Using taxonomy, discriminants, and signatures for navigating in text databases*

Soumen Chakrabarti   Byron Dom   Rakesh Agrawal   Prabhakar Raghavan

IBM Almaden Research Center

Abstract

We explore how to organize large text databases hierarchically by topic to aid better searching, browsing and filtering. Many corpora, such as internet directories, digital libraries, and patent databases are manually organized into topic hierarchies, also called *taxonomies*. Similar to indices for relational data, taxonomies make search and access more efficient. However, the exponential growth in the volume of on-line textual information makes it nearly impossible to maintain such taxonomic organization for large, fast-changing corpora by hand.

We describe an automatic system that starts with a small sample of the corpus in which topics have been assigned by hand, and then updates the database with new documents as the corpus grows, assigning topics to these new documents with high speed and accuracy.

To do this, we use techniques from statistical pattern recognition to efficiently separate the *feature* words, or *discriminants*, from the *noise* words at each node of the taxonomy. Using these, we build a multi-level classifier. At each node, this classifier can ignore the large number of "noise" words in a document. Thus the classifier has a small model size and is very fast. Owing to the use of context-sensitive features, the classifier is very accurate. As a by-product, we can compute for each document a set of terms that occur significantly more often in it than in the classes in which it belongs.

We describe the design and implementation of our system, stressing how to exploit standard, efficient relational operations like sorts and joins. We report on experiences with the Reuters newswire benchmark, the US Patent database, and web document samples from Yahoo!. We discuss applications where our system can improve searching and filtering capabilities.

1 Introduction

The amount of on-line data in the form of free-format text is growing extremely rapidly. As text repositories grow in number and size and global connectivity improves, there is a pressing need to support efficient and effective information retrieval (IR), search and filtering. A manifestation of this need is the recent proliferation of over one hundred commercial text search engines that crawl and index the web, and several subscription-based information multicast mechanisms. Nevertheless, there is little structure on the overwhelming information content of the web.

It is common to manage complexity by using a hierarchy[1], and text is no exception. Many

*A preliminary version of this work appeared in International Conference on Very Large Data Bases, August 1997, Athens, Greece. Submitted by invitation to the Very Large Data Base Journal on February 4, 1998.

[1] A hierarchy could be any directed acyclic graph, but in this paper we only deal with trees.

internet directories, such as Yahoo![2], are organized as hierarchies. IBM's patent database[3] is organized by the US Patent Office's class codes, which form a hierarchy. Digital libraries that mimic hardcopy libraries support some form of subject indexing such as the Library of Congress Catalogue, which is again hierarchical.

We will explore the opportunities and challenges that are posed by such topic hierarchies, also called *taxonomies*. As we shall show, taxonomies provide a means for designing vastly enhanced searching, browsing and filtering systems. They can be used to relieve the user from the burden of sifting specific information from the large and low-quality response of most popular search engines [9, 35]. Querying with respect to a taxonomy is more reliable than depending on presence or absence of specific keywords. By the same token, multicast systems such as PointCast[4] are likely to achieve higher quality by registering a user profile in terms of classes in a taxonomy rather than keywords.

The challenge is to build a system that enables search and navigation in taxonomies. Several requirements must be met. First, apart from keywords, documents loaded into such databases must be indexed on *topic paths* in the taxonomy, for which a *reliable* automatic hierarchical classifier is needed. As one goes deep into a taxonomy, shared jargon makes automatic topic separation difficult. Documents on stock prices and on impressionist art look very different to us, but may be carelessly filed as "human affairs" by a Martian. Second, the taxonomy should be used also to present to the user a series of progressively refined *views* of document collections in response to queries. Third, the system must be *fast*, especially since it will often be used in conjunction with a crawler or newswire service. Fourth, the system must efficiently *update* its knowledge when it makes mistakes and a human intervenes.

We describe such a taxonomy-and-path-enhanced-retrieval system called TAPER. For every node in the taxonomy, it separates *feature* and *noise* terms by computing the best *discriminants* for that node. When classifying new documents, only the feature terms are used. Good features are few in number, so the class models are small and the classification is speedy. In contrast to existing classifiers that deal with a flat set of classes, the feature set changes by context as the document proceeds down the taxonomy. This filters out common jargon at each step and boosts accuracy dramatically. Addition and deletion of documents is easily handled and discriminants recomputed efficiently. The text models built at each node also yield a means to summarize a number of documents using a few descriptive keywords, which we call their *signature* (these are distinct from, and not necessarily related to, the features).

In addition to the algorithmic contributions, we describe the design and implementation of our system in terms of familiar relational database idioms. This has the following benefits. Our system can handle extremely large numbers of classes, documents and terms, limited essentially by word size of the computer. It adapts to a wide range of physical memory sizes, while maximizing fast sequential scans on disk when needed. Moreover, our design leads to insights into how database text extenders can exploit the core relational engine.

We report on our experience with TAPER using the Reuters newswire benchmark[5], the US patent database, and samples of web documents from Yahoo!. Depending on the corpus, we can classify 66–87% of the documents correctly, which is comparable to or better than the best known numbers. We can process raw text at over seven megabytes a minute on a 133 MHz RS6000/43P

[2] `http://www.yahoo.com`
[3] `http://www.ibm.com/patents`
[4] `http://www.pointcast.com`
[5] `http://www.research.att.com/~lewis`

with 128 MB memory. Our largest training runs, taking up to a day (including fetching web pages), was with over 2,100 topics, over 266,000 URLs, and over 600,000 unique terms/tokens, is the most extensive web page classification experiment known to us.

Organization of the paper. In Section 2 we demonstrate that using a taxonomy, topic paths, and signatures can greatly improve retrieval. Next, in Section 3 we study the problems that must be solved to provide the above functionality. The problems are hierarchical classification, feature selection, and document signature extraction. These are explored in detail in Sections 3.2, 3.3, and 3.5 respectively. Section 3.4 describes the architecture of our prototype. The proof of quality of signatures is necessarily anecdotal at this point; some examples can be found in Section 2. More rigorous evaluation of feature selection and classification is presented in Section 4. Related work is reviewed in Section 5 and concluding remarks made in Section 6.

2 Capabilities and features

We discuss two important contexts in which accurate, high-resolution topic identification is needed: querying and filtering.

Querying: Most queries posted to search engines are very short. Such queries routinely suffer from the *abundance* problem: there are many aspects to, and even different interpretations of the keywords typed. Most of these are unlikely to be useful. Consider the wildlife researcher asking AltaVista[6] the query `jaguar speed` [9]. A bewildering variety of responses emerge, spanning the car, the Atari video game, the football team, and a LAN server, in no particular order. The first page about the animal is ranked 183, and is a fable. Thwarted, we try `jaguar speed -car -auto`. The top response goes as follows: "If you own a classic Jaguar, you are no doubt aware how difficult it can be to find certain replacement parts. This is particularly true of gearbox parts." The words `car` and `auto` do not occur on this page. There is no cat in sight for the first 50 pages. We try LiveTopics[7], but at the time of writing, all the clusters are about cars or football. We try again: `jaguar speed +cat`. The top two hits are about the clans Nova Cat and Smoke Jaguar; then there is LMG Enterprises, fine automobiles. All these pages include the term `cat` frequently. The 25th page is the first with information about jaguars, but not exactly what we need. Instead, we can go to Yahoo!, and visit the likely directory Science:Biology, and query `jaguar`. This takes us to Science:Biology:Zoology:Animals:Cats:Wild_Cats and Science:Biology:Animal_Behavior, but we could not find a suitable page about jaguars there.

Filtering: Another paradigm of information retrieval is *filtering*, in which a continual stream of documents are generated on-line, as in newsgroups and newsfeed. The system collects *interest profiles* from users and uses these to implement either content-based or collaborative filtering, i.e., it notifies the user only of the documents they are likely to be interested in [19, 41, 4, 29].

In its simplest form, a profile may be a set of terms and phrases specified explicitly by the user. This has the same problem as querying without topic context as discussed above. A better notion of a profile is the set of documents the user has seen and/or liked, perhaps with scores. This is a fine-grained characterization of the user, and may work well with small systems, but for

[6] `http://www.altavista.digital.com`
[7] `http://www.altavista.digital.com/av/lt/help.html`

thousands of users and the web at large, a system storing this level of detail will not scale. A promising alternative is to characterize profiles not at the individual document level, but at the level of narrow but canonical topics, such as the taxonomy of Yahoo!. Such an approach is used in Surf Advisor, a collaborative recommendation system [30].

We identify a few core capabilities from these examples. Text search must be based on topical context as well as keywords. Once the broad topic of a document is known, it can be characterized by terms that are frequent compared to the topic model. To identify document topics, an automatic classification system needs to separate words that contain topic information and words that are essentially noise. We discuss these at length next.

2.1 Querying in a taxonomy

In a database that associated not only keywords but also topics with documents, the query `jaguar speed` could elicit not a list of documents, but a list of topic paths:

Business_and_Economy:Companies:Automotive
Recreation:
  Automotive
  Games:Video_Games
  Sports:Football
Science:Biology:Animal_Behavior

The user can now restrict queries by *concept*, not by *keyword*. Using samples, it is possible to show the above response even as the user types the query, *before* actually issuing a search. At this point, the user can restrict the search to only a few topic paths. The artificial limit to the length of the response list from search engines, together with cars and video games, will not crowd out the cat. As we have shown above, enforcing or forbidding additional keywords cannot always be as effective. If new documents can be binned into these topic paths in real-time, this ability may be very useful for multicast channels as well. User profiles will be topic paths rather than keywords.

2.2 Context-sensitive signatures

An exhaustive keyword index, as in AltaVista, is perhaps more of a problem than a solution. A single occurrence of a term in a document, even one that is not a "stopword," no matter how useless an indicator of the contents, is indexed. The IR literature has advanced further; there exist prototypes that extract *signature* terms which are then used for indexing. These signatures can also be used as summaries or thumbnails; their descriptive power can often compare favorably with that of arbitrary sentences as extracted by popular search engines. They are also effective for describing a document cluster [2].

We claim that the common notion of a document abstract or signature as a function of the document alone is of limited utility. In the case of a taxonomy, we argue that a useful signature is a function of both the document and the reference node; the signature includes terms that are "surprising" *given* the path from the root to the reference node. In the above example, *car* and *auto* may be good signature terms at the top level or even at the Recreation level, but not when the user has zoomed down into Recreation:Automotive. Here is another illustration from a document[8] in Health:Nursing that starts like this:

[8] `http://www2.best.com/~goodnews/practice/faq.htm`

Beware of the too-good-to-be-true baby that is sleeping and sleeping and doesn't want to nurse. Especially monitor the number of wet diapers, as seriously jaundiced babies are lethargic.

The first-level classification is Health. We can compute the top signature terms with respect to Health as:

Jaundice, dampen, dehydration, lethargic, hydrate, forcibly, caregiver, laxative, disposable.

This tells us the document is about treating jaundice. The second level classification is Health:Nursing. Shifting our reference class, we compute the new signature to be:

Baby, water, breast-feed, monitor, new-born, hormone.

Now we know the document is about nursing *babies*; this information comes from both the path and the signatures. In Section 3.5 we shall propose some means of computing context-sensitive signatures. Thus, significant improvement in search quality may be possible by maintaining functionally separate indices at each taxonomy node, using only a few signature terms from each document.

Another application of context-sensitive signatures is finding term associations. Using phrases for search and classification can potentially boost accuracy. The usual way to find phrases is to test a set of terms for occurrence rate far above that predicted by assuming independence between terms. Unfortunately, associations that are strong for a section of the corpus may not be strong globally and go unnoticed. E.g., *precision* may be visibly associated with *recall* in a set of documents on information retrieval, but not in a collection also including documents on machine tools. Computing signatures at each node exposes all such associations.

2.3 Context-sensitive feature selection

Separating feature terms from noise terms is central to all of the capabilities we have talked about. In the above examples, *car* and *auto* should be "stopwords" within Recreation:Automotive and hence be pruned from the signatures. Feature and noise terms must be determined at each node in the taxonomy.

It is tricky to hand-craft the stopwords out of domain knowledge of the language; *can* is frequently included in stopword lists, but what about a corpus on waste management? The contents of a stopword list should be highly dependent on the corpus. This issue looms large in searching using categories and clusters. In hierarchical categories, the importance of a search term depends on the position in the hierarchy [35].

In Section 3 we will design an efficient algorithm to find, for each node in the taxonomy, the terms that are best suited for classifying documents to the next level of the taxonomy. Conversely, we detect the noise words that are of little help to distinguish the documents. We reuse the term "feature-selection" from pattern recognition to describe this operation.

Feature selection enables fine-grained classification on a taxonomy. For diverse top-level topics, a single-step classifier suffices. But as a document is routed deep into a taxonomy, shared jargon makes sophisticated feature selection a necessity. Together with feature selection, we have to pick models for each class and a classifier. Many options have been evaluated [40]. In spite of its simplicity, naive Bayesian classifiers are often almost as accurate as more sophisticated classifiers [24]. For a fixed number of features, naive Bayes is faster than more complex classifiers. However, to approach the latter in accuracy, naive Bayes typically needs many more features.

Finding feature terms for each node mitigates this problem. Often, fewer than 5–10% of the terms in the lexicon suffice to discriminate between documents at any node in the taxonomy. This can greatly speed up classification. The need for fast multi-level classification is not restricted to the time a text database is populated. With increasing connectivity, it will be inevitable that some searches will go out to remote sites and retrieve results that are too large for direct viewing. There are already several "meta-search" tools that forward queries to a number of search engines and combine the results; we have seen how a hierarchical view is much better.

Applications of feature selection: Feature selection is useful in any setting where salient distinctions are sought between two or more sets of documents. Consider the scenario where a set of documents (e.g., a keyword query result) has been clustered into subsets, and we wish to annotate the clusters with salient keywords. We can regard the clusters as given classes, and use feature selection to find these keywords. Other example applications include differentiating between patents filed by two companies, or by the same company at different times (to expose any trend). We shall see some examples of such applications in Section 4.

3 Models, algorithms and data structures

In this section we will deal with the core components of our system. Consider first the task of computing the terms that induce the best distinction between the sub-topics of a given topic. To do this we have to find terms that occur *significantly* more frequently in some sub-topics compared to others, as against those that show this property "by chance" owing to a finite sample. We can make such judgements only on the basis of some statistical model of document generation. This we discuss in Section 3.1. The model leads to a natural classification procedure, described in Section 3.2. Finding good features for the classifier to use is discussed in Section 3.3. Performing all the above functions efficiently on large databases ranging into tens to hundreds of gigabytes raises several performance issues that are discussed in Section 3.4; we also give details of our data structures and implementation there. Finally, in Section 3.5 we discuss how to use the class models to extract context-sensitive document signatures.

3.1 Document model

There have been many proposals for statistical models of text generation. One of the earliest indicators of the power of simple rules derived from both quantitative and textual data is Zipf's law [49]. The models most frequently used in the IR community are Poisson and Poisson mixtures [37, 43]. (If X is distributed Poisson with rate μ, denoted $X \sim \mathcal{P}(\mu)$, then $\Pr[X = x] = e^{-\mu}\mu^x/x!$ and if Y is distributed Bernoulli with n trials and mean np, denoted $Y \sim \mathcal{B}(n,p)$, then $\Pr[Y = y] = \binom{n}{y}p^y(1-p)^{n-y}$. As $n \to \infty$ and $p \to 0$, the distributions $\mathcal{B}(n,p)$ and $\mathcal{P}(np)$ converge to each other.) We will assume a Bernoulli model of document generation for most of the paper. In this model, a document d is generated by first picking a class. Each class c has an associated multi-faced coin[9]; each face represents a term t and has some associated probability $\theta(c,t)$ of turning up when "tossed."

Conceptually, as the training text is being scanned, our classifier database will be organized as a three-dimensional table. One axis is for *terms*; their string forms being replaced by 32-bit ID's,

[9]We use the term *coin* here for what is perhaps better called a *die*.

we call them TID's and denote them t in formulae. The second axis is for documents; these are called d in formulae. The third axis is for *classes* or topics. Topics have a hierarchy defined on them; for this paper we will assume a tree hierarchy. These classes are also assigned ID's and called CID's; we denote them c in formulae.

The measure maintained along these dimensions (t, d, c) is called $n(t, d, c)$, which is the number of times t occurs in $d \in c$. This number is non-zero only when $t \in d \in c$. $t \in d$ means term t occurs in document d, and $d \in c$ means d is a sample document in the training set for class c. A superclass of c, i.e., an ancestor in the topic tree, inherits all $d \in c$.

Aggregations along the dimensions give some important statistics about the corpus which are used by our classifier.

- The *length* of document d is given by $n(d, c) = \sum_t n(t, d, c)$. The length of all documents can be found using a `GROUP BY` on (d, c).
- The total length of training documents in class c, denoted $n(c)$.
- The total number of times term t appeared in training documents of class c.
- The *fraction* of times t occurs in $d \in c$, i.e., $f(t, d, c) = n(t, d, c)/\sum_t n(t, d, c)$. This can be computed from the above but we materialize this value for efficiency reasons. We will need the sum of f and f^2 over documents in a class as explained later.
- The *number* of training documents in class c that have at least one occurrence of a specific term t. This is denoted $m(t, c)$.
- The number of training documents in class c, denoted $|c|$.

We will describe the details of arranging this table later in Section 3.4.

Assuming the Bernoulli model with parameters $\theta(c, t)$,

$$\Pr[d|c] \quad = \quad \binom{n(d)}{\{n(d,t)\}} \prod_t \theta(c, t)^{n(d,t)}, \tag{1}$$

where $\binom{n(d)}{\{n(d,t)\}} = \frac{n(d)!}{n(d,t_1)!\,n(d,t_2)!\cdots}$ is the multinomial coefficient. The above Bernoulli model makes the assumption that the term occurrences are *uncorrelated*, which is certainly not correct. First, given a term has occurred once in a document it is more likely to occur again compared to a term about which we have no information. Second, the term frequency distributions are correlated.

Our independence assumption leads to what is called a *naive Bayes* classifier. (A naive Bayes classifier in essence builds density functions for each class that are marginally independent, and then classifies a data point based on which density function has the maximum value at that point.) In practice, these simple classifiers perform surprisingly well compared to more sophisticated ones that attempt to approximate the dependence between attributes.

Recently this phenomenon has been investigated in depth by Friedman [17]. A classifier that uses an estimate of class densities is subject to *bias* (decision boundaries that are shifted from the "bes t" position, because the model is inaccurate) and *variance* (decision boundaries fit to noisy data). Friedman analyzes how the low variance of naive density estimates can mitigate the high bias to give simple classifiers that can often beat more sophisticated ones. It will also be clear from Section 3.4 that this simplicity lets us design a system that can handle enormous problem sets.

Rare events and laws of succession: We return to the issue of estimating the model parameters $\theta(c, t)$, the rate at which term t occurs in documents of class c.

The average English speaker uses about 20,000 of the 1,000,000 or more terms in an English dictionary [36]. In that sense, many terms that occur in documents are "rare events." This means that with reasonably small sample sets, we will see zero occurrences of many, many terms, and will still be required to estimate a non-zero value of $f(c, t)$. The maximum likelihood estimate, $f(c, t) = n(c, t)/n(c)$, is problematic: a class with $f(c, t) = 0$ will reject any document containing t.

Finding such estimates, also called *laws of succession*, has been pursued in classical statistics for centuries. Laplace showed that given the results of n tosses of a k-sided coin, i.e., the number of times each face occurred, $n_1, \ldots, n_k$, the correct Bayesian estimate for the probability of face i, denoted $\Pr_L(i|\{n_i\}, n)$, is not n_i/n, but $\frac{n_i+1}{n+k}$ [26]. This is the result of assuming that all possible associated k-component vectors of face probabilities $(p_1, \ldots, p_k)$ are *a priori* equally likely. This is called the *uniform prior* assumption. The above value of $\Pr_L(i|\{n_i\}, n)$ is obtained by using Bayes rule and evaluating $\frac{1}{\Pr[n_i]} \int_0^1 \theta \Pr[n_i|\theta] d\theta$. Alternative priors have been suggested and justified. We experimented with many of these, and found that Laplace's law wins by a few percent better classification accuracy all the time. For lack of space, we refer the reader to Ristad's paper for details [36]. With this adjustment, (and returning to our earlier notation) $f(c, t)$ is estimated as $(1 + n(c, t))/(n(c) + L(c))$, where $L(c)$ is the size of the lexicon of class c.

The binary model: Thus far, our model has been quite sensitive to the number of times a term is repeated in a document. The obvious criticism is that given a term has occurred once we expect it to occur again and again. A model at the other extreme is the binary model, in which the repetition count is ignored. The parameter $\theta(c, t)$ then becomes the fraction of documents in class c that contained t at least once. Laplace's correction can be applied here as well.

3.2 Hierarchical classification

A *classifier* inputs a document and outputs a class. If the class is not the one from which the document was generated, we say the classifier *misclassified* that document. Typically, a classifier is *trained* by giving it example documents with class labels attached.

Our system has a classifier at each internal node in the taxonomy, with diverse feature sets. Given a new document d, the goal is to find a leaf node c such that the posterior $\Pr[c|d]$ is maximized among all leaves. There is a danger in greedily picking one's way down the tree: an irrrevocable error may be made early in the process [24]. Let the path to a leaf c from the root be $c_1, c_2, \ldots, c_k = c$. Since the root subsumes all classes, $\Pr[c_1|d] = 1$ for all d. Furthermore, we can write $\Pr[c_i|d] = \Pr[c_{i-1}|d] \Pr[c_i|c_{i-1}, d]$, for $i = 2, \ldots, k$. Taking logs, $\log \Pr[c_i|d] = \log \Pr[c_{i-1}|d] + \log \Pr[c_i|c_{i-1}, d]$. Suppose in the taxonomy we mark edge (c_{i-1}, c_i) with the edge cost $-\log \Pr[c_i|c_{i-1}, d]$. We are then seeking the least-cost path from the root c_1 to some leaf.

Computing the one-step conditional probability $\Pr[c_i|c_{i-1}, d]$ is straight-forward. For notational convenience, name c_{i-1} as r_0 and its children $\{r_j\}$. Then the probability that the document d belongs to the child node r_i given that it belongs to the parent node r_0 is given by: $\Pr[r_i|r_0, d] = \Pr[r_i|d]/\Pr[r_0|d]$ where $\Pr[r_0|d] = \sum_j \Pr[r_j|d]$ (where $\sum_j$ is over all the siblings of r_i. Note that $\Pr[r_i|d] = \Pr[d, r_i]/\sum_j \Pr[d, r_j]$ by Bayes rule. If we use the Bernoulli model as before, $\Pr[d|r_j] = \binom{n(d)}{\{n(d,t)\}} \prod_t \theta(r_j, t)^{n(d,t)}$. Care is needed here with finite-precision numbers, because the probabilities are very small (often less than 10^{-5000}) and the scaling needed to condition the probability prevents us from maintaining the numbers always in log-form.

3.3 Feature selection

Now we will discuss how to select terms that the classifier will use in its models. Suppose we are given two sets of points in n-dimensional Euclidean space, interpreted as two classes. Fisher's method finds a direction on which to project all the points so as to maximize (in the resulting one-dimensional space) the relative class separation as measured by the ratio of inter-class to intra-class variance. More specifically, let X and Y be the point sets, and μ_X, μ_Y be the respective centroids, i.e., $\mu_X = (\sum_X x)/|X|$ and $\mu_Y = (\sum_Y y)/|Y|$. Further, let the respective $n \times n$ covariance matrices be $\Sigma_X = (1/|X|)\sum_X (x-\mu_X)(x-\mu_X)^T$ and $\Sigma_Y = (1/|Y|)\sum_Y (y-\mu_Y)(y-\mu_Y)^T$.

Fisher's discriminant method seeks to find a vector α such that the ratio of the projected difference in means $|\alpha^T(\mu_X - \mu_Y)|$ to the average variance, $\frac{1}{2}\alpha^T(\Sigma_X + \Sigma_Y)\alpha = \alpha^T \Sigma \alpha$ is maximized. It can be shown that $\alpha = \Sigma^{-1}(\mu_X - \mu_Y)$ achieves the extremum when Σ^{-1} exists. Also, when X and Y are drawn from multivariate Gaussian distributions with $\Sigma_X = \Sigma_Y$, this is the optimal discriminator in that thresholding on $\alpha^T q$ for a test point q is the minimum error classifier [48, 14].

Computing α involves a generalized eigenvalue problem involving the covariance matrices. In applications like signal processing where Fisher's discriminant is used, n is typically a few hundred at most; in the text domain, n is typically 50,000 to 100,000; and the covariance matrices may not be suitably sparse for efficient computation. Moreover, it is hard to interpret a discriminant that is a linear sum of term frequencies, possibly with negative coefficients!

Our approach will be to take the directions α as given, namely, a coordinate axis for each term. We assign each term a figure of merit, which we call its *Fisher index*, based on the variance figures above, which is $\frac{|\alpha^T(\mu_X - \mu_Y)|}{\alpha^T \Sigma \alpha}$ in the two-class case. For each term t, $\alpha = e_t$ is a unit vector in the direction of t. Given the discriminating power of terms, we will pick terms greedily until we get good discrimination between classes.

In general, given a set of two or more classes $\{c\}$, with $|c|$ documents in class c, we compute the ratio of the so-called between-class to within-class scatter. Switching back to our term frequency notations, we express this as:

$$\mathsf{Fisher}(t) = \frac{\sum_{c_1,c_2}\Big(\mu(c_1,t) - \mu(c_2,t)\Big)^2}{\sum_c \frac{1}{|c|}\Big(x(d,t) - \mu(c,t)\Big)^2}, \tag{2}$$

$$\text{where} \quad \mu(c,t) = \frac{1}{|c|}\sum_{d \in c} x(d,t). \tag{3}$$

The information theory literature provides some other notions of good discriminants. One of the best known is *mutual information* [11]. Closer inspection shows that its computation is more complicated and not as easily amenable to the optimizations we implement in our system. We will discuss other related work in text feature selection later in Section 5.

The remaining exercise, having sorted terms in decreasing order of Fisher index, is to pick a suitable number starting at the best. Let F be the list of terms in our lexicon sorted by decreasing Fisher index. Our heuristic is to pick from F a prefix F_k of the k most discriminating terms. F_k must include most features and exclude most noise terms. A short F_k enables fast classification and holding a larger taxonomy in memory. Too large an F_k will fit the training data very well, but will result in degraded accuracy for test data, due to *overfitting*. There are various techniques for pruning feature sets. We minimize classification error on a set of documents kept aside for *model validation*, shown in Figure 1. Some others approaches are to use the minimum description length principle, resampling or cross validation. We randomly partition the pre-classified samples into $\mathcal{T}$, the training set and $\mathcal{V}$, the validation set. We compute the Fisher index of each term based on $\mathcal{T}$, and then classify $\mathcal{V}$ using various prefixes F_k. Let N_k be the number of misclassified documents using F_k; then we seek that value of k, say k^*, for which N_k is minimized.

For classification we choose the class c that maximizes the following *a posteriori* class probability based on the Bernoulli model introduced in Section 3.1:

$$\Pr[c|d, F_k] = \frac{\pi(c)\prod_{t \in d \cap F_k}\theta(c,t)^{n(d,t)}}{\sum_{c'}\pi(c')\prod_{t \in d \cap F_k}\theta(c',t)^{n(d,t)}}, \tag{4}$$

where π is the prior distribution on the classes. Let $c_*(d)$ be the "true" class of $d \in \mathcal{V}$, then $N_k = \sum_d N_k(d)$, where

$$N_k(d) = \begin{cases} 1, & \exists c \neq c_*(d) : \Pr[c|d, F_k] > \Pr[c_*(d)|d, F_k] \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

3.4 Data structures and pseudocode

The modules described so far are parts of a topic analysis system that we have built, called TAPER. In building an "industry grade" topic analyzer, we set the following goals:

- We must handle thousands of classes and millions of documents; the current limits are 2^{16} classes, 2^{32} unique tokens, and 2^{32} documents. We make the reasonable assumption that we can hold a simple pointer representation of the taxonomy tree in memory with a few words per node (class).
- Training must be essentially on-line, say, as in a crawling and indexing application. Testing or applying must be interactive. As we will see in Section 4, we can train at 140μs per token and test at 30μs per term. Training should preferably make a single pass over the corpus.
- Since our statistics modules maintain combinable aggregates, it is simple to incrementally update a fixed taxonomy with new document, and correct misclassifications by moving a document from an incorrect class to one adjudged correct. With some more work, it is also possible to reorganize entire topic subtrees.

A sketch of the TAPER system is shown in Figure 1. The training documents are randomly split into two subsets: one subset for collecting term statistics and estimating term parameters, and another subset for "pruning" the models by deciding which terms are noise. We typically do a $\frac{2}{3}:\frac{1}{3}$ split. Cross validation would make better use of small training sets, but would take too many passes over the corpus. The resulting class models restricted to features alone are stored on disk using an indexed access method. For some corpora, these statistics actually fit easily in main memory. During classification the classifier loads these model statistics on demand and emits a set of the most likely classes for the input document.

TAPER has the following modules:

- Maps between terms and 32-bit ID's, and between classes and 16-bit ID's.
- A tree data structure for storing class-specific information.

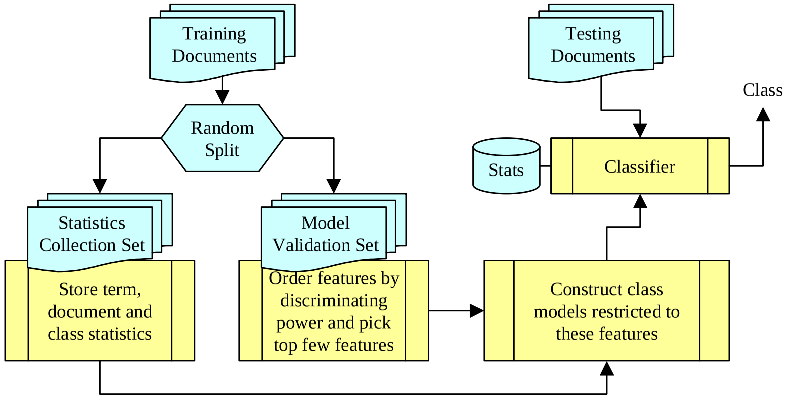

Figure 1: A sketch of the TAPER hierarchical feature selection and classification engine.

- A module for statistics collection.
- A module for feature selection.
- A module for applying the classifier to a new or test document.

We will discuss the last three in detail.

3.4.1 Statistics collection

The goal of this module is to collect term statistics from a document and dispense with it as fast as possible. After simple stopword filtering and stemming while scanning, the document is converted to a sequence of 32-bit TID's (term ID's). The main table maintained on disk is the *frequency table* shown in Figure 2. TID corresponds to a term that occurs in some document belonging to a class corresponding to KCID (kid class ID). CID's (KCID's and PCID's) are numbered from one onwards. PCID represents the parent of KCID (zero if KCID is the root). In the statistics collection phase this table is keyed on the TID and KCID. There are four other numeric fields per row. All these four numbers are additive over documents, so for each document d and term t we can just append a row to the frequency table, with SMC set to one, SNC set to the number of times t occurred in d, called $n(d,t)$, SF1 set to $n(d,t)/\sum_t n(d,t)$ and SF2 set to $(\text{SF1})^2$. SMC is used in the binary model; SNC is needed in the bernoulli model.

This trades off space for time, and the frequency table grows rather quickly, but with a lot a duplicate keys. Depending on how much disk space exists, once in a while we must pause to catch our breath and sort and aggregate the duplicate keys. For large corpora, this is vastly preferable to a disk hash table with random IO. Since sorting is a superlinear operation, it is good to keep the size of the frequency table small for faster sorting in addition to conserving disk space. We use simple heuristics to start a sort phase. First, we configure some upper bound to the number of rows

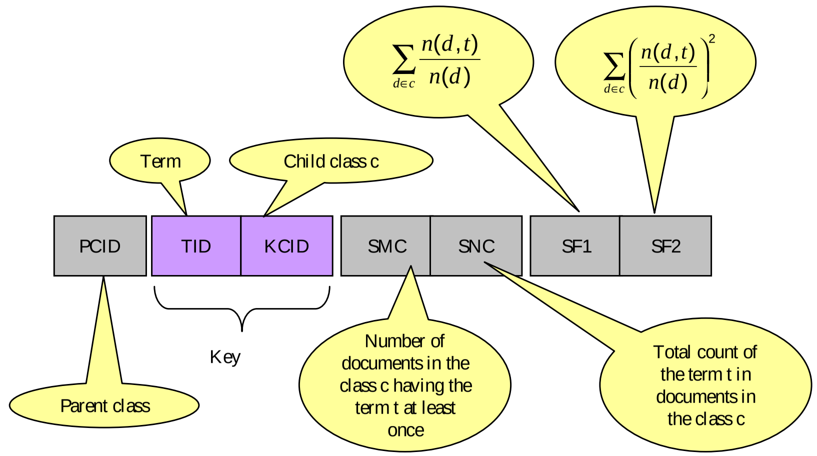

Figure 2: The frequency table.

for which a sort must be initiated to conserve space. We also estimate the sorting time given the number of rows and the average number of rows that a document adds to the table. From previous sorts, we also maintain an estimate of the fraction of rows compacted by aggregation. We can also do this by recent sampling techniques [1].

Given a moment in the execution and an infinite supply of documents, we can continue training until we hit the upper bound dictated by space limits, and then sort. We can estimate the time to complete this. We can also estimate the total time needed if we sorted right away, trained the same number of documents (which need not create an overflow) and sorted again. If the latter estimate is smaller we initiate a sort. In our current implementation, processing documents stops while the sorting is in progress. To meet tough real-time requirements, one can open a new frequency table and fork a thread, perhaps on another processor, to aggregate the last run while more documents continue to be accepted.

We could have chosen an indexed access method instead of the frequency table, and looked up and updated SMC, SNC, SF1 and SF2 as we scanned each document. That would have resulted in index lookups and random IO potentially for every term in the training set. It was far more efficient to append statistics in a logged fashion. The frequency table is a temporary file and no direct indexed access to it is actually required later. Another benefit is compactness: this is the most space-intensive phase of training, and we avoid the storage overheads of indexed access and take control of compaction explicitly. The space overhead of storing TID and PCID redundantly is moderate, as the rest of each row is already 18 bytes long.

3.4.2 Feature selection

Before beginning feature selection, we aggregate the frequency table one last time to eliminate all duplicates. We rewind the frequency table and prepare to scan it. At this stage, all rows with the same TID are collected in a contiguous run going through all CID's where that TID occurred (see Figure 3). We also prepare to output another file, called the *fisher table*. For the following discussion we will assume it has the format shown in Figure 4. Rows are keyed by CID and a floating point number FI (FI stands for *Fisher index*), where for each fixed CID the rows are sorted in decreasing order of FI. The last column is the TID (term ID) whose corresponding CID and FI are the first and second columns.

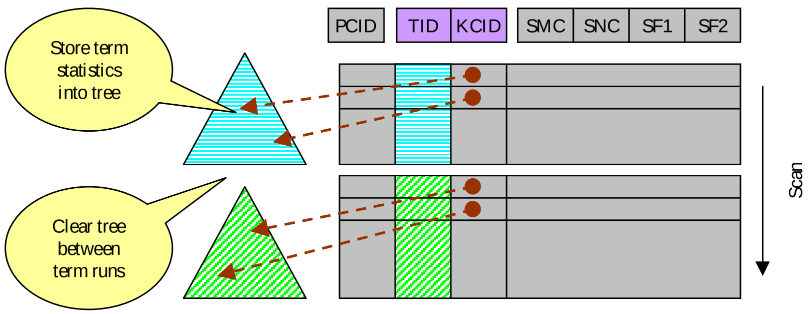

Figure 3: Scanning the frequency table and computing the Fisher index of each term.

Because TID is the primary key in the frequency table, as we scan it we will get a sequence of runs, each run having a fixed TID. Associated with each topic node in memory we keep a few words of statistics (derived from SMC, SNC, etc.). When we start a run for a given TID we clear these. As we scan through the various CID's for the given TID in the frequency table, we locate the node corresponding to the CID in the taxonomy and update these statistics. In a large taxonomy, very few of the nodes will be updated during a run. If a node is updated, its parent will be updated as well. We can therefore reset these statistics efficiently after each run.

When the run for a given TID completes, we can compute, exploring only the updated nodes, the Fisher index of that term for every internal node in the taxonomy (using (2)). For each of these PCID's, we append a row to the Fisher table. Next, we sort the Fisher table on the key (PCID, FI). This collects all classes into contiguous segments, and for each PCID, orders terms by decreasing values of FI.

Consider now the case in which, for each internal topic c, the number $k^*(c)$ of features to pick is specified to TAPER directly. (The next section discusses how k^* is determined in one pass over the portion of the training documents set apart earlier for model pruning.) Given $k^*(c)$, we scan the sorted Fisher table, copying the first $k^*(c)$ rows for the run corresponding to class c to an output table, and discarding the remaining terms. This involves completely sequential IO.

Once feature selection is performed on the Fisher table, both the frequency and fisher table are sorted once again, this time with (PCID, TID) as the key. After these sorts we perform a merge. We consider rows of the Fisher table one by one. For each row, once we find the beginning of a key-

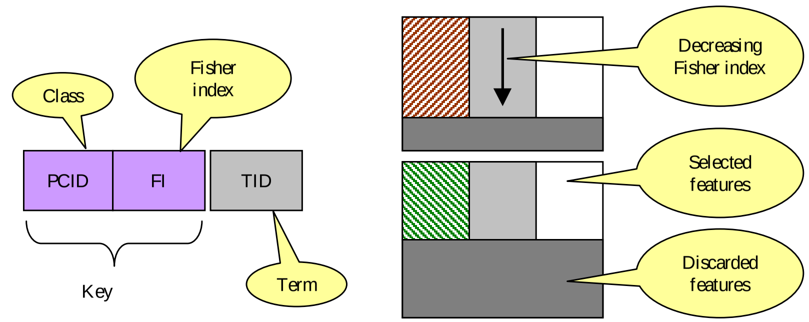

Figure 4: The Fisher table. When sorted using the (PCID, FI) key, we get a contiguous run for each internal topic node, the run containing the best features first. If a suitable feature set size if specified, we can truncate the run to produce a table of feature TID's for each internal topic.

matched row of the frequency table, we read it as long as the key remains unchanged, constructing a memory buffer of the form (KCID, SMC, SNC). This buffer is then written into a hash table on disk.

3.4.3 Finding a good cutoff k^*

Given terms in decreasing Fisher index order, we wish to find a good value for k^*, the number of chosen features. Fix a test document $d \in \mathcal{V}$ and consider what happens as we grow the prefix k. Typically, d will be misclassified upto some prefix because there aren't enough discriminating terms in the prefix, and then at some point it may get correctly classified owing to some good features being included. For some documents, at a larger prefix, a noise term gets into the feature set and causes d to be misclassified again. Let this 0-1 function (1 iff d is misclassified) be $N_k(d)$; then we seek to find $\sum_d N_k(d)$ for all k.

This we wish to do in only one pass over the validation documents $\mathcal{V}$. To do this, we will hold N_k in memory. Even if we store N_k for each value of k and the lexicon is of size 100,000, we need only 400 kB. Thus the obvious code on the left is modified to the one on the right:

Naive algorithm:
For all or suitably many values of k
    Prepare models with only k features
    For $d \in \mathcal{V}$
        Determine $N_k(d)$

IO-efficient algorithm:
Initialize $N_k = 0$ for $k = 0, 1, \ldots$
Compute feature list F ordered by decreasing FI as before
For each document $d \in \mathcal{V}$
    Let c', c'' be children of c
    Compute $\Pr[c'|c, d, F_0] = \pi(c') / \sum_{c''} \pi(c'')$ for all c'
    For $k = 1, 2, \ldots$
        Check if the k-th feature term occurs in d
        Find $\Pr[c'|c, d, F_k]$ using $\Pr[c'|c, d, F_{k-1}]$ and (4)
        Suppose the true class for d is child c_* of c.
        If $\Pr[c_*|c, d, F_k] < \max_{c'} \Pr[c'|c, d, F_k]$
            $N_k \leftarrow N_k + 1$

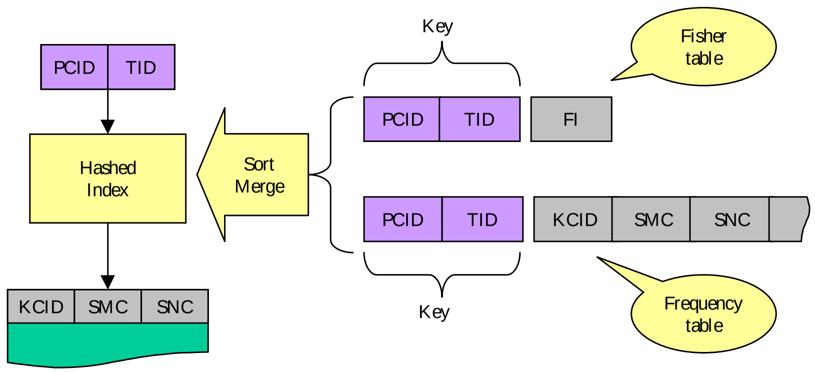

Figure 5: Creating an indexed topic model statistics table that is used by the classifier while classifying new documents.

3.4.4 Updates to the database

For a batch job, the large frequency and fisher tables can now be deleted, leaving the relatively smaller indexed topic statistics and the much smaller term-to-TID maps. If the system is used in a setting where new documents will be added to classes, it is necessary to preserve the frequency table. It continues to be used in the same way as before: rows are appended and occasionally it is compacted to aggregate duplicate keys. Running feature selection integrates the new data into the indexed statistics. Like running statistics generation for a relational server, feature selection is not an interactive operation. E.g., on a database with 2000 classes, average 150 documents per class, and average 100 terms per document, it may take a couple of hours. So this is invoked only when there is reason to believe that the refreshed statistics will improve classification. Automatically detecting such times is an interesting question.

Another issue is deletion of documents and moving of documents from one class to another (perhaps because classification was poor or erroneous for those documents). Since feature selection is always preceded by a frequency table aggregation, we can always place negative "correction" entries in it! I.e., we produce a frequency table row corresponding to each term in the deleted document and negate SMC, SNC, SF1 and SF2 for the class(es) the document is being deleted from. (Here we cannot ensure that the document was originally included in the aggregate, but that can be done by preserving ID's for training documents.) The API does not include deletion and move yet, but the changes are straight-forward.

A more difficult issue is the reorganization of the taxonomy itself. Although the current system leaves this issue unresolved, we believe its design will make reorganization relatively simple. Notice that in TAPER, a parent class inherits, in an *additive* fashion, the statistics of its children, since each training document generates rows for each topic node from the assigned topic up to the root. We thus envisage a procedure of reassigning CID's and writing out a new frequency table with some negative "correction" entries. Consider detaching a subtree under node c_1 and attaching it elsewhere under node c_2. Statistics at or above the least common ancestor c_ℓ of c_1 and c_2 remain unchanged.

Negative (respectively, positive) rows are appended to the frequency table corresponding to all classes between c_ℓ inclusive and c_1 (respectively, c_2) exclusive. And the parent and child links have to be modified in the taxonomy tree.

3.4.5 Classification

The rationale for the data organization described above becomes clear when we consider what happens when the classifier is invoked on a document. In the basic API, one loads a taxonomy and its pre-computed statistics, and submits a document (represented by term counts) to the classifier. In our model, the probability that the document is generated by the root topic is 1 by definition and decreases down any path in the taxonomy. Accordingly, also specified in the API is a probability cut-off for nodes reported back as close matches.

Consider the document d at some internal node c_0 with children c_1 and c_2. TAPER needs to intersect d with the feature set at c_0, then for each surviving term t, look up the class models for c_1 and c_2. It is thus best for both space and IO efficiency to index the statistics by (c_0, t) and include in the record a vector of statistics for each c_i, $i = 1, 2$. The obvious pseudocode has to be slightly modified to reflect this (p_c denotes log probabilities).

Naive index lookup:	**Optimized index lookup:**

For each child c_i of c_0, $i = 1, 2, \ldots$
    Initialize p_{c_i} to 0
    For each term $t \in d$
        Lookup term statistics for (c_i, t)
        Update p_{c_i}
Normalize $\sum_i \exp(p_{c_i})$ to one
Add p_{c_0} to each p_{c_i}.
	Initialize all p_{c_i} to zero
	For each term $t \in d$
	    Skip if key (c_0, t) is not in index
	    Retrieve record for (c_0, t)
	    For each c_i that appears in the record
	        Update p_{c_i}
	Normalize etc.

3.5 Context-sensitive document signatures

Up to a point, the user can sift a query response based only on the topic paths. However, even the leaf classes are necessarily coarser than individual documents; support is therefore needed to browse quickly through many documents without looking into the documents in detail. Most search engines attach a few lines from each document. Often these are the title and first few lines; or they are sentences with the most search terms. For many documents, better keyword extraction is needed. Moreover, as we have argued, these signatures should be extracted relative to a node in the taxonomy.

Given this reference node c, one approach is to concatenate the training documents associated with c into a super document d, and then rank terms $t \in d$ in decreasing order of the number of standard deviations that $x(d,t)$ is away from $f(c,t)$. Here our earlier simplistic document model gets into trouble: as mentioned in Section 3.1, a term that has occurred once in a document is more likely to occur again. Since the Bernoulli model does not take this into account, frequent terms often remain surprising all along the taxonomy path.

Matters are improved by moving to another simple model. First suppose we have a single test document d, and consider $t \in d$. If the observed fraction of training documents in class c containing term t is $\theta(c,t)$, we simply sort all $t \in d$ by increasing $\theta(c,t)$ and report the top few. If there are $\ell > 1$ test documents in c, we find the fraction $\phi(t)$ that contains t, and sort the t's in

(a) Patent			
	Communication	329	Modulator
		332	Demodulator
		343	Antenna
		379	Telephony
	Electricity	307	Transmission
		318	Motive
		323	Regulator
		219	Heating
	Electronics	330	Amplifier
		331	Oscillator
		338	Resistor
		361	System

(b) Yahoo	
	Arts
	Business and Economy
	Computers and Internet
	Education
	Entertainment
	Government
	Health
	News and Media
	Recreation
	Reference
	Regional
	Science
	Social Science
	Society and Culture

Figure 6: Sample of topic taxonomies we have experimented with. (a) A portion of the US Patent database taxonomy with numeric class codes. (b) The first level of a web taxonomy derived from Yahoo!, with a total of 2,218 nodes.

increasing order of $\frac{(\theta(c,t)-\phi(t))\sqrt{\ell}}{\sqrt{\theta(c,t)(1-\theta(c,t))}}$. Both in fact correspond to P-values computed using the normal approximation to the binomial distribution.

4 Performance

In this section we study the performance of our system. There are three aspects to performance: first, to what extent this paradigm assists in text search and browsing; second, how accurate our techniques for feature selection and hierarchical classification are; and third, how efficient or scalable our system is. The first item is at this point a matter of qualitative judgement, as is the evaluation of the signature-finding techniques. The quality of feature selection and classification can be measured precisely, and we present these results here. As regards efficiency, we restrict our discussion to quoting our running times on a specific platform, and show that they scale favorably with corpus size.

4.1 Datasets and measures

We used three data sources: the Reuters benchmark used widely in the IR community, the US Patent database, hereafter referred to as USPatent, and Yahoo!. For evaluation, the simple scenario is a m-class problem where each document belongs to exactly one class. We can draw up an $m \times m$ contingency table, entry (i, j) showing how many test documents of class i were judged to be of class j. This is called the *confusion matrix*. One important number to compute from a confusion matrix is the sum of diagonal entries divided by the sum of all elements: this gives the fraction of documents correctly classified. If each document has exactly one class, this number is the same as *microaveraged recall* and *precision* as defined by Lewis [27]. Matters are complicated by documents having multiple classes. Due to space constraint we omit our experiments with this setting. See Lewis for more details on evaluating classifiers [27].

4.2 Evaluation of feature selection

Although Reuters has provided a taxonomy for its articles, the data available does not include taxonomy codes in the class header. For this subsection we will work with other corpora where

such information is explicitly provided.

The sample of USPatent that we used has three nodes in the first level, Communication, Electricity and Electronics. Each has four children in the second level. Figure 6 shows the taxonomy we used. The overlap in vocabulary between some of the nodes, e.g., modulator, demodulator, amplifier, oscillator; and motive, heating, resistor make the classification task appear more challenging than Reuters, which deals with a more diverse set of topics.

4.2.1 Classification error vs. feature set size

Figure 7 shows the results of validation experiments over the patent database. 500 training patents and 300 validation patents were picked at random from each of the 12 leaves in Figure 6. The Fisher index ordering gives rapid reduction in classification error within just a few hundred feature terms, out of the roughly 30,000 terms in our lexicon. For some classes the error goes up slightly (not visible in the range shown) after a minimum due to overfitting. The smallest minima and corresponding errors are roughly at 160 terms, 25.1% for Patent; 200 terms, 11.7% for Communication; 890 terms, 17.8% for Electricity; and 9130 terms, 16.6% for Electronics. The minima are not very sharp, but the diversity of the feature set sizes still questions the common practice of picking a fixed number of most frequent terms in each class as features.

4.2.2 Best discriminants and applications

We list the best features in the patent taxonomy below; notice how the sets change down the levels.

Patent: Signal, modulate, motor, receive, antenna, telephone, transmit, frequency, modulation, modulator, demodulator, current, voltage, data, carrier, power, amplifier, phase, call, amplitude.

Patent:Communication: Antenna, telephone, modulator, demodulator, signal, modulate, output, call, modulation, input, demodulated, frequency, phase, communication, radar, demodulating, space, detector, line, demodulation, transmit, circuit.

Patent:Electricity: Motor, heat, voltage, transistor, output, circuit, connect, input, weld, extend, surface, current, position, gate, speed, control, terminal, drive, regulator, signal, rotor.

Patent:Electronics: Amplifier, oscillator, input, output, frequency, transistor, signal, laser, emitter, couple, amplify, gain, resistance, connect, extend, form, contact, differential, material, resistor.

We have also applied the feature selection algorithm to find salient differences between various sets of documents. One application is to find descriptions for clusters in unsupervised document clustering. For example, the query *mouse* gets hundreds of responses from the IBM Patent Server. To quickly zoom in to the right notion, one clusters the response and runs TAPER with the clusters treated as classes. A sample of results is shown:

Cluster 1: Tissue, thymus, transplanted, hematopoietic, treatment, exemplary, organ, immunocompromised, host, trypsin.

Cluster 2: Computer, keyboard, hand, edge, top, location, keys, support, cleaning.

Cluster 3: Point, select, environment, object, display, correspondence, direct, image.

These "cluster digests" lets the user easily refine the query.

In the context of the Patent Database, TAPER can also be used, for example, to compare and contrast the patent portfolios of two assignees or one assignee at two different times. For example, a comparison between *Sun Microsystems* and *Silicon Graphics* gives the following:

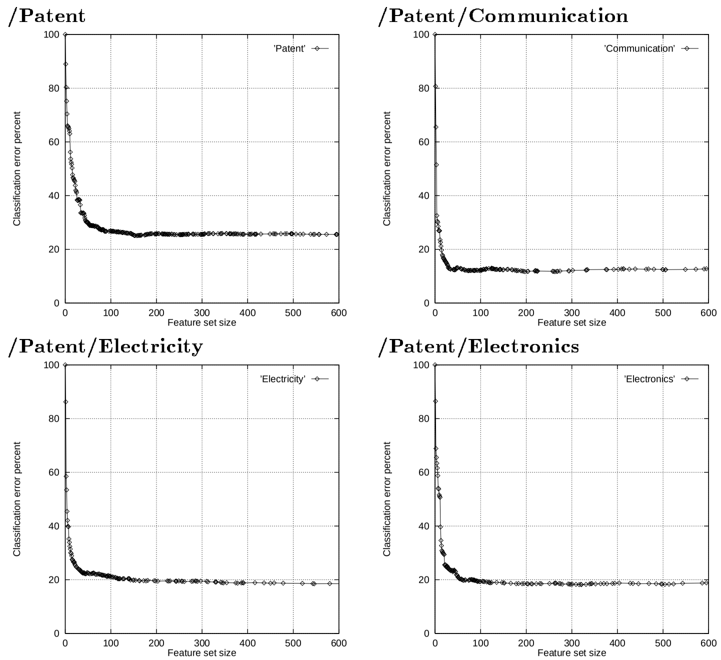

Figure 7: Evaluation of feature selection.

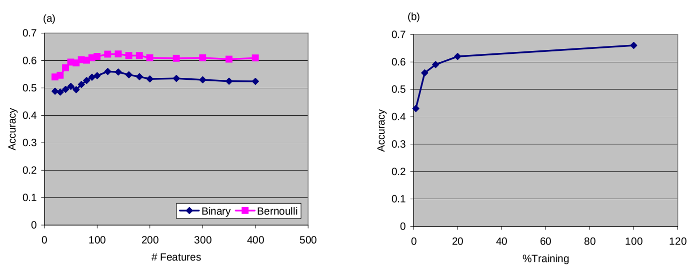

Figure 8: (a) Choice between the Bernoulli and binary text models for classification. Bernoulli appears consistently superior in our experiments. (b) Verification that the training set is adequate. The x-axis is the percent of the training set sampled for training. The y-axis is the fraction of documents correctly classified.

Sun Microsystems	Silicon Graphics
General purpose programmable digital computer systems	Information processing system organization
Electrical computers and data processing systems	Data presentation, computer graphics
Integrated circuit, processor, voltage, interface	Surface detail, texture
	Adjusting resolution or level of detail

An even more interesting example is comparing Intel patents in 1993-94 with those in 1994-95. TAPER detects a new line of research and patenting in the second year:

Intel, 1993-94	Intel, 1994-95
General purpose programmable digital computer systems	Interactive television bandwidth reduction system
Chip fabrication	Involving difference transmission
Counter, input	Field or frame difference
	Involving adaptive transform coding

(It is possible to get the coherent phrases because the patent database effectively stores them as single terms associated with patents.)

4.2.3 Choice of system configurations

Binary vs. Bernoulli: TAPER supports both the Bernoulli (term counts are significant) and the binary (term count is zero or one) text models. As we discussed in Section 3.1, the Bernoulli model might be criticized as paying too much attention to each additional occurrence of a term, so we compare Bernoulli with binary in Figure 8(a). The Bernoulli classifier appears to be more effective in exploiting the count information, so for the rest of the paper we restrict ourselves to this model.

Sufficiency of training data: Before we compare TAPER with other known approaches, we ensure that the training set is adequate. To do this, we train using different sized random samples from the training set and verify that the test performance converges. This is shown in Figure 8(b).

4.3 The Reuters benchmark

The Reuters benchmark has about 7,700 training documents and 3,500 testing documents from about 135 classes. Each document is about 200 terms long on average. We experimented with Reuters to ensure that our basic classifier is of acceptable quality. Less than a tenth of the articles are assigned multiple classes. In fact, in some cases, some class labels were refinements of others, e.g., *grain* and *wheat*, and it would be incorrect to regard them as classes at the same level since some classes imply others. For simplicity, we just removed all but the first class label from each article. Alternatively, for m classes, one can build m two-way classifiers; the c-th classifier discriminating between c and $\bar{c}$.

We only considered classes with at least 20 training documents. Only 30 classes were large enough, giving a 30×30 confusion matrix. The best accuracy was achieved using about 8000 features. The best microaveraged recall/precision is 0.87, which compares favorably with previous experiments [3], although those studies used the $(c, \bar{c})$ style classifiers. The numbers are not therefore directly comparable. However, only 9% of the documents had multiple topics, so even if we were incorrect on *all* such documents, we would not do much worse.

For this benchmark there is no benefit from hierarchy. To test the effect of our feature selection, we compared it with an implementation that performs singular value decomposition (SVD) on the

Class name	329	332	343	379	307	318	323	219	330	331	338	361
329_Demodulator	51	9	2	1	0	6	5	2	12	11	0	2
332_Modulator	21	27	3	2	3	7	6	4	10	12	2	2
343_Antennas	10	6	47	3	4	2	6	1	1	4	14	3
379_Telephony	9	1	1	65	1	5	2	5	3	1	5	3
307_Transmission	1	1	1	1	57	2	8	5	0	1	19	4
318_Motive	6	4	1	1	1	41	7	13	14	4	2	3
323_Regulator	8	3	1	3	7	4	59	7	2	1	2	5
219_Heating	2	1	0	0	0	18	9	49	12	1	2	5
330_Amplifier	6	5	1	0	1	17	1	8	53	3	4	1
331_Oscillator	10	2	3	0	6	9	4	7	10	33	13	4
338_Resistor	0	0	0	0	3	0	3	2	0	0	87	4
361_System	2	1	1	1	9	8	8	9	1	1	30	29

Table 1: Confusion matrix for the weighted cosine one-level classifier. Each row sums to 100, modulo rounding. The diagonal elements add up to only 0.48 of the total number of the documents. This is the micro-averaged recall.

Class name	329	332	343	379	307	318	323	219	330	331	338	361
329_Demodulator	80	5	0	0	0	2	0	3	5	4	0	0
332_Modulator	16	55	1	0	1	2	1	3	9	11	0	0
343_Antennas	5	5	63	1	1	0	2	0	0	2	15	6
379_Telephony	4	2	1	82	0	1	0	2	1	1	1	4
307_Transmission	0	0	0	0	55	2	3	3	0	2	26	8
318_Motive	6	4	0	2	3	48	5	16	8	5	1	2
323_Regulator	3	1	1	2	3	2	81	6	0	0	1	1
219_Heating	1	1	0	0	0	10	4	72	7	0	3	1
330_Amplifier	3	9	0	0	0	10	0	11	57	8	0	1
331_Oscillator	15	8	0	0	0	4	0	7	8	47	5	4
338_Resistor	0	0	0	0	1	0	2	0	1	0	92	4
361_System	1	0	0	0	2	6	6	10	1	1	12	61

Table 2: Confusion matrix for our multi-level classifier, showing much larger diagonal elements, i.e, more frequently correct classification. The micro-averaged recall is 0.66.

original term-document matrix, projects documents down to a lower dimensional space, and uses a Bayesian classifier in that space assuming the Gaussian distribution [42]. Our classifier was more accurate by 10–15%, in spite of its simplicity. Our explanation is that the SVD, in ignoring the class labels, finds projections along directions of large variance, which may not coincide with directions of best separation between documents with different classes.

4.4 Evaluation of hierarchical classification

In this section we describe our experience with the hierarchical USPatent dataset. We compare the hierarchical classifier with a standard vector-space [38] based classifier. Each document is a vector in term space; each class is the sum or centroid of its document vectors. The similarity between two vectors is their cosine. Weighting the terms usually results in better relevance ranking. There are over 287 variants of term weighting schemes with tuned magic constants reported [38]. We pick one version recommended by Sparck-Jones [20, 23].

$$\begin{aligned} n_{\max}(d) &= \max_{t \in d} n(d,t) \\ m &= \text{number of classes} \\ n_t &= \textstyle\sum_c \operatorname{sign}(n(c,t)) \end{aligned}$$

Classifier	Prefix	Parameters	Recall	Time/doc
Flat	250	2651	0.60	15 ms
Taxonomy	950;200;400;800	2649	0.63	6 ms

Table 3: The benefits of hierarchy. The prefix field in the second row correspond to the four internal nodes in the USPatent tree: /Patent, /Patent/Communication, /Patent/Electricity and /Patent/Electronics.

$$\begin{aligned} w(c,t) &= \left(1+\tfrac{n(d,t)}{n_{\max}(d)}\right)\left(1+\lg\tfrac{m}{n_t}\right) \\ \mathsf{Score}(c,d) &= \frac{\vec{x}_d \cdot \vec{w}_c}{|\vec{x}_d|\,|\vec{w}_c|} \end{aligned}$$

We see a substantial gain in accuracy over the standard weighted-cosine classifier. We did further experiments to see how much of the gains was from feature selection as against the hierarchy. To do this, we can fix the feature selection and classification modules, and only change the taxonomy: one will be the taxonomy in Figure 6, the other will have the root and the 12 leaves. We have to be very careful to make this a fair competition, making sure that the class models are represented with the same complexity (number of parameters) in the two settings. In counting the number of parameters we must also account for the sparsity of the term frequency tables; we have no direct control on this. By trial and error, we came up with the comparative evaluation shown in Table 3.

In this dataset, the accuracy benefit from hierarchy is modest compared to the benefit from feature selection. However, note that the flat classifier has a steep performance penalty because it has to compare too many classes all at once. This gap will be dramatic for larger taxonomies, such as Yahoo!. How to allocate a fixed number of parameters among the taxonomy nodes for best overall classification is an interesting issue.

Summarizing, we showed that our feature selection is effective, and that our classifier is significantly more accurate than cosine-based ones and comparable to the best known for flat sets of classes. Hierarchy enhances accuracy in modest amounts, but greatly increases speed.

4.5 Running times

TAPER has undergone extensive revisions to make it efficient. One can specify some maximum amount of memory it should use, and it uses the disk intelligently to stage larger data. Here, we will merely give some examples of the current performance, not compare it with all the preliminary versions. Also, comparison with other existing packages on a common hardware platform is left for future work. TAPER runs on two platforms: a 133 MHz RS 6000/43P with 128 MB RAM, and a 200 MHz Pentium-II with 256 MB RAM. On the former, we can train at 140 microseconds per term and test at 30 microseconds per term. These times are measured after the document is in memory, and they account for the database reorganization costs during training. The whole Reuters experiments, including IO time for text, takes less than 20 minutes. On the latter platform, 266,000 web documents from 2118 Yahoo! classes have been trained (from a disk image obtained by crawling) in about 19 hours. TAPER processes documents that are several hundred words long in about the time needed for a disk access. This makes it possible to directly connect TAPER to a web crawler and populate the search database with topic information as well as keyword index, on the fly.

5 Related work

We survey the following overlapping areas of related research and point out differences with our work where appropriate: IR systems and text databases, data mining, statistical pattern recognition, and machine learning.

Data mining, machine learning, and pattern recognition. The supervised classification problem has been addressed in statistical decision theory (both classical [45] and Bayesian[5]), statistical pattern recognition [14, 18] and machine learning [46, 32, 25]. Classifiers can be parametric or non-parametric. Two well-known classes of non-parametric classifiers are decision trees, such as CART [6] and C4.5 [34], and neural networks [21, 28, 22]. For such classifiers, feature sets larger than 100 are considered extremely large. Document classification may require more than 50,000.

IR systems and text databases. The most mature ideas in IR, which are also successfully integrated into commercial text search systems such as Verity[10], ConText[11] and AltaVista, involve processing at a relatively syntactic level; e.g., stopword filtering, tokenizing, stemming, building inverted indices, computing heuristic term weights, and computing similarity measures between documents and queries in the vector-space model [43, 39, 16].

More recent work includes statistical modeling of documents, *unsupervised* clustering (where documents are not labeled with topics and the goal is to discover coherent clusters) [2], *supervised* classification (as in our work) [3, 10], query expansion [40, 44]. Singular value decomposition on the term-document matrix has been found to cluster semantically related documents together even if they do not share keywords [13, 33]. For a survey see [15].

Contrast with our work. Our work emphasizes the performance issues of feature selection and classification raised by corpora ranging into tens to hundreds of gigabytes, rather than exploring many variants of learning algorithms on small corpora and lexicon (10,000 documents and terms).

Most closely related to our work [7] are the concurrent investigations made by Koller and Sahami [24] and Yang and Pedersen [47]. Koller and Sahami propose a sophisticated feature selection algorithm that uses a Bayesian net to learn inter-term dependencies. The complexity in the number of features is supralinear (e.g., quadratic in the number of starting terms and exponential in the degree of dependence between terms). Consequently, the reported experiments have been restricted to a few thousand features and documents. Yang and Pedersen's experiments appear to indicate that much simpler methods suffice, in particular, that the approach of Apte et al [3] of picking a fixed fraction of most frequent terms per class performs reasonably. There is a possible danger that this fraction is very sensitive to corpus and methodology (e.g., whether stemming and stopwording is performed). This is indicated by the poor performance of such simplistic methods observed in recent work by Mladenic [31].

Our goal has been to look for techniques that have good statistical foundation while remaining within almost linear time and one pass over the corpus, even when doing feature selection simulataneously for many nodes in a large topic taxonomy. Koller and Sahami also emphasize the importance of hierarchies, but they use a greedy search for the best leaf and point out the potential dangers of this approach. Our formulation fixes this problem. Also, our approach of computing context-dependent document signatures to aid search and browsing appears to be a new extension to the Scatter-Gather type of retrieval interface [12].

[10] `http://www.verity.com`
[11] `http://www.oracle.com/products/oracle7/oracle7.3/html/context_qa.html`

6 Conclusion

We have demonstrated that hierarchical views of text databases can improve search and navigation in many ways, and presented some of the tools needed to maintain and navigate in such a database. A combination of hierarchy, feature selection, and context-sensitive document signatures greatly enhanced the retrieval experience.

Our work raises several questions for future investigation. Usually, TAPER finds good feature set sizes independently for each internal node; the space needed to store the resulting model was not explicitly controlled. In Section 4.4 we raised the question of designing classifiers that maximize accuracy given bounded space, i.e., model size, and bounded time. Table 3 suggests the interesting problem of allocating the total space among nodes of a hierarchy for best overall accuracy. Resolcing the following performance issue can greatly cut down space and time during training: is it possible to prune off terms with poor Fisher index even as term statistics are being collected? Another issue related to accuracy is whether the classifier can reliably stop at a shallow level of the tree when classifying a document about which it is uncertain. Finally, in recent work we have found that adding hyperlink information to the feature set used by TAPER greatly improves accuracy [8].

Acknowledgments: Thanks to Mark Jackson, Alex Miller, Bruce Hönig, and Carol Thompson from the IBM Patent Server Team, to Chandra Chekuri, Mike Goldwasser, and Eli Upfal for helpful discussions, and to Sunita Sarawagi and Martin van den Berg for comments on the paper.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Symposium on the Theory of Computing (STOC)*, pages 20–29, Philadelphia, May 1996. ACM.

[2] P. Anick and S. Vaithyanathan. Exploiting clustering and phrases for context-based information retrieval. In *SIGIR*, 1997.

[3] C. Apte, F. Damerau, and S. M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 1994. IBM Research Report RC18879.

[4] M. Balabanovic and Y. Shoham. Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, Mar. 1997.

[5] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, New York; Berlin, 1985. ISBN: 0-387-96098-8.

[6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks/Cole, 1984. ISBN: 0-534-98054-6.

[7] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Using taxonomy, discriminants, and signatures for navigating in text databases. In *VLDB*, Athens, Greece, Aug. 1997.

[8] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD*. ACM, 1998. To appear.

[9] C. Chekuri, M. Goldwasser, P. Raghavan, and E. Upfal. Web search using automatic classification. In *Sixth World Wide Web Conference*, San Jose, CA, 1996.

[10] W. W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. In *SIGIR*. ACM, 1996.

[11] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

[12] D. R. Cutting, D. R. Karger, and J. O. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *SIGIR*, 1993.

[13] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407, 1990.

[14] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[15] C. Falaoutsos and D. W. Oard. A survey of information retrieval and filtering methods. Technical Report CS-TR-3514, University of Maryland, College Park, MD 20742, Aug. 1995.

[16] W. B. Frakes and R. Baeza-Yates. *Information retrieval: Data structures and algorithms.* Prentice-Hall, 1992.

[17] J. H. Friedman. On bias, variance, 0/1 loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[18] K. Fukunaga. *An Introduction to Statistical Pattern Recognition, 2nd ed.* Academic Press, New York, 1990.

[19] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, Dec. 1992.

[20] D. Harman. Ranking algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information retrieval: Data structures and algorithms*, chapter 14. Prentice-Hall, 1992.

[21] D. R. Hush and B. G. Horne. Progress in supervised neural networks. *IEEE Signal Processing Magazine*, pages 8–39, January 1993.

[22] A. K. Jain, J. Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, March 1996.

[23] K. S. Jones. A statistical interpretation of term specificity and its applications in retrieval. *Journal of Documentation*, 28(1):11–20, 1972.

[24] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *International Conference on Machine Learning*, volume 14. Morgan-Kaufmann, July 1997. To appear.

[25] P. Langley. *Elements of Machine Learning.* Morgan Kaufmann, 1996. ISBN: 1-55860-301-8.

[26] P.-S. Laplace. *Philosophical Essays on Probabilities.* Springer-Verlag, New York, 1995. Translated by A. I. Dale from the 5th French edition of 1825.

[27] D. Lewis. Evaluating text categorization. In *Proceedings of the Speech and Natural Language Workshop*, pages 312–318. Morgan-Kaufmann, Feb. 1991.

[28] R. P. Lippmann. Pattern classification using neural networks. *IEEE Communications Magazine*, pages 47–64, Nov. 1989.

[29] B. N. Miller, J. T. Reidl, and J. A. Konstan. Experiences with GroupLens: Making Usenet useful again. In *USENIX Annual Technical Conference*, pages 219–233, 1997.

[30] S. Mirsky. Untangling the web. *IBM Research Magazine*, 4:8–10, Dec. 1997.

[31] D. Mladenic. Feature subset selection in text-learning. In *10th European Conference on Machine Learning*, 1998.

[32] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan-Kaufmann, 1991.

[33] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent sematic indexing: A probabilistic analysis. Submitted for publication.

[34] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[35] P. Raghavan. Information retrieval algorithms: A survey. In *Symposium on Discrete Algorithms.* ACM-SIAM, 1997. Invited paper.

[36] E. S. Ristad. A natural law of succession. Research report CS-TR-495-95, Princeton University, July 1995.

[37] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *SIGIR*, pages 232–241, 1994.

[38] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[39] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[40] H. Schutze, D. A. Hull, and J. O. Pederson. A comparison of classifiers and document representations for the routing problem. In *SIGIR*, pages 229–237, 1995.

[41] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating word of mouth. In *Computer Human Interaction (CHI).* ACM, 1995.

[42] S. Vaithyanathan. Document classification using principal component analysis. Personal communication, May 1997.

[43] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979. Also available on-line at `http://www.dcs.gla.ac.uk/Keith/Preface.html`.

[44] E. M. Voorhees. Using WordNet to disambiguate word senses for text retrieval. In *SIGIR*, pages 171–180, 1993.

[45] A. Wald. *Statistical Decision Functions.* Wiley, New York, 1950.

[46] S. M. Weiss and C. A. Kulikowski. *Computer Systems That Learn.* Morgan-Kaufmann, 1990.

[47] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning*, pages 412–420, 1997.

[48] T. Y. Young and T. W. Calvert. *Classification, Estimation and Pattern Recognition.* Elsevier, 1974.

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