Dividing the Haystack: Web Page Classification

Jan Ulrich
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
ulrichj@cs.ubc.ca

Abstract

Automatic web page classification is an integral part of analyzing the world wide web. In this paper I build a complete automatic classification system and compare different classifiers. I also build a classifier without using negative examples, thus making it much more practical. This work shows how to build an automatic text based classifier with the appropriate machine learning algorithms.

1 Introduction

With the explosion of the internet, better ways of processing, analyzing, and navigating through the data are necessary. The data analysis is complicated and requires a number of steps. Initially the data is downloaded by webcrawling. Then features have to be selected for classification. In order to do searching, relevancy analysis has to be performed. In this paper I focus on classification. Classification can make internet searches more effective as users could search for ”tiger” under animals or under operating systems depending on what they needed. Classification is also important for webcrawlers since they could be specialized to only look for certain webpages. Today much classification is done by hand at places like Yahoo and the Open Directory Project, DMOZ. Automatic web classification is needed as the web is growing at a faster pace than can be classified manually.

There are many different approaches to webpages classification. One approach is multi-class classification where you divide the set of webpages into several mutually exclusive classes. Given a test point, the classifier can then label it with a specific class. Another approach is to use a single-class classifier which only has two classes and makes binary decisions. The problem that can arise with these approaches is that webpages often fall into multiple categories thus breaking the mutual exclusion principle. One way around this is to use a single-class classifier for each class and then produce the union of all these classifiers [2].

However there is a practical problem with these single-class classifiers. In order to train the classifiers, labeled data needs to be provided. In the case of multi-class classifiers labeled data is easier to obtain (e.g. DMOZ). For single-class classifiers this means finding positive and negative examples. For example, if your class was homepages, you would have to produce samples of homepages and non-homepages. It can be hard to say what a non-homepage is. All areas of non-homepages should be represented equally and it should also be taken into account that the set of non-homepages is much larger than homepages. A solution to this problem is to do single-class classification without negative samples [7].

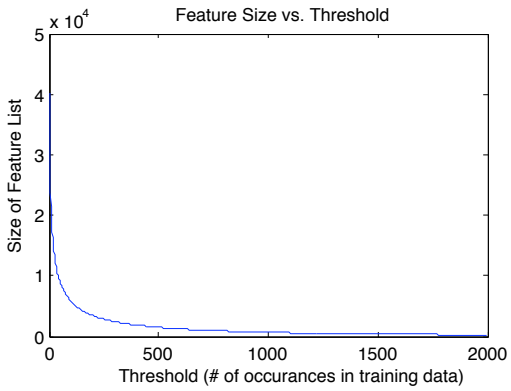

Figure 1: The number of text features decreases exponentially as the threshold increases. The features were cut off based on their frequency in the training set.

As I will demonstrate, the negative samples can be reconstructed from the positive samples and the universal set of webpages using a mapping-convergence algorithm.
I will first describe the system I built to get the data. In section 3, I will describe multi-class classification and present my results. Then in section 4 I present the PEBL algorithm for single-class classification along with my results. In section 5 I will describe the websitte that I built to test the classifiers. Finally, I will conclude in section 6.

2 The Data

The web contains billions of webpages. Such a vast number makes analysis a hard task. Since crawling the web was not the goal of this project, I used the Open Directory Project, DMOZ, as my web directory. Since I did not have the resources to process all the webpages in the database, I took a sample of 24737 pages for my training set which was one half percent of all the webpages. There are many different kinds of information to extract from a webpage, such as text, images, link structure, and layout. I decided to use only text as my features, but these techniques could be extended to a bigger set of features.

2.1 Feature Selection

Even though I decided to only use text as my feature space, feature selection was still a big issue since there are an infinite number of possible text features. I downloaded the webpages and extracted the text using an open source htmlparser, called HTMLParser. My first step in processing the downloaded text was to remove any symbols from the possible words. Then I verified words in an English dictionary. There are many languages on the web, but I decided to focus only on English webpages. Next I filtered out any stop words, from a stop words list. Stop words are common words that don't portray much meaning such as "I", "and", and "the". I included some French, Spanish, and German stopwords because I found that some of them were in my dictionary. For example in German "die" is just the article "the". In my downloaded data the feature space was 50077 words out of a possible 113000 words in my dictionary.
This many features was too much to process requiring me to reduce the feature space. In order to analyze the features, I created a frequency list of all the filtered words in my training set. I decided to narrow down my feature set based on frequency. There are more sophisticated was of doing this, but my project was focused on classification and

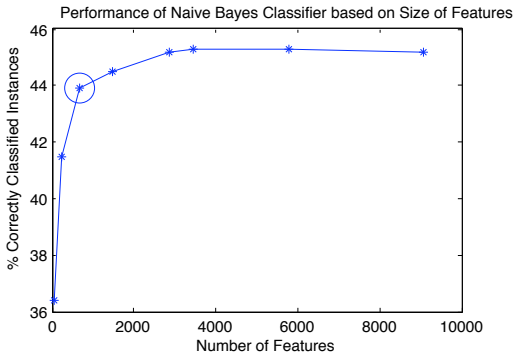

Figure 2: How performance varies with feature size. A Naive Bayes multi-class classifier is evaluated with 10 fold cross-validation using a training set of size 8672. The circled point is the size of the feature set that was chosen.

feature selection is just a prerequisite. The number of features varied exponentially with the threshold level as can be seen in Figure 1. In order to find the optimal number of features, I ran some tests with the Naive Bayes classifier on different number of features. As can be seen in Figure 2, the performance increases significantly in the beginning and has a miniscule peak around my sample point of 6000 features. Since larger feature set sizes become computationally intractable, I chose a feature set of size 657 with a threshold of 1000 because it gave good performance and was still computationally tractable.

3 Multi-class Classification

In multi-class classification, data is labeled with one mutually exclusive class. In my project, my training set was 24737 webpages as indexed by the DMOZ project. These webpages were categorized into 17 classes: Adult, Arts, Business, Computers, Games, Health, Home, Kids and Teens, News, Recreation, Reference, Regional, Science, Shopping, Society, Sports, and World. Regional and World were peculiar categories since they contained sub-labels such as American, Canadian, German, British, etc. However they contained all the other types of websites and therefore this classification was not mutually exclusive. Also I was not able to get good features from the World category (which was obviously the biggest) since I was using an English dictionary and the pages were in every kind of language. I therefore excluded these two categories from my training data and was left with 9526 webpages. I used the WEKA [5] data mining software to compare different classification algorithms on the data.

3.1 LogitBoost - The Most Successful Multi-Class Classifier

LogitBoost was the most successful algorithm, but it was not significantly better than SVM as can be seen in Table 1. LogitBoost is a boosting algorithm with a binomial log-likelihood loss function. Boosting was introduced by Freund and Schapire [3] as a powerful classification technique. In boosting a number of base classifiers are fitted iteratively to re-weighted data in order to build a strong classifier. In each iteration the misclassified data points are weighed more and the correctly classified data points are weighed less. Thus each time a base classifier is added, it works on correctly classifying the misclassified data [1].
The base classifier that I used was Decision Stumps, which are Decision Trees with two terminal nodes. Therefore a feature is selected to be decided upon for each Decision Stump.

Table 1: Multi-class classification results using 657 features and 10 fold cross-validation

Algorithm	Correctly Classified Instances
LogitBoost (with Decision Stumps)	50.32%
SVM (with a Linear Kernel)	49.40%
Naive Bayes	43.93%
C4.5 Decision Tree	40.64%
Stacking (of 0-R Classifier)	14.74%
Random	6.67%

Table 2: Correctly Classified Instances using the PEBL framework

Classifier	Testing Data	Testing: Pos	Testing: Neg	Training: Pos
	N = 221	N = 58	N = 163	N = 559
SVM (Linear Kernel)	86.42%	67.24%	93.13%	78.71%
LogitBoost (Dec. Stumps)	78.28%	24.14%	97.5%	25.04%

I used the LogitBoost [4] algorithm instead of the typical boosting algorithm, Adaboost, because it is more robust to outliers or mislabeled data. The robustness comes from the binomial log-likelihood loss function it uses instead of an exponential one. Since this manually labeled data is prone to errors, this algorithm is more fitting. Many webpages also cannot truly be put into only one category, so they are bound to be somewhat mislabeled.

4 PEBL: Single-class Classification

Another approach to classification is using single-class classifiers. The advantage of these classifiers is that they can be built specifically for any category. All that is needed to train a single-class classifier is positive and negative examples. However it is generally hard to produce negative examples. For example, what is a "non-sports" page. Is it a reference page, a shopping page, or a fashion page? It could be all of them and many more. The negative examples have to be representative of the entire spectrum of "non-sports" pages which is hard to do in practice. Therefore Positive Example Based Learning (PEBL) has been developed to create single-class classifiers with only positive examples and a set representative of the whole web [7]. PEBL uses a mapping-convergence algorithm to create the negative examples. The idea is to extract complement of the positive examples from the universal set.

4.1 The Mapping-Convergence Algorithm

There are two stages in this algorithm: the mapping stage and the convergence stage. In the mapping stage the algorithm uses a weak classifier to generate "strong negatives" based on the positive examples. We look at what features occur in the positive data and any of the unlabeled data without these features is labeled as negative. In the convergence step a margin maximizing classifier, such as SVM, is used to classify the unlabeled data given the positive and negative sets. Anything that is labeled negative is then included in the negative set and the algorithm iterates until the negative set converges. It therefore classifies as much as it can as negative without crossing into the positive space.

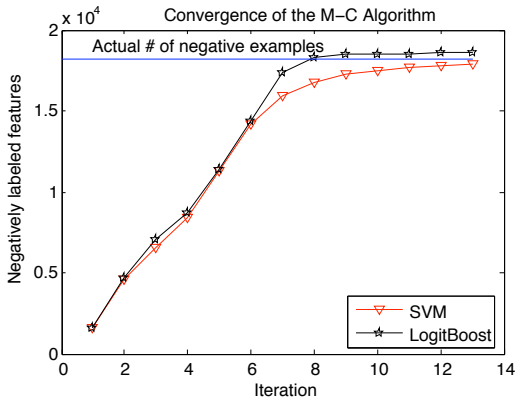

Figure 3: Two algorithms were used in the convergence step of building the PEBL classifier. The graph shows the progress towards convergence and the increasing size of the negatively labeled set.

4.2 PEBL Classification Results

I implemented the PEBL algorithm using SVM as stated in the PEBL paper [7] as well as LogitBoost for comparison. I chose LogitBoost with Decision Stumps because it performed the best in the multi-class experiment. Figure 3 shows the progress towards convergence for each of the algorithms. In the unlabeled data of size 18819, there were a total of 664 positive examples and therefore 18155 negative examples. The algorithm should converge close to the correct number of negative features. SVM does the best because it gets close to the actual number of negative examples, but does not overshoot. The LogitBoost algorithm classified too many examples as negative, thus mislabeling some positive examples. I conjecture that this is because LogitBoost does not maximize the minimum margin where as SVM does since the support vectors are defined to be the minimum margin. However boosting could still be applied to this problem. The algorithm would just have to be changed to account for the minimum margin. In [6], AdaBoost has been modified to maximize the minimum margin just as needed for PEBL. The results from classification are summarized in Table 2. It can be seen that SVM performed better, with more correctly classified websites in the new test data. It can be seen that LogitBoost classified too many websites as positive by the poor performance in the positive training data. However booth algorithms performed much better than in the multi-class classification.

5 Website - www.janulrich.info/web

I created a website to allow people to test the classifier themselves. The user can enter the URL of a website, and the server will then download the given website's content, extract the text from the content, and then create a frequency count based on the features I have selected for my classifier. Both the multi-class classifier and the PEBL classifier can be tested on the website. For the multi-class classifier, the LogitBoost algorithm I have trained before will then classify the given webpage into one of 15 categories. The PEBL classifier is a single-class classifier and has been trained on the class of webpages related to computers. Therefore it will predict whether the website is related to computers or not.
This website provides people with the opportunity to see the results of my work themselves and also to inspire them to think of new uses for automatic webpage classification.

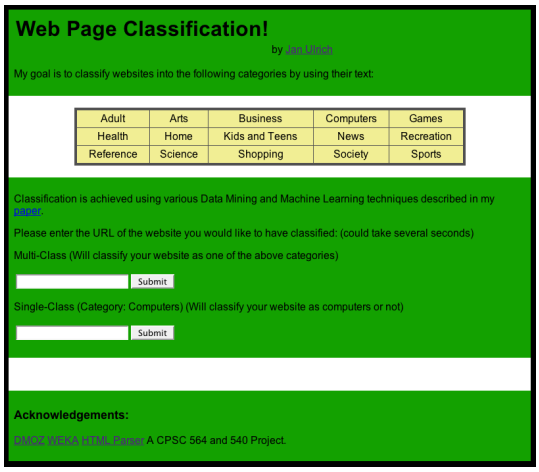

Figure 4: http://www.janulrich.info/web

6 Conclusion

I successfully developed a web page classification scheme for both single-class and multi-class classifiers. Both types of classifiers are the best in different cases. The single-class classifier should be used when building a classification for something specific. The PEBL algorithm makes it easier to build a classifier because it only needs positive examples. The multi-class classifier is useful when there is a standard set of categories and the goal is just to find the best match.
Logitboost was slightly better than SVM as a multi-class classifier. It classified just over half of the webpages correctly which is much better than 6.67% of the webpages with random classification. In single-class classification, I was able to get much better results for the one class. For PEBL, SVM was the appropriate algorithm. It is possible to use boosting as long as the algorithm is modified to maximize the minimum margin. The PEBL classifier did take a very long time to build. The computation required for each model should be considered when deciding between a single-class and a multi-class classifier.

References

[1] Marcel Dettling and Peter Buehlmann. Boosting for tumor classification with gene expression data. *Bioinformatics Journal*, 2003.

[2] Susan Dumais and Hao Chen. Hierarchical classification of web content. *Proc. Eighth Int'l Conf. Knowledge Discovery and Data Mining (KDD'02)*, 2002.

[3] Yoav Freund and Robert Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings to the Thirteenth International Conference*, 1996.

[4] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Ann. Stat.*, 2000.

[5] The University of Waikato. Weka 3: Data mining software in java. `http://www.cs.waikato.ac.nz/ml/weka/`.

[6] Gunnar Raetsch and Manfred Warmuth. Efficient margin maximizing with boosting. *Journal of Machine Learning Research*, 2005.

[7] Hwanjo Yu, Jiawei Han, and Kevin Chen-Chuan Chang. Pebl: Web page classification without negative examples. *IEEE Trans. on Knowledge and Data Engineering*, 2004.