable by the neural network.

This method, used alone, can achieve slightly better results than a naïve Bayes classifier, as most vandalism, even that which only inserts bad-words, has many statistical properties that allow it to be identified as such.

Combining the Methods

One possible way to combine the outputs of these two different strategies is simply to average them, or perhaps perform a weighted average. But then, the problem becomes, what weights should be used? The combination of scores itself should also be able to benefit from machine learning.

The weights could be automatically adjusted, and that would achieve fair results, but still not optimal.

Using the output of the naïve Bayes classifier as a single input to the neural network seems to be the best way to combine the two. The neural network not only finds optimal weights, but can also recognize patterns and relationships between the Bayesian score and other statistics. Tests show that this works as well in practice as it does in theory.

Finding an Appropriate Result Threshold

The result from the above algorithms is a number from 0 to 1, with 1 being most likely vandalism. Selecting different thresholds will yield different false positive and false negative rates. As previously stated, existing machine learning attempts optimize this threshold for total accuracy – ie, percentage of edits correctly classified. They select the threshold to maximize this value. While this may look good on a research report intended to impress a professor, it is not the ideal method in practice.

For production Wikipedia bots, it is better to have ten uncaught vandalism edits than a single false positive. Optimizing a threshold for total accuracy will likely yield a threshold of around 50%, and approximately equivalent false positive and false negative rates. As previously stated, this is unacceptable.

Cluebot-NG provides facilities for selecting a target false positive rate, and calculating a threshold based on that. All analysis and development is based on maximizing the percentage of vandalism detected, given a fixed false positive rate. The total accuracy will be lower (ie, it will catch less vandalism than it is theoretically capable of), but there will be many fewer false positives.

Estimates given to me for an acceptable false positive rate range from 1% to 0.5%. During development, I chose to use the lowest estimated value – 0.5% - although this can be changed easily at any time.

The core engine is still in development, but preliminary results are extremely promising. Using that 0.5% false positive rate as a maximum, I am currently getting around 65% vandalism detection rate (the total accuracy is much higher, at the cost of more false positives). The threshold for an 0.5% false positive rate is around 95%. This means that 65% of vandalism is being identified correctly with a confidence of over 95%.

Discussion of the Training and Trial Datasets

For machine-learning algorithms such as this, the datasets used to train and test it are of utmost importance. If the training dataset is not accurate, the accuracy of the bot will be much lower than its potential. If the trial dataset is not accurate, statistics may be reported incorrectly. Incorrect statistics that are lower than actual production performance are fine – however, statistics that indicate a better performance than can actually be achieved in production are dangerous.

The dataset of "good" edits being used here comes from a set of human-reviewed edits, so these are guaranteed to be accurate, and are also close to a random sampling.

The dataset of vandalism edits consists of vandalism edits reverted by humans – NOT by existing bots. I decided this because existing bots generate a fair number of false positives, many of which go unreported, and it's not acceptable to have the training set polluted by false data. Using only human-reverted edits has a few effects on the outcome.

First, it can be fairly certain that the outcome is an accurate measure of production performance, although actual production performance may be better (see next). This is a good indicator when considering the effectiveness of the bot using the trial set. Additionally, the trial dataset does not include edits reverted by bots. This means that the statistics presented here are statistics on edits not currently detected by other bots. It is also reasonable to assume that if Cluebot-NG has such a high accuracy with hard-to-detect vandalism (vandalism not detected by simple heuristics bots), that it will catch all of the vandalism currently caught by heuristics bots as well, without increasing the false positive rate. So, in fact, the actual performance in production will be **better** than what is indicated by trial statistics.

This differs from the datasets used by other machine-learning attempts. Their datasets not only include bot-reverted edits, but may even have a disproportionate number of them, as bot-reverted edits with consistent edit summaries are easier to detect than human-reverted vandalism in some cases (as humans can revert edits for reasons other than vandalism).

Speed

One goal from the beginning for Cluebot-NG is that it has to be fast. If a human can revert an edit in less time than the bot can, the bot is pointless. The bot should act as a first line of defense before humans get a chance to spend time reverting it. Other proposed machine-learning methods probably suffer from speed problems, as do many existing simple heuristics bots.

Cluebot-NG is written in a compiled language, C++, using high-efficiency principles. Most other existing bots are written in slower scripting languages, and are not designed for efficiency. For example, existing heuristics bots greatly overuse regular expressions. Regular expressions can be very useful, and individually are fast, but when hundreds of regexes are used in sequence to scan the same text, performance suffers significantly.

Cluebot-NG consists of three main parts. Normalization/Statistics on the edit, Bayesian processing, and neural network processing. All three have been optimized. The part that takes the longest is the normalization and statistics processing. For this, dedicated routines have been written to replace the functions of regular expressions in most other bots. These routines calculate many statistics at once, to avoid traversing the text multiple times. Regular expressions are still used for certain things, but to a much lesser extent than in other bots. When regexes are used, they are fast system native POSIX regexes, not whatever is shipped with whatever scripting language the bot is written in.

Bayesian processing is very fast, as it uses Berkeley DB for its datastore instead of a slower SQL database or, worse, a flat file. ANN processing is also fast, using libfann (Library for Fast Artificial Neural Networks).

Overall, a single edit can be fully processed and scored in about 0.02 seconds (on my desktop machine).

In addition to this raw speed, processing can be parallelized, allowing throughput to increase almost by a factor of the number of CPU cores on the system. This is actually largely useless in production, as the Wikipedia edit rate is significantly below what can be handled by a single thread, but it is useful in training, when an entire training dataset has to be processed.

Architecture

Cluebot-NG is designed to be clean and modular. This modular capability comes in two aspects. First, C++ modules can be easily written to extend the core functionality of the bot. Second, module parameters and order can be extensively configured by configuration files. Almost all existing metrics can be configured and changed using this method, without changing the source code or recompiling.

Conclusion

Once put into production, this bot could revolutionize vandalism prevention on Wikipedia, cutting down the undetected vandalism rate by a factor of two or more. Currently, statistics show it catching 65% of vandalism with a conservative false positive rate of 0.5%. As described above, this includes only vandalism not caught by current bots – and as Cluebot-NG is likely faster, it should be able to catch what is handled by current bots as well. As the dataset grows, and the code is fine-tuned, performance can only improve from here.

Anyone wishing to learn more about Cluebot-NG or help in its construction should contact Chris Breneman (nick: Crispy) on irc.cluenet.org #cluebotng
Chris Breneman is being assisted by Tim Sears and Cobi Carter in the creation of Cluebot-NG. Tim is generating the datasets and may write the production Wikipedia interface, and Cobi is providing some resources for development.