Building a Dynamic Classifier for Large Text Data Collections

Pavel Kalinov **Bela Stantic** **Abdul Sattar**

Institute for Integrated and Intelligent Systems
Griffith University,
Brisbane, Australia,
Email: {P.Kalinov, B.Stantic, A.Sattar} @griffith.edu.au

Abstract

Due to the lack of in-built tools to navigate the web, people have to use external solutions to find information. The most popular of these are search engines and web directories. Search engines allow users to *locate* specific information about a particular topic, whereas web directories facilitate *exploration* over a wider topic. In the recent past, statistical machine learning methods have been successfully exploited in search engines. Web directories remained in their primitive state, which resulted in their decline. Exploration however is a task which answers a different information need of the user and should not be neglected. Web directories should provide a user experience of the same quality as search engines. Their development by machine learning methods however is hindered by the noisy nature of the web, which makes text classifiers unreliable when applied to web data. In this paper we propose Stochastic Prior Distribution Adjustment (SPDA) - a variation of the Multinomial Naïve Bayes (MNB) classifier which makes it more suitable to classify real-world data. By stochastically adjusting class prior distributions we achieve a better overall success rate, but more importantly we also significantly improve error distribution across classes, making the classifier equally reliable for all classes and therefore more usable.

Keywords: web directory; categorization; classification; MNB

1 Introduction and Motivation

Search engines are currently the main way for users to find information on the web, therefore the main source referring visitors to web sites. There are two sets of problems related to the dominant search engine model: problems for users and problems for the web itself. From the users' point of view, the main issue stems from the *keyword search paradigm*, which relies on the following assumptions:

- **Users know what they are looking for.** While this is often true, it is not the majority of cases (Yoshida et al. 2007). Many people are looking for something they have a vague idea about. This type of search for background information generates more search queries than those for specific information. Since people don't know beforehand what exactly they want, they cannot define it by using a list of search terms so they issue *navigational queries* ((Broder 2002, Lee et al. 2004)), essentially trying to use the search engine as a web directory.
- **Users know how to find it.** This assumption also doesn't always hold. Even if users know what they want, they may not necessarily be able to formulate it in terms of keywords. It is a paradoxical situation - in order to find a document, you have to know some words it contains; in order to know them, you must have already read the document (or similar ones), i.e. you must have already found it by some other means. This reveals the underlying assumption that the user should have prior knowledge which cannot have come from the search engine. The situation was best formulated by the science fiction writer Robert Sheckley in *Ask a foolish question*: "in order to ask a question you must already know most of the answer". Additionally, there are the *synonymy* and *polysemy* issues (Deerwester et al. 1990), namely - that different people refer to the same concepts in different ways, and that the same word may mean different things to different people (or, even more confusingly, to the same person in different contexts).

Another side of the problem arises from the second aspect mentioned above - that search engines have become the main source of traffic to web sites. This has had an effect "not unlike the Heisenberg Uncertainty Principle [...] The act of Google trying to understand the web caused the web itself to change." (Zawodny 2003). This change has created an economy where a significant part of web content is now published for the benefit of search engines and not human users; content is published for the purposes of *Search Engine Optimization*, which essentially means trying to exploit search engine algorithms by providing content "they like", to the detriment of human users. Search engine spam has thus become a major source of digital garbage (Fetterly et al. 2003, Gyöngyi & Garcia-Molina 2005, Gyöngyi et al. 2006); as a consequence, a large part of development efforts are now targeted at *adversarial information retrieval* (Fetterly 2007).

The proliferation of this class of sites can be directly attributed to weaknesses of the search engine model, and an alternative model should be developed to diminish its impact.

An alternative model should also exist which does not depend on users' prior knowledge and ability to formulate their queries with specific words. Such is the web directory model which allows *discovery* of documents, as opposed to *locating* them by a search engine.

The main drawback of the existing web directory model is its reliance on manual labour for both information gathering and classification. The fact that web directories have not been combined with web spiders has also created an issue specific to them - a problem with outdated information (sites that were listed once, then not re-checked and updated). This form of *web decay* has significant impact on directories such as Yahoo! (Bar-Yossef et al. 2004) (also confirmed by our experiments).

There are a number of machine learning methods used for text classification, which is the basis needed for automatically building a web directory. An example is the *Multinomial Naïve Bayesian* (MNB) classifier, a simple probabilistic model which classifies data instances into multiple classes using Bayesian statistics and relying on the *independent feature model* (Frank & Bouckaert 2006). However, this method works well only for simple cases, while for unbalanced classes it is not accurate. Another method of text classification is *Support Vector Machines* (SVM); Liu et al. (2005) made a large-scale study using SVM over data from the Yahoo! directory and found performance "far from satisfactory", due to noisy and sparse data.

Statistical methods in general usually suffer from unbalanced classes and give preference to classes with more instances. This is particularly detrimental for web directories, since they are very unbalanced as content. Furthermore, an unbalanced end result is disproportionally perceived as useless by users, e.g. - if the classifier makes an error in 5% of instances, but they are all in a particular category and as a result this category remains empty, users would declare the algorithm totally unsuccessful and not just 5% unsuccessful.

In this study, we aim to build a viable web directory by improving existing classification methods, which suffer from unbalanced classes. We build on top of the MNB classifier and propose an addition to it by adding a stochastic adjustment of class prior distributions, which enables the modified algorithm to learn from its mistakes. While the classic algorithm and its many variants are static (i.e. - they will return the same values at every pass over the document collection), our method shows an improvement with successive passes over data. Furthermore, while the classic algorithm suffers from unbalanced classes and shows preference for the dominant class and very high error rates in classes with less instances, our method achieves an even distribution of the error rate, making it equally reliable for any class.

The remainder of the paper is organised as follows: in the next section we review the relevant prior work. In section 3 we outline our methodology. Section 4 describes the experiments we conducted. In section 5 we show and comment experimental results. Finally in section 6 we conclude the paper and outline the course for future work.

2 Relevant Work

There has been extensive research on the use of classification methods for web documents: (Lang 1995, Chakrabarti, Dom, Agrawal & Raghavan 1998, Xue et al. 2008); some systems have later been employed in real-world web directories (Attardi et al. 1999), but none persisted or are in use by major directories at present.

2.1 Data Processing

Text classification is usually based on the *bag of words* model (Baeza-Yates & Ribeiro-Neto 1999) where a document is considered a collection of non-dependent words (or n-grams) and is analyzed based on statistics such as their occurrence in documents of different classes. Zipf's law (according to which a large number of words appear in a text only a few times, while a few words occur orders of magnitude more often (Zipf 1949)) is ignored, which for a number of reasons does not render the algorithms ineffective.

Before starting a learning process, any algorithm needs to first obtain the data. Document text is usually parsed by a *tokenizer*, which splits it into words and normalizes them according to some rules (for example forcing lower case only, so that *Word* becomes the same as *word*). Some algorithms use these tokens directly, while others construct *n-grams* - a series of n tokens. This preserves some information about word order in texts, but requires much more storage and computation. Another alternative is to build Markov models; in them word order is not just preserved, but is in fact more important than the actual words used. Unfortunately, this approach provides better success rate at the cost of exponential increase in processing time (every document classified has to be matched to every existing Markov model in the training set), making it impractical to use for a large-scale classifier.

After tokenizing, text is sometimes passed through linguistic processing; for example words may be *stemmed* - words with a common root are combined into one, so that *Word* becomes the same as *word*, *wording* and *worded*.

Word counts may be scaled or normalized in some manner, for example, by their *Inverse Document Frequency*: $IDF(w) = \ln \frac{N}{df_w}$ where N is the number of documents in the collection, and df_w is the number of documents that contain the term w. This weight is then multiplied by the frequency of the term (TF) in the document to achieve its TF-IDF value.

A major issue with text classification is the high dimensionality of data. In a simple model, such as Naïve Bayes, every word is a *feature* itself. The typical representation of a document as a vector where each dimension is a word in the vocabulary means that the number of dimensions is equal to the number of words in the vocabulary. Reducing this number would increase the tractability of all related tasks. Therefore some other models apply feature selection or feature construction to reduce the number of features - *dimensionality reduction*, which is a form of lossy compression where precision is traded for computation costs. It is guaranteed to lose accuracy (Lang 1995), but in most cases compensates this by a great reduction in computation time.

Dimensionality reduction by feature construction can be achieved in a number of ways, such as unsupervised clustering like k-means performed over the dictionary of the document collection, *Latent Semantic Indexing* (LSI) (Deerwester et al. 1990), *Semantic Hashing* (Salakhutdinov & Hinton 2007) or a random projection of the high-dimension word vector onto a much lower-dimensional space (Kaski 1998).

2.2 Approaches to Classification

The MNB classifier is a simple probabilistic model which classifies data instances into multiple classes using Bayesian statistics and relies on the *independent feature model* - i.e., it assumes word occurrences are independent of each other. This violates Zipf's law (actually, texts on the web were found to follow a double Pareto distribution (Chierichetti et al. 2009)) and independence is never the case, but the model works surprisingly well even though it is very inaccurate in estimating correct probabilities. As Domingos & Pazzani (1997) point out, the algorithm needs to only find out the class with highest probability and not what this probability actually is and how exactly it compares to those of the other classes. The winning class (the one with the *maximum likelihood*) is found as:

$$\underset{c}{\operatorname{argmax}}\ p(C=c) = \ln \frac{p(C|D)}{p(\neg C|D)} = \ln \frac{p(C)}{p(\neg C)} + \sum_i \ln \frac{p(w_i|C)}{p(w_i|\neg C)} \quad (1)$$

where C is the class the instance belongs to, c is each class, $p(C|D)$ is the prior probability of a document for a class and $p(w_i|C)$ is the prior probability of word i for that class, i including all words in the document.

This works for simple cases, but unbalanced classes are a serious problem: the classifier tends to favour incorrectly the larger classes since they have a high prior probability. The naïve assumption is also a problem where classes are not only unbalanced as numbers (i.e. - one class has many more instances than another) but unbalanced as content. If documents in one class are typically longer than documents in the other classes, then word occurrence rates in it would be higher and this class would be favoured incorrectly as well. As a solution, Frank & Bouckaert (2006) propose to normalize word probabilities:

$$n'_{wd} = \alpha \times \frac{n_w d}{\sum_{w'} \sum_{d \in D_c} n_{w'd}} \quad (2)$$

where $n_{w'd}$ are class-specific word counts (occurrences of the word in documents of class c), replacing w_i in *Eq.1.* They find that α can be equal to 1, so in our experiments we followed that advice (see Algorithm 1).

Algorithm 1: MNB with word count normalization

```
Calculate prior distributions over whole collection
Calculate normalization based on word occurrences in each class
ClassifyQueue
foreach document do
    foreach class do
        Calculate log-likelihoods of words in document
        Apply normalization
        Add global log-likelihood of class
    end
    Find class with highest probability
    Announce winner class
end
```

Rennie et al. (2003) propose some steps to overcome systemic errors in the MNB model. They introduce Complement Naïve Bayes (CNB) to overcome skewed training due to skewed classes (training the algorithm to distinguish a class on examples *not* in that class, as opposed to the normal practice of using instances *in* that class), and weight normalization to compensate for cases where the term independence assumption of the model doesn't hold.

Rocchio's *Relevance Feedback* (Rocchio 1971) can also be used for classification by creating a representative document vector based on all documents in a class; each instance is classified into the class with which it has the smallest cosine distance.

Support Vector Machines (SVM) in effect build as many classifiers as there are classes, each one separating the class from all others. Each instance is evaluated by every classifier, then the decision is made by the one with highest confidence. Liu et al. (2005) used SVM over the Yahoo! directory and found the results "far from satisfactory", due to the same problems that we faced - noise and sparse data.

All hierarchical classifiers have a common shortcoming - if they make an error on a document at one of the higher levels, this error is then propagated down the hierarchy. Xue et al. (2008) address this by dynamically training a specialized classifier for each document in the collection, using a subset of existing categories as training data. This allowed their classifier to maintain an acceptable accuracy down to Level 5 of the classification tree. They used MNB since its accuracy is comparable to that of SVM, but has lower complexity.

Most methods treat classification as a "one off" task: they take a document collection, process it and finish. For a web directory this cannot be a solution, since it updates its document collection constantly (adds, edits and removes documents from it), as well as evolves the classification structure. Furthermore, classification of documents changes with time - sometimes instances are moved from one class to another by editors not because they were wrongly classified initially, but because the perception of what they belong to may have changed (i.e. - "Hilton" used to be a hotel chain and was associated with "Business" but is now associated with "Paris" and goes to "Entertainment" or "Junk" classes). This question of ontology evolution has been studied in terms of creating different ontologies for different periods and then comparing them (Enkhsaikhan et al. 2007); however, this does not facilitate a gradually changing system such as a web directory. An online (incremental) indexing method has been proposed: (Gorrell & Webb 2005, Gorrell 2006), the results of which converge to those of LSI but process only one instance at a time, i.e. - it accounts for streaming addition of documents to the collection, but doesn't offer a solution to documents being removed from the corpus, or evolving classes. Katakis et al. (2005) propose an algorithm which is incremental in both respects - it couples an incremental feature ranking method with an incremental learning algorithm that can consider different subsets of the feature vector during prediction; for evolving classes though it would need to be retrained from scratch.

2.3 Approaches to Training

Some classifiers use different heuristic criteria to minimize training by selecting only a small subset of the data to train on. It is worthwhile to see what approaches spam filters use for learning, since they also use a Bayesian filter (though not multinomial). They employ several different training strategies (Zdziarski 2005):

- *TEFT* (train-everything) is the classic approach of all text classifiers, including Bayesian, and is also the most intuitive - learn from all the data. However, it is computationally expensive and, as our experiments show, does not always provide the best results. According to Zdziarski, the experience of spam filters using this approach is that it suffers from unbalanced classes (where spam is much more than non-spam) and only works well if the ratio is not worse than 70:30. This is not the case in our experiment (see data below).
- *TOE* (train-on-error) - the algorithm runs the classifier part first, then compares the result to a (manual) label for the instance and learns from it only if it has made an error (the so-called *if it isn't broken, don't fix it* philosophy). Such filters

are more “static” than TEFT filters, i.e. - they take longer to learn. On the other hand, they work better for large datasets and for highly unbalanced classes, which matches our case.

- *TUM* (train-until-mature) filters try to be the middle ground between TEFT and TOE - initially they learn on everything, then they stop learning and only retrain when they make a mistake. As with the others, they need labeled data in order to recognize a mistake, plus some heuristics to decide when a token is *mature* so as to stop learning it.
- *TUNE* (train-until-no-errors) algorithms learn until they make no mistakes, or very few mistakes. The downside is that when they start making new mistakes, they have to be retrained over the whole document corpus.

2.4 Alternatives

An interesting alternative to analyzing the documents themselves is the work of Chakrabarti, Dom & Indyk (1998) who propose a method to extend a human-classified directory by applying its manual labeling to *neighbouring* unlabeled sites, where they define “neighbours” in terms of outgoing links and use these links as features. The *naïve* version of their algorithm propagates classification to all unlabeled data. This was found to work well in some restricted domains, but its performance on data from the Yahoo! web directory was extremely poor which they attributed to the generally noisy nature of the web. By iterative application of the algorithm and some engineering of features, they managed to outperform classifiers based on text classification only. Gyöngyi et al. (2006–2007) developed the idea further, but found the method extremely computationally expensive which forced them to use a host-to-host connection graph instead of page-to-page, introducing significant errors. They also found that while the method works well in some restricted domains, in others it doesn’t work at all.

Some other alternative methods worth mentioning are social bookmarking (Yanbe et al. 2007) and collaborative filters. Social bookmarking relies on people labeling all documents, while collaborative filters are trained by many people then used by an individual user, such as news.google.com (Das et al. 2007). Another interesting development is WEBSOM (Lagus et al. 2004), based on a Kohonen neural network called SOM (Self-Organizing Map) (Kohonen 1995). It is not classification but a projection of data (such as a number of documents) onto a map. Similar documents end up geographically near each other on the map, facilitating browsing by analogy. The method is computationally expensive though, and the resulting map questionable in terms of usability: to achieve good resolution for browsing, the number of map units has to be proportional to the number of documents. This leads to maps with millions of points which are not easy to navigate, defeating the purpose.

3 Methodology

We aim to create the basis for a large hierarchical classifier which can accommodate a web directory comparable to the Open Directory Project (DMOZ - *www.dmoz.org*), where we use the DMOZ entries as labeled instances for training and will later complement them with unlabeled data downloaded by a web spider. The method should take into account the addition and deletion of documents which in a live directory combined with a web spider would be continuous, as well as occasional relabeling of instances by a human. For our tests though, we work with static data.

DMOZ has a directory tree with over 763,000 nodes, so to duplicate it we need a hierarchical classifier with as many classes. For our prototype, we built only the first level, which has 15 categories. The development of the top level of the hierarchy should equip us with the necessary techniques for all the lower levels, since by definition it is the worst case as it contains all the lower levels in itself. We decided to use a Multinomial Naïve Bayes (MNB) classifier, because it can work well incrementally, e.g. - small changes to the document collection or the reclassification of a small number of documents would not require full re-training. We tested the “classic” MNB, as well as several variations.

3.1 Dataset, Processing and Systemic Problems

For training/test data we used a complete data dump from DMOZ, which contained 4.16 mln web site records and 4.60 mln manual labels (some sites were classified into more than one class). We downloaded a random sample of 120 788 documents (excluding errors) from a total of 2.24 mln in the English language categories only (to minimize the dictionary).

We did not use the DMOZ descriptions for classification, but the actual text from the downloaded sites. All documents were passed through a filter which stripped HTML tags, then discarded very short and very long phrases (e.g. - shorter than 3 words and longer than 20 words) on the basis that a) one or two words are obviously not part of a sentence so are not part of any sensible text (this removes all site navigation and other clutter), while more than 20 words without punctuation indicate a machine-generated list of keywords and not a human-written text.

Words were not stemmed or pre-processed in any other manner. Arnaud (2004) attributes the poor results of a project trying to create a browsable web index based on Self Organizing Maps to the use of a bad text pre-processor. This may be a case of just selecting the wrong pre-processor; or perhaps it is a fundamental problem: pre-processing creates some bias in the data which then reflects on the ordering algorithm. Furthermore, linguistic processing has to know the structure of the language, which in our case would mean a different pre-processor for every different language sub-tree of the directory (currently numbering 81). We decided to skip it, incurring higher computational costs. We only filtered out too short and too long words (less than 3 and more than 20 characters long), and normalized word counts by TF-IDF.

Since the data is extremely high-dimensional, the usual approach would be to apply dimensionality reduction. However, we decided against this and worked with the raw data. The reasoning was that any initially successful projection of the documents into a lower-dimension representation would deteriorate as we alter the collection, since *success* is being measured by information entropy over the collection; as the collection changes, that would change too. In other words, if we achieve a set of constructed features which best split the data, this “best split” is only guaranteed to be best for the initial data. If we then remove, for the sake of argument, all documents containing the terms which constitute one of these features, we would get a feature with a probability of zero. Evolving the document collection would mean for us that a) we need to update the dictionary constantly, b) we need to perform dimensionality reduction again and again, c) as a result, documents will have to be re-mapped to the new low-dimension vectors constantly, and d) the classifier will need to be retrained to the new features with every change. While this is not impossible to do, it adds another level of complexity to the system and doesn't seem to save too much in the way of computation, so the precision/cost trade-off doesn't seem justified. Moreover, dimensionality reduction would project the data into (typically) 64 or 16 dimensions, and since we only have 15 classes we might as well project the data into them without this intermediary. We also have to keep in mind that at lower levels of the classifier we would need separate dimensionality reduction as the document collections there would be different.

We treat the hierarchical classifier as a hierarchically ordered series of normal classifiers where the output of one classifier is the input for those at the lower level. Each classifier splits the data instances into a number of classes, which the lower-level classifiers process further using their own training data. If the higher-level classifier moves an instance from one class to another at a later point in time, this instance gets removed from the document collection of the respective lower-level classifier and put into another one. Since each lower-level classifier works with only a part of the data, it calculates all static measures (such as TF-IDF values for words) locally - i.e., taking into account only its own part of the document collection. This means there can be no global stop-words, because they are stop words in some context only. The usual example for such a word is the article "the", which is so common in the English language that it cannot be used to distinguish between different types of texts. But, it shows that the text is in English in the first place - so it is very useful to distinguish between English and Bulgarian texts, for example (i.e. - it is a valuable feature at the top level of the classifier).

We also decided to ignore *hapaxes* (Zdziarski 2005) - tokens with very low confidence, which we defined as words that occur in less than 10 documents.

Document numbers above may seem sufficient for learning but, as Liu et al. (2005) noted, data is in fact very sparse - 76% of the categories of Yahoo! Directory when they studied it contained less than 5 documents. Similarly, DMOZ has an average of 6.02 labels per category. This is due to a very fragmented categorization structure where many nodes are either too specific and have almost no content that matches them, or are placeholders (Liu et all call them *conceptual nodes*) serving only to organize lower-level branches, with no content of their own. An omission by design (of DMOZ, Yahoo! and all similar directories) is that sites are supposed to be indexed only in one node of the directory tree - e.g., if a site is classified in the *Business: Investing: Exchanges* subcategory, it is not listed in the higher levels, i.e. *Business* and *Business: Investing*, nor in any lower levels. This makes the above situation even worse, since even high-level nodes such as *Business* are practically empty. In our implementation we did the same as Liu et al. (2005) - we "folded" high level categories by including in them the content of their sub-categories.

As regards classification duplication, the number of labels per URL in DMOZ is 1.02, as opposed to 2.23 in Yahoo!, so the level of classification noise is much lower. We have ignored this (although it harms the classifier), but in a future implementation it should be accepted as additional data and not noise, which would mean using a fuzzy classifier.

Dimensionality of data is another problem. In theory, the document collection should only contain words from a limited dictionary (provided we only deal with documents in one language). In practice though, even when we apply ourselves to the English-language part of the directory only, there are many occurrences of names of people or places, foreign language words inserted into English texts, foreign language sites wrongly categorized in an English-language node, typos, deliberate attempts at filter poisoning (or *Bayesian poisoning* (Graham-Cumming 2006, Zdziarski 2005)) like a large collection of "words" like "btsnwgdguf", bringing the overall dictionary to over half a million words for our (rather small) sample of documents. Our full dictionary had 593,718 words, 388,329 of which with a DF of one (i.e. - they were seen in only one document). For comparison, the *20 newsgroups* document collection usually used for benchmarking has a dictionary of slightly more than 61,000 words. Ignoring words that occur in less than 10 documents brought the dictionary used for training to a more manageable 62,492 words (about 2% less in the case of train-on-error variations).

The most significant problem though is that classes are highly unbalanced in a number of ways.

Firstly, they are unbalanced as number of instances they contain: the largest top-level English-language category ("Regional") contains 1.10 mln instances, while the smallest ("News") has less than 9 thousand. Thus, the dominant category has 42.4% of all instances, with the remaining instances spread in 14 categories.

Secondly, classes are unbalanced as average length of the documents they contain - for example, documents in the "Business" category of our sample had an average of 96.54 unique terms each (after filtering), while the "Adult" category (rather surprisingly) had more - 121.76, and "News" (not so surprisingly) had 235.05.

Furthermore, as mentioned above, the MNB model's assumed term independence doesn't always hold - some terms have a strong correlation, like "San" and "Francisco" in texts on American cities (Rennie et al. 2003), creating some bias in the algorithm. If this bias applied to terms evenly distributed across classes, it would just lower the success rate in general. What happens in reality though is that different classes violate the independence assumption to different degrees (the third type of unbalancing); word count normalization doesn't compensate for this, since it accounts for length of documents only and not for the actual terms they contain.

Classes are unbalanced also by the fact that the "Regional" category contains practically a bit of everything - local business, local entertainment, local news etc. so its keywords to a large extent coincide with those from all other classes, unlike for example the "Adult" or "Business" class which have more distinctive dictionaries. A high word occurrence rate for common terms, coupled with the high prior probability of the class leads to a situation where the "classic" MNB predicts the dominant class for almost every instance and has a success rate of ≈ 1 for that class and 0 for some others.

The combination of all these factors results in significant variation in classification success from class to class. Each of them is usually "tackled" by a separate normalization, but the interaction between them is not well studied. We decided to work from the opposite end and compensate not for each factor separately, but for their cumulative effect as measured by the class-specific error rate.

3.2 Algorithm Variations

The changes we introduced include:

- *Weight decay* of learned word weights: words with a word count of 1 are removed from our training data after each pass, and counts for the remaining words are divided by 2. In this way we can remove weights for words that have not been used for some time (e.g. - when the documents that contain them were removed from a node or from the collection altogether). This measure is introduced to account for a gradually changing document collection being re-classified constantly, such as a live web directory.
- We calculate *prior distribution* differently and use stochastically adjusted values for class-specific error distribution smoothing.

The second item needs explanation in more detail, since it is the core of the novelty we propose.

Prior distribution in principle should be the distribution of documents into the respective classes, where "documents" is taken to mean all documents in the collection. Spam filters though, as discussed above, employ a *train-on-error* policy where they train only on those documents they cannot classify correctly. In effect, the classifier only sees a part of the collection and bases its statistics on it only, so the prior distribution it uses is not the distribution over the whole collection but over the errors. Since classes are very unbalanced and this approach unbalances them even further, they try to compensate that by a heuristic giving a *false positive* error more weight than a *false negative* error. This heuristic is parameter-based and there is no methodology to calculate this parameter - it is based on experiments or personal preferences (where a spam filter can be tuned by a person). This parameter is a pre-set value and does not change in the life of the filter, irrespective of how it affects its efficiency or what data it processes.

Furthermore, Zdziarski's assertion that these filters are "static" means not only that they learn slowly, but also that they unlearn slowly - for example, they will recover very slowly after a Bayesian poisoning attack. In our case such attacks are not a realistic consideration, but re-classifying documents by human editors, which is a daily occurrence, has the same effect on the classifier (it has learned some classification of a number of terms, then has to unlearn it). The junk filter in the *Thunderbird 3* mail client (part of the Mozilla suite) has a partial solution to the problem in the form of weight decay (the same as we use). However, it is triggered by a rather arbitrary event - the dictionary reaching a particular size, which is a) an arbitrarily-set parameter, and b) not guaranteed to happen at all (e.g. - if the document collection is from a restricted domain with a small dictionary).

Initially we tried this approach by running the classifier in several passes over the collection with a *train-on-error* policy, triggering weight decay and re-calculating prior distributions at the end of each pass. This lead to significant fluctuations in performance - success rate dropped sharply after the first update, owing to the fact that the classifier had done very well predicting instances from the most populous class, hence it had not trained on it too often and the prior distribution and word counts for this class suddenly became too low when we initialized it for the next pass. On the next update it became too dominant again, since during this pass the algorithm had made too many errors on it. The end result was a "seesaw" between these two states.

We then decided to use a constantly updated rolling count of errors, which exhibited the best results among all the algorithms we tried.

There are three sets of data used to classify an instance: word count distribution by class (document *evidence*), *prior distribution* of classes and *word count normalization* by class. The first we did not try to manipulate: we used a word count of all errors (for all time), decaying after each pass over the whole collection. Changing that would require an enormous complexity of the database, since we would have to record not only word counts, but when each individual incrementation of the counter happened so that we could undo it later.

The other two measures we redefined as *based on the last 10 000 trainings of the classifier*, e.g. - we calculate class prior distribution over the last 10 000 errors, and normalize word counts based on the word counts of the last 10 000 documents on which the classifier made an error (see Algorithm 2).

Algorithm 2: MNB with distributions over last N errors

```
Calculate normalization based on word
occurrences in each class
ClassifyQueue
foreach document do
    Calculate prior distributions over last $N$
    classification errors (from error log)
    foreach class do
        Calculate log-likelihoods of words in
        document
        Apply normalization
        Add current log-likelihood of class
    end
    Find class with highest probability
    Announce winner class
    Compare with manual label
    if winner class and manual label are
    different then
        Log error
    end
end
```

In effect, we use statistics from a non-random, stochastically selected sample of the data in order to calculate the static measures needed by the classifier, making them dynamic measures.

4 Experimental Study

In order to test the advantages of our approach we conducted extensive experimental evaluation and compared our method with the classic MNB, as well as a *train-on-error classifier* as used by spam filters, and one with word count normalization as in (Frank & Bouckaert 2006). We tried the classifiers on the filtered text from sites we downloaded and compared results to the manual labels from DMOZ. We did a number of test runs, each with several passes over the data (results reported here are from a typical run).

All algorithms used the same filtered data, with word counts normalized by TF-IDF:

$$\textit{TF-IDF}_w = \frac{n_{i,j}}{\sum_k n_{k,j}} \ln \frac{N}{df_w}$$

where $n_{i,j}$ is the number of occurrences of the term t_i in document d_j, the denominator is the sum of the number of occurrences of all terms in document d_j, N is the number of documents in the collection and df_w is the number of documents that contain the term w.

4.1 Query Set

We designed the experiments with view to compare our stochastic approach to other variations of the MNB classifier in terms of:

- Overall accuracy (success rate of the classifier)

- Class-specific accuracy (separate success rate for each class)
- Computation cost for classification
- Computation cost for training
- Storage requirements for training data

A benchmarking mechanism was implemented to record classification results for each algorithm, classification and training times (in microseconds) and dictionary size of each algorithm's training set. To avoid bias from a fixed ordering of algorithms (file cache, database cache etc.) they were called in random order when classifying each instance. Preparatory operations (reading the document, fetching IDF values etc.) were common for all algorithms and were not taken into account.

4.2 Noise in the Data

The first issue we faced was the significant level of noise of several types in the data, each of which has different impact and has to be dealt with differently:

- Dead links - more than 8% of listed sites returned a 404 HTTP response code(*Document not found* error) on download, or were unavailable (connection timeout). These we removed automatically.
- Web decay - sites that have been removed but generate *soft 404 errors* and now serve other content, not what was categorized by DMOZ. They do not issue an error response code and are the most difficult to remove, as they require a human's decision for each instance. They harm the classifier by making it learn wrong classifications.
- Intentional noise - legitimately-looking sites that embed noise within their own code (comment spam, hidden text due to using *search engine optimization* techniques etc.). Same as above, only more difficult to spot and remove.
- Entry pages - sites that have only a logo and an "Enter" link on their homepage which was indexed by DMOZ, but has no usable text to classify. They add noise to the classifier's training phase and reduce its success rate at the classification phase significantly.
- Classification duplication - although contrary to DMOZ policy, many sites have been indexed in more than one category.
- Wrong classification - some sites have been classified into a category not suitable for them. This one is highly subjective and the initial intuition was to just ignore it, but it turned out to present the most serious problem of all since it is the reason for the dominance of the "Regional" category (which category, in our opinion, should not exist at all - it should be a separate hierarchy).

4.3 Variations Tested

We tested the following variations of the classifier:

- **MNB** Multinomial Naïve Bayesian classification by the classic formula.

  Classifier: MNB as in Equation 1.

  Training: save data for all words for all documents.
- **MNB-TOE** Multinomial Naïve Bayesian classification by the classic formula, with *train on error* policy.

  Classifier: MNB as in Equation 1, using prior distribution over the whole collection.

  Training: save data only for those documents which the classifier did not classify correctly.
- **MNB-WN** Multinomial Naïve Bayesian classification by the classic formula, with normalized word counts.

  Classifier: As Equation 1, but word counts are additionally normalized as per Equation 2 - see Algorithm 1.

  Training: save data for all words for all documents.
- **MNB-SPDA** Multinomial Naïve Bayesian classification with *train on error* policy, using a prior distribution over the last N errors as in Algorithm 2. We shall call this "MNB with Stochastic Prior Distribution Adjustment".

  Classifier: As Equation 1, but word counts are additionally normalized as per Equation 2. Prior distribution is calculated over the last N documents on which the classifier made an error ($N = 10000$ for our experiment). Word counts are normalized over the whole collection.

  Training: save data only for those documents which the classifier did not classify correctly.
- **MNB-SPDWNAC** Multinomial Naïve Bayesian classification with *train on error* policy, using a stochastic prior distribution and with word count normalization as per Equation 2, both calculated over the last N errors as in Algorithm 2. We shall call this "MNB with Stochastic Prior Distribution and Word Normalization Adjustment - Corrected".

  Classifier: As Equation 1, but word counts are additionally normalized as per Equation 2. Both prior distribution and word normalization are calculated over the last N documents on which the classifier made an error ($N = 10000$ for our experiment) in 80% of cases. *Corrected* stands for an addition we made which was necessitated by some problems explained below; the correction was that for the other 20% of classified instances we applied usual word normalization and not the stochastic variant.

  Training: save data only for those documents which the classifier did not classify correctly.

For all algorithms, we applied weight decay at the end of each pass.

5 Results and Analysis

In this section we provide results obtained from our experiments and some analysis of our findings.

5.1 Success Rates

Apart from our two variations using a dynamic sample of the last 10,000 unsuccessfully classified documents, all algorithms are static - i.e., they use the whole document collection and produce the same results at each pass (variation between passes is within 0.01% due to randomized order of testing). Our initial intuition was that our variations would continuously improve, since the document sample they use is stochastically built and should work towards minimizing errors. This does not happen: after 3 or 4 passes they converge to a success rate which does not change significantly thereafter. Variation between passes is within 2.5% for MBN-SPDA - it consistently beats MNB-WN, but with a varying margin (the one reported here is an average-to-low value).

On further reflection, we saw the obvious - the stochastic adjustment does not work towards minimizing the error itself, but only minimizes its variance across classes. Overall success rates can be seen in *Table 1*.

You will note that we have not quoted MNB-SPDWNA results apart from its overall performance. It turned out that MNB-SPDWNA has an important drawback: the value for normalized class word counts is in the denominator part of Equation 2. If, for any reason, the algorithm makes no errors or very few errors in one class, the values for it become too low. Since we are dividing by them, we increase its score and the algorithm starts predicting this class for all instances.

The consequences are illustrated in Fig. 1, which shows the result of a bad initialization in a test run. We did not randomize the training sequence well enough and one of the classes was not represented in the first 100,000 trained instances. The algorithm made more than 10,000 errors in that time, which resulted in an adjusted prior distribution where this class was not present. Its word normalization values then became extremely large. Interestingly, for the next 30,000 instances the algorithm actually exhibited a better success rate than before, then it became drastically unsuccessful. The initial improvement was due to weight decay being applied, which also made the respective word counts much lower and lessened the effect of the wrong normalization. This also happened at the end of the next pass when weight decay was applied again, but to a lesser extent since weights were large already. It then converged to a somewhat higher (but still much worse than normal) success rate and remained there after successive passes, i.e. - it did not recover from the error. Unlike it, the MNB-WN variant suffered a short and sharp drop in performance which it quickly overcame with increasing word counts as more documents were classified. Later applications of weight decay, as expected, did not affect it.

We got similar outcomes on a number of other runs even with correct initialization, meaning that the MNB-SPDWNA (not corrected) algorithm in its pure form is too unreliable. There are two stable states to which it can converge: the optimal state with an even error distribution, which is a form of dynamic equilibrium that needs to be maintained constantly; and the worst case, where the last N errors do not include an error in one of the classes; in this case the algorithm predicts one class for all instances, thereby making no further errors in this class and entering a positive feedback loop from which it cannot recover.

Apparently, MNB-SPDWNA needs some form of correction. We tried to do stochastic word normalization only for 80% of classified instances and full normalization for the rest, in order to give the algorithm an opportunity to occasionally make errors in all classes and introduce some negative feedback. Unfortunately, this only partly mitigates the positive feedback effect and does not make the algorithm usable.

MNB	**TOE**	**NW**	**SPDA**	**SPDWNA**
47.91%	67.86%	69.44%	70.40%	42.84%

Table 1: Success rates

It can be seen that the "classic" MNB algorithm without any corrections also performs badly. It is interesting to note though how it fails: it's extremely good at guessing the dominant category and extremely bad at some of the others (with literally zero success rate for the "News" and "Reference" categories, and close to zero for "Business" and "Shopping") - see error distributions in *Table 3.* The only one performing worse was the earlier experiment we conducted with a train-on-error algorithm using a prior over errors updated once per iteration.

Train-on-error policy by itself improved the situation enormously - a jump from 47.91% to 67.86%. Word normalization as per (Frank & Bouckaert 2006) worked better - 69.44%, but the best results were exhibited by MNB-SPDA, which had a 70.40% success rate. This may not sound impressive if compared to results reported elsewhere, but we consider it a success given the enormously noisy data we worked with.

Category	**All**	**Stochastic**	**Boost**
Arts	9.51%	11.46%	+20.50%
Business	9.06%	9.52%	+5.08%
Computers	4.97%	5.94%	+19.52%
Games	2.20%	2.00%	-9.09%
Health	2.37%	2.74%	+15.61%
Home	1.25%	1.18%	-5.60%
News	0.25%	0.24%	-0.04%
Recreation	3.97%	4.14%	+4.28%
Reference	2.19%	2.23%	+1.83%
Regional	42.07%	35.59%	-15.40%
Science	4.61%	4.97%	+7.81%
Shopping	3.53%	4.14%	+17.28%
Society	8.60%	10.98%	+27.67%
Sports	3.85%	3.92%	+1.82%
Adult	1.58%	0.95%	-39.87%

Table 2: Stochastic adjustment to distribution

Prior distribution over the whole collection, and the values to which MNB-SPDA converged after 8 iterations, can be seen in *Table 2.* The last column provides an explanation of the improved results of the algorithm. What we see in it is an artificial "boosting" of categories with diluted content (e.g. "Society" and "Arts" which are difficult to define) where the classifier had problems, while clear-cut categories such as "Adult" and "Games" are corrected downwards since the algorithm can guess them anyway based on word distributions alone, no matter what the prior is. The "Regional" category was also adjusted downards, for the opposite reason - its high prior probability makes it easier to guess. The end result of the adjustment is a significantly higher success rate in the problematic categories at the expense of a slight worsening of results elsewhere. This provides a much smoother spread of the error across categories: while the standard algorithms are very good in the dominant category and very poor in some of the others, MNB-SPDA has a much smaller variation in its success rates (see *Table 3*).

For the purposes of building a web directory, we think it is much more valuable to have equal treatment of all categories (i.e. - equal error levels) than higher overall accuracy. Luckily though, MNB-SPDA provides both.

If we look at the results in terms of computation costs, it also saves significantly (more than four times) on training times. A comparison of total training times (normalized) can be seen in *Table 4*.

There is also a small saving in the training set: while the *TOE* algorithms used 61,255 words, MNB-SPDA used 59,832 to train its classifier. Both of these savings - in time and in storage requirements - are due

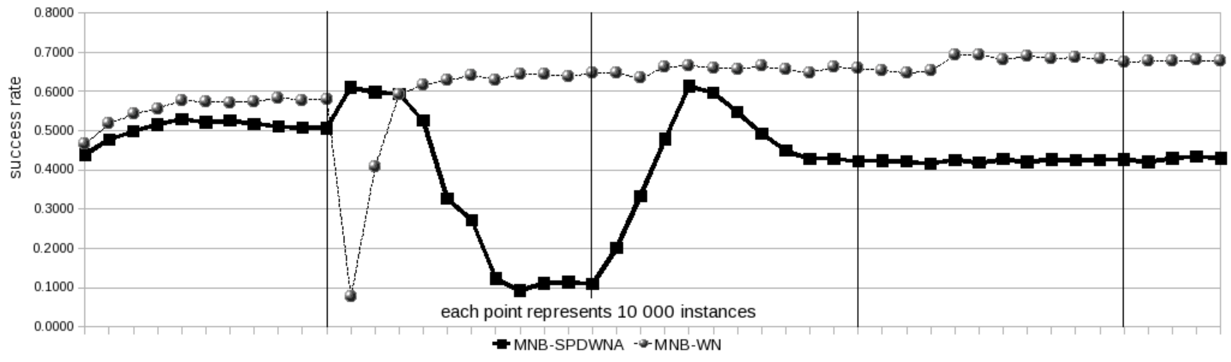

Figure 1: Consequences of bad initialization

Category	MNB	TOE	NW	SPDA
Arts	23.00%	66.27%	69.25%	65.03%
Business	00.59%	49.88%	39.92%	68.02%
Computers	13.12%	54.10%	68.06%	65.47%
Games	10.88%	49.83%	76.40%	71.36%
Health	6.12%	41.10%	61.66%	68.10%
Home	14.61%	40.90%	56.97%	73.51%
News	0.00%	0.99%	48.34%	72.19%
Recreation	0.98%	39.96%	38.31%	68.63%
Reference	0.00%	22.78%	47.00%	71.70%
Regional	99.87%	89.91%	87.29%	74.67%
Science	15.94%	55.59%	60.12%	66.80%
Shopping	0.07%	32.60%	54.71%	68.13%
Society	14.87%	62.30%	46.05%	60.82%
Sports	2.65%	50.25%	66.95%	72.25%
Adult	17.20%	54.00%	86.88%	83.22%
Deviation	**0.2394**	**0.1945**	**0.1493**	**0.0500**

Table 3: Class-specific success rates

MNB	TOE	NW	SPDA
4.13	1.13	4.17	1.00

Table 4: Algorithm training times

to the fact that the algorithm trains only on a subset of the documents. With a less noisy dataset, where the algorithm would make fewer errors, these savings would be much higher.

MNB	TOE	NW	SPDA
1.00	1.01	1.01	1.46

Table 5: Algorithm classification times

MNB-SPDA is heavier in classification cost though; classification times can be compared in *Table 5*. This is due to the fact that the algorithm performs an adjustment to its prior distributions after each training, which adds some complexity. It has to maintain a stack with the last 10,000 errors and summarize it before classifying each instance - an operation other algorithms do not need. In this area, computation can be optimized in the future.

6 Conclusion and Future Plans

In this work we presented the basis for a working mechanism that will make the automatic building of web directories practical. One of the variations we introduced: stochastically adjusting prior probabilities, allows the algorithm to learn from its errors and achieve not only better overall success, but better error distribution across classes as well. This makes it equally reliable for all classes, unlike the other available methods.

The greatest challenge we faced though is not algorithm efficiency but noisy data. As often noted, the web is noisy and a significant part of the noise can be filtered out only by people. Training a successful classifier seems to need more human labeling than is freely available at this time. Our method can be used for initial training of the classifier, but then collaborative filtering based on user feedback may need to be employed for further tuning.

This study makes the following contribution to the field:

- We treat data as dynamic and achieve better classification results on this basis.
- Our method saves significantly on training time.
- Our method makes some savings in terms of storage requirements for training data.
- The classifier based on our method has smaller variation of its success rate across classes.

In our future work we intend to further enhance the ideas presented in this paper by incorporating the following: we plan to implement boosted learning which will apply the knowledge learned from labeled data to the unlabeled data obtained by a web spider; we will make our classifier fuzzy, allowing an instance to belong to many classes; we will try to optimize management of the *error stack* on which our method is based in order to make it as efficient in terms of classification times as the other algorithms; we will conduct further experiments on a larger dataset.

References

Arnaud, D. (2004), ‘Study of a Document Classication Framework Using Self-Organizing Maps’, `users.swing.be/aundro/websom/repository/som.pdf`.

Attardi, G., Gullí, A., Dato, D. & Tani, C. (1999), ‘Towards automated categorization and abstracting of web sites’, `di.unipi.it/~gulli/papers/submitted/automated_classification.htm`.

Baeza-Yates, R. & Ribeiro-Neto, B. (1999), *Modern Information Retrieval*, ACM Press Books, Harlow.

Bar-Yossef, Z., Broder, A. Z., Kumar, R. & Tomkins, A. (2004), Sic transit gloria telae: Towards an Understanding of the Web’s Decay, *in* ‘Proceedings of the 13th conference on World Wide Web’, New York, NY, USA, pp. 328–337.

Broder, A. (2002), 'A Taxonomy of Web Search', *SIGIR Forum* **36**(2), 3–10.

Chakrabarti, S., Dom, B., Agrawal, R. & Raghavan, P. (1998), 'Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies', *The VLDB Journal* **7**(3), 163–178.

Chakrabarti, S., Dom, B. E. & Indyk, P. (1998), Enhanced Hypertext Categorization Using Hyperlinks, *in* L. M. Haas & A. Tiwary, eds, 'Proceedings of SIGMOD-98, ACM International Conference on Management of Data', ACM Press, New York, US, Seattle, US, pp. 307–318.

Chierichetti, F., Kumar, R. & Raghavan, P. (2009), Compressed Web Indexes, *in* '18th International World Wide Web Conference', pp. 451–451.

Das, A., Datar, M., Garg, A. & Rajaram, S. (2007), Google News Personalization: Scalable Online Collaborative Filtering, *in* 'Proceedings of the Sixteenth International World Wide Web Conference (WWW2007)'.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W. & Harshman, R. A. (1990), 'Indexing by Latent Semantic Analysis', *Journal of the American Society of Information Science* **41**(6), 391–407.

Domingos, P. & Pazzani, M. J. (1997), 'On the Optimality of the Simple Bayesian Classifier under Zero-One Loss', *Machine Learning* **29**(2-3), 103–130.

Enkhsaikhan, M., Wong, W., Liu, W. & Reynolds, M. (2007), Measuring Data-Driven Ontology Changes using Text Mining, *in* P. Christen, P. J. Kennedy, J. Li, I. Kolyshkina & G. J. Williams, eds, 'Sixth Australasian Data Mining Conference (AusDM 2007)', Vol. 70 of *CRPIT*, ACS, Gold Coast, Australia, pp. 39–46.

Fetterly, D. (2007), 'Adversarial Information Retrieval: The Manipulation of Web Content', *ACM Computing Reviews* .

Fetterly, D., Manasse, M., Najork, M. & Wiener, J. (2003), A Large-Scale Study of the Evolution of Web Pages, *in* 'WWW '03: Proceedings of the 12th international conference on World Wide Web', ACM, New York, NY, USA, pp. 669–678.

Frank, E. & Bouckaert, R. R. (2006), Naïve Bayes for Text Classification with Unbalanced Classes, *in* 'Proc 10th European Conference on Principles and Practice of Knowledge Discovery in Databases', Berlin, Germany, Springer, pp. 503–510.

Gorrell, G. (2006), Generalized Hebbian Algorithm for Dimensionality Reduction in Natural Language Processing, PhD thesis.

Gorrell, G. & Webb, B. (2005), Generalized Hebbian Algorithm for Latent Semantic Analysis, *in* 'Proceedings of Interspeech 2005'.

Graham-Cumming, J. (2006), 'Does Bayesian Poisoning Exist?', `virusbtn.com/spambulletin/archive/2006/02/sb200602-poison`.

Győngyi, Z., Berkhin, P., Garcia-Molina, H. & Pedersen, J. (2006), Link Spam Detection Based on Mass Estimation, *in* 'VLDB '06: Proceedings of the 32nd international conference on Very large data bases', VLDB Endowment, pp. 439–450.

Győngyi, Z. & Garcia-Molina, H. (2005), 'Web Spam Taxonomy', `citeseer.ist.psu.edu/gyongyi05web.html`.

Győngyi, Z., Garcia-Molina, H. & Pedersen, J. (2006–2007), 'Web Content Categorization Using Link Information', `infolab.stanford.edu/~zoltan/publications.html`.

Kaski, S. (1998), Dimensionality Reduction by Random Mapping: Fast Similarity Computation for Clustering, *in* 'IJCNN98 International Joint Conference on Neural Networks', Vol. 1, Piscataway, NJ: IEEE, pp. 413–418.

Katakis, I., Tsoumakas, G. & Vlahavas, I. (2005), 'On the Utility of Incremental Feature Selection for the Classification of Textual Data Streams', *Advances in Informatics* pp. 338–348.

Kohonen, T. (1995), *Self-Organizing Maps*, Springer Verlag, Berlin.

Lagus, K., Kaski, S. & Kohonen, T. (2004), 'Mining Massive Document Collections by the WEBSOM method', *Inf. Sci.* **163**(1-3), 135–156.

Lang, K. (1995), NewsWeeder: Learning to Filter Netnews, *in* 'Proceedings of the 12th International Conference on Machine Learning', Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, pp. 331–339.

Lee, U., Liu, Z. & Cho, J. (2004), Automatic Identification of User Goals in Web Search, Technical report, UCLA Computer Science.

Liu, T., Yang, Y., Wan, H., Zeng, H., Chen, Z. & Ma, W. (2005), 'Support Vector Machines Classification with a Very Large-Scale Taxonomy', *SIGKDD Explorations* **7**, 2005.

Rennie, J. D. M., Shih, L., Teevan, J. & Karger, D. R. (2003), Tackling the Poor Assumptions of Naïive Bayes Text Classifiers, *in* 'Proceedings of the Twentieth International Conference on Machine Learning', pp. 616–623.

Rocchio, J. (1971), *Relevance Feedback in Information Retrieval*, pp. 313–323.

Salakhutdinov, R. & Hinton, G. (2007), 'Semantic Hashing', `www.cs.utoronto.ca/~hinton/`.

Xue, G. R., Xing, D., Yang, Q. & Yu, Y. (2008), Deep Classification in Large-Scale Text Hierarchies, *in* 'SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval', ACM, New York, NY, USA, pp. 619–626.

Yanbe, Y., Jatowt, A., Nakamura, S. & Tanaka, K. (2007), Can Social Bookmarking Enhance Search in the Web?, *in* 'JCDL '07: Proceedings of the 2007 conference on Digital libraries', ACM Press, New York, NY, USA, pp. 107–116.

Yoshida, T., Nakamura, S. & Tanaka, K. (2007), WeBrowSearch: Toward Web Browser with Autonomous Search, *in* 'WISE', pp. 135–146.

Zawodny, J. (2003), 'PageRank is Dead', `jeremy.zawodny.com/blog/archives/000751.html`.

Zdziarski, J. A. (2005), *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*, No Starch Press.

Zipf, G. K. (1949), *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*, Addison-Wesley.