Statistically Tagging Blog Posts

Jonathan Hohle – 993867167
jonathan.hohle@asu.edu

Abstract

This paper describes the implementation of Bayesian tagging system, and its application on personal and commercial blog posts. Multiple accurate tags can be determined for a post, with increasing accuracy as the blog corpus gets larger.

Author Keywords

Bayesian Classification, Blogs, Tagging

The Problem

According to Technorati, there are 55 million blogs, most of which have some kind of categorization system. This categorization, or tagging is largely a manual process, typically executed by the authors of the posts that make up each blog.

There are two major types of blogs on the internet today: commercial blogs, which feature bloggers who are paid to post (or encouraged by their employers) with many authors, and personal blogs, generally with one author.

Commercial blog tagging seems to be a mishmash of very relevant tags combined with tags chosen at the whim of the author. Sometimes posts get made with no tags at all. Personal blogs may be more consistent, but like commercial blogs, generally have a decent corpus of manually tagged posts. Couldn't blog tagging be automated?

Many people use text classifiers every day without even thinking about it. The spam filters included with most email clients are generally some form of statistical classifier which is attempting to put email in a spam category or a ham category. Spam filtering is very effective, so it seems like a good starting point for a blog post classifiers. Blog post classification is more than a two class problem however, and optimally, multiple classes could be assigned to a post. How do we get their from a two class problem?

Previous Work

Paul Graham is responsible for A Plan for Spam[2], a seminal work on the topic of spam classification. His paper spawned an entire industry of spam classification products. His approach is simple, look at all the tokens in an block of text, and find determine how many spam emails contain those tokens compared to how many not spam emails. If the probability is in favor of the spam emails, the email never makes it to the inbox. With even very low thresholds, Paul Graham's techniques are very effective at keeping junk out of our inboxes.

Ben Kamens saw the usefulness of these Bayesian classification method, but was interested in sorting mail for his comany's FogBugz software into more categories that just spam and ham. Of course he wanted that functionality as well, but he also wanted to automatically sort mail based on its content. While investigating this problem he discovered a something which helped him increase his accuracy. Comparing each message in a category against all the messages not in the category was not effective. However, comparing a message to only the messages in two categories at a time was very effective.

In Beyond Binary Classification [3], Ben Kamens describes what he calls a Bayesian tournament algorithm. Two categories are chosen and evaluated. The category with the highest probability moves to the next round, where it is compared to the next category and so on, until all of the categories have been evaluated. The category which makes it through to the end is the most likely category. This method proved to be so effective, that it was integrated and now ships with Fog Creek's FogBugz software.

This research gets us to being able to accurately choose one category, or tag, for a blog post, but blogs posts generally have several tags? The rest of this paper will discuss the implementation of a system which is able to tag blog posts with several relevant tags.

Data Set and Aggregator

For my corpora I chose to use an individual's blog and a commercial blog. For the individual blog I chose to use my own (http://hohle.net), which consisted of 174 posts, each tagged with a single category. This data was readily available to me, and the live database was copied over into a database specifically for this project.

For a commercial blog, I chose The Unofficial Apple Weblog (http://tuaw.com/). I chose this blog because it was familiar, it has multiple authors, each with their own tagging style, and the corpus of posts was large (over 9500 tagged posts). To aggregate the posts and tags from the blog, I wrote a small spider which scrapped each of their posts, and dumped their content to my database. After the initial scraping I would retrieved deltas daily, however, stopped retrieving posts during the final week of the project. Each post from the blog was tagged 0 to 5 times.

Implementation Details

Several implementation choices were made which may have affected the accuracy of the classifier. The first major decision was how to "tokenize" blog posts. I chose to use Paul Graham's approach and ignore characters which weren't alphanumeric, dashes, dollar signs, or single quotes. Put simply, I split the title and content strings using the regular expression /[^a-zA-Z0-9\-$']/ and combined the two arrays. Duplicates were removed from the array as well as any empty strings and strings with only whitespace.

My first step was to implement Ben Kamens' algorithm as a base classifier and expand from there. I was not satisfied with the speed of my initial implementation and spent several iterations trying to improve the performance while the algorithm was still simple. Most of the processing time was spent at the database going through the corpus data. While the tables were optimized and indexed appropriately, looking up the count of posts with a certain token and tag was very expensive. I simplified the queries significantly, but there might still be several hundred token lookup queries per classification, multiplied 90 times, once for each tag belonging to the commercial blog. Since the database was the bottleneck, I had plenty of RAM to spare, and my corpus was no longer changing, I made the decision early on to only run queries once if necessary. I also reduced most of the queries to only operate on primary and foreign keys, which were cached as described earlier. This significantly improved the performance of the classifier. An initial classification would take nearly a minute, but subsequent classifications would only take a few seconds.

The probability of each token was defined as $p(w)$ and was defined as

$$p(w)=b_1(w)/\ b_2(w)$$

where

$$b_n(w)=(\#\text{ of posts with tag } n \text{ and token } w)/\ (\#\text{ with tag } n)$$

with $b_1(w)$ being the tag which is considered the winner and $b_2(w)$ being the tag which is considered the challenger.

In Paul Graham's paper, he suggests not looking at every token, but only looking at the most interesting tokens. The most interesting tokens are those whose probability is closest to 1.0 (favoring the winning tag) or closest to 0.0 (favoring the challenging tag). To simply sort on interestingness, I wrote a function which would return from 0.0 to 0.5 – 0.0 being not very interesting and 0.5 being extremely interesting. This function was defined as $i_w=|p(w)-0.5|$. I chose to use the 25 most interesting tokens.

To choose multiple relevant tags, I ran the post

through Ben Kamens' algorithm a second time, excluding the tag that was selected during the first iteration. I then looked at the probability of the post being the best tag (the first chosen), or the tag which as just selected. If that probability was over a certain threshold, I kept the new tag, and repeated the process, always comparing the newly selected tag with the original tag. When the probability fell below the threshold or 5 tags were retrieved, the evaluation was stopped. The threshold chosen for testing was 0.54.

My implementation included a webapp which mimicked a simple blog authoring interface. As you type, the editor would attempt to classify your post and presents the author with tags that are relevant to the content being produced. Because the classifier would use 25 words to classify, it generally took about that many for the classifier to get in the general ballpark of what the article was about. Anything above that would provide mostly accurate tags.

Accuracy

To test my initial implementation of Ben Kamens' algorithm, I classified posts in 200 post groups. I soon realized that a programatic indicator of accuracy would be difficult because the tags given to posts weren't necessarily meaningful or accurate. A post might be tagged “analysis / opinion,” but other posts with that tag are wide ranging. To convince myself the classifier was working, I automated the process of classifying groups of posts and checking to see if the first tag selected was chosen by the post author. For my own blog, with limited data and only single tags for posts, the classifier was able to correctly tag the post 90% of the time. For The Unofficial Apple Weblog, the results were much higher with the lowest grouping above 98% and the highest at 99.5%. This was enough to convince me that the classifier was working.

Once I moved to multiple tags any validation was done by hand. This was much more time consuming and getting multiple parties to validate the results was difficult. In all, about 1 and 20 posts included a tag which was not appropriate for the content.

Conclusion

While automated tagging might not be a solvable problem, a classifier with a large corpus can accurately suggest tags to an author. These tags might have to be manually selected, or massaged, but they provide a good starting point for consistent site-wide tagging.

There are many improvements that could be made to the current implementation. Currently, the probabilities of the top 25 tokens are average to get an aggregate probability. Using something like chi squared might be more accurate. Also, while html entities are valuable for classifying spam email, they might not be so valuable for classifying blog posts. Because they are relatively benign, they probably don't hurt classification, but taking them out might reduce the number of uninteresting tokens evaluated (and if nothing else, speed up processing).

References

1. http://technorati.com/ (2006)
2. Graham, P., A Plan for Spam. http://www.paulgraham.com/spam.html. (2002)
3. Kamens, B., Beyond Binary Classification. http://www.fogcreek.com/FogBugz/Downloads/KamensPaper.pdf. (2005)