

# Let's Trust Users - It Is Their Search

Pavel Kalinov, Bela Stantic, Abdul Sattar,  
Institute for Integrated and Intelligent Systems (IIIS)  
Griffith University, Brisbane, Australia  
{P.Kalinov, 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 that category. In contrast to the existing search engines, users can explore the web without any keywords, guiding the exploration engine with relevance feedback alone.

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

## I. INTRODUCTION AND MOTIVATION

A tendency exists, not only in web development but in product development in general, to simplify features and interfaces in order to make them more attractive to the “average user” (who presumably prefers simpler interfaces and less features). This leads to the mass development of products which might indeed be useful for most people but have limited use for an advanced user who may need those omitted features and be prepared to use them. While this may make good business sense for the developing company, it is not necessarily in the best interests of the user. After all, everyone starts from being “average” but after a while might grow into “advanced” and would expect advanced features to be available.

This simplification has also been the case with information retrieval systems on the web, among which search engines have become predominant for a number of reasons. Several issues arise from their dominance:

- Search engines by design address the *information locating* need, where the user knows something exists and needs to find out where it is. They have not been designed to address the *information discovery* need, where the user does not know something and needs to find out that it exists. The latter need was addressed by web directories, but they are in decline - mainly because they have not implemented machine learning techniques as search engines did, but also because they

are deemed too complicated to use (which may explain why companies do not try to improve them).

- Search engines employ ranking algorithms to order search results. Since this affects the amount of traffic they send to web sites, these sites have a strong incentive to try to manipulate those algorithms in their favour. This has created an enormous *search engine optimization* industry, which generates a significant amount of *digital garbage* in its efforts to achieve *click arbitrage* and in effect works against the interests of the end user (one author called this business model the creation of fast-food content [1]). The user, however, is not taken into account in any way by the ranking algorithms and is not allowed to weed out such content.
- The model rests on a number of underlying assumptions which have been taken for granted in the beginning and were never re-examined later. These result in a fast and economically effective service of a low quality, while the user is not allowed to select a different option (for example - slower but better service).

The last item needs some in-depth explanation. To further the above analogy, the current search space is like a world where every food outlet is a fast-food place and there are no restaurants with quality food. Users can get serviced really fast, receive *something* to eat, even have some choice (ordinary burger or cheeseburger, chips or no chips...), but the option to get a soup, steak, salad, some fruit and a glass of wine is out of the question. In the real world fast food exists because it fills a niche - people do want fast service sometimes and are prepared to accept its limitations. There exist other places though for the cases when you do not find these limitations acceptable; however, no analogue of these places currently exists on the web.

What are some of the implicit assumptions that modern search engines make? They assume 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.
- The user requires a fast answer.
- The user cannot be relied upon to make his query more specific by supplying its context.
- The search engine interface should be simple, even if it means sacrificing features in order to simplify it.

These assumptions result in the development of 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, add their own snippets of knowledge to it, save their results and resume the search later, customize the results etc.
- Search engines are not personalized - the user's search context and "background knowledge" is not taken into account, but rather a "common denominator" is used leveling all users. This effectively ignores the *synonymy* and *polysemy* issues [2], 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 to the same person in different contexts).
- Optimizing search speed is important, so most elements of the search result have been pre-computed, again using a "common denominator" for all users and contexts.
- Search queries are limited to a small number of words. Users lack a more expressive way to define what they are looking for.
- There are no features that allow users to specify in what context they are searching, control search engine settings (change the relative importance of different ranking factors) or to collaborate with other users.

Furthermore, a standard search query is practically very limited in length: the search engine assumes that the user wants a document to contain all the words in the query, which is realistic only if they are not more than five or six. Longer queries return no match, so the search engine invisibly substitutes a shorter query in place of the original one, removing or replacing some of the original terms. This has an additional implication which is usually overlooked. Every document can be described by a (not too large) finite number of search queries. Search results are ranked by a number of factors, most of them external to the document itself and unrelated to the user and his search context. Search engines limit the number of results they show to users (Google's limit is 1000 shown documents). With billions of indexed documents on the web, it is then very probable that a document will rank behind this "decision boundary" for every conceivable query that can be associated with it - i.e., this document becomes unreachable through keyword search; for users who only rely on search engines to find information, this document is practically non-existent.

All of the above shows the search engine as an independent machine which users are allowed to use, but never on their own terms; they are not treated as individuals who can influence the search engine's behaviour, or use it as a tool they can customize. It is also apparent that the search engine model cannot serve all the information finding needs and cover all of the web, so an additional model is needed.

## II. RELEVANT WORK

The issues discussed above have been recognized and addressed in a number of ways.

- Attempts are made to reconstruct search context from *search trails* (queries during one search session) [3].
- Web site revisitation is treated as a personal problem of the user, hence solved by a personal agent [4]. Google offers some rudimentary functionality by a "Search history" feature, which leaves a lot to be desired in terms of usability.
- Google's "Search in results" feature looks like context search (search within the context of results from a previous one), but on closer inspection is just query expansion which starts a new search with the second query added to the initial one. The name of the feature is a misnomer: search results for  $A + B$  are not a subset of search results for  $A$ .
- Google tailors its results based on the user's IP address (*geotargeting*) and/or preferred language, with the unfortunate extreme case of Google China (now defunct). This aims to create some user context, but the important point is that the user is not asked to agree on such profiling, is not even told it is happening and is not given the ability to cancel it.
- Clustering of search results aims to show the user several contexts within which the query can be found. It is done at search time: first some results are found matching the query, then fast clustering is performed over the result set only and not the whole database [5].
- Query expansion is another tool aiming to introduce some user context. It can happen on the user side, where a personal agent monitors user browsing, builds a personal context and uses that to add more terms to outgoing queries [6]. Or, it can be done on the search engine side: the ARCH search architecture [7] uses an ontology (based on the Yahoo! Directory) to expand queries; it works by letting the user select a category from a classification tree, then adding (weighted) typical words from that category to the original query. Another approach is to find associated terms by analysing prior searches in search logs [8].
- Personal search agents ([6], [9]) filter or re-order search results before they are shown to the user. Users don't find them too useful as they work "behind the scenes", so *mixed initiative* can be introduced to let the user actively request assistance [10]. However, the problem remains that such filters are passive in the sense that they do not guide the search engine - it sends results as normal, then some operations are performed over those results; 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

In order to overcome the shortcomings of the search engine model highlighted above, we propose a concept which will rely on a combination of statistical learning techniques and human classification. In this combined search engine / web directory model users will be able to:

- Browse the web directory, satisfying their *information discovery* need.
- Search the whole database (same as a search engine), 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 user-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.
- Collaborate with others by sharing research sessions.

The web directory will use the data collected by a search engine’s web spider, so it will have the same amount of web information to classify as a search engine. It will necessarily be an extremely large structure, which is why machine learning methods are needed to build it. Since the hierarchy can be arbitrarily deep, the system can have relatively few documents per leaf node, so every document will be reachable by exploration/browsing. However, it will not be too difficult to navigate: if we suppose an average of 100 documents per leaf node and 10 branches per node, users would be able to browse something in the order of 100 billion documents with only ten navigational clicks.

#### A. Browsing the Directory

A simple mechanism exists for browsing the directory, which practically replicates the Open Directory (DMOZ): a static hierarchical structure, where we only added two improvements. The user has an additional sorting option: to see more *typical* documents first, where ordering is by the classifier scores for each instance. Thus sorted on top are documents that (according to the classifier) match the category best, or in other words are more typical and representative for it. Another 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 for the end user, and training a classifier on only 6 instances is practically impossible. To address this, we “folded” categories to include all instances from descendant nodes as well (the same approach was used in other projects based on Yahoo! Directory [11] [7], where category fragmentation is even worse).

In every category we also display additional documents downloaded by our spider, which have not been manually classified but belong to the category according to the classifier. In a real-world implementation, this should strengthen

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. The user can start exploring the web by submitting an optional query in the form of a short text or a whole document. It is optional because the user can, alternatively, just start browsing the directory (i.e. - have an empty query); in both cases, this query can later be expanded (see *Query and Query Expansion* below).

Results returned to the user come in the form of a path through the directory tree, together with some document listings. The user’s query is treated as a document and is passed through the same pre-processing filter that the backend classifier uses, 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). This gives the user a way to correct the system, by following not the suggested path through the directory but an alternative path; even if the classifier is correct, the user can still “wander off” by clicking on an alternative link in order to explore areas not actually fitting the query, but still somewhat relevant to it.

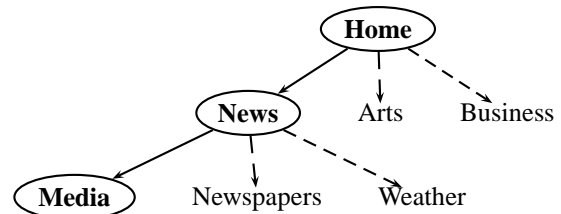


Figure 1: Classifier proposes path through directory

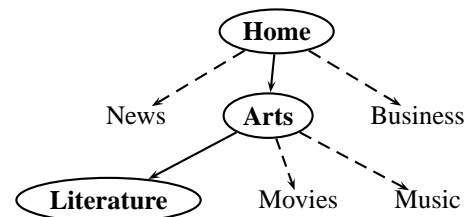


Figure 2: User makes correction, gets new proposed path

Together with each suggested category, the system also offers a number of documents from it. They can be ordered in a number of ways between which the user can choose:

- The default is *Editor’s choice*: a human has decided which documents are most important for a category.
- The *Typical* ordering lists documents by their fitness to the category (based on the classifier score); seeing the most typical documents of the category assists the user in deciding whether it is relevant to his query or not.

- *Relevant* documents first: this lists documents in order of relevance to the user’s query. It uses the *Search* mechanism (see below), performing a local search with the whole complex *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 and uses the same mechanism which orders *exploration* results by relevance. It is based on an *inverted index* such as search engines use. This index is the inverted version of the document description table: instead of a list of all words contained in a document, it indexes all documents that contain a word.

In our implementation though this is not a flat occurrence list, where values are either *yes* or *no* (contains or doesn’t contain) or number of occurrences. We have a normalized TF\*IDF weight for each word for each document, where 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 we can say for example: word  $w_i$  is twice more important for document  $D_j$  than for document  $D_k$ . We use these values to calculate the relative scores of each candidate document and order them by relevance to the search query. Every query term is also weighted by TF\*IDF before we multiply it by this score.

TF\*IDF is used to discount common, or “stop” words and to emphasize distinctive words. However, a word is a stop-word or distinctive in some context. The word *software* for example can provide a good decision boundary when splitting IT-related from non-IT-related 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 and search in just one of them. In a real-world scenario, this could easily scale to millions of such fragments, allowing the user to select from millions of different contexts with only several clicks.

### D. Query and Query Expansion

A major feature of the system is that the user’s query can be expanded by relevance feedback. In the result list the user can mark some results as either relevant or not relevant and add them to the query. When the user clicks on a link next a document, all the (weighted) keywords contained in the document are added to the query. The user also has a list of those added documents which can be edited. This is a one-click query expansion where the user is not required to enter any keywords - the system supplies them instead. Since some of the added documents are positive (relevant) and others are negative (not relevant) examples, the query is necessarily a complex object with a positive and negative component which have to be weighted. We do not use RSJ weighting [12] because it assumes binary index descriptions of documents and we would prefer a more accurate document representation vector. Instead, we use a Rocchio relevance feedback weighting scheme [13]:

$$Q = \alpha Q_u + \beta Q_p - \gamma Q_n \quad (1)$$

where the query  $Q$  fed to the classification algorithm consists of the original user query  $Q_u$  plus the query vector  $Q_p$  from positive feedback (documents the user marked as relevant) minus the negative query  $Q_n$  (documents marked as non-relevant). Those vectors are calculated as  $Q_p = \frac{\sum D_p}{n_p}$  and  $Q_n = \frac{\sum D_n}{n_n}$ , where  $D$  are the respective document vectors and  $n$  is their cardinality.  $\alpha$ ,  $\beta$  and  $\gamma$  are weights signifying how important each component is. The defaults are  $\alpha = 1$  and  $\beta = \gamma = 0.5$ , i.e. the original query is as important as the whole relevance feedback, and the positive and negative feedback components are equally important. Users can adjust these values on a case-by-case basis though.

Unlike the ARCH architecture [7], our query expansion works on the document level and not the category level. Where ARCH adds a feature vector derived from the “concept” (category), we only add the feature vector for a particular document. This gives control to the user and is not only more precise (provides several orders of magnitude more granularity), but is also realistic. Documents contain a couple of hundred words each, while a category has hundreds of thousands (we derived 127,830 unique terms from our fairly small sample of the *Business* category of DMOZ; a real-world scenario would be much worse): apparently, an expansion adding hundreds of thousands of terms would dilute the user’s initial query so much as to make it irrelevant, not to mention computation costs.

Also, unlike *incremental relevance feedback* [14], we do not limit the size of any of the query components; this limitation was intended to accommodate streaming filters and not a large-scale classifier as our system. The price we have to pay is slower processing, but - as discussed - we

prefer to make no assumptions on behalf of the user, so we do not assume the user wants a “fairly good” answer fast; we aim to provide as good an answer as possible even if it will be slower. The user though has the option to select the faster algorithm (see below) which will tilt the compromise towards faster but more inaccurate search.

Our *research query* is very different from a standard search engine query, in that a) it can be arbitrarily long (though of course orders of magnitude less than a concept-based expanded query), and b) it is not a flat list of words but an array of the type *word|weight*; weight values can also be negative as result of negative feedback. This query is treated as a constructed document which summarizes what the user is searching for; we call this a *research query* to distinguish it from standard search queries. This dummy document is passed through the hierarchical classifier structure where all terms in it are taken into account when calculating the result; 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. Enhancements

Some other improvements we have introduced can be considered usability enhancements and not real innovation, but they help our exploration engine become user-centric and not server-centric as the dominant current model.

Such is the *saved researches* feature, which allows users to save and re-use later research queries they have developed. Each query may include search terms, relevance feedback, start category (the user can skip the top levels of the directory and start at a specific node) and related bookmarks.

In terms of collaboration, we have implemented a feature which allows users to make their *researches* public. Other users can load such research sessions, modify them and then save them as their personal researches.

We also need user collaboration to help develop the web directory itself. Our experiments with Open Directory Project data show that the amount of freely available labeled data is not enough to train a realistic hierarchical classifier for the volumes of documents we want to categorize. Furthermore, even if we did train our classifier on such data, it would need constant re-tuning and corrections in the future due to the constantly changing nature of the web [15].

One way to tackle this is to employ editors to classify resources, which is not economically attractive. An alternative approach is to use volunteer editors, but the state of DMOZ shows that this doesn’t work too well.

We propose a collaborative filter instead, where every user will be a small-time editor. By providing feedback on whether a document is a good fit to a directory or not, millions of users can easily help train our classifiers to a more satisfactory level. This feature has not been implemented in our prototype yet; we have only provided a feedback mechanism for users to report errors or spam,

but these are manually verified by us before change in classification occurs.

## IV. IMPLEMENTATION DETAILS

We needed to adapt some standard methods in order to account for the specifics of a constantly evolving system where both the underlying data and classification change constantly. We built a proof-of-concept prototype and tested a number of approaches and algorithms from which we selected the best-fitting to the task.

### A. Preprocessing

We pass all documents through a filter which strips HTML tags, then discards very short and very long phrases (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), while more than 20 words without punctuation indicate a machine-generated list of keywords and not a human-written text. We also filter out too short and too long words (less than 3 and more than 20 characters long).

Words are not stemmed or pre-processed in any other manner. There are a number of justifications for this decision [16], the main one being that linguistic processing has to know the structure of the language. In our case this would mean a different pre-processor for every different language sub-tree of the directory (currently numbering 81), so we decided to skip it, incurring higher computational costs.

Since the data is extremely high-dimensional, the usual approach would be to apply dimensionality reduction. However, we decided against this and work with the raw data. The reasoning is 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. Evolving the document collection would mean 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) all classifiers 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; since we only have an average of 13.43 classes per classifier, 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 hierarchy we need separate dimensionality reduction since document collections there are different.

## B. Hierarchical Classification

For our proof-of-concept prototype, 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 downloaded 149,178 documents from these URLs and trained a hierarchical structure of Multinomial Naïve Bayesian classifiers using these documents as training data. We downloaded a further 9,517 documents for validation, plus some others to expand the search database (the latter are not taken into account when benchmarking algorithms since we don't have manual labels for them). DMOZ data is very noisy (classification errors, web decay etc.); we did not perform a clean-up so as not to skew experiment data, hence the relatively low accuracy results reported. We expanded classification only over the top three levels of the English-language directory, with a total of 474 classifiers and 6,368 classes.

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 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 (this applies only for instances that are not manually labeled as belonging to a class, which can only be moved by a human editor). 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 (as discussed above) they are stop words in some context only.

## C. Classification Algorithm

We chose Multinomial Naïve Bayes as our main classification tool because it learns fast, generalizes well and is not too expensive computationally.

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. It classifies using *maximum likelihood*, where the winning class is found as:

$$\begin{aligned} \operatorname{argmax}_c p(C = c) &= \ln \frac{p(C|D)}{p(-C|D)} = \\ &= \ln \frac{p(C)}{p(-C)} + \sum_i \ln \frac{p(w_i|C)}{p(w_i|-C)} \end{aligned} \quad (2)$$

where  $C$  is the class the instance belongs to,  $c$  is each class,  $p(C)$  is the *prior probability* of a document for a class and  $p(w_i|C)$  is the prior probability of the  $i$ -th word

for that class,  $i$  including all words in the document (this sum is the *document evidence*).

Unfortunately, this “classic” MNB suffers badly when classes are unbalanced, which turns out very much to be the case; for example, the largest top-level category (*Regional*) of DMOZ has more than 1.1 mln instances, while the smallest category at the same level (*News*) has only 9,000 - more than 100 times less. Prior distributions are so skewed that the classifier tends to classify every instance in the dominant class whatever the *document evidence* is, achieving almost 100% accuracy in this class, while it puts nothing in the smaller classes resulting in exactly zero accuracy in some. The method's main naïve assumption, feature independence, is also badly violated by data on the web, a sample of which we downloaded. In reality features are not independent but follow a double Pareto distribution [17].

We apply several mitigation measures to account for the various problems.

First, we discount word counts by standard TF\*IDF [18].

In calculating it, we apply word count normalization [19] to compensate for the varying average length of documents between classes (the *Business* category for example has an average of 96.54 unique words per document, while in *News* there are 235.05):

$$n'_{wd} = \alpha \times \frac{n_{wd}}{\sum_{w'} \sum_{d \in D_c} n_{w'd}} \quad (3)$$

where  $n_{w'd}$  are class-specific word counts (occurrences of the word in documents  $d$  of the part of the corpus  $D_c$  belonging to class  $c$ ); we took the smoothing parameter  $\alpha$  (vector length measured in the  $l_1$  norm) to be equal to 1, as that was shown to work well [19]. This improved the situation dramatically in terms of overall accuracy, although classification errors still varied widely between classes (see results for algorithm WN in comparison table).

Term frequency distribution also varies greatly between classes, so we normalize these values by the  $l_2$  norm which provides some “subtle benefits” for classification [20] (in our experiments - to the tune of a 2% increase in accuracy).

In  $f_i = \frac{n_i}{\sum_k n_k}$  (term frequency of word  $i$  equals its count divided by word counts of all  $k$  words in document), we normalize  $f_i$  as  $f'_i = \frac{f_i}{\sqrt{\sum_k (f_k)^2}}$  and use this value as the

TF part of our TF\*IDF estimation. The overall accuracy thus achieved was more acceptable, but classification errors still displayed great variation between classes.

We then adopted an approach used by spam filters: a *train on error* policy, meaning that the classifier trains only on documents which it misclassified. In spam filters this is an online process, since they work on a stream of documents. In our case it means that we have to do a number of passes over the whole document collection, so there appears an element of the algorithm *converging* to a stable state after several

passes (owing to our next adaptation as well), unlike the original which is static. The main effect of this *train on error* policy is that the classifier in fact sees only “borderline” documents, which represent a small subset of the data collection (and the better it gets, the smaller this subset becomes). The approach is counter-intuitive and skews word counts significantly, but has been shown in practice to work best so it’s being used in most industrial spam filters [21].

Lastly, we calculate class prior distributions dynamically: over the last 10 000 errors and not over the whole document collection [16]. In effect, this introduces a negative feedback loop in the system: classes where the classifier has problems become over-represented in this stochastic sample so the probability for it to place a new instance in them becomes higher and compensates for the problems; note that the system does not have 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.

A side effect of repetitive passes over the collection is that word counts would grow indefinitely if no further modification was made. We apply time decay of word counts: divide by 2 after every pass, round up to the nearest integer and delete those that remain equal to 1. This introduces an element of *unlearning* which is extremely important, having in mind that not only our document collection is dynamic, but classification labels change as well (editors may move a document from one class to another). Any knowledge not reinforced by recent examples is eventually forgotten, while normal classification is not affected since time decay is applied proportionately to all weights: it changes the numbers but their proportions remain the same.

In terms of complexity, the new algorithm is still linear to the number of instances, as the original, but training is several times slower; where the original needs one pass over the data in order to learn its statistics, the new classifier needs several passes until it converges to a stable state. Our experiments show 6 to 10 iterations to be enough for this to happen (with larger collections it would be faster). Classification is marginally slower due to the dynamic recalculation of priors (this element can be optimized by caching). Another drawback is that the algorithm does not generalize too well, which has to be overcome by more training. Since this is offline though, it will not affect users.

In conclusion, we have a classifier which is equally reliable in all classes and is robust in the face of the many small changes in the web and our view of it (as represented by the opinion of human editors). We call this classifier MNB-SPDA (Multinomial Naïve Bayes - Stochastic Prior Distribution Adjustment). We use it for the backend of our system to (offline) pre-process documents into the directory hierarchy, as well as in the frontend to classify user queries.

#### D. Alternative Classification Algorithm

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. We do this based on category TF\*IDF values for all terms:

$$TF * IDF_{ic} = TF \times IDF = \frac{n_i}{\sum_k n_k} \times \log \frac{D}{d: n_i \in d_c} \quad (4)$$

where  $TF * IDF_{ic}$  is the value for a word in a category,  $n_i$  is the number of documents in that category  $d_c$  that contain the word,  $n_k$  is the number of all word occurrences in the category and  $D$  is the total number of documents. Normally, it should not be  $d_c$  in the denominator the number of categories. However, in a big enough collection (such as the web) a word would most probably be seen in every category at least once, so this value would become meaningless. Using  $d_c$  instead manages to extract the most meaningful terms for every category.

After we calculate this value for every word for every category, we end up with values which are not comparable since they are not normalized; this is a hard task given all the broken independence assumptions discussed above. So we “normalize” with a brute-force method: in every category we sort terms in order of ascending  $TF * IDF_{ic}$ , then substitute the value with the order number. Thus, if we have a dictionary of 100,000 terms, the least important of them for a category would have a value of 0 in that category and the most important would have a value of 100,000. To classify, we just sum these values for every category and find the winner as the category with the highest count. The method is crude, but turns out to be practical for the same reason MNB works: it only needs to find the winner and not to estimate probabilities precisely. It is fast because we sum integers as opposed to summing logarithms of floating-point values in MNB.

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

Table I: Comparison between classic MNB, Word count Normalized, SPDA and Discretized Centroid Classifier. **Bold** is best, *italic* is second-best.

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 [16]). On the other hand, it is more than five times faster so 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 for their query, then switch to the precise method further into their research.

## V. CONCLUSION AND FUTURE WORK

In this paper we propose a *web exploration engine* as an alternative information-finding model. While current systems try to guess algorithmically search or user context by implicit feedback, we propose a large number of pre-computed contexts from which the user can easily select one. The model allows users to not only research a topic but also 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 web documents have been pre-classified in a large number of categories; users can explore them 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 a category.
- In contrast to search engines, in this model users can explore the web without any keywords, guiding the exploration engine with 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] M. Arrington, "The End Of Hand Crafted Content," *Tech Crunch*, Dec. 2009.
- [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] 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.
- [4] N. Milic-Frayling, R. Sommerer, and K. Rodden, "WebScout: Support for Revisitation of Web Pages within a Navigation Session," in *WI '03: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*. Washington, DC, USA: IEEE Computer Society, 2003, p. 689.
- [5] Vivisimo, "Tagging vs. Clustering in Enterprise Search," online, Aug. 2006, [www.vivisimo.com/html/download-tagging](http://www.vivisimo.com/html/download-tagging).
- [6] 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.
- [7] 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.
- [8] 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.
- [9] D. Godoy and A. Amandi, "PersonalSearcher: An Intelligent Agent for Searching Web Pages," in *In Advances in Artificial Intelligence*. Springer, 2000, pp. 43–52.
- [10] 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.
- [11] 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.
- [12] S. E. Robertson and S. K. Jones, "Relevance weighting of search terms," *Journal of the American Society for Information Science*, vol. 27, no. 3, pp. 129–146, 1976.
- [13] J. Rocchio, *Relevance Feedback in Information Retrieval*. Englewood Cliffs, NJ: Prentice Hall, 1971, pp. 313–323.
- [14] J. Allan, "Incremental Relevance Feedback for Information Filtering," in *Proceedings of the 19th International Conference on Research and Development Information Retrieval (SIGIR 96)*. ACM Press, 1996, pp. 270–278.
- [15] D. Fetterly, M. Manasse, M. Najork, and J. Wiener, "A Large-Scale Study of the Evolution of Web Pages," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM, 2003, pp. 669–678.
- [16] 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.
- [17] F. Chierichetti, R. Kumar, and P. Raghavan, "Compressed Web Indexes," in *18th International World Wide Web Conference*, Apr. 2009, pp. 451–451.
- [18] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for IDF," *Journal of Documentation*, vol. 60, no. 5, pp. 503–520, 2004.
- [19] 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.
- [20] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the Poor Assumptions of Naïve Bayes Text Classifiers," in *Proceedings of the Twentieth International Conference on Machine Learning*, 2003, pp. 616–623.
- [21] J. A. Zdziarski, *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*. No Starch Press, July 2005.