

Let's Trust Users - It Is Their Search

Pavel Kalinov, Bela Stantic, Abdul Sattar

Institute for Integrated and Intelligent Systems (IIS), Griffith University, Brisbane, Australia

pavel.kalinov@griffithuni.edu.au, {B.Stantic, A.Sattar} @griffith.edu.au

Abstract—The current search engine model considers users not trustworthy, so no tools are provided to let them specify what they are looking for or in what context, which severely limits what they are able to achieve. Instead, search engines try to guess that, which is currently done using “implicit feedback”.

In this paper we propose a “web exploration engine” - a model where users can use the search engine as their tool and explicitly specify the context of their search. Information about the web has been pre-classified in a large number of categories; users can explore this hierarchy by providing relevance feedback or search within a particular category. Search is truly “local” in the sense that keyword relevance is not global, but specific to the category. In contrast to using a search engine, users can guide the exploration engine with relevance feedback alone without entering keywords.

Keywords-Web Directory; Rocchio Relevance Feedback; Search Engine; Web Exploration;

I. INTRODUCTION AND MOTIVATION

A tendency exists to simplify features and interfaces so as to make them more attractive to the “average user”, leading to the mass development of products which have limited use for an advanced user who may need more features and be prepared to use them. This simplification has particularly been the case with web information retrieval systems, among which search engines have become predominant for a number of reasons. Several issues arise from their dominance:

- Search engines satisfy the *information locating* need (the user knows something exists and needs to find out where it is). They are not designed to address the *information discovery* need (the user does not know something and needs to find out that it exists). This was addressed by web directories, now mostly in decline.
- Search engines ranking algorithms are a target for exploitation by the *search engine optimization* industry, which generates significant amounts of *digital garbage*. Users are not taken into account by these ranking algorithms and are not allowed to weed out such content.
- A number of underlying assumptions have never been questioned, resulting in a fast but low quality service where users have no option of slower but better service.

What assumptions do modern search engines make? That:

- every search is separate and independent;
- every search is generated by a short-term interest;
- every user means the same thing with the same word;
- users require fast answers;

- users cannot explicitly supply context for their queries;
- features can be sacrificed to simplify search interfaces.

As a result, we have search engines where:

- Search is *stateless* - every search is treated as a new one, unrelated to previous searches of the same user.
- *Research*, as opposed to *search*, is not supported - users cannot develop a research topic over some time, save results and resume the search later, customize them etc.
- Search context and user “background knowledge”, as well as *synonymy* and *polysemy* issues [1] are ignored.
- To optimize search speed, most search results are pre-computed and identical for all users and contexts.
- Queries are limited to a (short) list of words. Users lack a more expressive way to define what they search for.
- Users cannot control ranking settings (adjust relative weights of ranking factors) or collaborate with others.

Furthermore, it is assumed that documents should contain all the query terms, which is realistic for short queries only. Longer queries return no match, so the search engine invisibly substitutes a shorter query in place of the original one. However, every document can be described by a finite number of short queries. Search engines limit the number of shown results; with billions of indexed documents it is then very probable that a document will rank behind the cut-off boundary for every possible query associated with it - i.e., the document becomes unreachable through keyword search, hence practically non-existent for people using such search.

It is apparent the current search engine model cannot serve all the information finding needs of users, or cover all of the web. An additional model should cover the full web, and let users influence result ranking and provide their own context.

II. RELEVANT WORK

All issues discussed above have long been recognized and addressed in a number of separate ways.

- Attempts are made to reconstruct search context from *search trails* (queries during one search session) [2].
- Google’s “Search in results” looks like search within the context of results from a previous one. In fact it’s just query expansion: a new search with the new query added to the initial one. However, search results for $A + B$ are not a subset of search results for A .
- Google creates user context by tailoring results based on *geotargeting* and preferred language. Users are not told this is happening, or given the ability to cancel it.

- Clustering of search results shows several contexts for the query. It is done at search time and is performed over the top N results only, not the whole database [3].
- Query expansion aims to introduce additional context. It can be done on the user side by a personal agent adding more terms to outgoing queries [4], or on the search engine side: the ARCH search architecture [5] uses an ontology based on *Yahoo! Directory* to expand queries; it lets the user select a category, then adds weighted typical words from that category to the original query. Another approach is to find associated terms by analysing prior searches in search logs [6].
- Personal agents ([4]) filter or re-order search results shown to the user. *Mixed initiative* allows users to actively request assistance [7]. This is, however, passive: the search engine sends normal results which are then processed; there is no way to request different results.

All of the above methods are basically just mitigation techniques on top of a standard search engine, which have little effect and do not change the fundamental search model.

III. THE EXPLORATION ENGINE

We propose a concept relying on a combination of statistical learning and human classification. This combined search engine / web directory model will enable users to:

- browse, satisfying their *information discovery* need;
- search, satisfying the *information locating* need;
- search within only a branch of the directory tree, enabling search in a narrower (predefined) context;
- create and/or expand a query by supplying relevance feedback, enabling explicitly specified 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;
- collaborate with others by sharing research sessions.

Unlike traditional directories, the web directory uses data collected by a web spider and has a comparable amount of information to that of as a search engine. It will become an extremely large structure, so machine learning methods (a statistical classifier) are used to build it. The hierarchy can be arbitrarily deep, with relatively few documents per leaf node, so every document will be reachable by browsing. If we suppose an average of 100 documents per leaf node and 10 branches per node, users would be able to reach about 100 billion documents by only ten clicks on average.

A. Browsing the Directory

Our directory prototype can be browsed in a simple way, similar to the Open Directory (DMOZ), data from which we used: a static hierarchical structure, to which we have added two improvements. There is an additional sorting option: more *typical* documents first, where ordering is by the classifier scores for each instance. The other improvement we had to make is related to the sparsity of the data. DMOZ contains

an average of only 6.02 documents per category; browsing such categories is impractical, and 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 was done by projects based on Yahoo! Directory [8] [5], where category fragmentation is worse).

To every category we also add documents downloaded by our spider, which have not been manually classified but belong to the category according to the classifier. This is the innovation which strengthens the categories by adding to them more content by several orders of magnitude.

B. Exploring the Directory

The main feature of the system is its *Exploration* mode. Users can start exploring the web by submitting an optional query in the form of 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.

Results returned to the user come in the form of a path through the directory tree, together with some document listings. The query is treated as a document and is passed through the pre-processing filter of the backend classifier, then goes through the many levels of classifiers. 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). User can then correct the system by following not the suggested path through the directory but an alternative path; even if the classifier is correct, users can still click on alternative links and explore related areas, so this acts as a recommendation feature.

Together with each suggested category, the system offers a number of documents from it. They can be ordered by:

- *editor’s choice* (default): a human has decided which documents are most important for a category;
- *category score*: list by fitness to the category;
- *relevance score*: order by relevance to the query, based on a local search using the *research query*.

In a “real world” implementation of the system (coupled with a large search engine), users would be provided with a deep path through a large directory tree, seeing hundreds of nodes to chose from (both “best fit” to their query and suggested “next best”). In effect it’s a cascading classifier which recommends directory categories but also allows users control in the sense that its recommendations can be overridden at any point, enabling true *information discovery*.

C. Searching in the Directory

Keyword search is also available based on the mechanism which orders *exploration* results by relevance. It uses an *inverted index*, which is the inverted version of the document description table: instead of indexing words contained in a document, it indexes documents that contain a word.

In our implementation this is not a flat occurrence list with *yes* or *no* values or number of occurrences. Instead, we

have a normalized TF*IDF weight for each word for each document: we record the ratio between the word's TF*IDF value for that document and the average TF*IDF of all words in the document. Based on this 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 . We order candidate documents by these scores to rank them by relevance to the search query.

Query terms, as well as document terms, are weighted by TF*IDF. Normally, TF*IDF is used to discount common words and to emphasize distinctive words. However, a word is common or distinctive in some context only. The word *software* for example 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 where our proposed innovation is. At the top level, the user receives search results from the whole document collection, ordered by global relevance. When a lower-level category is selected, we perform local search in it: over a partial document collection and ordered by local relevance. In effect, the user is provided with a series of increasingly specialized search engines allowing focused search in narrow pre-defined contexts. In our prototype, with just two clicks the user can fragment the web in 6,368 pieces to search in just one of them. With more data, it could easily scale to millions of fragments, allowing the user to select from millions of different contexts with only several clicks.

D. Query and Query Expansion

The initial 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. We use Rocchio relevance feedback weighting [9]: $Q = \alpha Q_u + \beta Q_p - \gamma Q_n$, 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 component 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 ARCH [5], our query expansion works on the document level and not the category level. This gives control to the user and is much more precise (provides several orders of magnitude more granularity). The resulting *research query*

is very different from a standard search engine query, in that 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. This query is passed through the hierarchical classifier structure which takes into account all terms to calculate the results; results though do not need to contain all or even most of the terms - they are just those that best relate to the query.

E. Additional Enhancements

Some improvements are only usability enhancements and not real innovation, but they help our exploration engine become user-centric and not server-centric as the current dominant model. Such is the *saved researches* feature, which allows users to save and re-use research queries they have developed. We also allow users to make their *research sessions* public; others can then load such sessions, modify them and then save them as their personal researches.

IV. IMPLEMENTATION DETAILS

We needed to adapt some standard methods to account for the specifics of a constantly evolving system where both the data and its classification change constantly.

A. Hierarchical Classification

We used a list of 4,228,645 labeled URLs from the Open Directory Project (DMOZ.org) which have been classified by human editors into 763,529 categories. We trained a hierarchical structure of Multinomial Naïve Bayesian classifiers using 149,178 of these documents we downloaded as training data, and a further 9,517 documents for validation. We expanded classification over the top three levels in English language, with 474 classifiers and 6,368 classes.

Our hierarchical classifier is a tree of separate classifiers; outputs from each one are inputs for those at the lower level. Each classifier works with data from its category only; static measures (such as TF*IDF) are calculated locally. This means there are no global stop-words; as discussed, they are such in some context, and each classifier has its own context.

B. Classification Algorithm

We chose Multinomial Naïve Bayes as our main classification tool because it learns and classifies fast and generalizes well. However, it has serious problems with noisy or unbalanced data, and web documents are both so we had to apply several mitigation measures to account for them.

After TF*IDF weighting, we apply word count normalization [10] to compensate for the varying average length of documents between classes: $n'_{wd} = \alpha \times \frac{n_{wd}}{\sum_{w'} \sum_{d \in D_c} n_{w'd}}$, where $n_{w'd}$ are l_2 normalized class-specific word counts.

We borrowed an approach from spam filters: a *train on error* policy; this means the classifier trains only on documents which it misclassified. It is counter-intuitive and skews word counts, but has been shown to work best with unbalanced data and most industrial spam filters use it [11].

Lastly, we calculate class prior distributions dynamically: over the last 10 000 errors and not over the whole document collection [12]. This introduces a negative feedback loop: problematic classes become over-represented in this stochastic sample so their prior probability is adjusted upwards which compensates for the problems (note that we do not need to know what the problems are). 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 as compared to the best results without this approach (see comparison table). We call this classifier MNB-SPDA (Multinomial Naïve Bayes - Stochastic Prior Distribution Adjustment). We use it in the backend to pre-process documents into the directory hierarchy, and in the frontend to classify user queries.

	MNB	WN	SPDA	DCC
normalized classification time	<i>4.1851</i>	4.2812	5.5180	1.0000
overall accuracy	0.4775	<i>0.6923</i>	0.7157	0.6836
worst class accuracy	0.0000	0.3861	0.6152	<i>0.5631</i>
interclass accuracy deviation	0.2393	0.1473	0.0492	<i>0.0866</i>
accuracy on validation set	0.5293	<i>0.4865</i>	0.4104	0.3487

Table I: Results of MNB, Word count Normalized [10], SPDA and Discretized Centroid Classifier. **Bold** is best, *italic* is second-best.

For the user frontend, we provide an alternative classification algorithm of comparable accuracy which is much faster than MNB. It is essentially a centroid classifier with discretized centroid vector values for faster computation. Compared to MNB-SPDA, the method loses only about 3% in overall accuracy, and its accuracy deviation between classes is only 1.8 times higher (other methods are much worse [12]). We offer it to users who prefer a fast result to a more precise but slower one. They can use it to first get some rough idea of what results are available, then switch to the precise method further into their research.

V. CONCLUSION AND FUTURE WORK

We propose a *web exploration engine* as a new information-finding model. While current systems try to guess search or user context by implicit feedback, we propose a large number of pre-computed contexts from which the user can easily select one and then modify it. The model also allows users to expand their research into related topics.

This study makes the following contributions:

- We have proposed a *web exploration engine* which lets users explicitly specify the context of their search.
- In our concept documents have been pre-classified in a large number of categories; users can *explore* them or perform keyword search within a particular category.
- Search is truly *local*: term relevance is not global, but specific to a category.
- Unlike search engines, keywords input is not required; the system can be guided by relevance feedback alone.

As of future work, we need to implement a collaborative filter to allow users to train the classifier, and to combine the system with the web spider of a commercial search engine so we can test for scalability and user acceptance.

REFERENCES

- [1] 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.
- [2] H. Cao, D. Jiang, J. Pei, E. Chen, and H. Li, "Towards Context-Aware Search by Learning A Very Large Variable Length Hidden Markov Model from Search Logs," in *WWW '09: Proceedings of the 18th international conference on World wide web*. New York, NY, USA: ACM, 2009, pp. 191–200.
- [3] Vivisimo, "Tagging vs. Clustering in Enterprise Search," online, Aug. 2006, www.vivisimo.com/html/download-tagging.
- [4] L. Chen and K. Sycara, "WebMate: A Personal Agent for Browsing and Searching," in *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, K. P. Sycara and M. Wooldridge, Eds. New York: ACM Press, 1998, pp. 132–139.
- [5] A. Sieg, B. Mobasher, S. Lytinen, and R. Burke, "Concept Based Query Enhancement in the ARCH Search Agent," in *International Conference on Internet Computing*, 2003, pp. 613–619.
- [6] B. M. Fonseca, P. Golgher, B. Póssas, B. Ribeiro-Neto, and N. Ziviani, "Concept-based interactive query expansion," in *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. New York, NY, USA: ACM, 2005, pp. 696–703.
- [7] M. Armentano, D. Godoy, and A. Amandi, "Personal assistants: Direct manipulation vs. mixed initiative interfaces," *International Journal of Man-Machine Studies*, vol. 64, no. 1, pp. 27–35, 2006.
- [8] G. R. Xue, D. Xing, Q. Yang, and Y. Yu, "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*. New York, NY, USA: ACM, 2008, pp. 619–626.
- [9] J. Rocchio, *Relevance Feedback in Information Retrieval*. Englewood Cliffs, NJ: Prentice Hall, 1971, pp. 313–323.
- [10] 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*, ser. Berlin, Germany. Springer, 2006, pp. 503–510.
- [11] J. A. Zdziarski, *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*. No Starch Press, July 2005.
- [12] P. Kalinov, B. Stantic, and A. Sattar, "Building a Dynamic Classifier for Large Text Data Collections," in *Proceedings of the 21st Australasian Database Conference (ADC 2010)*, ser. CRPIT, vol. 104. Australian Computer Society, 2010.