A Scalability Analysis of Classifiers in Text Categorization *

Yiming Yang
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, U.S.A.
yiming@cs.cmu.edu

Jian Zhang
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, U.S.A.
jian.zhang@cs.cmu.edu

Bryan Kisiel
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, U.S.A.
bkisiel@cs.cmu.edu

ABSTRACT

Real-world applications of text categorization often require a system to deal with tens of thousands of categories defined over a large taxonomy. This paper addresses the problem with respect to a set of popular algorithms in text categorization, including Support Vector Machines, k-nearest neighbor, ridge regression, linear least square fit and logistic regression. By providing a formal analysis of the computational complexity of each classification method, followed by an investigation on the usage of different classifiers in a hierarchical setting of categorization, we show how the scalability of a method depends on the topology of the hierarchy and the category distributions. In addition, we are able to obtain tight bounds for the complexities by using the power law to approximate category distributions over a hierarchy. Experiments with kNN and SVM classifiers on the OHSUMED corpus are reported on, as concrete examples.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; H.4.m [**Information Systems Applications**]: Miscellaneous; I.5.4 [**Pattern Recognition**]: Applications–*Text processing*

General Terms

Algorithms, Theory, Experimentation

Keywords

complexity analysis; hierarchical text categorization; power law

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

SIGIR'03, July 28–August 1, 2003, Toronto, Canada.

1. INTRODUCTION

Real-world applications of text categorization (TC) often require a system to deal with tens of thousands of categories defined over a large taxonomy (such as Yahoo). Although many candidate classification methods have been published, it is difficult to tell which one(s) would scale to applications with such a large number of categories. A part of the difficulty comes from the fact that many of them were evaluated using only a relatively small number of categories (up to a few hundred), leaving the scalability in larger applications an open question.

For example, the OHSUMED corpus, which consists of 233,445 article abstracts in medical journals with 14,321 unique category labels (a large subset of the Medical Subject Headings (MeSH)), has become an evaluation benchmark in text categorization since 1994[19, 17]. To our knowledge, only one TC method was evaluated using the full domain of MeSH categories in OHSUMED, the k-nearest neighbor approach reported in[19, 17]. Evaluations of other methods used much smaller subsets (23, 28 or 49 categories) [9, 6] until the Text Retrieval Conference (TREC-9) in 2000, wherein a subset of 4904 categories from OHSUMED were selected for the filtering track[13]. However, only three systems were able to submit complete results on that subset of categories; the remaining systems used a subset of that subset, consisting of 500 categories. Since 2001, the Operational TC workshops [1] have been focused on the real-world applications of text categorization techniques, in which dealing with large sets of categories is an issue. Yet, comparable results on benchmark collections with respect to scaling remain rare. It is not only difficult to assess the state of the art with respect to the scalability of TC methods; it is also difficult to draw thorough conclusions about their effectiveness in large applications when only small subsets of the categories (say, 1% of the total) from the application domains have been arbitrarily chosen for the evaluations.

Given the insufficient empirical evidence for assessing the scalability of TC methods, a logical step at this stage is to analyze the theoretical complexities of TC algorithms under the assumption of dealing with very large sets of categories. Presenting such an analysis is the main contribution in this paper. First, we provide a formal complexity analysis for five popular algorithms in conventional (non-hierarchical) categorization, including Support Vector Machines (SVM), k-Nearest Neighbor (kNN), Ridge Regression (RR), Linear Least Squares Fit (LLSF), and Logistic Regression (LR). Second, we analyze the computational cost in hierarchical

[1] http://www.info.uta.fi/sigir2002/html/ws5.htm

text categorization for two alternative approaches: training a binary classifier for each category, and training a m-way classifier for each group of m categories where $m \leq 2$. Note that by "m-way" we do not mean that the classifier has to choose only one from m classes for each document. To the contrary, the classifier can assign any number (0 to m) of categories to a document. Third, we propose a new way to assess the scalability of classifiers based on category distributions (often skewed in TC applications). In particular, we show how to use the power law to formally bound the computational costs of classifiers in hierarchical classification).

The organization of this paper is as the follows: Section 2 introduces individual categorization methods and provides a formal complexity analysis for each in conventional, non-hierarchical categorization; section 3 analyzes the scaling issues in hierarchical classification, and proposes a new way to use the power law in category distributions to bound the complexities of classifiers; section 4 reports our experiments with kNN and SVM on the OHSUMED benchmark collection; section 5 summarizes the concluding remarks.

2. COMPLEXITY ANALYSIS FOR NON-HIERARCHICAL TC

We choose five methods which have been popular in text categorization and with high performance in benchmark evaluations, including SVM, kNN, RR, LLSF and LR.

2.1 Algorithms and Notation

One classification method may have more than one algorithm for its implementation. We choose one algorithm for each method which has had representative performance in published evaluations. The SVM, LR and RR algorithms are designed for binary classification, which requires one classifier per category for "yes" or "no" decisions on documents with respect to that category. The kNN and LLSF algorithms, on the other hand, are designed for m-way classification, which uses one classifier in total for all the categories. Thus our analysis on the scalability in non-hierarchical categorization with M categories covers two cases: using M binary classifiers, and using one M-way classifier, depending on the type of the algorithm chosen for the method. In both cases, we assume that efficient data structures, such as inverted indexing (when appropriate), are used for the sparse representations of documents and categories. We neglect issues of tokenization (stopword removal and stemming), feature selection and term weighting computation, since they are common to all algorithms and have been heavily reported on in the literature. We also leave parallel computing out of the focus in this paper.

The variables used in the following subsections are:

- N – the number of training documents
- V – the number of features (the vocabulary size)
- M – the number of training-set categories ($M < N$ in general)
- I – the number of iterations for iterative algorithms
- $\vec{x_i} \in R^V$ – the vector of the ith training document whose elements are the within-document term weights
- $y_i \in \{+1, -1\}$ – the category indicator in a binary classification model
- $\vec{y_i} \in \{1, 0\}^M$ – the category-indicator vector in an M-way classification model
- $\vec{w} \in R^V$ – the coefficient vector ("weight vector") of a linear model for a particular category
- $\langle \vec{x_i}, \vec{x_j} \rangle$ – the dot-product of two vectors $\vec{x_i}$ and $\vec{x_j}$
- L_d – the average document length (word count)
- L_v – the average number of unique words in a document
- L_f – the average number of documents indexed by a word, and it is obvious that $NL_v = VL_f$

Table 1 summarizes the time/space complexities for the algorithms that we analyze in the following sub-sections.

2.2 SVM

SVM is a promising classification method developed by Vapnik[14]. It applies Structural Risk Maximization, which aims to minimize the *generalization error* instead of the *empirical error* on training data alone. Multiple variants of SVM have been developed [14, 6]; here we limit our discussion to linear SVM due to its popularity and high performance in text categorization[6, 18, 8].

The optimization of SVM (dual form) is to minimize:

$$\vec{\alpha}^* = \arg\min_{\vec{\alpha}} \{-\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \vec{x_i}, \vec{x_j} \rangle\}$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0; \quad 0 \leq \alpha_i \leq C$$

The prediction is given by:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \langle \vec{x_i}, \vec{x} \rangle + b = \langle \vec{w}^*, \vec{x} \rangle + b$$

where $\vec{w}^* = \sum_{i=1}^{N} \alpha_i y_i \vec{x_i}$, and the bias b can be computed using any "unbounded support vector" ($0 < \alpha_{usv} < C$):

$$b, = y_{usv} - \langle \vec{w}^*, \vec{x}_{usv} \rangle$$

To train a SVM, we need to solve the optimization problem using Quadratic Programming (QP) techniques. Generally speaking, QP can be complex and inefficient for large data collections. Since, for SVM, both the objective function and feasible region are convex, it becomes a convex QP, and can be solved more easily than general quadratic programming. Furthermore, there exist algorithms that can solve this more efficiently by utilizing the sparseness of the text data.

Here we discuss the algorithm used in *SVM-Light* [7], which is one of the most popular SVM packages in text categorization. The basic idea [3] is to iteratively decompose the big QP problem into smaller ones (called "working set") and solve them sequentially until convergence is obtained. The training-time complexity for each iteration is:

$$O(q^2 L_v) << O(N^2 L_v)$$

where q is the size of working set, bounded by the training-set size, and L_v is the number of unique words per document on average. The space required is $O(q^2 + NL_v)$. The number of iterations I is usually around one thousand for the Reuters-21578 benchmark corpus, for example, and sometimes can go beyond ten thousand. Note that I would be affected by the choice of q, making a purely theoretical complexity analysis difficult. However, Joachims[7] showed a set

Table 1: Complexities of classifiers in non-hierarchical text categorization

Classifier	Training Time on M Categories	Testing Time per Document	Space Complexity	Type of Algorithm
SVM	$O(MN^c)$ $c \approx 1.2 \sim 1.5$	$O(ML_v)$	$O(NL_v + q^2)$	binary
kNN	$O(NL_d)$	$O(\frac{N}{V}L_v^2) + O(N)$	$O(NL_v)$	M-way
LLSF	$O(N^2 k_s)$	$O(ML_v)$	$O(NV)$	M-way
RR	$O(MINL_v)$	$O(ML_v)$	$O(NL_v)$	binary
LR	$O(MINL_v)$	$O(ML_v)$	$O(NL_v)$	binary

of empirical observations about the super-linear time complexity of SVM in training with respect to N, the number of training documents: $O(N^{1.2})$ on a web page collection[12], and $O(N^{1.5})$ on the OHSUMED corpus.

The complexity analysis above is for the training time on one category. Since the algorithm in *SVM-Light* (and other SVM algorithms applied to text categorization) requires one binary classifier for every category, the total training-time complexity for M categories is therefore $O(MN^c)$, where c is some domain-specific or corpus-specific constant.

In the testing phase, the dominating part is to compute the dot-product of the input vector (document) and the model vector for every class, which has a complexity of $O(ML_v)$. For simplicity, we could consider L_v as a parameter whose value depends on the domain or application but not on the training-set size. Thus, we can say that the online response time of SVM is $O(M)$ per document.

2.3 kNN

Different from SVM (and other methods in this paper), kNN is a "lazy-learning" algorithm, which means that it does not have an off-line learning phase. The so-called "training" phase in kNN is simply to index the training data for later use. Building the inverted index of documents for classification[15] is a mature technique with a complexity of $O(NL_d)$. If we consider L_d, the average length of documents, as an application-specific constant, then the training complexity of kNN is $O(N)$, i.e., linear in the training-set size both in time and space.

In the testing phase, each new document is compared against all the training documents using a pre-defined similarity metric. The k top-ranking training documents are then selected, and used to compute a score for every category as follows:

$$\text{score}(c_j|\vec{x}) = \sum_{\vec{x}_i \in \text{kNN} \& \vec{x}_i \in c_j} sim(\vec{x}, \vec{x}_i)$$

The document scoring part takes $O(L_f L_v) = O(\frac{NL_v^2}{V})$ time [15] where L_f is the number of postings per word on average in the inverted indexing, and V is the vocabulary size. Using a modified version of *quicksort*, one can obtain the k top-ranking documents (not sorted) in $O(N)$ time. Recall that the average computing time for standard quicksort is $O(N \log_2 N)$, accomplished by splitting the input list into two portions, and then splitting each portion again and again recursively. However, since our purpose is to find the top k neighbors, not to sort the list, we only need to apply quicksort to the larger portion in each step of the recursion, until a portion of size k is obtained. Sorting the resulting k documents, if desirable, takes $O(k \log_2 k)$ time in addition, which is negligible since $k << N$. The category scoring part is $O(k)$, assuming the use of a hash table to hold the cumulative scores for categories during the computation. This part is also negligible in the complexity analysis.

2.4 Linear Regression Using Truncated SVD

The Linear Least Squares Fit (LLSF) mapping method has had performance competitive with SVM and kNN in benchmark evaluations[17, 18]. The optimization problem in LLSF is defined as follows:

$$\begin{aligned} \mathbf{W}^* &= \arg\min_{\mathbf{W}} \|\mathbf{XW} - \mathbf{Y}\|_F^2 \\ &= \arg\min_{\mathbf{W}} \{\sum_{i=1}^{N}\sum_{j=1}^{M}(\langle \vec{x}_i, \vec{w}_j\rangle - y_{ij})^2\} \end{aligned}$$

where $\mathbf{X}$ is a document-term matrix whose elements are term weights in training documents, and $\mathbf{Y}$ is a document-category matrix whose elements $y_{ij} \in \{0, 1\}$ indicates whether the ith training document belongs to the jth category. The solution $\mathbf{W}$ is a term-category matrix whose element w_{tj} is the regression coefficient ("weight") of term t in the prediction of category c_j. The Frobenius matrix norm is defined as $\|\mathbf{A}\|_F^2 = \sum_{i,j} a_{ij}^2$.

If the matrix $\mathbf{X}^T\mathbf{X}$ is non-singular, then the LLSF solution has the form of

$$\mathbf{W}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

However, $\mathbf{X}^T\mathbf{X}$ is a term-term matrix whose inverse may not exist. The "if" condition is equivalent to assuming that all the words are independent of each other, which is obviously not true in reality. A more viable approach is to compute the *pseudo-inverse* $\mathbf{X}^+$ using the truncated Singular Value Decomposition (SVD) of matrix $\mathbf{X}$, which is guaranteed to find a solution in the form of

$$\mathbf{W}^* = \mathbf{X}^+\mathbf{Y}$$

A nice property of the pseudo-inverse is that it only depends on matrix $\mathbf{X}$, not on matrix $\mathbf{Y}$, which means that we only need to compute it once, based on the terms in training documents, and then use it again and again in fitting the regression functions with respect to categories. In this sense, this method is quite efficient when M is very large, as compared to training binary classifiers M times repeatedly and independently.

The Lanczos algorithm introduced in [2] and thoroughly analyzed by [1] is particularly efficient for solving LLSF on very large and sparse matrices, and has a good convergence property. The step-wise complexities are given below:

Step 1. Compute the truncated SVD

$$\mathbf{X} = \mathbf{USV}^T,$$

where $\mathbf{U}$ is matrix of right singular vectors, $\mathbf{V}$ is the matrix of left singular vectors, $\mathbf{S}$ is a diagonal matrix with the k_s largest singular values (on the diagonal in a descending order), and k_s is the user-specified number of singular values to compute. The time complexity of SVD is $O(Lk_s)$ where $L = \max\{N, V\}$, and the space

requirement is $O(Nk_s)$ for storing either $\mathbf{U}$ or $\mathbf{V}$. The value of k_s can be empirically chosen through validation. In our experiments with LLSF on benchmark collections (Reuters news stories, MEDLINE documents, etc), we observed the optimal ranges of k_s to be between a few hundred and one thousand[16].

Step 2. Compute the pseudo-inverse

$$\mathbf{X}^+ = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T = \mathbf{X}^T\mathbf{U}\mathbf{S}^{-2}\mathbf{U}^T$$

The time complexity here is $O(k_sN^2)$, dominated by the computation of $US^{-2}U^T$. The space complexity is $O(NV)$, for storing matrix $\mathbf{X}^+$.

Step 3. Compute the solution matrix $\mathbf{W}^* = \mathbf{X}^+\mathbf{Y}$. Since the matrix $\mathbf{Y}$ (document-category matrix) is very sparse, it is more efficient to compute the transpose

$$(\mathbf{W}^*)^T = \mathbf{Y}^T(\mathbf{X}^+)^T$$

which enables the use of the inverted indexing of $\mathbf{Y}$. The time complexity is $O(NL_c)$ where L_c is the average number of categories per document. The space needed for the inverted indexing is also $O(NL_c)$. Matrix $\mathbf{X}^+$ would make the space complexity $O(VM)$ if we need to keep it in a dense form. However, our previous work showed that aggressive elimination of non-influential elements from that matrix would not cause any loss of classification accuracy[16].

Among the above steps, the dominating part in the training time of LLSF is the matrix multiplication in Step 2, with a complexity of $O(k_sN^2)$. As for the space complexity, the dominating part is the storage required for matrix $\mathbf{X}+$, with a complexity of $O(NV)$. For the testing phase, the complexities of LLSF are the same as those in SVM. That is, both use one vector representation of a linear model per category, and both compute the dot-product of a new document and a category vector in the same way.

2.5 Ridge Regression (RR)

Ridge regression is a "regularized" version of linear regression. We limit our discussion to a binary version of the LR algorithm, which has been reported in the literature with a performance competitive with SVM and linear regression on benchmarks data sets[21, 10]. The classification problem is defined as minimizing the the following objective function:

$$\vec{w}^* = \arg\min_{\vec{w}}\{\frac{1}{n}\sum_{i=1}^{N}(y_i - \langle\vec{w},\vec{x}_i\rangle)^2 + \lambda\langle\vec{w},\vec{w}\rangle\}$$

It has a close-form solution as:

$$\begin{aligned}\vec{w}^* &= (\sum_{i=1}^{N}\vec{x}_i\vec{x}_i^T + N\lambda\mathbf{I})^{-1}\sum_{i=1}^{N}\vec{x}_i y_i \\ &= (\mathbf{X}^T\mathbf{X} + N\lambda\mathbf{I})^{-1}X^TY\end{aligned}$$

Note that using $(\mathbf{X}^T\mathbf{X}+N\lambda\mathbf{I})^{-1}$ instead of $(\mathbf{X}^T\mathbf{X})^{-1}$ makes the transformation matrix non-singular, provided $\lambda > 0$. As a result, we can compute the matrix inverse without SVD.

A variant of Gauss-Seidel[5] has been used for solving the ridge regression problems[21], which is an iterative algorithm for fitting a binary classification model for every category. The training time complexity is $O(INL_v)$ per category, or $O(MINL_v)$ for all the categories. The testing phase time and space complexities are exactly the same as those for SVM and LLSF. The space required in training is to store the inverted index of training documents; that is, on the order of $O(NL_v)$. As for the value of I, several hundred iterations have usually been enough for convergence in our experiments.

It is possible, in principle, to improve the efficiency of the current RR algorithm by computing the $(\sum_{i=1}^{N}\vec{x}_i\vec{x}_i^T + N\lambda\mathbf{I})^{-1}$ just once, and then sharing the resulting transformation matrix in the per-category computation cycles. This would yield a significant efficiency gain when the number of categories is large. We leave this potential to future work, and limit our discussions on RR to the current algorithm.

2.6 Logistic Regression (LR)

The logistic regression model estimates the conditional probability

$$p(y \mid \vec{x},\vec{w}) = \frac{1}{1+\exp(-y\langle\vec{w},\vec{x}\rangle)}$$

where the model is fitted by maximizing the conditional log-likelihood $\frac{p}{1-p}$, which is equivalent to minimizing the following:

$$\vec{w^*} = \arg\min_{\vec{w}}\{\frac{1}{N}\sum_{i=1}^{N}\log(1+\exp(-y_i\langle\vec{w},\vec{x}_i\rangle))\}$$

In practice, the "regularized" version is preferred:

$$\vec{w^*} = \arg\min_{\vec{w}}\{\frac{1}{N}\sum_{i=1}^{N}\log(1+\exp(-y_i\langle\vec{w},\vec{x}_i\rangle)) + \lambda\langle\vec{w},\vec{w}\rangle\}$$

Another variant of the Gauss-Seidel algorithm[21] has been used for solving LR, having exactly the same big-O notion complexities as we presented above for RR.

3. COMPLEXITY ANALYSIS FOR HIERARCHICAL TC

We have presented two types of categorization algorithms for non-hierarchical classification: binary classification algorithms, with complexities that grow linearly in the number of categories (SVM, LR and RR), and m-way classification algorithms (kNN and LLSF), with complexities that do not. Now we analyze their use and the computational cost in hierarchical text categorization. SVM and kNN are used as examples to illustrate the attendant issues, which should generalize to the other classifiers as well.

3.1 Classifier Allocation over the Hierarchy

Given a pre-defined taxonomy of categories (mostly in a tree structure but sometimes with acyclic graph parts), hierarchical classification is accomplished by allocating classifiers over individual nodes in the hierarchy, making local predictions for each test document and combining the local predictions (binary decisions with or without weights) for final decisions. Without loss of generality, we assume that all categories are assigned to leaf nodes [2].

The two strategies typically used in hierarchical categorization are:

1) allocating an m-way classifier on each non-leaf node;

2) allocating a binary classifier on each pair of parent-child nodes;

[2]For cases where non-leaf nodes have their own category labels, we can convert them to the above situation by adding one child node for each, representing the non-leaf category.

Let K be the number of leaf nodes on a hierarchy, K' be the number of non-leaf nodes, and $b > 1$ be the branching factor of on average. It follows that $1 + (b-1)K' = K$.

For example, if $K = 10,000$ and $b = 10$, then $K' = 1,111$. Thus, the first strategy requires 1,111 b-way classifiers over the hierarchy, while the second requires $K + K' - 1 = 11,110$ binary classifiers.

3.2 Complexities in Hierarchical Settings

Using Q_m and Q_2 to denote the total complexity of using m-way and binary classifiers over the hierarchy respectively, we have:

$$\begin{aligned} Q_m = \sum_{i=0}^{h-1}\sum_{j=1}^{m_i} O(n_{ij}^c) &= \sum_{i=0}^{h-1} O(N_i^c)\sum_{j=1}^{m_i}\pi_{ij}^c \\ &\leq O(N_0^c)\sum_{i=0}^{h-1}\sum_{j=1}^{m_i}\pi_{ij}^c \qquad (1) \end{aligned}$$

and

$$Q_2 = b \times \sum_{i=0}^{h-1}\sum_{j=1}^{m_i} O(n_{ij}^c) \leq b \times O(N_0^c)\sum_{i=0}^{h-1}\sum_{j=1}^{m_i}\pi_{ij}^c \qquad (2)$$

where

- h is the depth of the hierarchy;
- m_i is the number of categories defined at the ith level;
- $i = 0, 1, \ldots, h$, and $i = 0$ corresponds to the root level;
- $j = 1, 2, \ldots, m_i$ are ranks of categories based on their sizes at level i;
- n_{ij} is the number of local training documents;
- $N_i = \sum_{j=1}^{m_i} n_{ij}$, and we have $N_i \leq N_0$ (the number of document-category pairs in the training set);
- $\pi_{ij} = \frac{n_{ij}}{N_i}$ and $\sum_{j=1}^{m_i}\pi_{ij} = 1$;
- $O(n^c)$ is the complexity of a single classifier, assuming polynomial time in the size of the local training set.

When using kNNs, for example, we have $c = 1$ if considering L_v (the average number of unique words per document) invariant with respect to the training-set size.

Substituting $c = 1$ in formula 1 yields a complexity of:

$$Q_{\text{knn}} \leq O(N_0)\sum_{i=0}^{h-1}\sum_{j=1}^{m_i}\pi_{ij} = h \cdot O(N_0)$$

This complexity does not depend on the category distributions or on the number of classifiers needed over the hierarchy; it is simply linear in the depth (h) of the hierarchy and the single-classifier complexity on the top-level training set.

When using SVMs, on the other hand, we do not get as simple-formed a complexity as that of kNNs because the single-classifier complexity is super-linear (or higher) in SVM. For example, $c \approx 1.5$ for SVM on the OHSUMED corpus, based on empirical observations[6]. Nevertheless, the complexity can be computed using formula 2, where it does depend on category distribution over the hierarchy.

3.3 Compleixty Estimation Using the Power Law

Given the dependency between category distributions over a hierarchy and the complexities of classifiers (especially for those with super-linear or higher complexity), we wonder whether we could characterize the typical distributions by using a well-known family of statistical density functions, and whether we could use a parametric analysis to make the complexity assessment simpler yet still accurate. These questions lead us to the examination of the "power law" in text categorization.

3.3.1 Power law as a general phenomena

As an interesting phenomenon, the power law has been observed in multiple domains, including an early discovery in cognitive science regarding human learning rate through repetitive tasks[11], and the recent observations about the Internet topology[4]. The application of Zipf's law to word distribution over documents is another well-known example, which has been commonly observed in Information Retrieval. The power law has an exponential form: $y = \alpha x^{-\beta}$ (or, equivalently, $\log y = \log\alpha - \beta\log x$), where $\alpha > 0$ and $\beta > 0$ parameterize the equation. The law simply says that two variables, x and y, are linearly correlated with a negative slope in log-scale. When $\beta = 1$, the power law reduces to Zipf's law. We do not know whether the observation of the power-law phenomena in multiple domains is merely a coincidence or whether a deeper philosophical interpretation could be made. Regardless, we have found that category distributions in data from real-world applications are often highly skewed, including news stories, journal articles and web pages [18, 20]. The power law is therefore a natural candidate for the characterization of those skewed distributions - more appropriate than using Zipf's law (assuming a fixed slope of -1) or a uniform distribution (assuming a flat slope of 0).

3.3.2 Power law in category distributions

Assuming the category distribution in a corpus obeys the power law, we have

$$n_j = \alpha j^{-\beta}$$

where $j = 1, 2, \ldots, M$ are the size-based ranks of categories, and n_j is the document frequency of the jth category in that corpus. Parameter α can be determined by setting $j = 1$, which yields $\alpha = n_1$, the number of documents in the most common category. Parameter β can be computed by fitting a regression line for the observed rank/frequency pairs in that corpus, and the slope of that is line is the value of $-\beta$. Thus, the equivalent formula for the power law is:

$$n_j = n_1\, j^{-\beta} \qquad (3)$$

Figure 1 shows the category distributions in several benchmark corpora of documents, including RCV1 (Reuters news stories), Hoover (web pages of industrial companies) and OHSUMED[17, 20, 8]. The x-axis is category rank; the y-axis is document frequency. Each curve is obtained by interpolating the rank/frequency pairs in a corpus. The slope of each curve shows how skewed the category distribution is, and the slice corresponding to a particular category reflects the amount of data that category has. Obviously, all the curves differ substantially from a flat line (uniform distribution), but the high-frequency region of each curve is approximately linear, although the slopes differ from node to node. This implies that we can use the power law to estimate the document frequencies with high accuracy, for the subset of relatively common categories in a corpus. This is

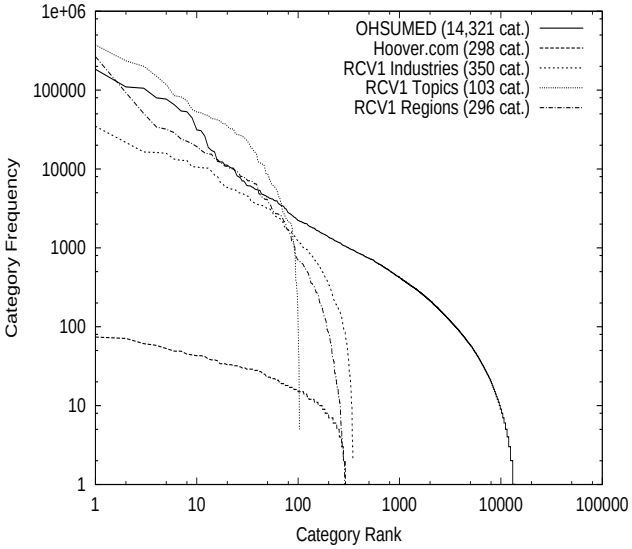

Figure 1: Category rank vs. frequency in benchmark corpora

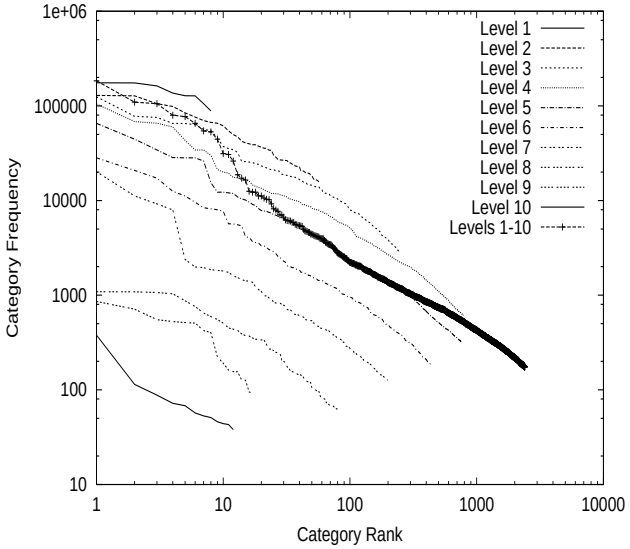

Figure 2: Category distributions at different levels of MeSH in OHSUMED (rare categories corresponding to 15% of the data at each level are not shown in the graph).

important for analyzing the complexity of hierarchical categorization, since those frequencies dominate the complexity in the quantity $Q_m = \sum_{i=0}^{h-1} \sum_{j=1}^{m_i} O(n_{ij}^c)$, particularly when $c > 1$. The same statement applies to the complexity analysis for Q_2.

Figure 2 shows the multi-level category distributions in OHSUMED, where the MeSH categories are defined over 10 levels on the hierarchy. To make the linear portions of those curves more visible, we cut off the right-most portion of each curve to make the area underneath that curve correspond to 85% of the data in that level. Interestingly, all the curves have a similar trend, except those at the top one or two levels. If we fit a straight line as the approximation of each curve, then the slopes of all of the lines are similar to each other, and, all can be approximated by the the slope of the global curve.

Next, we show how to use the power law to focus on the dominating components at each level in the complexity analysis, and to obtain simpler formulas for computing the exact complexity (or tight bounds) in several interesting cases.

3.3.3 *Bound analysis*

The power law for individual levels has the form

$$n_{ij} \approx n_{i1}\, j^{-\beta_i} \tag{4}$$

where n_{i1} is the number of training documents in the most common category at the ith level, and β_i is the level-specific slope of the regression line. Using this to estimate of the complexity Q_m, we have:

$$\begin{aligned} Q_m &= \sum_{i=0}^{h-1} \sum_{j=1}^{m_i} O(n_{ij}^c) \\ &\approx \sum_{i=0}^{h-1} \sum_{j=1}^{m_i} O(n_{i1}^c j^{-c\beta_i}) \\ &= \sum_{i=0}^{h-1} O(n_{i1}^c) \sum_{j=1}^{m_i} j^{-c\beta_i} \end{aligned} \tag{5}$$

The upper bound of the complexity is:

$$Q_m \le \sum_{i=0}^{h-1} O(N_0^c) \frac{g^{(U)}(m_i, c\beta_i)}{[g^{(L)}(m_i, \beta_i)]^c} \tag{6}$$

where N_0 is the size of the top-level training set, and function $g(.)$ is defined as:

$$g(m, a) \stackrel{def}{=} \sum_{j=1}^{m} j^{-a}.$$

We now show the upper bound $g^{(U)}(m, a)$ and the lower bound $g^{(L)}(m, a)$ of this function, conditioned on the parameter values:

$$\begin{cases} g(m,a) = \sum_{j=1}^{m} j^a = m & : \quad a = 0 \\ \ln(1+m) < g(m,a) < 1 + \ln m & : \quad a = 1 \\ \frac{1-(m+1)^{1-a}}{a-1} < g(m,a) < \frac{a-m^{1-a}}{a-1} & : \quad a \neq 0, 1 \end{cases}$$

The first cast ($a = 0$) corresponds to a uniform distribution of categories. The second case ($a = 1$) corresponds to a Zipf's law distribution, whose bounds are derived based on the following inequality:

$$\int_1^{m+1} \frac{1}{x} dx < \sum_{j=1}^{m} j^{-1} < 1 + \int_1^m \frac{1}{x} dx.$$

For the third case ($0 < a < 1$ or $1 < a$), we used the following inequality to derive the bounds:

$$\int_1^{m+1} x^{-a} dx < \sum_{j=1}^{m} j^{-a} < 1 + \int_1^m x^{-a} dx.$$

We need one more piece to establish the proof of Formula 6, that is: $n_{i1} \le \frac{N_0}{g(m_i, \beta_i)}$.

Proof. Given $\sum_{j=1}^{m_i} n_{ij} = N_i \le N_0$, we have:

$$\sum_{j=1}^{m_i} n_{ij} = n_{i1} \sum_{j=1}^{m_i} j^{-\beta_i} \le N_0$$

$$n_{i1} \leq \frac{N_0}{\sum_{j=1}^{m_i} j^{-\beta_i}} = \frac{N_0}{g(m_i, \beta_i)} \quad (7)$$

$$\begin{aligned} Q_m &= O(\sum_{i=0}^{h-1} n_{i1}^c \sum_{j=1}^{m_i} j^{-c\beta_i}) \\ &\leq O(\sum_{i=0}^{h-1} \frac{N_0^c}{[g^{(L)}(m_i, \beta_i)]^c} g^{(U)}(m_i, c\beta_i)) \\ &= O(N_0^c) \sum_{i=0}^{h-1} \frac{g^{(U)}(m_i, c\beta_i)}{[g^{(L)}(m_i, \beta_i)]^c} \quad (8) \end{aligned}$$

Formula 6 suggests that we can get a relatively accurate assessment of the total complexity by running one m-way classifier at the top level with the full training set (for a realistic measure of $O(N^c)$), and plugging the corpus-specific parameters (m_i and β_i values) into the function $g(m, x)$. To estimate the Q_2 complexity is similar: we just need to run one or a few binary classifiers with the full training set to get some assessment of $O(N_0^c)$, and then use Formula 6 enlarged by a factor of b. These close-form approximations and bounds greatly simplify the complexity assessment.

4. CONCRETE EXAMPLES

Let us use SVM and LLSF classifiers on OHSUMED for examples. Let the complexity parameter c be 1.5 in SVM, and let c be 2 in LLSF, considering k_s as a constant. The hierarchy has a depth $h = 10$, an average branching factor $b \approx 9$, and a global power-law parameter $\beta \approx 0.84$ (Figure 2). For a rough estimate, we set $\beta_i \approx \beta = 0.84$, and $m_i \approx b^i$. Substituting those parameters in Formula 6 and multiplying the result by a factor of b, we have a complexity of:

$$\begin{aligned} Q_{\text{svm-train}} &\leq b \cdot O(N_0^c) \sum_{i=0}^{h-1} \frac{\frac{1}{c\beta-1}(c\beta - m_i^{1-c\beta})}{[\frac{1}{c\beta-1}(1-(m_i+1)^{1-c\beta})]^c} \\ &\approx 9 \cdot O(N_0^{1.5}) \sum_{i=0}^{9} \frac{3.85 \times (1.26 - (9^i)^{-0.26})}{[3.85(1-(9^i+1)^{-0.26})]^{1.5}} \\ &\approx 61.8 \cdot O(N_0^{1.5}) \quad (9) \end{aligned}$$

Similarly, the training-time complexity for using m-way LLSFs over the hierarchy is computed as:

$$\begin{aligned} Q_{\text{llsf-train}} &\leq O(N_0^c) \sum_{i=0}^{h-1} \frac{\frac{1}{c\beta-1}(c\beta - m_i^{1-c\beta})}{[\frac{1}{c\beta-1}(1-(m_i+1)^{1-c\beta})]^c} \\ &\approx O(N_0^2) \sum_{i=0}^{9} \frac{1.47 \times (1.68 - (9^i)^{-0.68})}{[1.47(1-(9^i+1)^{-0.68})]^2} \\ &\approx 13.3 \cdot O(N_0^2) \quad (10) \end{aligned}$$

To get some concrete measures, we ran kNN and SVM on a large subset of OHSUMED data in a "flat" fashion, which means to use one kNN for the entire set of categories and documents, or to use one SVM for each category but on the entire set of training documents. With kNN, we conducted the experiments both on the full domain of MeSH categories and on the Heart Diseases (HD) sub-domain. With SVM, we conducted the experiment only with the HD sub-domain. The corpus contains a total of 233,445 documents dated 1987-1991, with 14,231 unique categories assigned to those documents. We arbitrarily chose the documents from 1987 for training and the documents from 1988 for testing. The same set of documents were used in the full-domain (MeSH) and the sub-domain (HD) experiments; the only difference in those cases was in document labeling. For the full-domain experiments, we used the original category labels; for the sub-domain experiments, we replaced each non-HD category label with a dummy name of "None". Table 2 summarizes the data set statistics, and Table 3 shows the CPU times and the performance scores (in $F1$) for those runs. A 2.0 GHz Pentium 4 Xeon PC with 2GB RAM was used for these experiments.

Table 2: Statistics of Data Sets

Documents	Uniq. Cat.	Assigned Cat.	Domain
OH.87: 36,890	10,889	439,956	MeSH
OH.88: 47,054	11,474	566,690	MeSH
HD.87: 36,890	94	3,768	HD
HD.88: 47,054	98	4,961	HD

Table 3: CPU times for flat kNN and SVMs

	SVM/HD	kNN/HD	kNN/MeSH
training time	40 min	11 sec	12 sec
testing time	25 min	311 min	328 min
micro-avg F1	0.516	0.526	0.478

Based on these results, we can make further projections for these classifiers with respect to scaling.

SVM training time. Using the training time of SVM on 94 HD categories to estimate the training time of using SVMs in non-hierarchical TC with the full domain of 14,321 MeSH categories, we have:

$$\frac{40}{94} \times 14,321/60 \text{ min} = 102 \text{ hours}$$

For hierarchical SVMs, we can use formula 9:

$$61.8 \cdot O(N_0^{1.5}) = 61.8 \cdot \frac{40}{94} = 26.3 \text{ min}$$

Clearly, using hierarchical TC can substantially enhance the scalability for SVM in terms of training time. Whether or to what degree this efficiency gain comes at the cost of effectiveness is a question for future research.

SVM online response time. In the HD experiment, the average response time per document was:

$$\frac{25 \text{ min}}{47,054} \times 60 \approx 0.03 \text{ sec}$$

When using flat SVMs, since the response time is linear in the number of training-set categories, the projected response time for the problem with the full-domain of 14,321 categories in OHSUMED would be:

$$\frac{14,321}{94} \times 0.03 \approx 4.6 \text{ sec}$$

When usnig hierarchical SVMs, and with a pachinko-machine type of propagation (i.e., a lower level SVM is activated only if its parent makes a "yes" decision), the online computation time would be:

$$\frac{0.03}{94} \times h \times b' \approx 0.003 \times b' \text{ sec}$$

where b' is the branching factor on average in the beam search.

KNN online response time. Recall that the main computations in kNN are in the testing phase. The complexity of each level of kNNs in the hierarchy is roughly the same

(or less) as that of flat kNN with the full set of documents. Therefore, a hierarchical use of kNNs always increases the computational cost by a factor of h, which is the height of the hierarchy. Thus, hierarchical classification with kNNs is worth doing only if the effectiveness can be significantly improved by doing so. The online response time per document of flat kNN was empirically measured (Table 3) as:

$$\frac{60 \times 328 \text{ min}}{47,054} = 0.4 \text{ sec}$$

Using this result to project the online response time of kNNs in hierarchical categorization with $h = 10$ and $b = 9$ (the parameter values in OHSUMED), and assuming the pachinko-machine policy again, we have the estimated CPU seconds per document as:

$$0.4 \times (1 + \frac{b^{-1} - b^{-h}}{1 - b^{-1}} \cdot b') < 0.4 \times (1 + \frac{b^{-1}}{1 - b^{-1}} \cdot b')$$

$$= 0.4 \times (1 + 0.125b') = 0.4 + 0.05b' \text{ sec}$$

This formula is based on the assumption that the size of the local training set at each level reduces by a factor of $b = 9$ on average.

5. SUMMARY

In this paper, we first present complexities of five popular algorithms in non-hierarchical text categorization, then introduce their usage in hierarchical settings in types of binary and m-way classifiers. By using the power law to model category distributions, we are able to derive bounds of complexities for polynomial-time algorithms in hierarchical text categorization. Finally, concrete examples with kNN and SVM are given on the OHSUMED benchmark corpus.

We hope this study provides useful insights to the assessment of current techniques in text categorization with respect to scaling. For future work, we plan to conduct large-scale experiments with several high-performing classification methods in order to investigate the tradeoff between the efficiency and effectiveness of those methods in solving very large classification problems.

6. ACKNOWLEDGMENTS

We thank Jamie Carbonell at Carnegie Mellon University for the fruitful discussion about the power law observations in early AI research. This research is sponsored in part by the National Science Foundation (NSF) under grants EIA-9873009 and IIS-9982226, and in part by the DoD under award 114008-N66001992891808. However, any opinions or conclusions in this paper are the authors' and do not necessarily reflect those of the sponsors.

7. REFERENCES

[1] M. Berry. Large-scale singular value computations. volume 6-1, pages 13–49, 1992.

[2] J. Cullum and R. Willoughby. Lanczos algorithm for large symmetric eigenvalue computations. In *Theory*, volume 1, Boston, MA, 1985.

[3] R. F. E. Osuna and F. Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII–Proceedings of 1997 IEEE Workshop*, New York, 1995.

[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.

[5] G. Golub and C. V. Loan. *Matrix Computations (3rd edition)*. Johns Hopkins University Press, Baltimore, MD, 1996.

[6] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *European Conference on Machine Learning (ECML)*, pages 137–142, Berlin, 1998. Springer.

[7] T. Joachims. *The Maximum-Margin Approach to Learning Text Classifiers: Methods, Theory, and Algorithms*. Ph.D. thesis, University of Dortmund, 2000.

[8] D. Lewis, F. Li, T. Rose, and Y. Yang. The reuters corpus volume i as a text categorization test collection. In *Journal of Machine Learning Research*, page (to appear), 2003.

[9] D. D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. In *19th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'96)*, 1996. 298-306.

[10] F. Li and Y. Yang. A loss function analysis for classification methods in text categorization. In *ICML*, 2003 (submitted).

[11] A. Newell and P. Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J. Anderson, editor, *Cognitive Skills and Their Acquisition*, pages chapter 1, pp 1 –55, Hillsdale, NJ, 1981. Lawrence Erlbaum Associates, Inc.

[12] J. Platt. Sequetial minimal optimization: A fast algorithm for training support vector machines. In *Technical Report MST-TR-98-14*. Microsoft Research, 1998.

[13] S. Robertson and S. Walker. Microsoft cambridge at trec-9. In D. Harmon, editor, *Proceedings of the Nineth Text REtrieval Conference (TREC-9)*, 2001.

[14] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.

[15] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *17th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, pages 13–22, 1994.

[16] Y. Yang. Noise reduction in a statistical approach to text categorization. In *Proceedings of the 18th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'95)*, pages 256–263, 1995.

[17] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval*, 1(1/2):67–88, 1999.

[18] Y. Yang and X. Liu. A re-examination of text categorization methods. In *The 22th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 42–49, 1999.

[19] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In J. D. H. Fisher, editor, *The Fourteenth International Conference on Machine Learning (ICML'97)*, pages 412–420. Morgan Kaufmann, 1997.

[20] Y. Yang, S. Slattery, and R. Ghani. A study of approaches to hypertext categorization. In *Journal of Intelligent Information Systems*, volume 18, pages 219–241. Kluwer Academic Press, 2002.

[21] T. Zhang and F. J. Oles. Text categorization based on regularized linear classificatin methods. In *Information Retrieval*, volume 4, pages 5–31, 2001.