Detecting ads in a machine learning approach

Di Zhang (zhangdi@stanford.edu)

1. Background

There are lots of advertisements over the Internet, who have become one of the major approaches for companies to advocate their products and for webmasters to make money by posting these ads. Some big players in this market are Google and Facebook, with a lot of other companies.

These ads can sometimes be useful, while can also be very annoying at some other occasions. One of the purposes for this project is to analyzing the distributions of ads provided by different sources, so we could have a rough idea about the environment of Internet.

2. Model selection

Online ads are usually in the form of Javascript, which dynamically loads advertisement contents from server at the time a page is rendered. These code are often copied from the provider, or requests same files from provider, which means they may have a well-discernable pattern in the HTML file. Therefore, the inputs are a bunch of static HTML files, while the output is whether a file is considered as containing or not containing ads. This led me to think about the classic algorithms of text classification. More specifically, Naive Bayes and SVM.

3. First analysis

3.1 Dataset

I randomly got 2000 webpages for both positive and negative. (A page is positive means it contains ads, while negative means it contains no ads.) Among these pages, I used 1000 positive pages and 1000 negative pages as my training set, and the other 1000 positive pages and 1000 negative pages as my test set.

3.2 Pre-processing

Advertisements on webpages are typically present in the form of external javascript, so I only considered the http links in the page, and ignored everything else. For each link in a page, it is tokenized into multiple words, with all non-alphanumerical characters being dropped. For example, http://www.amazon.com/ will be tokenized into {“http”, “www”, “amazon”, “com”}. And all tokens being parsed out from a page makes an training/testing example fed into the learning algorithm.

3.3 Learning algorithms

At this point, the problem is pretty much simplified to a standard text classification problem, so Naive Bayes might be the right thing to do as a starting point. Without much surprise, I got the following training and validation accuracy.

	Training accuracy	Testing accuracy
Naive Bayes	81.32%	75.85%
SVM	100.00%	80.55%

Figure 1. Training and testing accuracy for Naive Bayes and SVM

One step further, if we think a little deeper about the nature of advertisement links, they usually have a pretty stable pattern, or even exactly same url. Therefore, instead of using the unigram tokens, it might make sense to use n-gram tokens. This indeed helped the performance, and I got the result of Figure 2 by varying the token size, i.e. how many words makes a token.

We can see that both Naive Bayes and SVM get a gain when we have multiple words in a token. Due to the difference in their natures, these two algorithms reach peak at different sizes of tokens. Naive Bayes peaked at 7, while SVM at 3.

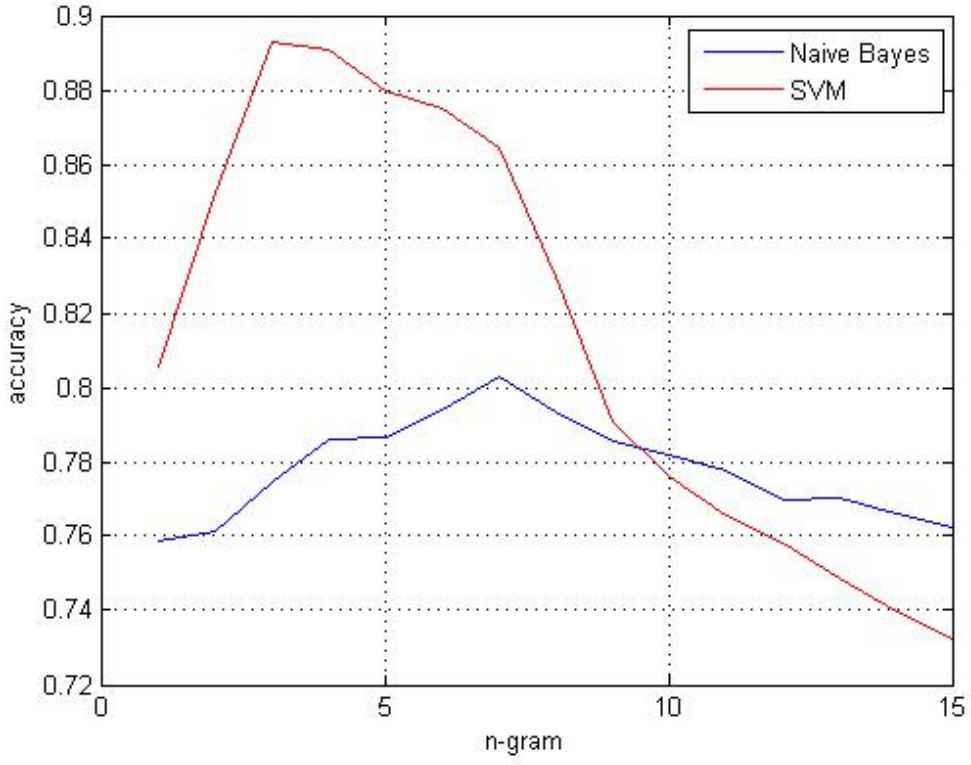

Figure 2. accuracy against n-gram terms

4. Discussions

4.1 Motivation

Figure 2 showed that SVM algorithm can make a prediction with a little less than 90% accuracy when I used tri-gram as input. It would be interesting to study more about the properties about SVM algorithm.

4.2 Kernel

Kernel is probably the most important part of the SVM algorithm, and Figure 3 shows some result for polynomial and Gaussian kernel. However, to my surprise, the performance degrades very fast when the degree of polynomial kernel increases. And it basically becomes random guess when the degree goes to infinity (Gaussian kernel).

Kernel	Linear	2nd-order	3rd-order	4th-order	Gaussian
Testing accuracy	0.805500	0.758000	0.734500	0.717500	0.535500

Figure 3. Testing accuracy for kernels in different order

4.3 Rational

[1] would be inspiring on explaining why Gaussian kernel is not good at text classification problems. If we take the Fourier transformation of the Gaussian function, we get

$$F(e^{-\frac{x^2}{2\sigma^2}}) = \int_{-\infty}^{\infty} e^{-\frac{x^2}{2\sigma^2}} e^{j\omega x} dx = \sqrt{2\pi}\sigma e^{-\frac{\sigma^2\omega^2}{2}}$$

where the result is essentially a low-pass signal filter, i.e. The function value decreases as frequency ω increase. However, when we work on text classification problems, the appearance of most features are pretty sparse, which means each appearance is like a Dirac function. As we know, Dirac function has abundant high frequency information, which would be filtered out by the Gaussian kernel, and thus the performance degrades badly .

4.4 Stop words

One widely applied technique in text classification is stopword elimination. These words are frequently appeared and don't have much real meaning. In our case, they are "http", "www" and "com". However, to our surprise, the performance degrades a little bit when these stopwords are removed. Naive Bayes gives 74.55% testing accuracy, while SVM 79.95%, which are both a little bit lower than the original result.

5. Distribution analysis

In the previous sections, each webpage is classified by whether it has an advertisement on it, regardless of what kind of ad it is. Therefore, it has been a classic text classification problem. The problem becomes a multi-label one when we want to detect what kind of advertisements are present on each individual webpage.

One straightforward extension from what I have already done would be running the previous algorithm for each individual advertisement. This can be time consuming, but is still doable given I only have a few dozens of different ads in total.

Here, I used homepages of top 10,000 sites given by Alexa [2].

Rank	Name	Count
1	Google Analytics	2947
2	Facebook Connect	1164
3	Google +1	1088
4	Facebook Social Plugins	1079
5	Google Adsense	1031
6	Omniture	840
7	Twitter Button	731
8	Quantcast	644
9	ScoreCard Research Beacon	636
10	DoubleClick	514

Figure 4. Most widely distributed advertisements

From Figure 4 we could see that Google and Facebook are occupying a large portion of the market. In fact, people are more careful when deciding whether to an ad on homepage. Users are easily irritated when they see too many ads on a page. So fewer ads would appear on homepages than other pages to better capture users. That being said, our Internet may be much more overloaded by ads what we see here.

6. Conclusion and future work

As we have seen in this work, advertisement detection can be considered as text classification problem with reasonable amount of data and accuracy. And n-gram tokenization could help overall performance while higher order polynomial kernel hurts. Also, we could have a rough idea about how overloaded by ads the Internet is.

For the future, it would be nice to train regular expressions [3] using input webpages for each advertisement. Also, more webpages are needed to make the distribution analysis more thorough and accurate.

References

[1] https://charlesmartin14.wordpress.com/2012/02/
[2] http://www.alexa.com/
[3] Li et al., “Regular Expression Learning for Information Extraction”, Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 21–30, Honolulu, October 2008.