Towards Real Intelligent Web Exploration

Pavel Kalinov, Bela Stantic, and Abdul Sattar

Institute for Integrated and Intelligent Systems,
Griffith University, Brisbane, Australia
`pavel.kalinov@griffithuni.edu.au`
`{A.Sattar,B.Stantic}@griffith.edu.au`
`http://iiis.griffith.edu.au/`

Abstract. A significant problem of the dominant web search model is the lack of a realistic way to acquire user search context. Search engines use implicit feedback, which is extremely sparse and does not allow users to properly define what they want to know, or what they think of search results. In our proposed "web exploration engine", which we implemented as a prototype, documents have been automatically pre-classified into a large number of categories representing a hierarchy of search contexts. Users can browse this structure or search within a particular category (context) by explicitly selecting it. Keyword relevance is not global but specific to a category. The main innovation we propose is the "floating" query resulting from this feature: the original search query is re-evaluated and the importance of its features re-calculated for every context the user explores. This allows users to search or browse in a truly local (context-dependent) way with a minimum of effort on their part.

Keywords: Web Directory, Relevance Feedback, Adaptive Ranking

1 Introduction and Motivation

Scientific research on web information retrieval systems is concentrated primarily on search engines, which are also the dominant information-finding model on the web. However, many unresolved issues have arisen from their dominance.

Search engines by design satisfy the *information locating* need only (the user knows something exists and needs to find out where it is). They do not address the complementary *information discovery* need (the user does not know something and needs to find out that it exists). This is served by web directories, which are now in decline (mainly due to not implementing machine learning).

A major issue that recently began to be recognised arises from the fact that search engines limit the number of results shown to users. With a finite number of keyword combinations related to every document, it is very probable that (given a consistent ranking algorithm) a significant number of documents will always rank below the "decision boundary" and be unreachable through keyword search [1], *even if people search for them using the correct keywords.* A different ranking algorithm might put these documents before the threshold, but users are not allowed to influence the algorithm or select a different one.

Originally, search engines were built around some basic assumptions: that every search is separate and independent; every search is generated by a short-term interest; every user means the same with the same word; users require fast answers; users cannot (or will not) explicitly supply context for their queries; advanced features should be sacrificed to simplify search interfaces. These assumptions are the basis of the current system-centric model; many features and methods are introduced to address issues arising from them, but these are only mitigation efforts while the underlying model remains the same.

Additionally, search context and the user's "background knowledge" (cognitive context), as well as *synonymy* and *polysemy* [2] are largely missing from the picture (again, with many mitigation efforts which do not change the underlying model). To optimise for speed, most search results are pre-computed and identical for all users and contexts, with minimal personalisation at retrieval time to fit certain pre-computed criteria [3]; search queries are limited to a (short and flat) list of words: users lack an expressive way to define what they search for; they cannot adjust result ranking factors, or even know what they are.

Furthermore, some exploitable features of popular search engine algorithms have had a large negative impact on the web as a whole. A whole "search engine optimisation" industry arose out of these features, employing practices such as "link farming" (creating sites for the express purpose of filling them with links to other sites in order to increase their "popularity"), "keyword stacking" (publishing automatically generated text filled with desired keywords to increase "keyword density"), "keyword bombing" (linking to sites with specific keywords in the link to improve their ranking for those words), the creation of "fast food content" (low-quality texts aimed exclusively at gaining high search engine ranking so as to place advertising on them) etc. They lead to a significant and growing proportion of web content being "digital garbage" aimed at search engines and not humans, making the information-finding task ever more difficult for users.

Web directories, if they manage to overcome some of their problems, could provide a viable alternative addressing some of these issues. For example, a large enough browsable hierarchy of topically ordered documents could allow users to reach any document with a finite number of navigational clicks, whatever its ranking is; while browsing, users would see branches they never knew existed (thus would never search for) and would discover new information; search within a category would limit the scope of their search, providing some search context.

However, web directories face even more serious problems than search engines. Some are inherent, like the difficulty users have of navigating a large tree-like structure (the Open Directory has over 700,000 categories), where they have to take a decision at every level: which branch down to follow? This is not as trivial as it seems, since it requires users to a) know exactly what they want, and b) classify it the same way as the directory maintainers do (e.g., somebody looking for dolphins under "Fish" will never find them).

Information acquisition is manual: entries are individually added by humans. This results in web directories having several orders of magnitude less data than search engines, and in whole types of web documents completely missing from them. For example, editors may link to the home page of an online newspaper, but not to its individual articles. They also avoid documents which tend to disappear or change too often, since maintaining the integrity of already published information is another significant problem related to the limitations of the manual maintenance model, which suffers heavily from "web decay" [4].

Another problem is the difficulty of using machine learning for building a web directory. While it is essentially a text classification problem, there are some web-specific issues making the use of popular text classification algorithms impractical. The biggest of them is that the web grows and changes constantly [5]. Human perception of resources also changes, so a resource may be initially classified into a category and then re-classified to another. A "batch" text classifier would need to be retrained from scratch to reflect such changes, which is not a feasible task if it has to be done often on a real-world web scale. "Streaming data" algorithms can deal with online addition of data, but not with removal/modification of instances already processed, or their re-classification.

A further problem is that statistical text classification algorithms are based on some assumptions: feature independence, relatively small variance in basic document statistics like word count per document etc.; these assumptions are usually broken in real-world texts, but classifiers manage to overcome this by applying different normalisations. It turns out though that documents from different categories break these assumptions differently and some vital statistics vary greatly between classes (our experiments found that documents in the "Business" category of our sample of downloaded Open Directory sites had an average of 96.54 unique terms each while "News" had 235.05 [6]). As a result, a classification method that works in one category ("Sports" sites) does not work for another ("Business" sites) [7]. Every separate issue can be overcome by some heuristic, normalisation or parameter setting, but the variance of issues between categories means that different solutions have to be found for almost every category. This task is comparable in complexity to the manual classification of all instances and is probably why machine learning has never been applied to a major directory.

One way to create a hybrid between a search engine and web directory is to apply clustering of search results and present clusters to the user as search contexts [3], for which also an open-source framework exists[1]. However, this clustering is applied over the (limited) search results provided by the search engines and expressly cannot classify documents into pre-existing categories. These clusters are also typically a small number and only one level deep, i.e.: they cannot achieve a deep topical hierarchy.

An alternative approach could combine some features of search engines with some features of web directories and use the advantages of one model to overcome inherent problems of the other. For example, using an automated data collection mechanism (a web spider) would be highly beneficial to web directories, while the ability of users to browse topically ordered information will provide the "information discovery" lacking in a search engine, and the ability to narrow down search to a particular category will provide some search context.

[1] Carrot^2: http://project.carrot2.org/

2 Web Exploration Engine

We propose a system which is a cross between a web directory and a search engine: it collects and indexes documents using a search engine-like web spider, but then classifies them into a hierarchically ordered set of categories. These act as pre-made contexts and allow users to limit their search (when performing keyword search), but are also browsable allowing users to find information without having to supply keywords which is difficult in cases where users cannot specifically define what they are looking for (i.e. half of search cases [8]).

Users can search the directory or specific parts of it, and are able to influence result ranking by providing explicit relevance feedback which is then integrated into the research query. We have named it the "research query" as opposed to "search query" to stress the fact that it is not a flat keyword list but a complex user-adjustable vector where term weights can even be negative (if the user indicated negative preference towards some resource); the user builds this query over a period of time, and can then save it and re-use it later, allowing long-term research over a topic. The initial query is optional since the user can start by just browsing (having and empty query) and supplying relevance feedback ("this site seems like what I want", "this site is something I do not want"), which then creates a query from the feedback instead of just adding to the initial one.

This "Web Exploration Engine", as we have named it, enables users to:

– browse a directory structure (providing *information discovery*);
– search by keywords (providing *information locating*);
– limit the search within a branch of the directory tree, enabling a focus on a narrower (predefined) context;
– create and/or expand a query by supplying relevance feedback, providing explicitly specified user context;
– expand the query in a "session" manner, with small increments leading to a detailed, in-depth query;
– save a query for later re-use and expansion, enabling long-term research.

To build this structure, we use a statistical classifier trained on human-labelled examples. Its hierarchy can be arbitrarily deep, with relatively few documents per leaf node, so that every document is reachable by simple browsing. If we suppose an average of 100 documents per leaf node and 10 branches per node, users would be able to browse 100 billion documents by ten clicks on average.

2.1 Building the Directory

As opposed to the traditional model, our system [9] uses a web spider to obtain its data. The spider is semi-automated: it does not follow all links automatically but is guided by a human editor (URLs have to be approved before being downloaded). This assures a certain level of quality, e.g. - the editor may decide to not index forum postings or other user-generated content, and allows a certain focus by topic if we want to build a "vertical" (topic-specific) directory: the editor can approve only links outgoing from relevant sites. Processing the download queue is assisted by auto-approval by manually added URL patterns ("if the URL starts with *http://www.bbc.co.uk/news/* - download it without specific approval").

We train a hierarchical structure of Multinomial Naïve Bayesian classifiers [6] on labelled URLs from the Open Directory which are classified by human editors into a tree of categories. Expanding this information in our system is assisted by heuristic pattern-based classification: we added classification rules such as "if the URL starts with *http://www.transdat.com.au/* and contains machine*.html" - add it to the Business / Construction / Equipment / Machines subcategory" (where * represents a wildcard). Documents downloaded by the spider that are not classified manually or by pattern are added to categories based on the classifier's decision; this increases content by several orders of magnitude.

We can make human labour more efficient by using the classifier scores: editors can assess and classify only "borderline" cases with low classifier score, and not waste time on clear-cut cases. Conversely, cases with too high classifier confidence can also be examined to prevent "search engine optimised" documents (which can be considered a form of spam) from taking advantage of some feature of the algorithm. Additionally, the editors' attention can be focused on any atypical documents. For example, our system calculates the average term TF-IDF of every document; our experiments showed that most outliers (documents with extreme average values) are either errors or web spam: machine-generated text used for "keyword stacking", or random gibberish to confuse spam filters ("Bayesian poisoning" [10]). These can be weeded out at a relatively low cost, as compared to a human checking every document in the directory.

For hierarchical classification we use not one multi-level classifier but a tree of separate independent classifiers; outputs from each one (in the form of lists of documents belonging to a category) are inputs for those at the lower level. Each classifier works with data from its category only and calculates static measures locally. Among other things, this means there cannot be global stop-words. Stop-words, and word weights in general, are context-specific: the usual example are words such as "the" and "and" which are too common in the English language and useless for classification. However, their frequent appearance in a text means that the text is (very probably) in English, so they are useful at the first level of our tree where we separate English-language from "other language" documents. This applies to a differing extent to all words in all contexts, so we calculate their weights separately for every context (category). Consequently, we do not use linguistic pre-processing which relies on stop-word filtering. On the plus side, using a number of independent classifiers allows us to create a distributed system to work with web-scale data, with separate servers for each category.

To develop the classification part of our prototype, we experimented [6] using labelled data from the Open Directory Project. We downloaded a sample of documents and trained on their original texts and not the ODP descriptions which tend to be very short, as well as repetitive hence not very distinguishing.

The first issue we faced was lack of sufficient labelling data, which is extremely sparse. The Open Directory contains an average of only 6.02 documents per category. From an end-user point of view browsing such categories is impractical, and from our point of view training a classifier on only 6 instances is impossible. To address this, we "folded" categories to include all instances from descendant nodes as well (the same approach was used in other projects [11, 12] based on the Yahoo! Directory, where category fragmentation is even worse).

We chose Multinomial Naïve Bayes as our main classification tool because it learns incrementally, classifies fast and generalises well. However, it has serious problems with very noisy or unbalanced data (such as web documents).

We applied several mitigation measures. The first was TF-IDF term weighting to discount frequent terms, which is standard practice in text classification. However, as already discussed, we use local values for the IDF part as it is calculated over a partial document collection (those documents belonging to a category), so the same word has different IDF values in different parts of the classification tree. We also applied word count normalisation [13] to compensate for the varying average length of documents between classes:

$$n'_{wd} = \alpha \times \frac{n_{wd}}{\sum_{w'} \sum_{d \in D_c} n_{w'd}} \tag{1}$$

where $n_{w'd}$ are l_2 normalised class-specific word counts (number of occurrences of the word w in documents d of the part of the corpus D_c belonging to class c); we took the smoothing parameter α (vector length measured in the l_1 norm) to be equal to 1, as that was shown to work well [13] and we did not want to introduce a new point of failure by experimenting with it further. This normalisation improved classification success considerably, but results were still not satisfactory as a whole. The main issue was not so much the overall classification success rate as its variance between classes: for some categories the classifier was almost perfect while in some categories it had a success rate of practically zero.

We then decided to apply a policy which proved to be effective for industrial spam filters [10]: *train on error*, meaning the classifier trains only on documents which it misclassified. Obviously, this skews word counts and prior distributions, since the classifier "sees" only a small part of the document collection (the borderline cases) and ignores most of the typical documents. It is counter-intuitive, but works in the real-world, which was also confirmed by our experiments.

Apparently, modifying the basis on which word counts and prior distributions are calculated serves to improve the situation dramatically. We then tried several unsuccessful new variations until we made a breakthrough which actually changed the fundamental workings of the algorithm. We decided to calculate class prior distributions dynamically: over a sliding window of the last 10 000 errors and not over the whole document collection. This introduces a negative feedback loop: problematic classes become over-represented in this stochastic sample (there are disproportionately more errors in them) so their prior probability is adjusted upwards to compensate for the problems (note that we do not even need to know what the problems are). Of course, a price has to be paid: increased accuracy in some classes happens at the expense of classes where the classifier was previously more accurate. Nevertheless, overall accuracy increases and - more importantly - error variation between classes drops four times [6], meaning that the classifier is now equally reliable for all classes.

Fundamentally, we now have a different type of algorithm. It is based on Multinomial Naïve Bayes but is no longer a static batch-classifying algorithm. Instead, it is dynamic in the sense that it needs to keep classifying (and make some new errors!) in order to learn. The negative feedback loop keeps it in a dynamically stable state - in this case, minimising error variance between classes (note that it does not minimise errors themselves, just their inter-class variance, so it can never be completely error-free).

This approach leads to some important consequences that have to be noted:

– The algorithm has a learning stage: we have to keep classifying (and making errors) until it converges to a stable state. We found experimentally that this happens after four or five iterations over the whole document corpus.
– Classification order matters! If we do not randomise iterations properly and, for example, start classifying with instances of one class only it can easily get unbalanced and go into wild fluctuations that are difficult to recover from.
– After we have classified the whole document corpus, we re-initialise and keep classifying in order to stay up-to-date. Essentially, the classifier never stops.
– The classifier works in parallel with the web spider, which keeps updating the document corpus (adds new documents, deletes documents that were removed from the web and updates documents that changed), and with the editors who add documents but may also move them from one category to another. To avoid potential conflicts, the classifier works not with current data but with a snapshot of the data as it was when an iteration started.
– Since the classifier keeps re-training, word counts would potentially reach enormous values. To avoid this, we introduced weight decay applied at the start of each iteration. This means that a term which has not been encountered for some time will have its weight decreased and will be eventually deleted. We get two benefits from this: if we move a document from one category to another, the classifier "forgets" the initial classification after some time (automatic re-learning), and if a document gets removed from the collection altogether the terms associated with it are eventually deleted. This automatically deals with Bayesian poisoning, because documents containing it tend to be short-lived (comment spam or spam text injected into hacked legitimate sites which gets deleted when the site owners discover it).
– Since the method deals automatically with any imbalances, noise and other problems in the data, there are no heuristics or parameter settings to modify for its various use cases (which is a major issue for industrial spam filters [10]). We can apply it in every part of the category tree without modification.

Having this algorithm, we can build a usable large-scale web directory with relatively low human labour. It has to be stressed though that the method is not fully automatic as it inherently needs manual labelling of some data instances, as well as creating the category structure. Nevertheless, we believe it is much more viable from an economic point of view than the current dominant model.

2.2 Browsing the Directory

The directory can be browsed as a simple static hierarchical structure similar to the Open Directory. We have added some document ranking options though to improve usability. Users can browse documents alphabetically, ordered by editor's choice, by *typicality* (how typical a document is for the category - by its classifier score) or by relevance (similarity) to the current user query.

The main feature of the system however is its *Exploration* mode. Users can start exploring by submitting a query in the form of several keywords, a short text or a whole document, or they can just start browsing the directory - i.e., have an empty query; in both cases, this query can later be expanded.

The engine returns results in the form of a path through the directory tree, as well as some document listings. It treats the query as a document, passes it through the pre-processing filter of the backend classifier and converts it into a TF-IDF weighted vector (we have TF because words can have more than one occurrence in the query, and initially we use IDF values for the whole document corpus). This weighted vector then goes through the many levels of classifiers, where the IDF parts of term weights are substituted with their local values. At each level, the most probable category is returned as the answer but other categories are listed as well, in order of probability (i.e. - how well they fit the query). They are illustrated by several "most typical examples": the top N instances by order of classifier score.

Users can follow the suggested path through the directory and see listings at different depth levels. The ability to "jump" straight to a low-level category without having to manually select branches at higher levels significantly assists users from a usability point of view but also from a cognitive point of view: those who would otherwise look for dolphins under "Fish" would now learn that they are "Mammals/Marine" instead. Users can also correct the system by following an alternative path; or, even if the classifier is correct, they can still click on alternative links and explore related areas using it as a recommendation feature.

2.3 Searching in the Directory

As all search engines do, we use a document "posting list": an inverted index which is the reversed version of the document description table (instead of indexing words contained in a document, it indexes documents that contain a word). In our case though this is not a flat occurrence list with *yes* or *no* values or number of occurrences. We have a normalised weight for each word for each document: $sw_i = \frac{tf\text{-}idf_{wi}}{\overline{tf\text{-}idf_i}}$ - the specific weight sw_i for the word in document i is the ratio between the word's TF-IDF value for that document and the average TF-IDF of all words in the document. As with every other use of IDF values in our system, these are local, calculated for the specific category.

On this basis we can compare the relative importance of words in documents and can say for example: word w_i is twice more important for document D_j than for document D_k **in category** C. We order candidate documents by these scores to rank them by relevance to the research query.

As discussed, a word is common or distinctive in some context only. The word *software* can provide a good decision boundary when splitting IT-related from other texts, but once we have texts about software only it stops being distinctive and should already be treated as a stop-word. This is why having local weights is an advantage for our system: at the top level, users receive results from the whole document collection, ordered by global relevance (just as if they search a normal search engine); when they select a specific category though, we perform local search: over a partial document collection and *ordered by local relevance.* In effect, we provide a cascading series of increasingly specialised search engines allowing focused search in pre-defined contexts. Search can be narrowed down by selecting from millions of nested contexts with only several clicks.

For technical reasons, we limit the number of search results returned to the user to 1000. Limiting the number of search results would normally produce the effect that some documents become unreachable by search. In our model however, the user can browse the directory tree into more specific categories thus a) narrowing the document collection, b) reordering search results due to the different local ranking, and/or c) conceivably reaching a node with a handful of entries only, all visible at a glance, where search will not even be needed.

As additional usability enhancements, we added two features lacking in modern search engines: the ability of users to "bookmark" a research (the initial query plus all feedback in the form of sites marked as relevant/not relevant) and later re-use it, as well as the ability to add their own bookmarks to a list of search results, whether the exploration engine considers them relevant or not.

2.4 Query Expansion, Floating Query

The initial user query can be expanded interactively by relevance feedback. In the result list users can mark documents as either relevant or not relevant; when that happens, all the (weighted) keywords of the document are added to the query. This one-click query expansion does not require the user to enter keywords - the system supplies them instead. Since some of the added documents are positive (relevant) and others are negative (not relevant) examples, the query becomes a complex object with a positive and a negative component. To balance its three parts, we use Rocchio relevance feedback weighting [14]:

$$Q = \alpha Q_u + \beta Q_p - \gamma Q_n \tag{2}$$

where the resulting query Q consists of the original user query Q_u plus the query vector Q_p from documents marked as relevant minus Q_n (non-relevant documents). Query vectors are calculated as $Q = \frac{\sum D}{n}$, where D are the document vectors and n is their cardinality. α, β and γ are weights signifying how important each part is. By default $\alpha = 1$ and $\beta = \gamma = 0.5$, i.e. the original query is as important as all the feedback, and the positive and negative components weigh equally. Users can adjust these values on a case-by-case basis though.

Unlike other query expansion methods (such as the ARCH system [12]), our query expansion works on the document and not the category level; they use all terms from all documents in a category while we only add/subtract terms from user-selected documents. This gives full and explicit control to the user and is more precise: it provides several orders of magnitude more granularity.

The resulting *research query* is very different from a standard search engine query: a) it can be arbitrarily long, and b) it is not a flat list of words but a weighted array, where weights can be negative. The query is passed through the hierarchical classifier which takes into account all terms to calculate results. It has to be noted though that results do not need to contain all or even most of the terms - they are just those that (probably) best relate to the query. An empty query would still return results: this will be "the most probable" path through the hierarchy, which is "the most populated categories at each level" as probability estimation would in this case be based on prior probabilities only.

Further to being complex, the query also changes as the user moves around the directory structure. Keywords in the query have an initial weight, which is then multiplied by the local weight of that keyword in the category which the user is currently in. Thus, we get a *floating query* which changes with every click the user makes, adapting to the current context. An example can be seen in the table below (values are not realistic but chosen for illustration purposes only).

	Original	English	Equipment	Aerospace
and	0.01	**0.92**	0.00	0.00
repair	**0.47**	0.69	0.15	0.06
radar	0.23	0.03	**0.89**	0.08
helicopter	0.20	0.61	0.85	**0.98**

Table 1. Evolution of the original search query as the user moves through the English-language category into Equipment and Aerospace subcategories.

Note how the word *and* is initially distinctive, as it is a useful indicator to distinguish between English and other languages, then becomes a stop-word (practically disappears) as all documents in the category become English-language only. In the top level of English-language sites, *repair* is a good discriminator between Equipment and other sites, but then becomes common within that context and its importance shrinks accordingly. In the Aerospace category, *helicopter* becomes important as it distinguishes different types of Aerospace equipment.

2.5 Advantages of the Exploration Engine

The web exploration model we describe has several advantages over the existing search engine and web directory models:

– Unlike PageRank [15] and its derivatives, our ranking relies on features of the document itself and not on external factors. This renders ineffective most of the arsenal of the *search engine optimisation* industry: "keyword bombing" or "link farming" become irrelevant; "keyword stacking" is counter-productive, as adding more keywords to a document dilutes the TF-IDF values for all of these keywords and it will not rank high for any of them. Furthermore, any *optimisation* will work in one category only, since every

category has its own IDF values; thus, a document cannot be "optimised" for one category without harming its ranking in all the (millions of) others. If the model becomes wide-spread, it may deter creation of "digital garbage".
- The model uses the much more expressive explicit as opposed to implicit user feedback, allowing better user control.
- The only ranking parameter the system uses (the ratio between α, β and γ in feedback weighting) is user-adjustable, so the user has full control: we make no assumptions on his behalf.
- Unlike the dominant search engine model, in our model all documents are reachable: both by simple browsing and by focused search.
- The dynamic nature of our classifier automatically takes care of changes in classification (where editors move documents between categories), as well as Bayesian poisoning and other noise.
- The independent nature of the individual classifiers in our hierarchical classification tree allows a relatively easy implementation of a distributed system.
- Some of the features of our method facilitate more effective human editing by pointing the editors to the most problematic entries requiring attention.
- Heuristic pattern-based classification adds orders of magnitude more content into categories, used also for training data for automated classification.
- Users control the search process through explicit feedback, allowing them to better specify what they search for.
- Users can develop a query over a period of time, save it and later re-use or expand it, enabling long-term research.
- Users can add their own snippets of data to search results, even though the exploration engine may lack this data or not consider it relevant.

3 Conclusion

We propose a *Web Exploration Engine* as a hybrid information-finding model. Where current systems try to guess search or user context by implicit feedback, we use pre-computed contexts from which the user can select one and then modify it. The model also allows users to expand their research into related topics, assisting *information discovery.*

Document ranking in our search results is category- (context-) specific: the same document is ranked differently against the same keywords, depending on the context in which the user is searching. The "floating query" which enables this is the main innovation of this work.

In our future work, we will add real-world scale data, since the advantages of the engine will be best visible when it contains billions of documents in millions of categories. If using manually labelled examples on this scale this proves to be prohibitively expensive, a future development could be to allow users to assist the training process by adding a collaborative filter to the user experience.

We believe that the introduction of our classification method for dynamic data, and our approach to information locating and information discovery as a whole, make the revival of web directories practical and will allow them to easily grow into *Web Exploration Engines.*

References

1. I. H. Witten, M. Gori, and T. Numerico, *Web Dragons: Inside the Myths of Search Engine Technology.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.
2. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.
3. Vivisimo Inc., "Tagging vs. Clustering in Enterprise Search." `http://vivisimo.com/html/download-tagging`, Aug. 2006.
4. Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins, "Sic transit gloria telae: Towards an understanding of the web's decay," in *Proceedings of the 13th conference on World Wide Web* (S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, eds.), WWW '04, (New York, NY, USA), pp. 328–337, ACM, May 2004.
5. D. Fetterly, M. Manasse, M. Najork, and J. Wiener, "A large-scale Study of the Evolution of Web Pages," in *Proceedings of the 12th International Conference on World Wide Web*, WWW2003, (New York, NY, USA), pp. 669–678, ACM, 2003.
6. P. Kalinov, B. Stantic, and A. Sattar, "Building a Dynamic Classifier for Large Text Data Collections," in *Proceedings of the 21st Australasian Database Conference* (H. T. Shen and A. Bouguettaya, eds.), vol. 104 of *ADC2010*, pp. 113–122, Australian Computer Society, 2010.
7. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen, "Web Content Categorization Using Link Information," tech. rep., Stanford University, 2006-2007.
8. T. Yoshida, S. Nakamura, and K. Tanaka, "WeBrowSearch: Toward Web Browser with Autonomous Search," in *Proceedings of the 8th International Conference on Web Information Systems Engineering* (B. Benatallah, F. Casati, D. Georgakopoulos, C. Bartolini, W. Sadiq, and C. Godart, eds.), WISE'07, (Berlin, Heidelberg), pp. 135–146, Springer-Verlag, Dec. 2007.
9. P. Kalinov, B. Stantic, and A. Sattar, "Let's Trust Users - It is Their Search," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (J. X. Huang, I. King, V. V. Raghavan, and S. Rueger, eds.), vol. 1 of *WI-IAT 2010*, (Los Alamitos, CA, USA), pp. 176–179, IEEE Computer Society, 2010.
10. J. A. Zdziarski, *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification.* No Starch Press, July 2005.
11. G. R. Xue, D. Xing, Q. Yang, and Y. Yu, "Deep Classification in Large-Scale Text Hierarchies," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (S.-H. Myaeng, D. W. Oard, F. Sebastiani, T.-S. Chua, and M.-K. Leong, eds.), SIGIR '08, (New York, NY, USA), pp. 619–626, ACM, 2008.
12. A. Sieg, B. Mobasher, S. Lytinen, and R. Burke, "Concept Based Query Enhancement in the ARCH Search Agent," in *International Conference on Internet Computing*, pp. 613–619, 2003.
13. E. Frank and R. R. Bouckaert, "Naïve Bayes for Text Classification with Unbalanced Classes," in *Proc 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Berlin, Germany, pp. 503–510, Springer, 2006.
14. J. Rocchio, *The SMART Retrieval System*, ch. Relevance Feedback in Information Retrieval, pp. 313–323. Englewood Cliffs, NJ: Prentice Hall, 1971.
15. L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," tech. rep., Stanford Digital Library Technologies Project, 1998.