CATEGORIZATION IN A HIERARCHICALLY STRUCTURED TEXT DATABASE

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ferhat Kutlu
February, 2001

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. H. Altay Güvenir (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. İlyas Çiçekli

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Attila Gürsoy

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

Categorization in a Hierarchically Structured Text Database

Ferhat Kutlu
M.S. in Computer Engineering
Supervisor: Assoc. Prof. H. Altay Güvenir
February 2001

Over the past two decades there has been a huge increase in the amount of data being stored in databases and the on-line flow of data by the effects of improvements in Internet. This huge increase brought out the needs for intelligent tools to manage that size of data and its flow. Hierarchical approach is the best way to satisfy these needs and it is so widespread among people dealing with databases and Internet. Usenet newsgroups system is one of the on-line databases that have built-in hierarchical structures. Our point of departure is this hierarchical structure which makes categorization tasks easier and faster. In fact most of the search engines in Internet also exploit inherent hierarchy of Internet. Growing size of data makes most of the traditional categorization algorithms obsolete. Thus we developed a brand-new categorization learning algorithm which constructs an index tree out of Usenet news database and then decides the related newsgroups of a new news by categorizing it over the index tree. In learning phase it has an agglomerative and bottom-up hierarchical approach. In categorization phase it does an overlapping and supervised categorization. k Nearest Neighbor categorization algorithm is used to compare the complexity measure and accuracy of our algorithm. This comparison does not only mean comparing two different algorithms but also means comparing hierarchical approach vs. flat approach, similarity measure vs. distance measure and importance of accuracy vs. importance of speed. Our algorithm prefers hierarchical approach and similarity measure, and greatly outperforms k Nearest Neighbor categorization algorithm in speed with minimal loss of accuracy.

Keywords: learning, categorization, clustering, hierarchy, Usenet, newsgroup, top-level, header-line, posting, frequency, norm-scaling, similarity measure, distance measure, agglomerative, bottom-up, stemming, stopword, index

ÖZET

Hiyerarşik Yapıda Olan Bir Veritabanının Kategorizasyonu

Ferhat Kutlu
Bilgisayar Mühendisliği, Yüksek Lisans Programı
Tez Yöneticisi: Doç. Dr. H. Altay Güvenir
Şubat 2001

Son yirmi yıldır İnternet alanındaki gelişmelerin etkisiyle veritabanlarında saklanan verinin boyutunda ve on-line veri akışında büyük bir artış oldu. Bu artış beraberinde, bu büyüklükteki veri yığınını ve akışını yönetebelicek araçlara olan ihtiyaçları açığa çıkardı. Hiyerarşik yaklaşım, bu ihtiyaçları tatmin için en iyi yoldur ve İnternet ve veritabanlarıyla uğraşanlar arasında da çok yaygındır. Usenet haber grupları sistemi, içinde yapısal bir hiyerarşi bulunduran on-line veritabanlarından biridir. Bizim hareket noktamız da kategorizasyon işlerini daha kolay ve hızlı hale sokan bu hiyerarşik yapıdır. Aslında İnternetteki arama motorlarının çoğu İnternetin yapısal hiyerarşisinden faydalanmaktadır. Verilerin artan boyutu birçok geleneksel kategorizasyon algoritmasını kullanılmaz hale sokmuştur. Bu sebeple Usenet haberlerinden oluşan bir veri tabanından indeks çıkartan ve daha sonra bu indeks üzerinden kategorizasyon yaparak yeni bir haberin ilgili haber gruplarını belirleyen yeni bir kategrizasyon öğrenme algoritması geliştirdik. Bu algoritma öğrenme safhasında birleştirici ve aşağıdan yukarıya hiyerarşik bir yaklaşıma sahiptir. Kategorizasyon safhasında ise örtüşümlü ve denetlemeli bir kategorizasyon yapmaktadır. Algoritmamızın kompleksite ölçütünü ve doğruluğunu kıyaslamak için k En Yakın Komşu kategorizasyon algoritması kullanılmıştır. Bu kıyaslama sadece iki algoritmanın kıyaslanması demek değil, hiyerarşik yaklaşımın düz yaklaşımla, benzerlik ölçütünün mesafe ölçütüyle ve doğruluğun öneminin hızın önemiyle kıyaslanmasıdır. Algoritmamız hiyerarşik yaklaşımı ve benzerlik ölçütünü benimsemekte ve küçük bir doğruluk kaybıyla k En Yakin Komşu algoritmasından çok daha hızlı çalışmaktadır.

Anahtar sözcükler: öğrenme, kategorizasyon, bölümleme, hiyerarşi, Usenet, haber grubu, üst seviye, başlık satırı, postalama, frekans, norm ölçekleme, benzerlik ölçütü, mesafe ölçütü, birleştirici, asağıdan yukarı, eklerinden ayırma, yaygın kelime, indeks

ACKNOWLEDGMENTS

First and foremost I would like to express my thanks and gratitude to my advisor and thesis supervisor Assoc. Prof. H. Altay Güvenir for his invaluable supervision, motivation and support during my post graduate study and this research.

I am also grateful to Assist. Prof. İlyas Çiçekli and Assist. Prof. Attila Gürsoy for their instructive comments.

I would also like to thank Prof. Erol Arkun, Prof. Mehmet B. Baray, Prof. Varol Akman, Prof. Bülent Özgüç, Assoc. Prof. Cevdet Aykanat, Assoc. Prof. Özgür Ulusoy, Assist. Prof. Uğur Doğrusöz and Assist. Prof. Uğur Güdükbay for their teaching during the two and a half years that I spent in Bilkent University.

Anneme, babama ve kardeşime...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Amount of on-line information is growing at an ever-increasing rate and the needs for concepts to help manage this huge size of information are rising each day. One of these concepts is the *categorization* of every kind of data so that data parts with similar contents are in the same category. To date there have been many categorization algorithms implemented. This is because different types of data require different types of techniques, and growing sizes of data creates the need not only for new hardware but also for new software originating from algorithms faster than the previous ones.

In this thesis we try to figure out the benefits of built-in hierachical structures in text databases by developing a brand-new algorithm and implementing it. We used a sample database downloaded from Usenet newsgroups because:

- Usenet system has such a built-in hierachical structure in itself which makes us able to run our algorithm without a pre-arrangement of the input data.
- Usenet system is a source of large number of documents and there is always new data available for training and testing [39].
- Most of the news are text-based and there is no need to worry about removing HTML commands or interpreting image files.
- There are a large variety of subjects covered, so it is possible to study a particular area or more general topics.

- Postings are grouped together by category, so initiating a supervised learning is very easy.

1.1 Usenet System

As a global definition Usenet system is a set of people with common interests exchanging messages tagged with one or more universally-recognized labels called *newsgroups*. More technically, Usenet system is a world-wide distributed discussion system including a set of newsgroups with names that are classified *hierarchically* by subject. Messages are posted to those newsgroups by people on computers with the appropriate software, and then they are broadcasted to other interconnected computer systems via a wide variety of networks, but the bulk of modern Usenet traffic is transported over the Internet as a line of the *Information Superhighway*. In short, Usenet system is such a conference system embedded in Internet.

```
From:  Buday Gergely <gergoe@math.bme.hu>
Date:  23 Feb 2000 16:59:08 +1100
Subject:  iterative deepening
Organization:  Technical University of Budapest
Newsgroups:  comp.ai, comp.ai.philosophy

Hi Folks,
I've bumped into the expression 'iterative deepening'.
I've tried to understand it from a 1985 AI paper
(R.Korf, Vol.  27 pp.  97-109),but his explanation is not
really clear, at least for me.
Could anybody explain it clearly, or provide a
(preferrably electronical) reference to it?

Thanks in advance...Gergely
```

Figure 1.1: A Sample News

A sample news is given in Figure 1.1. Most common header-lines in a news are *From* (name and e-mail address of sender), *Date* (posting date), *Subject*

(a few words about the matter), *Organization* (name of university, company, etc.), *Newsgroups* (names of newsgroups to be posted). Below the header-lines there is the text part of news and attaching a file to news is also possible.

Important rules to obey while using Usenet system[1]:

- Never forget that the person on the other side is a human being.
- Do not blame system administrators for their users' behaviour.
- Never assume that a person is speaking for an organization.
- Be brief and pay attention to what you say about others.
- Your postings reflect upon you, be proud of them.
- Use descriptive titles.
- Think about your audience.
- Be careful with humour and sarcasm.
- Only post a message once.
- Please rotate material with questionable content.
- Summarize what you are following up.
- Use mail, don't post a follow-up.
- Read all follow-ups and do not repeat what has already been said.
- Check your return e-mail address and expect responses.
- Double-check follow-up newsgroups and distributions.
- Be careful about copyrights and licenses.
- Cite appropriate references.
- Mark or rotate answers or spoilers.
- Spelling flames considered harmful.
- Do not overdo signatures.

[1] Available from http://www.hypernews.org/HyperNews/get/usenet.html

- Limit line-length and avoid control characters.
- Do not use Usenet system as a resource for homework assignments.
- Do not use Usenet system as an advertising medium.
- Avoid posting to unnecessarily multiple newsgroups.

These rules are to make Usenet system a functional, helpful, and dependable platform. Most of them are easy to obey but the last rule is the hardest one to realize. Because there are a large number of newsgroups embedded in Usenet system and their organization is not very clear. Sometimes it is impossible or takes much time to detect which newsgroup is dealing with what. Fortunately each day new newsgroups are popping up and enriching Usenet platform. But unfortunately the total number of available newsgroups is getting near to the point of becoming impossible to keep track of even for a frequent user.

1.2 Motivation

Top-level names of the newsgroups already available in Usenet system are *alt* (alternative newsgroups), *bionet* (Biology Network), *bit* (originating from BITNET (IBM mainframe)), *clari* (Clarinet News Service (commercial)), *comp* (computer newsgroups), *gnu* (GNU operating system), *misc* (miscellaneous newsgroups), *news* (Usenet news), *rec* (recreational newsgroups), *sci* (science newsgroups), *soc* (social issues newsgroups), *talk* (talk newsgroups)[2]. All other newsgroup names are produced by adding different names to top-level names and they are seperated by dots such as *comp.ai.philosophy*.

Multiple copies of a posting appearing in unrelated multiple newsgroups are called *spamming*, and a posting that has multiple newsgroups on its *Newsgroups* header line is *cross-posted*. It might look easy to find out a related newsgroup to your new message by beginning with top-level names and going down the hierarchy. But this is neither useful most of the times nor does it find out other related newsgroups placed at some other braches of the hierarchy for real cross-posting but not spamming.

[2]Available from http://www.faqs.org/faqs/usenet/hierarchy-list/

For a new user brute force approach is that: Read as many news as possible from each newsgroup and construct something such as a map to see his/her way to post a message and keep this map updated. Most probably such a user will no longer be a user after a few days. In reality Usenet users always post their messages to their most favorite newsgroups. Therefore they are not aware of other related newsgroups and randomly cross-post their messages without caring for the relation issue which gives way to spamming most of the time.

Our approach is quite different. Technology brings its imperfection with its perfection but this imperfection also could be solved by the same technology. Imagine that you have a software which takes your new message as an input and looks up an index which is ready and updated beforehand and finally fills in the *Newsgroups* header line of your message form. All you need is to install such a software and supply its initial database for constructing an index or in other words for *learning*.

In this work we challenge the unpredictable nature of the Usenet newsgroups. Usenet system has an evolving structure. We want to invent an intelligent algorithm which learns the most recent structure of Usenet system, solves almost all of the problems discussed above, and gives way to the implementation of the imagined software mentioned in the previous paragraph. The main target of this algorithm is to make Usenet users feel comfortable about where to post their newly written messages.

In general terms, our purpose in choosing such a problem area, or such a database is to deal with its built-in hierarchical structure. We want to show that if a database has an already present and substantially built-in hierarchy in itslef then categorization could be done faster and more accurately than the traditional categorization algorithms do.

The outline of our algorithm can be summarized in a few sentences. It begins with a database that contains various messages from all newsgroups or at least from some newsgroups under the top-level names we are interested in. This database should have sufficient number of documents downloaded from each newsgroup in itself to be able to do an exact learning - lets say a number between 50 - 100 news for each newsgroup. Then we run our algorithm to construct an index tree which is as high as the longest hierarchy in the database

such that each node of the tree contains different number of features inherited from its children. Finally a new document/a set of documents travel(s) down the tree by the guidance of a similarity measure and a *threshold* value of this similarity measure to find its/their related category or multiple categories.

We are doing *Categorization in a Hierarchically Structured Text Database* in this thesis. So we get the abbreviation **CHSD** for our algorithm. Newsgroups have a hierarchy implied in their names such as a newsgroup called *comp.groupware.lotus-notes.admin* is placed at depth four in the hierarchy of Usenet system. This is the feature that we exploited in our algorithm to construct the index tree at the learning phase. In other words, the index tree constructed by our algorithm is an exactly the same copy of the already present hierarchy in Usenet system. Actually this is the main force behind our challenge for speed in categorization. As for the correctness, we use a few tools such as an exclusive stoplist of words to be ignored, norm-scaling formula for weightening the features, a similarity measure to find the best matches and sufficient amount of data for training.

Unfortunately there are some problems in realizing this solution:

- Usenet system does not have a service to download already posted messages in newsgroups. People can get such a database by saving messages one by one and spending great amount of time on this work. This is also the way we used to get our test database.
- Even after constructing the initial database users should periodicaly collect new posted messages for the sake of the maintanence of their software. Because in long term the database needs to be updated in order not to loose accuracy in its results.
- Unpreventable spamming in Usenet system is decreasing the accuracy of categorization algorithms by creating *noisy data*.
- Great amount of disk space for database is needed if it is necessary to cover all newsgroups in Usenet system.

1.3 Organization of the Thesis

In Chapter 2 we give background information and mention related works done up to now about *hierarchical clustering algorithms*, *text categorization algorithms*, and *projects dealing with Usenet news*. In Chapter 3 we present the pseudocodes of our algorithm, explain them in details in two parts called *learning* and *categorization*, simulate a run time example done by hand and finally give the details of a small experiment done with a database of 1000 documents and a query document. In this last example we also explain the issues of *stopword* elimination and *stemming* of words which we have dealt with in the implementation of CHSD. In Chapter 4 we do complexity analysis and empirical evaluation of our algorithm, present our data sets and test results, and compare results of our algorithm with the results of KNN algorithm which we ran over the same sample database. At the end of Chapter 4 we give details and results of a scalability test that we did with as many documents as we could collect. The aim of this scalability test is to show that CHSD is scalable in terms of time and space complexities whatever the size of the database is. In Chapter 5 we give our conclusions and determine the future works which are to be done later for more improved versions of CHSD.

Chapter 2

Previous Work

Popularity of text categorization is increasing by the growing interest and usage of text data available on World Wide Web. In fact explosion of on-line information created a great deal of demand for categorization concept to allow users to easily access this information. Compulsorily some clustering algorithms are modified or new algorithms are developed from scratch for categorization. The most succesful approach for organizing this huge information is to make it comprehesible by categorizing into multiple topics where topics are organized in a hierarchy with a downwards increasing specificity [7, 16].

In this chapter we give the background information of hierarchical clustering and categorization algorithms in general terms. We explain the most popular algorithms in the first two sections and in the final section we give the details of two different projects implemented on Usenet news.

2.1 Clustering Algorithms

Clustering is segmenting a collection of items into subsets that are called clusters. Each cluster is a collection of related items such that its members are more similar to each other than they are to members of any other cluster. A cluster may be contained in multiple clusters or it may contain multiple clusters in itself. Items are in the same cluster if they share some characteristics such as certain attributes, certain values for certain attributes, certain range

of values for some combination of attributes and some general predicates.

Clustering algorithms divide the set of objects into previously unknown clusters as shown in Figure 2.1 and the main principle may be summarized as follows : the most similar objects are put in the same cluster and the less similar ones in distinct clusters. PDDP (Principal Direction Divisive Partitioning) algorithm is one of the newest works in this area and a typical example of clustering [6].

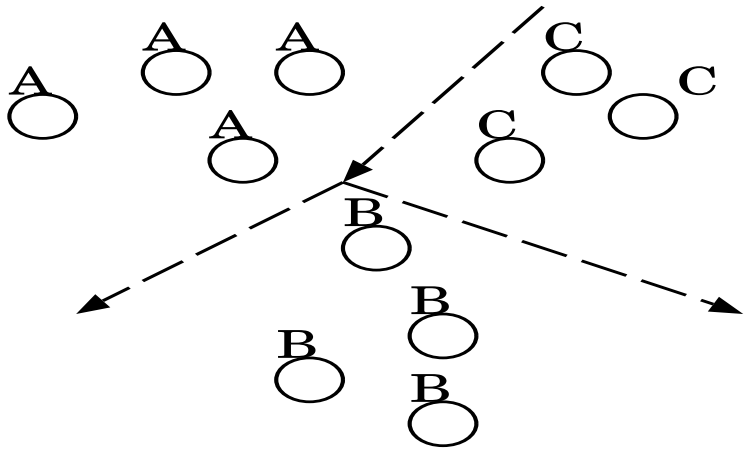


Figure 2.1: Clustering

A function of proximity is used to evaluate the likeness degree of pair of objects. The aim of a proximity measure is to evaluate to what extent a pair of objects are alike/unlike with respect to the available information about them.

Most of the existing algorithms for document clustering are based on either probabilistic methods or distance and similarity measures. Distance-based algorithms such as k-means analysis and hierarchical clustering use a selected set of words appearing in different documents as features. In these algorithms each document is represented by a feature vector, and can be viewed as a point in a multidimensional space.

There are a number of problems with clustering in a multi-dimensional space using traditional distance or probability-based algorithms [28, 31]. First, it is not trivial to define a distance measure in this space. Feature vectors must be scaled to avoid skewing the result by different document lengths or possibly by how common a word is across many documents. Second, the number of different words in the documents can be very large. Distance-based schemes

generally require the calculation of the mean of document clusters, which are often chosen initially at random.

Advantages of clustering:

- Provides a framework for managing locality.
- Allows performance tuning to different configurations and architectures by allowing the size of the clusters to be adjusted.
- Simplifies lock structuring issues, and hence reduces the code complexity which can lead to improved performance and scalability.
- Can differentiate heterogeneous collections.
- Creates an overview of collection and meaningful themes.
- Reflects emphases present in the collection of documents

Disadvantages of clustering:

- Many of the ways documents could be grouped together are not shown.
- Does not always give easy to understand results.
- Creates different levels of granularity.
- There is always variability in quality of results.
- Does not work well for differentiating homogenous collections.
- Results always require interpretation.

2.1.1 Hierarchical Clustering Algorithms

Hierarchical clustering is a kind of work that transforms a set of data points with a given measurement for similarity into a sequence of nested partitions [22]. In other words a hierarchical clustering algorithm yields a *dendogram* representing the nested partitions and similarity levels at which partitions change.

There are two basic approaches in hierarchical clustering:

- Agglomerative (starting with each data point as a single cluster, at each step merge two of them together)
- Divisive (starting with all data points in one cluster, at each step divide one cluster into two clusters)

Step 1: Create n clusters such that each cluster contains exactly one item

Step 2: Search the two clusters i and j that have most similarity

Step 3: Merge clusters i and j into a cluster (ij)

Step 4: Repeat Steps 2 and 3 until the number of clusters is equal to one

---

Figure 2.2: Agglomerative Hierarchical Clustering Algorithm

Due to its advantages and easy to implement speciality agglomerative hierarchical clustering algorithms are used frequently. In Figure 2.2 we give a sample algorithm that consists a mainframe for such an algorithm. For the realization of step 2 in this algorithm most of the hierarchical methods use distance measure between data points and there are four different approaches to determine the most similar cluster to the current cluster:

- *Single Link Method* takes the sum of *minimum* distances between the data points of clusters to calculate the distances between cluster and then chooses the cluster which has the minimum distance to the current cluster to be merged. This method is so versatile that it can even extract heavily concentric clusters but it is unsuitable for isolating spherical or poorly seperated clusters.
- *Complete Link Method* takes the sum of *maximum* distances between the data points of clusters to calculate the distances between clusters and then chooses the cluster which has the minimum distance to the current cluster to be merged. This method creates small, tightly bound and compact hierarchies in many applications.
- *Average Link Method* takes the sum of *averages* of the distances between the data points of clusters to calculate the distances between clusters and

then chooses the cluster which has the minimum distance to the current cluster to be merged. This method creates intermediate results between the methods above.

- *Ward's Method* is also known as the minimum variance method because it joins at each step the two clusters whose merger minimizes the increase in the total *within-group error sum of squares*. It tends to produce homogeneous clusters and a symmetric hierarchy [4].

Besides, hierarchical clustering algorithms have some disadvantages worth considering beforehand:

- Results can be influenced by extraneous data or noise in the data set.
- Since misclassification is probable final results need to be checked.
- Possibility of having the same minimum distances between different clusters enforces arbitrary choices.

2.1.2 k-Means Algorithm

Step 1: Choose k cluster centers to coincide with k randomly chosen patterns or k randomly defined points inside the hypervolume containing the pattern set

Step 2: Assign each pattern to the closest cluster center

Step 3: Recompute the cluster centers using the current cluster memberships

Step 4: If the convergence criterion is not met go to Step 2

Figure 2.3: k-Means Clustering Algorithm

k-Means algorithm is popular because it is easy to implement, and its time complexity is $O(n)$ where n is the number of patterns [21]. Sole disadvantage of k-Means Algorithm is its being so sensitive to the selection of the initial partition - first step in Figure 2.3 - and its possibilty to converge to a local

minimum of the criterion function value if the initial partition is not properly chosen.

Typical convergence criteria checked at last step of the algorithm are:

- No or minimal reassignment of patterns to new cluster centers.
- Minimal decrease in squared error.

2.2 Categorization Algorithms

We need to categorize everything in order to make sense of the world and simplify our perception, but once we have done so we respond to objects in terms of their class membership rather than their uniqueness. Nonetheless most of the objects have multiple memberships and this phenomenon stimulates the need for categorization. Actually what really needed is *classification* but categorization is a kind of classification task.

Document categorization is the automated assigning of natural language texts to predefined categories based on their content [9]. Most of the document categorizing systems regard documents as bags of words where each word is represented with its occurance number called *frequency* or zero if not present.

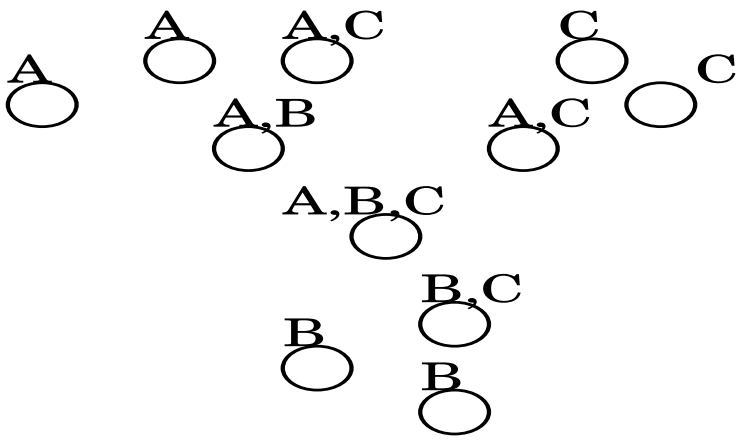


Figure 2.4: Categorization

Slight differences between Figures 2.1 and 2.4 explain how categorization differs from clustering. Categorization finds out label(s) for each object instead

of determining the partitions in the sample space. Therefore categorization gets rid of most of the disadvantages of clustering mentioned in Section 2.1 by creating easy to understand results and determining almost all possible labelings with highest possible accuracy.

Present theories of document categorization rely too much on similarity computation. Given a tree of category descriptors, there are two possible approaches to perform document categorization:

- *Document-centered categorization*: Given a document, find the most appropriate categories it belongs to. Complexity is proportional to the number of documents.
- *Category-centered categorization*: Given the category descriptors, search the database to find the documents that satisfy best each category descriptor. Complexity is proportional to the number of categories which is typically much smaller than the number of documents to categorize.

2.2.1 k Nearest Neighbor(KNN) Algorithm

KNN algorithm is based on the assumption that the nearest neighbor of an unclassified instance in the training dataset should belong to the same class as that instance [11] similar to work done in Figure 2.4. KNN algorithm assigns each unclassified instance to the category of its nearest neighbor instance if the distance to that labeled neighbor is below a threshold and the process continues until all instances are labelled or no additional labelings occur [35, 2, 1].

More technically KNN algorithm is an instance-based learning algorithm which performs training by simply storing the instances in the memory [5, 27]. Each training instance is represented as a set of feature-value pairs. Predictor feature values may be of categorical (nominal) or linear (ordered) type, whereas target feature values are of only linear type [26].

The querying phase of the KNN algorithm tries to predict the target feature value of a query instance as a function of most similar instances' target feature values. The k value is selected as the number of nearest (most similar) neighbors that will be taken into account in the querying phase.

Training:
[1] $\forall \mathbf{x}_t \in$ Training Set Store $\mathbf{x}_t$ in memory

Querying:
[1] $\forall \mathbf{x}_q \in$ Query Set

[2] $\forall \mathbf{x}_t \{\mathbf{x}_t \neq \mathbf{x}_q\}$: Calculate $Similarity(\mathbf{x}_q, \mathbf{x}_t)$

[3] Let $Similars$ be set of k most similar instances to $\mathbf{x}_q$ in Training Set

[4] Let $Sum = \sum_{\mathbf{x}_t \in Similars} Similarity(\mathbf{x}_q, \mathbf{x}_t)$

[5] Then $\bar{y}_q = \sum_{\mathbf{x}_t \in Similars} \frac{Similarity(\mathbf{x}_q, \mathbf{x}_t)}{Sum} \mathbf{y}_t$

Figure 2.5: The k Nearest Neighbor Approach

There is a variety of approaches to KNN algorithm in the literature. The algorithm, shown in Figure 2.5, is the simplest approach. For a given query instance, k nearest (similar) training instances are determined by using the $Similarity$ function given in equation 1.

The similarity between the query instance x_q and a training instance x_t is determined as:

$$Similarity(\mathbf{x}_q, \mathbf{x}_t) = \sqrt{\sum_{i=1}^{p} Sim(x_{qi}, x_{ti})} \quad (1)$$

where $Sim(x_{qi}, x_{ti}) = (\frac{x_{qi} - x_{ti}}{range\ (i)})^2$ and i is the feature dimension. Finally, the weighted sum of the target values of the k nearest neighbors of x_q is used as the predicted target value, $\bar{y}_q$, of the query instance x_q [26].

KNN algorithm assumes that all the predictor features are equally relevant. This assumption makes it less effective when the database contains irrelevant, weakly relevant and noisy features. However, the prediction accuracy of the model can be improved if the predictor features are assigned proper weights to denote their relevancy in the prediction process [15]. These weight values can be either obtained from database experts or automatically determined by some feature weight learning algorithms [32, 29]. In terms of interpretability, KNN algorithm is very poor, since it is a lazy approach. It does not induce models that enable interpretation of the underlying data set [36].

2.2.2 Rainbow Algorithm

Rainbow is a Naive Bayes classifier for text classification tasks [36], developed by Andrew McCallum at CMU[1]. It estimates the probability that a document is a member of a certain category using the probabilities of words occurring in documents of that category independent of their context.

Rainbow makes use of the naive independence assumption. More precisely, the probability of document d belonging to class C is estimated by multiplying the prior probability $Pr(C)$ of category C with the product of the probabilities $Pr(w_i \mid C)$ that the word w_i occurs in documents of this class and then this product is normalized by the product of the prior probabilities $Pr(w_i)$ of all words as shown in equation 2 below.

$$Pr(C \mid d) = Pr(C) \prod_{i=1}^{n} \frac{Pr(w_i \mid C)}{Pr(w_i)} \tag{2}$$

As many of the probabilities $Pr(w_i \mid C)$ are typically 0:0 (hence their product will be 0:0), Rainbow smoothes the estimates using the technique proposed in [14]. A more detailed description of this smoothing technique and of Rainbow in general can be found in [24].

2.2.3 Ripper Algorithm

Ripper[2] [37] is an efficient, *noise-tolerant rule learning* algorithm based on the incremental reduced error pruning algorithm [19, 17]. It learns single rules by greedily adding one condition at a time (using Foil's information gain heuristic [30]) until the rule no longer makes incorrect predictions on the growing set, a randomly chosen subset of the training set. Thereafter, the learned rule is simplified by deleting conditions as long as the performance of the rule does not decrease on the remaining set of examples (the pruning set). All examples covered by the resulting rule are then removed from the training set and a new rule is learned in the same way until all examples are covered by at least one rule. Thus, Ripper is a member of the family of *separate-and-conquer rule*

[1] Available from http://www.cs.cmu.edu/afs/cs/ project/theo11/www/naivebayes.html.
[2] Available from http://www.research.att.com/ wcohen/ripperd.html.

learning algorithms [18].

What makes Ripper particularly well-suited for text categorization problems is its ability to use set-valued features [38]. For conventional machine learning algorithms, a document is typically represented as a set of binary features, each encoding the presence or absence of a particular word in that document. This results in a very inefficient encoding of the training examples because much space is wasted for specifying the absence of words in a document. Ripper allows to represent a document as a single set-valued feature that simply lists all the words occurring in the text. Conceptually, Ripper's use of such a set-valued feature is no different than the use of binary features in conventional learning algorithms, although it makes use of some optimizations.

2.2.4 Fuzzy Algorithm

Step 1: Select an initial fuzzy partition of N objects into K categories by selecting the NxK membership matrix U such that an element u_{ij} of U represents the grade of membership $\mathbf{x_i}$ in category $\mathbf{c_j}$ (Typically $u_{ij} \in [0,1]$)

Step 2: Using U, find the value of a fuzzy criterion function and then reassign patterns to categories to reduce this criterion function value and recompute U

Step 3: Repeat Step 2 until entries in U do not change significantly

---

Figure 2.6: Partitional Fuzzy Categorization Algorithm

Traditional categorization algorithms tends to partition data items such that each item belongs to only one category. This is a kind of *hard categorization*. On the other hand fuzzy algorithms extend this notion to associate each pattern with every category using a membership function. So that each category turns out to be a fuzzy set of all patterns. In Figure 2.6 the outline of partitional fuzzy categorization algorithm is given [21].

2.3 Projects Dealing with Usenet News

Many researchers have worked for Usenet news categorization or they have taken these news as the input of their experiments just as we did in this thesis. Most of the projects have been done in filtering tasks such that users submit their interest profiles and then receive the news in accordance with their profiles [23, 34]. However there are new type of projects such as Websom [33] which uses a *self-organizing map* to automatically group similar documents into a two-dimensional space. In such projects clustering and categorization concepts are prefered according to the purpose of the project. In this section we mention two projects called as *Newsgroup Clustering Based on User Behavior* and *Usenet News Categorization* in detail.

2.3.1 Newsgroup Clustering Based on User Behavior

Jussi Karlgren created an algorithm called *Newsgroup Clustering Based on User Behavior* [20] in 1994 which aims the retrieval of useful information from the Usenet news domain that a user really needs. In this algorithm the distance measure is based on knowledge about the user or knowledge about the use of the document rather than knowledge about the content or genre of the document. This knowledge is extracted from user models that indicate the preferences of users. The user model contains a vector of user grades. The documents in the document base are graded by a user to be good (+), bad (–), or not accessed (0) such that :

- User is interested in a document if the grade of that document is (+).
- User is uninterested in a document if the grade of that document is (–).
- User does not know the document if the grade of that document is (0).

These grades are quantified by an algebra similar to Table 2.1 and then proximities between each document pair are calculated by formula 3 and a proximity-matrix is obtained after this process.

$$proximity(doc_A, doc_B) = \sum_i (interest(reader_i, doc_A) \bigotimes interest(reader_i, doc_B)) \quad (3)$$

where $\otimes$ stands for the quantitive recommendation algebra given in Table 2.1.

$\otimes$	+	-	0
+	1	0	0
-	0	0	0
0	0	0	0

Table 2.1: Quantitative Recommendation Algebra

The base hypothesis of this algorithm is : If a user A is interested in documents K and L, and another user X is interested in K, it is likely that X will also like L. In the light of this hypothesis documents are categorized by runing an *average-link agglomerative* method over the matrix of proximities.

2.3.2 Usenet News Categorization

A different project was done by a team from Johns Hopkins University in 1996 [39]. Their approach was to create a model for each newsgroup and to compare new documents with each model to find the best match.

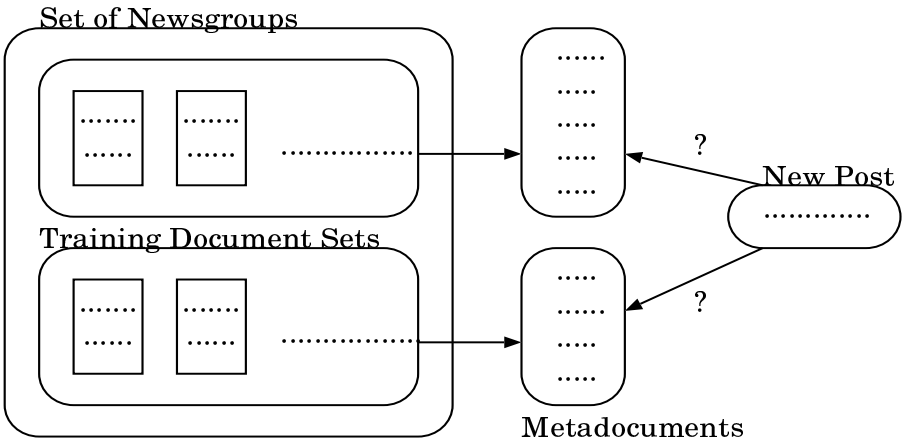


Figure 2.7: Usenet News Categorization System

Models are created from a collected set of documents from each newsgroup

as shown in Figure 2.7. Thus a model is equivalent to a metadocument in this project. Three different methods are proposed for the metadocument creation phase:

- *Full-text Concatenation*: For each newsgroup form a single document which is concatenation of all documents in itself - creates a great metadocument.
- *Document Selection*: Identify the documents that are most useful in discriminating topics and concatenate them into a single document - creates a smaller metadocument.
- *Discriminating Term Selection*: Pick out the terms that exemplify the concept of a particular newsgroup and concatenate them into a single document - creates fairly small metadocument.

The angle between two vectors has been exploited as an effective measure of content similarity, and many systems use the *cosine similarity* measure to compute the similarity among document and profile representations [10]. The cosine similarity between two vectors, v_1 *and* v_2 is based on the inner (dot) product of v_1 *and* v_2, and can be formulated as:

$$cosine(v_1, v_2) = \frac{\sum_t w_{t,v_1}.w_{t,v_2}}{\sqrt{\sum_t w_{t,v_1}^2}.\sqrt{\sum_t w_{t,v_2}^2}} \tag{4}$$

where $w_{t,x}$ stands for the frequency of word t in vector x, and the result is the similarity between vectors v_1 *and* v_2 such that 1 for identical vectors and 0 for the vector with no common terms.

After metadocuments are constructed *cosine similarity* is used to find the best match for new documents and a comparison is made between each newsgroup and each query document. This approach accepts a great deal of redundant time complexity because unnecessary comparisons are highly probable. Actually the main aim in this project is to determine the most appropriate topic labels for a given document to be posted to Usenet newsgroups. Thus accuracy is prefered to speed in the query phase.

Chapter 3

CHSD Algorithm

CHSD algorithm is similar to the other *categorization learning* algorithms. First of all similar to others it executes a learning phase which takes more time than the others, but after such a heavy work it achieves the capability of doing faster categorization. Because an index tree is constructed to the cost of many data replications in the learning phase but this makes it easier and faster to categorize new documents.

CHSD operates on a sample space of m categories in which each category consists of n documents. Each category is represented by a three-row vector in which rows contain *words*, *frequencies* and *norm-scaling values* respectively. Beginning with m vectors an **agglomerative, bottom-up hierarchical** approach is applied that ends up with a single node at root and an index tree with nodes that have different numbers of children.

CHSD is **supervised** since labels of categories are predefined, **overlapping** since a document is allowed to be present in more than one category, **polythetic** since all features of database are used simultaneously, **document centered** since it takes in a query document as input and finds out its related categories one by one.

In this chapter all details of CHSD are explained exclusively by pseudocodes. There are 7 interrelated functions in the algorithm given in separate sections named as *learning* and *categorization* as the phases of a typical categorization task. Finally in Section 3.3 a small run time simulation and the

results of a categorized news are given and discussed with other certain issues.

3.1 Learning

At learning phase CHSD begins with raw data and takes in a database which consists of multiple groups such that each group has multiple documents and a group name which is a concatination of multiple names implying the hierarchy of database as shown in Figure 3.1. For instance X.Y.Z means this group is a child of X.Y and X.Y is a child of X.

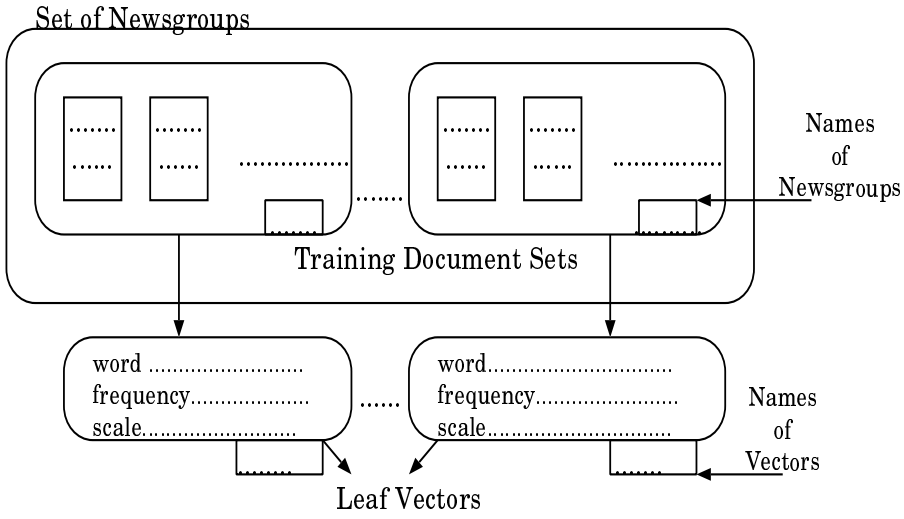


Figure 3.1: Initial Database System of CHSD

First of all InitTree function which is given in Figure 3.2 takes database as input and passes it to ProcessData(Figure 3.3).

ProcessData deals with each word of each document in each group in lines 5-17. In lines 10-14 frequency table is filled up and in lines 15-17 denominator of the norm-scaling formula (equation 1) is calculated for each word. Hash function mentioned in line 8 could be any hash function which generates a definitely distinct number for each different word in the database.

While dealing with words only the words that are not member of a pre-defined stoplist are counted. Making words not case sensitive by turning all

```
InitTree (DB) /* DB: database of newsgroups */
[1]  GroupNo ← ProcessData(DB)
[2]  Create RootNode[GroupNo]
[3]  for each group_i ∈ DB do
[4]      Create NewNode /* a node with a name and a words array */
[5]      NewNode.name ← group_i.name
[6]      for each word_k ∈ group_i do
[7]          NewNode.words[k].name ← word_k.name
[8]          NewNode.words[k].frequency ← word_k.frequency
[9]          NewNode.words[k].scale ← word_k.scale
[10]     RootNode[i] ← NewNode
[11]BuildTree(RootNode, 1)
```

Figure 3.2: InitTree Function

capitals into small letters and getting rid of punctuations before stoplist memberance check will be helpful. In addition, as explained in Section 3.3 stemming the words that pass this check increases the accuracy of results since it prevents missing most of the words that are similar to each other in the meaning.

After all documents are processed in the current group, norm-scaling values are calculated and a vector file is written out for each group which contains names, frequencies and norm-scaling values of words (line 21). The *if check* in line 19 prevents zero values to be written out thereby CHSD deals with only non-zero values. After all groups are processed ProcessData returns the number of groups in the database.

We scale the word frequencies by norm-scaling method [6] which is given by equation 1. By this scaling unimportant words for similarity calculation

gain lower values while important words gain higher values. For instance a widespread word which occurs once or more in each document of database gains a scaling value which is greater than one, while another word which occurs only once in a few documents gains one or less than one as a scaling value.

Norm-scaling method can be formulated as:

$$d_i = \frac{TF_i}{\sqrt{\sum_j TF_j^2}} \tag{1}$$

where d_i stands for the relative frequency of word i, TF_i stands for the total frequency of word i, and TF_j stands for the frequency of word i in particular document j.

In addition, there is another scaling method called as TFIDF [12]. But this scaling results in a non-zero value for each word and does not help to produce distinctly better results. It also needs a complicated coding and more time to be applied.

In line 2 of InitTree a root node is created which has enough number of pointers for leaf nodes. For each newsgroup in the database a new node is created such that each node keeps the name of the group and a word vector to include leaf vector of the current group. We can read these vector files written out by ProcessData and fill in the current new node's word vector as shown in lines 6-9. Initialization is done after all nodes get filled up and joined to the root.

At the next step, InitTree calls BuildTree (Figure 3.4) by passing the root and number 1 to it. Number 1 stands for the first parts of the hierarchical names mentioned above. Thus BuildTree begins to construct the index tree by merging the leaf nodes with similar first names at its first recursion. That is, new nodes for top-level names in Usenet system are created such as *comp*, *bionet* and their children are joined to them.

BuildTree takes in a node and a key value as input. It detects the groups which have similar first key many name parts. In other words siblings are found first and a new node is created for each sibling group. Each new node takes concatination of those key many names as its name (line 4). Then BuildTree

fills in the words vector of new node with the words of its children as shown in lines 7 through 14 of Figure 3.4.

One of the most crucial points in BuildTree function is to sieve the words of leaf nodes. In our experiments we determined $\sqrt{2}$ as the **threshold of norm-scaling value** for a word to be copied up to the nodes over the leaf level. That is, the words with norm-scaling values less than $\sqrt{2}$ will be present only in leaf nodes. The *if check* in line 9 of BuildTree function does this work. Purpose of such an elimination is to get rid of ignorable words and to lessen the number of words in the vectors of nodes that are higher than the leaf level. So that categorization phase becomes faster thereby supporting our main goal in this thesis in regards to time complexity. In addition, we believe that this method prevents the negative effect of noisy data to some extent.

As for the space complexity, it is obvious that this method saves a lot of memory. In addition, threshold of norm-scaling value could be bigger than $\sqrt{2}$ in accordance with the size of the database dealt with.

Recursion of BuildTree stops when there are no siblings to be merged by a new parent node. That is, all hierarchies of the database are constructed and the index tree is ready for categorization phase. In other words, learning is done sufficiently and control may return to main function.

```
ProcessData (DB)

[1]  StopList /* list of words to be ignored */
[2]  SubDictionary /* list of words in current group under DB */
[3]  Frequency /* keeps frequency of each word in SubDictionary */
[4]  Scale /* keeps normal scale value of each word in SubDictionary */

[5]  for each group_i ∈ DB do

[6]      for each document_j ∈ group_i do

[7]          for each word_k ∈ document_j do

[8]              hashValue ← Hash(word_k)
[9]              /* Hash function returns a bucket number for current word */

[10]             if word_k ∉ StopList and word_k ∉ SubDictionary then
[11]               SubDictionary[hashValue] ← word_k
[12]               Frequency[hashValue] ← Frequency[hashValue]+1

[13]             if word_k ∉ StopList and word_k ∈ SubDictionary then
[14]               Frequency[hashValue] ← Frequency[hashValue]+1

[15]         for each word_k ∈ SubDictionary do

[16]             if Frequency[k] ≠ 0 then
[17]               Scale[k] ← Scale[k] + Frequency[k] * Frequency[k]

[18]     for each word_k ∈ SubDictionary do

[19]         if Frequency[k] ≠ 0 then
[20]           Scale[k] ← Frequency[k] / sqrt(Scale[k])

[21]           WriteToFile(SubDictionary[k], Frequency[k], Scale[k])

[22]           SubDictionary[k] ← NULL /* reset arrays */

[23]           Frequency[k] ← 0

[24]           Scale[k] ← 0

[25] return i
```

Figure 3.3: ProcessData Function

```
BuildTree (Node, key)
[1] for each Group_i in the children of Node
[2] such that first key many parts of their names are similar do
[3]     Create NewNode /* a node with a name, a words array, child pointers */
[4]     NewNode.name ← (concatenation of key many similar parts detected)
[5]     wordCounter ← 0
[6]     childCounter ← 0
[7]     for each node_j ∈ Group_i do
[8]         for each word_k ∈ node_j do
[9]             if node_j.words[k].scale ≥ √2 then /* eliminate ignorable words */
[10]                NewNode.words[wordCounter].name ← node_j.words[k].name
[11]                NewNode.words[wordCounter].frequency ← node_j.words[k].frequency
[12]                NewNode.words[wordCounter].scale ← node_j.words[k].scale
[13]                wordCounter ← wordCounter + 1
[14]        NewNode[childCounter] ← node_j
[15]        childCounter ← childCounter + 1
[16]Node[i] ← NewNode
[17]BuildTree(NewNode, key+1) /* go on recursively */
```

Figure 3.4: BuildTree Function

3.2 Categorization

After such a learning phase categorization becomes the fastest and easiest phase of CHSD. Because all we need for the categorization of a new document is to produce its word vector and make it travel down the tree from root to a leaf or to multiple leaves and report final name(s) of the leaf/leaves as the result.

FindCategories (document, threshold)

[1] StopList /* list of words to be ignored */

[2] FoundCategories /* keeps names of categories found */

[3] Create NewNode /* a node with a words array only */

[4] for each $word_k \in$ document do

[5] if $word_k \notin$ StopList and $word_k \notin$ NewNode.words then

[6] NewNode.words[k] $\leftarrow word_k$

[7] if $word_k \notin$ StopList and $word_k \in$ NewNode.words then

[8] NewNode.words[k].frequency $\leftarrow$ NewNode.words[k].frequency+1

[9] HierarchicalSearch(NewNode, RootNode, threshold, FoundCategories)

[10] return FoundCategories

Figure 3.5: FindCategories Function

FindCategories takes a text document and a **threshold value of similarity** as input (Figure 3.5). First of all it creates a new node for this document and fills in the words vector of this new node from the document as shown in the lines 3-8. At this step it is a must to process data with the same functions used in ProcessData (Figure 3.3) for consistency. Otherwise accuracy decreases too much. After the new node gets filled up it is passed to HierarchicalSearch (Figure 3.6) in line 9 to make it travel down the tree and to get

its FoundCategories array filled up with the names of the leaf nodes visited.

HierarchicalSearch is a recursive function which takes in a node, a threshold value and an empty array as input as shown in Figure 3.6. Beginning by the root's children it calculates the similarity value between the new node and the nodes of index tree by our Similarity function (Figure 3.7) and it gives way to recursion through the nodes which has similarity to new node higher than the threshold value of similarity.

HierarchicalSearch (NewNode, Node, threshold, FoundCategories)

[1] i $\leftarrow$ 0

[2] while Node.child$_i$ $\neq$ NULL do

[3] sim $\leftarrow$ Similarity(NewNode.words, Node.child$_i$.words)

[4] if sim $\geq$ threshold then

[5] if Node.child$_i$.child = NULL then /* if NewNode met a leaf node */

[6] Insert(FoundCategories, Node.child$_i$.name, sim)

[7] else

[8] HierarchicalSearch (NewNode, Node.child$_i$, threshold, FoundCategories)

[9] i $\leftarrow$ i+1

Figure 3.6: HierarchicalSearch Function

Actually the threshold value of similarity is an option of user which floats between 0 - 1. It acts as a measure which determines the sensitivity of the algorithm. If we use **0** as a threshold value then the new node will visit all other nodes in the index tree and we will get the names of all leaf nodes as a result of our categorization request and because of this the query will take much time. If we use **1** as a threshold then we will most probably get no result of our categorization request and the query will take very short time. So there

is a trade-off between time complexity and accuracy in here and the threshold value of similarity is the key to solve this trade-off. We are certain that the best threshold value can be found by trying out the algorithm on different types and sizes of databases with certain number of different threshold values, as we did in our evaluation tests of CHSD.

Similarity (guest, host)

[1] number $\leftarrow$ 0 /* upper part of formula */

[2] divisor1 $\leftarrow$ 0 /* first part of divisor */

[3] divisor2 $\leftarrow$ 0 /* second part of divisor */

[4] for each word$_k$ $\in$ guest do

[5] for each word$_m$ $\in$ host do

[6] if word$_k$ = word$_m$ then

[7] number $\leftarrow$ number + word$_k$.frequency * word$_m$.frequency

[8] divisor1 $\leftarrow$ divisor1 + word$_m$.frequency * word$_m$.frequency

[9] divisor2 $\leftarrow$ divisor2 + word$_k$.frequency * word$_k$.frequency

[10] return (number / ($\sqrt{divisor1}$ * $\sqrt{divisor2}$))

Figure 3.7: Similarity Function

Similarity function given in Figure 3.7 takes in two word vectors - containing words and their frequencies - as input and calculates similarity between them according to the equation 2 :

$$Sim(v_1, v_2) = \frac{\sum_t w_{t,v_1}.w_{t,v_2}}{\sqrt{\sum_t w_{t,v_1}^2}.\sqrt{\sum_t w_{t,v_2}^2}} \quad (2)$$

where $w_{t,x}$ stands for the frequency of word t in vector x, and the result is the similarity between vectors v_1 *and* v_2 such that 1 for identical vectors and 0 for the vectors with no common terms. We only deal with the common words of

both vectors in this formula. In addition, binary search technique can be used instead of *for loop* in line 5 to make Similarity function faster.

Finally in line 6 HierarchicalSearch fills in the FoundCategories array with the names of **visited leaf nodes** by the help of Insert function which is given in Figure 3.8.

```
Insert (FoundCategories, category, sim)

[1]  i ← 0

[2]  while FoundCategories[i].sim ≥ sim do
[3]       i ← i+1 /* find the right place in sorted order */

[3]  temp ← FoundCategories[i]

[4]  FoundCategories[i] ← category

[5]  i ← i+1
[6]  j ← i

[7]  while FoundCategories[j] do
[8]       j ← j+1 /* go to the end of FoundCategories */

[9]  j ← j+1

[10] while j ≥ i do /* one right shift until i'th element */
[11]      FoundCategories[j] ← FoundCategories[j - 1]
[12]      j ← j - 1

[13] FoundCategories[j] ← temp
```

Figure 3.8: Insert Function

Insert function takes in an array of catagory names, a new category name and a similarity value belonging to that new category name as input as shown in Figure 3.8. It inserts the new name according to its similarity value in the array, so that the input array is kept sorted by similarity values in non-increasing order. Actually this work is done in a similar way to Insertion Sort

technique. So the first category name in FoundCategories array is the best match for the new document.

After all recursions are popped up in HierarchicalSearch, control returns to FindCategories and FoundCategories array becomes filled up with the category names determined by CHSD and then it is returned as the result of the query (line 10).

3.3 Examples

In this section we present a simulated categorization experiment such that we have three groups/categories in our database and each group has three documents as shown in Figure 3.9. Next we give a real document categorization example in a sample database of 10 newsgroups - 100 news per each - and discuss about stoplist and stemming issues on this example.

3.3.1 Learning

Assume that we have 3 newsgroups named as X.Y.Z, X.K, L.M such that each newsgroup contains 3 documents with a few words. Our first job is to process our sample database by the function ProcessData which is given in Figure 3.3. After processing data we get a representative vector for each group that consists of name, frequency and norm-scaling value of each word in the group as shown in Figure 3.9.

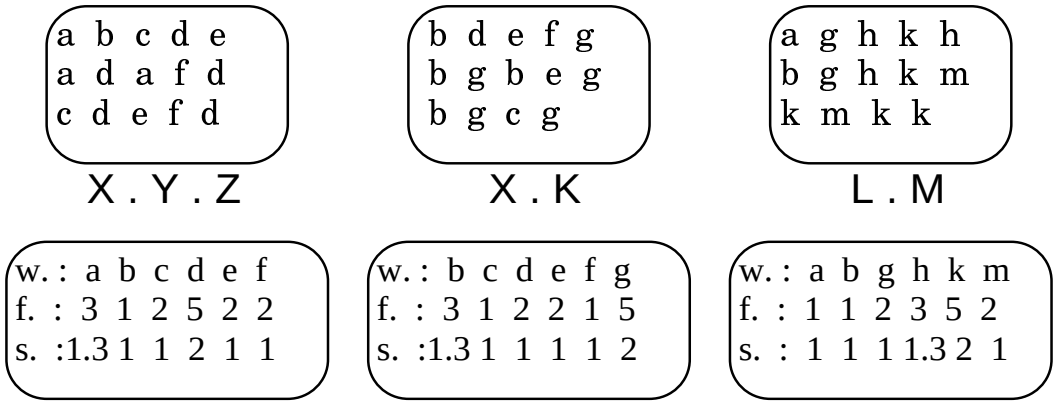
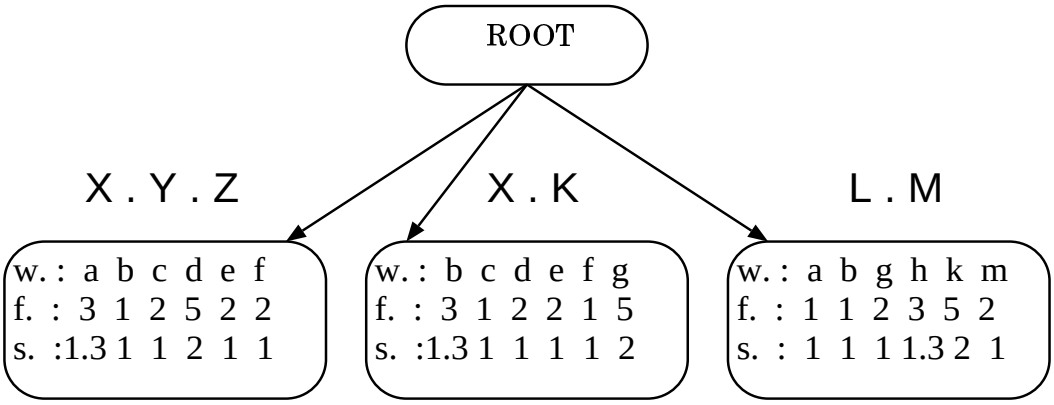


Figure 3.9: Sample Database and Produced Vectors

The next step is initiating the tree by creating a dummy root and connecting those representative vectors as leaves to the root as we did in Figure 3.10. InitTree function which is given in Figure 3.2 does this step.

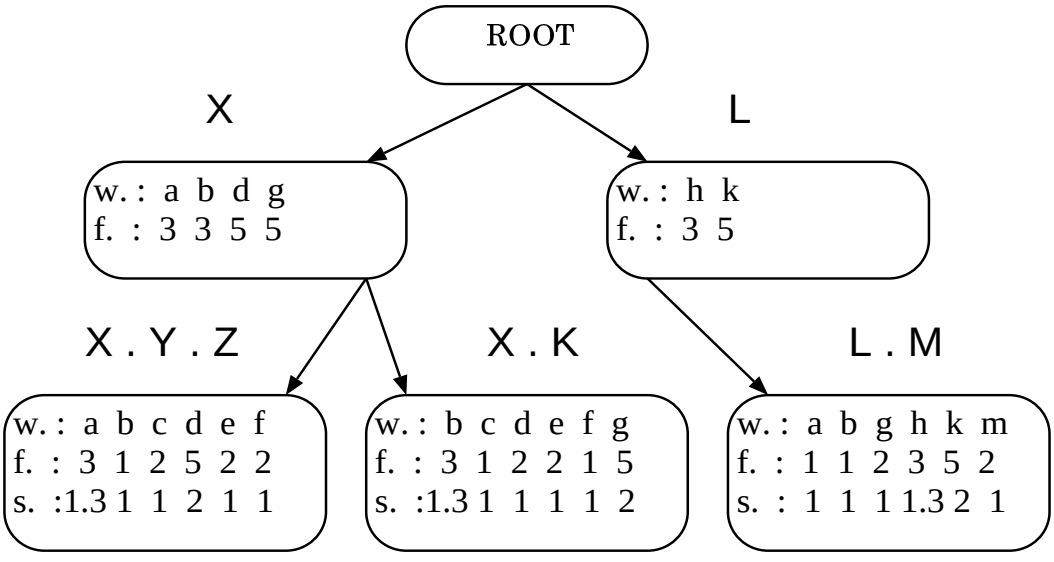
The initial tree is the starting point of BuildTree function (Figure 3.4) and the result of its first step is given in Figure 3.11. A couple of new nodes are



(w. : name of word ; f. : frequency of word ; s. : normal scaling value)

Figure 3.10: Work Done by InitTree

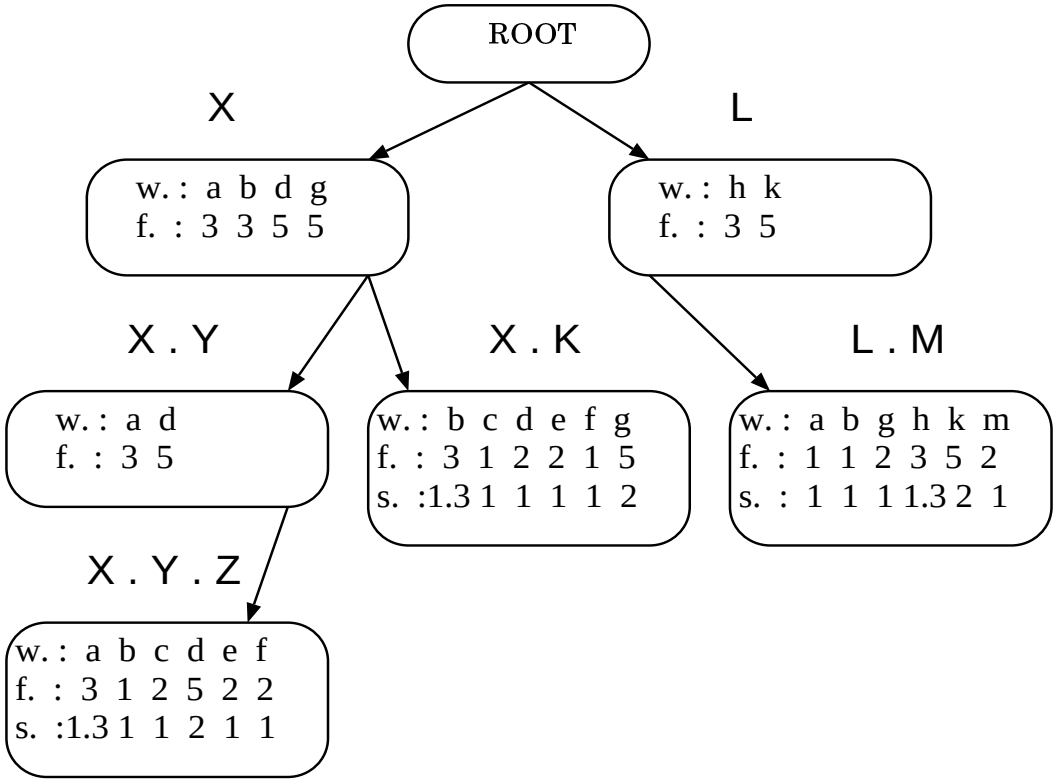
created and named as X and L respectively. Leaf nodes X.Y.Z and X.K became children of X and L.M became a child of L. At this step the crucial point is the elimination of words of representative vectors which have norm-scaling value as less than 1.2. So that vectors of nodes other than leaf nodes gain less number of words which are actually the most important ones in the current group. For instance words **h** and **k** are copied to new node L while words **a, b, g, m** stay intact in node L.M.



(w. : name of word ; f. : frequency of word ; s. : normal scaling value)

Figure 3.11: First Step of BuildTree

The last step of BuildTree is given in Figure 3.12 which shows a tree of height 3 because the longest hierachy is 3 in the database which is represented by the group X.Y.Z. To determine the end of the recursion of BuildTree, CHSD exploits the structure in group names such that if the difference between the numbers of parts in the names of a parent and a child is bigger than one then recursion goes on, otherwise it stops. Since there is a missing level between X and X.Y.Z a new node is created and named as X.Y. By this last step learning phase ends up with a well built index tree consisting of three leaf nodes, three inner nodes and a root.



(w. : name of word ; f. : frequency of word ; s. : normal scaling value)

Figure 3.12: Last Step of BuildTree

3.3.2 Categorization

For the categorization phase let us assume that we have three different documents, and the index tree shown in Figure 3.12. As a threshold value of similarity let us take 0.5 in all of the following experiments. In the following

figures we represent input documents as vectors such that the first row contains the words and the second row contains frequencies of the words. Actually these vectors are the results of first step in FindCategories function (lines 3-8).

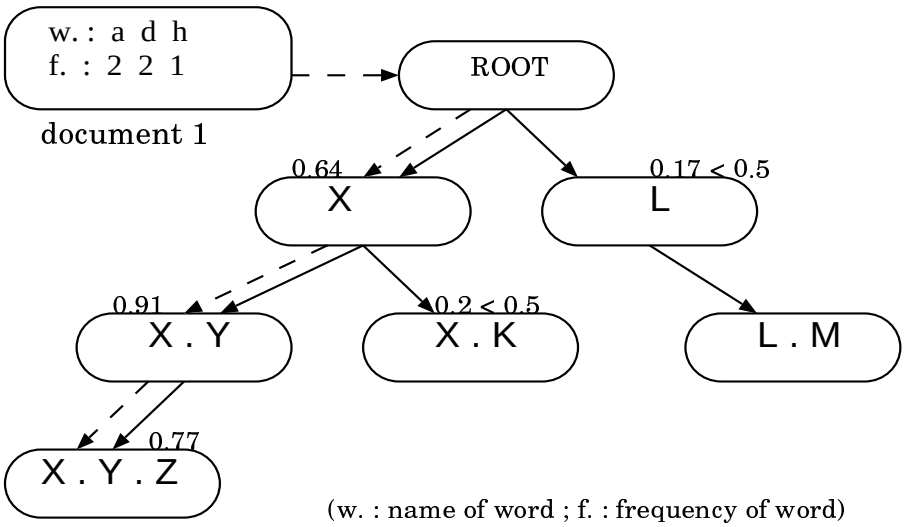


Figure 3.13: Categorization of Document 1

Beginning by the root, first document follows the path of X $\rightarrow$ X.Y $\rightarrow$ X.Y.Z and it is determined by 77% similarity that this documnet belongs to group X.Y.Z as shown in Figure 3.13. This document does not deviate from this path because children of the nodes with similarity less than the threshold value are not visited according to our algorithm. Therefore node L with 0.17 similarity and node X.K with 0.2 similarity are not visited.

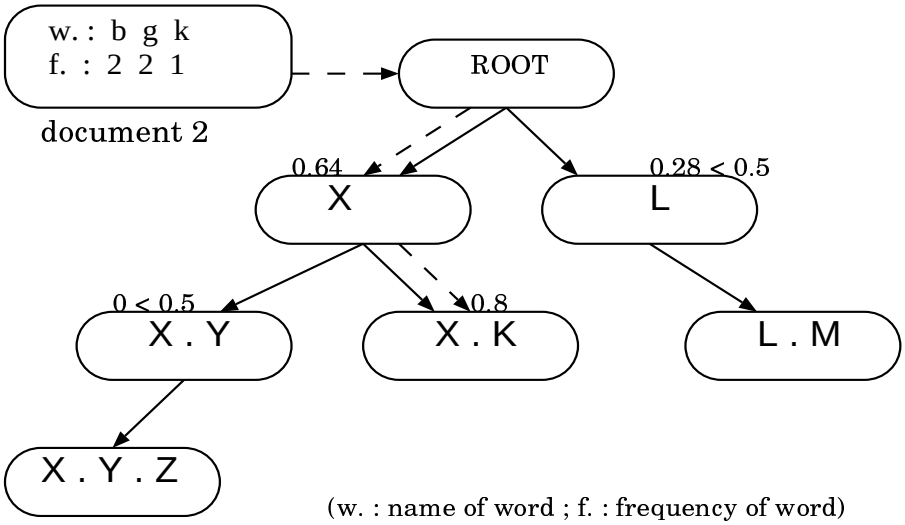


Figure 3.14: Categorization of Document 2

Second document follows the path of X → X.K and it is determined by 80% similarity that this documnet belongs to group X.K as shown in Figure 3.14. If we had taken a threshold value of less than 0.28 this document would have also gone through the path L → L.M. thereby determining that this document not only belongs to group X.K but also to group L.M. Most probably this would not be a right categorization. Thus the importance and the functionality of threshold value is very high in our algorithm.

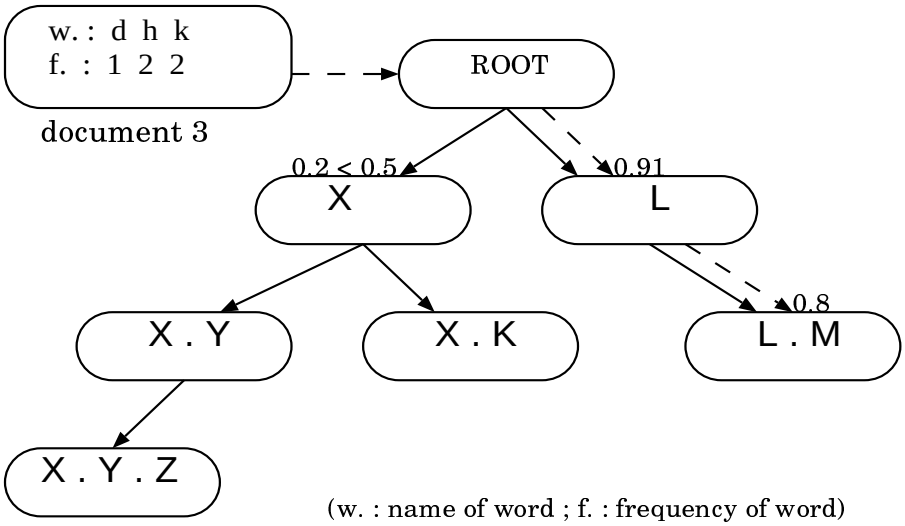


Figure 3.15: Categorization of Document 3

Third document follows the path of L → L.M and it is determined by 80% similarity that this document belongs to group L.M as shown in Figure 3.15. Since similarity of the node X to the third document is 0.2 which is less than 0.5 all of the left subtree is totally ignored as a correct decision. One of the best sides of CHSD is that it prevents doing unnecessary calculations.

3.3.3 A real document categorization

To give a more specific example and explain certain other details about processing raw data we collected 100 news documents for each newsgroup shown in Table 3.1. After learning phase the query document given in Figure 3.16 is categorized and the results are discussed in this section. The query document is a news which we do not know the newsgroups to post it.

It has been recognized since the earliest days of information retrieval that

Name of Newsgroup	Documents	Words	Similarity
comp.ai	100	2184	0.4021
comp.ai.alife	100	1376	0.5494
comp.ai.edu	100	2098	0.5737
comp.ai.fuzzy	100	1276	0.4823
comp.ai.genetic	100	1250	0.5474
comp.ai.nat-lang	100	2132	0.5503
comp.ai.neural-nets	100	937	0.5692
comp.ai.philosophy	100	2119	0.4879
comp.compilers.tools.javacc	100	758	**0.6431**
comp.compilers.tools.pccts	100	639	0.4982
Total Number of News	1000	-	-

Table 3.1: Sample Database

many of the most frequently occuring words in English are worthless as index terms such as *the, of, and, to, for* etc. These words make up a large fraction of the text of the documents. Eliminating such words from consideration early in the automatic indexing speeds up processing, saves huge amount of space in indexes and does not damage retrieval effectiveness [4]. Conventionally the group of these words is called *stoplist* or *negative dictionary.*

As for the categorization problems, determining the stoplist content is a crucial step that must be taken beforehand. This step requires a highly cautious examination of the database. Because each type of data has certain features which are required to be eliminated and some others which are required to be kept intact. In our database of Usenet news words such as *from, newsgroups, subject, date, message-id, hello, help* occur in all documents. Thus we expanded a basic stoplist with these words and filtered all of them while constructing a leaf vector for each group. For instance in the query document given in Figure 3.16 words such as *the, is, from, date, subject, newsgroups, hello, everybody, I, yours, into* are eliminated by an efficient stoplist memberance-check algorithm. We also eliminated e-mail and web site addresses in news documents.

In general terms *stemming* is the work of relating morphologically similar words to improve effectiveness and reduce the size of indexing files. Since a single stem typically corresponds to several full terms, by storing stems instead of terms, compression factor of over 50% can be achieved [8]. The sole disadvantage of stemming is that information about the full terms will be lost if

---

From: fkutlu@bilkent.edu.tr
Date: Fri, 03 Mar 2000 21:05:14 -0500
Subject: Calculate numbers in *java*
Newsgroups: ?

Hello Everybody,
I like sincerity, suggestive approaches and *java*.
I am a working for a commercial company.
Nowadays I am dealing with *java packages*.
I have built a new *multitask application form* where you can choose things that you prefer to buy among different *products* from a catalog.
I used an *applicable* and satisfactory *javac compiler*.
Actually it is a *package* of certain *number* of *applets* added into our *web-site* that are runing at background pretty well.
In future we want to make our system *capable* of calculating the prices in *java* and return it simultaneously.
What I need is a *javascript* where you can *convert* strings into *numbers* (Integers), and then by the *send command* (*Send button* in a *form*) the *script* calculates the choosen *products* (Prices, Integers) to a *total*.
Product id *numbers*, prices and results are integer.
It should be a *javascript* that directly converts text to integer and another *javascript* that calculates in a new *HTML* or *ASP* file a *total* of the *products*. Even some support of not so highly *pseudocode* suggestions might be a satisfaction for me.
You can find a detailed description and gui of this project at:
http://www.cs.bilkent.edu.tr/~fkutlu
It is also possible to download a *freeware* beta *version* of it.
I hope to get some help in time. Sincerely yours...

Figure 3.16: Query Document

stemming is done more heavily than required.

We used *affix-removal stemmer* algorithm in our experiment. This algorithm removes suffixes and prefixes from terms leaving them as stems or replaces some suffixes with some default letters. Suffixes such as *-ing, -ed, -s* are deleted completely while *-ies* parts of *-ies, -eies, -aies* are replaced by *-y*. Prefixes such as *fore-, extra-, kilo-, micro-, milli-, intra-, intro-, ultra-, mega-, nano-, pico-, pseudo-, multi-, mono-, pro-, tele-, over-* are also deleted. Implementation of this algorithm required a few more if-checks other than its own rules to prevent producing wrong stems. For instance the word *skies* may have

been reduced to the stem *ski* which will not match the word *sky*.

A vector of 76 words is produced from the query document and to draw notice to the stemming done by our software the words and their stems are given in Table 3.2 - which is an extraction from the query document and its vector. It should be noticed that there are many words with similar stems and they also have similar meanings. A more detailed example of a newsgroup leaf vector which CHSD uses at its initial step is given in Apendix A.

Word	Stem	Word	Stem
multitask	task	product	duct
Sincerely	sincer	sincerity	sincer
suggestion	sugges	suggestive	sugges
application	applic	applicable	applic
satisfaction	satisfac	satisfactory	satisfac
commercial	commerci	highly	highli
simultaneously	simultan	Integers	integ

Table 3.2: Stemmed Words of The Query Document

In this experiment a sample database of 10 newsgroups is taken from top-level *comp* to categorize the query document given in Figure 3.16. Choosing the sample database out of sibling newsgroups is a challenge to see if CHSD could distinguish them. After building the index tree out of this database the query document is given as an input to CHSD and 0.5 is determined as the threshold value of similarity to travel down the tree.

Word	Doc-Fr	Gr-Fr	Word	Doc-Fr	Gr-Fr	Word	Doc-Fr	Gr-Fr
version	1	19	javascript	3	7	code	1	2
java	4	155	send	2	4	duct	4	8
task	1	2	form	2	3	html	1	9
asp	1	3	javac	1	16	total	2	4
compil	1	38	capable	1	2	package	2	37
applic	2	26	applet	1	11	number	3	11
website	1	8	script	1	3	command	1	7
button	1	2	freeware	1	2	convert	2	8

Table 3.3: Common Words

Firstly, word-frequency vector of the query document is constructed and then this vector is allowed to follow its path. Since similarity of the query document to node *comp.ai* is calculated as 0.4021 which is less than 0.5, all

of the *comp.ai*'s subgroups are ignored. Thus the vector followed the path *comp* - *comp.compilers* - *comp.compilers.tools* - *comp.compilers.tools.javacc* and *comp.compilers.tools.javacc* is determined as the best newsgroup to post this news by 64.31% similarity.

The similarity of 64.31% is the result of our Similarity function of CHSD. Common words and their frequencies of the query document and the newsgroup *comp.compilers.tools.javacc* are given in Table 3.3 and they are also italized in Figure 3.16. Similarity function detects these common words and calculates the similarity by using their frequencies according to the similarity formula given by equation 2 in Section 3.2.

In fact this is a pre-arranged experiment and the italized words in Figure 3.16 are the leading factors to this result. But Table 3.1 implies that there are many comp.ai subgroups with similarity over 50% to query document and it might be considered to post it to these groups as well whereas they are ignored.

There are three main reasons of such an implication :

- The sample database is choosen out of sibling newsgroups.
- Despite that the number of documents are equal in all newsgroups some of them contain excessively long news resulting in huge leaf vectors for our algorithm such that they become dominant over other newsgroups. This causes an unfounded increase in similarity measure and a biased result.
- It needs a great deal of manual work to expand a stoplist to such an optimal extent that filtering stopwords prevents this kind of unbalanced stiuation.

In addition a stopword determined for a group might be a characteristic feature for another one. The best way is to have a different stoplist for each newsgroup or at least for each top-level newsgroup and then to produce different vectors out of the query document according to these stoplists. This will definitely increase the time complexity to such an extent that people might not prefer to use such a software. Another way is to use auxilary algorithms for selecting representative words or features of each group as explained in [25].

Priority	Name of Newsgroup	Similarity
1	comp.compilers.tools.javacc	0.6431
2	comp.ai.edu	0.5737
3	comp.ai.neural-nets	0.5692
4	comp.ai.nat-lang	0.5503
5	comp.ai.alife	0.5494
6	comp.ai.genetic	0.5474

Table 3.4: Proposed Newsgroups

On the other hand, such results are not bad for our approach since we want to propose more than one newsgroups for a news for posting. According to the results in Table 3.1, CHSD will propose the newsgroups given in Table 3.4 by their priority and the user will be free to choose one more newsgroups in the proposed list.

Chapter 4

Evaluation

In this section we do complexity analysis of CHSD and give details of our experiments that have been done so far. To reveal the best sides of CHSD more clearly we used KNN algorithm as a competitor in our first experiment. Our last experiment is a scalability test for CHSD such that the database used is 2.5 times bigger than the database of the first experiment and the algorithm has been run with the optimal threshold value found in the previous experiment.

4.1 Complexity Analysis

As a function of input sizes, we deal with the running time of CHSD in time complexity analysis and the memory space that CHSD requires in space complexity analysis.

4.1.1 Time Complexity

Notations given in Table 4.1 are used to explain complexity analysis. Actually these notations are abbreviations retirieved by concatinating initial letters of the input data features such as numbers of words, documents and newsgroups.

Time complexities of each function of CHSD are given in Figure 4.1. As

NOTATION	MEANING
twon	total words of node
twod	total words of a document
tcon	total children of a node
tgodb	total groups of database
tdog	total documents of a group
twog	total words of a group
tcf	total categories found
h	height of the index tree

Table 4.1: Notations Used in Complexity Analysis Formulations

explained in Chapter 3 ProcessData, InitTree and BuildTree are implemented sequentially in the learning phase. Thus the time complexity of learning phase is $O(tgodb \cdot tdog \cdot twog)$.

FindCategories is the main function of the categorization phase. It calls HierarchicalSearch and HierarchicalSearch calls auxillary functions Similarity and Insert. Since HierarchicalSearch and Similarity functions are the most time consuming functions, time complexity of categorization phase goes to $O(h \cdot \log tcon \cdot twon^2)$.

As for the overall time complexity of CHSD, learning phase is so dominant that time complexity of CHSD is $O(tgodb \cdot tdog \cdot twog)$. Because main goal of CHSD is to learn the built-in hierarchy in the database and to make the categorization phase easier and faster. But it should be noticed that in this thesis the learning phase is accepted as the work which begins with raw data and ends up with an index-tree that represents the learned hierarchy of the database. Briefly the time complexity of CHSD grows by the number of groups in the database, the number of documents in the groups and the number of words in the documents.

4.1.2 Space Complexity

An algorithm which requires only constant memory space such that the memory required is independent on input size is called as *in-place algorithm*. Unfortunately CHSD is not an in-place algorithm because its space compelxity is $O(now \cdot non)$ in the worst case where now is the number of words in the global

FUNCTION	TIME COMPLEXITY
	LEARNING PHASE
ProcessData	$O(tgodb(tdog(twod + twog) + twog))$ $= O(tgodb \cdot tdog \cdot twog)$
InitTree	$O(tgodb(tdog(twod + twog)) + twog) + h \cdot \log tgodb \cdot twog)$ $= O(tgodb \cdot tdog \cdot twog)$
BuildTree	$O(h \cdot \log tgodb \cdot twog)$
	CATEGORIZATION PHASE
Insert	$O(tcf)$
Similarity	$O(twon^2)$ ($O(twon \cdot \log twon)$ when Binary Search is used)
HierarchicalSearch	$O(h \cdot \log tcon \cdot twon^2)$
FindCategories	$O(twod + h \cdot \log tcon \cdot twon^2)$ $= O(h \cdot \log tcon \cdot twon^2)$

Figure 4.1: Time Complexities of Functions

dictionary of the database and *non* is the number of nodes in the index tree. That is each word occurs in all documents and copied to all nodes in the index tree. However it is not possible for this worst case to be realized since CHSD eliminates the words according to their norm-scaling values as explained in Chapter 3.

An algorithm which has consistent space complexity for all cases is called as *every-case space complexity algorithm*. CHSD is not an every-case space complexity algorithm either. Because its space complexity is dependent on the number of words extracted from the database and the number of nodes that the hierarchy requires. In addition, words are chosen to be copied into upper nodes according to the norm-scaling value which is a variable.

We implemented CHSD in C++ programming language. In C++, function calls are handled by using an internal stack of activation records. Every activation record needs a piece of memory space to hold it. As the activation records become larger and larger, the stack to hold them requires more and more memory space. In recursive algorithms, recursive function calls can build up the stack very quickly while in sequential algorithms memory requirement for this stack is limited. CHSD is a recursive algorithm in constructing index tree and searching over it. In the space vs. speed trade-off CHSD prefers speed.

4.2 Emprical Evaluation

4.2.1 Performance Measures

Text categorization is the assignment of free text documents to one or more of a predefined set of categories. While a number of different accuracy measures have been used in evaluating text categorization in the past, almost all have been based on the same model of decision making by the categorization system [8]. Some of these measures are *recall and precision, accuracy or error, break-even point, micro average, macro average* and *11-point average precision.*

For the evaluation of our test results of CHSD and KNN we used interpolated 11-point average precision measure method which is especially designed for category ranking.

Interpolated 11-point Average Precision

Category ranking can be evaluated by using measures similar to the conventional measures for evaluating ranking-based document retrieval systems such as recall, precision, and 11-point average precision. Given a classifier whose input is a document, and whose output is a ranked list of categories assigned to that document, the recall and precision can be computed at any threshold on this ranked list [40]:

$$recall = \frac{categories\ found\ and\ correct}{total\ categories\ correct} \tag{1}$$

$$precision = \frac{categories\ found\ and\ correct}{total\ categories\ found} \quad (2)$$

where *categories found* means categories above the decision threshold.

For the global evaluation of a classifier on a collection of test documents, we adapt the procedure for the conventional *interpolated 11-point average precision*, as described below:

- For each document, compute the recall and precision at each position in the ranked list where a correct category is found.
- For each interval between recall thresholds of 0%, 10%, 20%, ..., 100%, use the highest precision value in that interval as the *representative* precision value at the left boundary of this interval.
- For the recall threshold of 100% the representative precision is either the exact precision value if such a data point exists, or the precision value at the closest point in terms of recall. If the interval is empty, use the default precision value of zero.
- *Interpolation:* At each of the above recall thresholds, replace the representative precision using the highest score among the representative precision values at this threshold and the higher thresholds.
- *Per-interval Averaging:* Average per-document data points over all the test documents, at each of the above recall thresholds respectively. This step results in 11 per-interval average precision scores.
- *Global Averaging:* Average of the per-interval average precision scores to obtain a single-numbered performance average (*11-pt AVGP*).

4.2.2 Data Set

For the evaluation tests of CHSD a sample database of 2000 documents is collected from Usenet top-level groups called **comp** and **bionet** as shown in Table 4.2. Each newsgroup under comp contains 100 documents and each newsgroup under bionet contains different number of documents. This is done on purpose to test robustness of CHSD. Because in real life it is not possible to

find a balanced database such that each group contains equal number of documents. In fact categorization algorithm has to prevent the effect of different datasizes and irrelevant features on results [3]. This database does not include all newsgroups of comp because it is one of the biggest top-levels in Usenet system. However we collected all news in all newsgroups of under the top-level bionet.

Label	Name of Newsgroup	Documents	Words
b1	bionet.agroforestry	97	4265
b2	bionet.announce	46	3419
b3	bionet.audiology	30	937
b4	bionet.biology.cardiovascular	40	1492
b5	bionet.biology.computational	9	618
b6	bionet.biology.grasses	29	1112
b7	bionet.biology.n2-fixation	10	527
b8	bionet.biology.symbiosis	13	698
b9	bionet.biology.tropical	25	971
b10	bionet.biophysics	42	1773
b11	bionet.celegans	30	1003
b12	bionet.software	97	2480
b13	bionet.toxicology	32	1931
c1	comp.apps.spreadsheets	100	1799
c2	comp.arch	100	2481
c3	comp.arch.arithmetic	100	2113
c4	comp.arch.fpga	100	2379
c5	comp.arch.storage	100	2524
c6	comp.compilers.tools.javacc	100	1801
c7	comp.compilers.tools.pccts	100	1746
c8	comp.compression	100	1423
c9	comp.databases.btrieve	100	2045
c10	comp.databases.informix	100	2139
c11	comp.databases.ms-access	100	2183
c12	comp.os.linux.development.apps	100	1793
c13	comp.os.linux.development.system	100	2026
c14	comp.os.linux.networking	100	1902
c15	comp.os.linux.setup	100	2128
-	Total Number of News	2000	-

Table 4.2: Sample Database

Table 4.3 shows inner nodes and number of words in their vectors constructed by CHSD. There are 9 inner nodes over 28 leaves - in other words 28 categories - in this database and number of words in the global dictionary is

6495. In Table 4.2 and Table 4.3 all these newsgroups and innner nodes are labeled and the index tree structure out of this experiment is given in Figure 4.2 which is drawn by using these labels.

Appendix A shows the leaf vector of the newsgroup *bionet.biology.n2-fixation* which has only 10 documents and 527 words. That many documents are not enough to gain the most representative features of a newsgroup. When we examine the leaf vector in Appendix A, it is seen that there are many irrelevant words with higher norm-scaling values such as free, job, make, call.

Label	Name of Inner-Node	Number of Words
b14	bionet	2696
b15	bionet.biology	941
c16	comp	2576
c17	comp.compilers	1187
c18	comp.compilers.tools	1187
c19	comp.databases	1574
c20	comp.os	1903
c21	comp.os.linux	1903
c22	comp.os.linux.development	1219
c23	comp.arch	1981
c24	comp.apps	1421

Table 4.3: Inner Nodes of Index-Tree

4.2.3 Test Results

There are three measures taken for the evaluation of test results :

- *Train Time* is the time spent during learning phase but does not include the time spent for processing raw data. Begins with matrix upload and ends at the time categorization begins.
- *Test Time* is the time spent during categorization of query documents.
- *Accuracy* is the value calculated by interpolated 11-point average precision method which takes in the real labels and the labels found by the running algorithm.

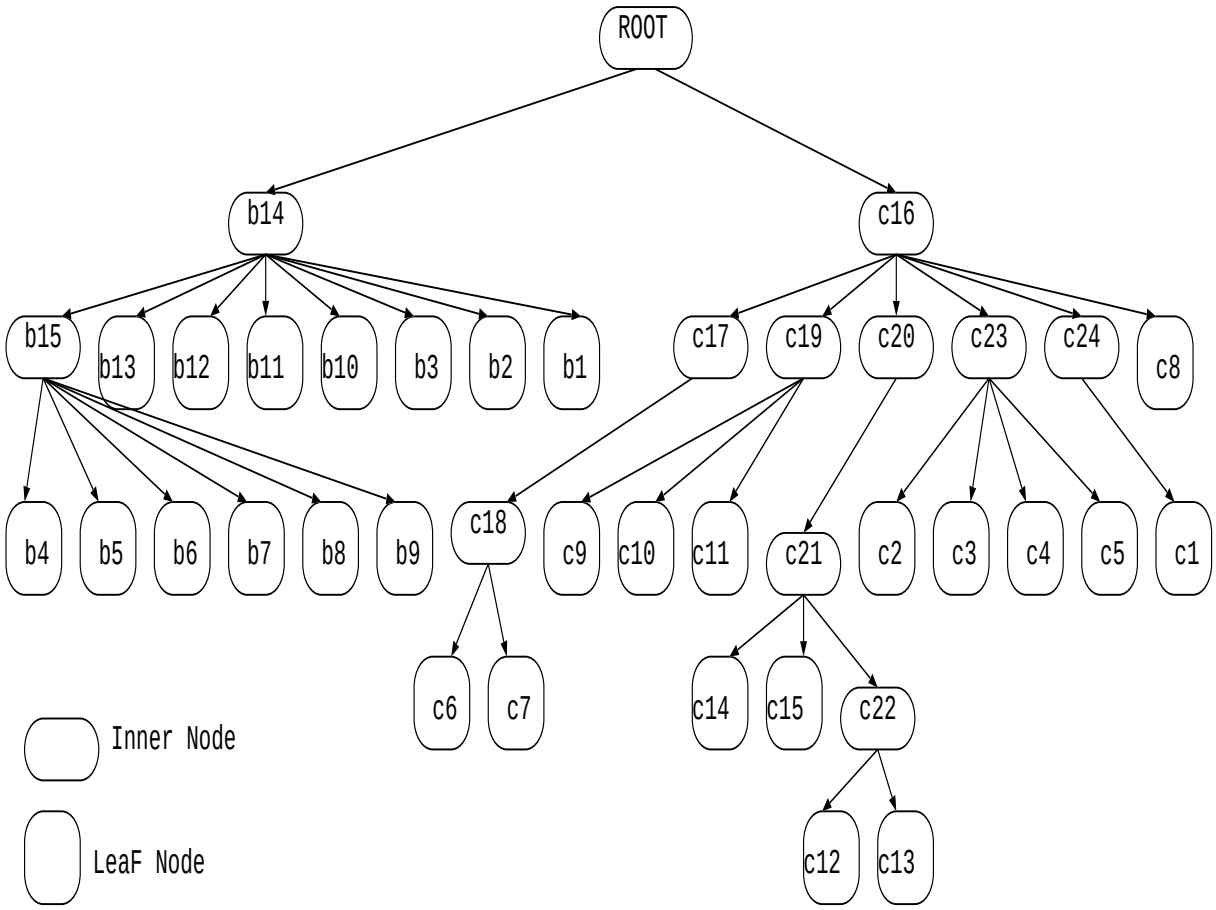


Figure 4.2: Index Tree of The Sample Database

10-fold cross validation technique [13] is used in the experiments. Therefore, the accuarcy of algortihms on data set is computed as the average of 10 runs in each of which a disjoint set of 1/10 of the data set is used in the querying, and the remaining 9/10 in the training phase. To calculate the accuracy of each fold Interpolated 11-point Average Precision method is used as explained in the previous section.

Firstly we run[1] CHSD over database of 2000 documents shown in Table 4.2 for 16 times with different threshold values floating between 0.5 - 0.875. At each run we increased the threshold value by 0.025 to determine the best threshold value which gives the highest accuracy.

The results are given in Table 4.4. The best accuracy value that we found is **0.87** with threshold value **0.825** at the 14^{th} run which took **4724** msec. train

[1]We used a computer with 64 MB memory and an Intel Celeron 333 Mhz. processor in our experiments.

No.	Threshold	Train(msec.)	Test(msec.)	Accuracy
1	0.5	4707	6721	0.22
2	0.525	4657	6679	0.29
3	0.55	4602	6610	0.33
4	0.575	4658	6551	0.38
5	0.6	4687	6509	0.42
6	0.625	4668	6454	0.46
7	0.65	4671	6406	0.48
8	0.675	4642	6392	0.51
9	0.7	4635	6350	0.55
10	0.725	4606	6305	0.59
11	0.75	4721	6289	0.63
12	0.775	4604	6237	0.72
13	0.8	4664	6191	0.81
14	**0.825**	**4724**	**6173**	**0.87**
15	0.85	4623	6122	0.79
16	0.875	4723	6099	0.65

Table 4.4: Results of CHSD on the Database Given in Table 4.2

time and **6173** msec. test time. We accept these values as the representative values of CHSD to compare it with KNN. We did not run CHSD with further more threshold values since accuracy began to decrease after 0.825 as shown in Figure 4.3.

Since we shuffled data at the beginning of each run of CHSD, the train time changed sporadically as shown in the chart given in Figure 4.4. However the test time decreased as the threshold value increased because number of unvisited nodes becomes higher by the increase in threshold value. That is, as threshold value approaches to 1 test time approaches to 0 as shown in the chart given in Figure 4.5.

Next we ran KNN over the same database with k parameter as 10 and got **0.93** accuracy, in **205492** msec. train time and **27196800** msec. test time. In other words, for each data fold KNN spent nearly 3.5 minutes to train and 454 minutes to test the data. It took 76 hours to gain the results of the database given in Table 4.2. The version of KNN implemented by Bilkent University Machine Learning Group is used in our experiments. It makes use of Euclidean distance to determine the similarities of the training instances to the query point.

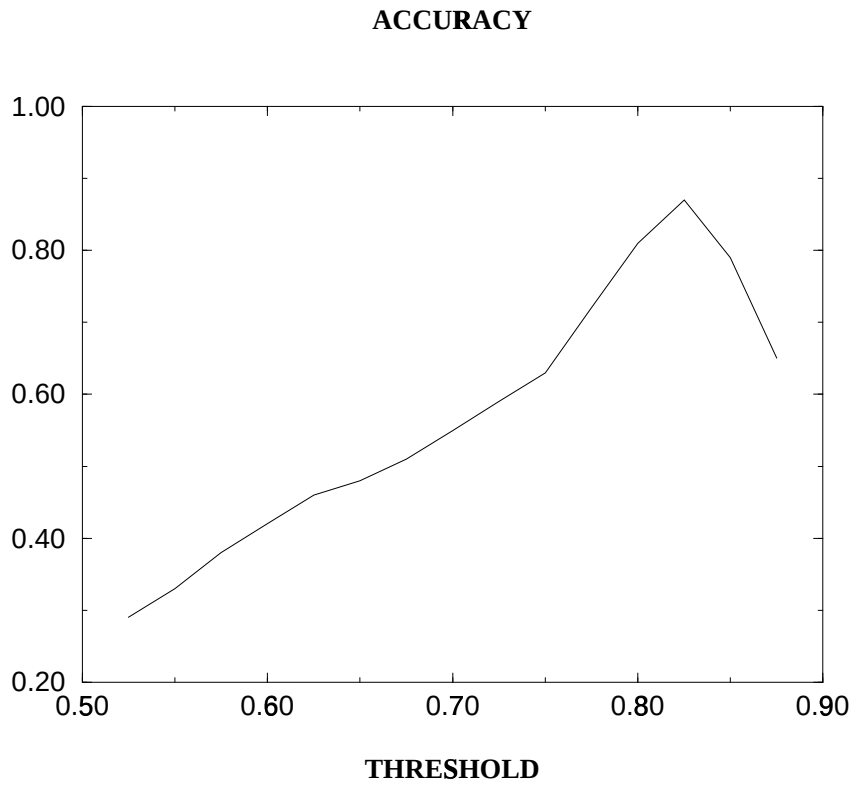


Figure 4.3: Accuracy vs. Threshold

As compared with KNN, CHSD resulted in 0.06 less accuracy even with the most optimal threshold value as shown in Table 4.4. Nevertheless this is not an untolerable loss of accuracy as we consider the speed difference between them. According to the results of this experiment CHSD was faster than KNN **43** times in the training time and for **4405** times in the test time. We claim that CHSD is more optimal and functional than KNN in these circumstances. Time scalability of an algorithm is so important because the amount of data is getting bigger each day.

4.2.4 Scalabilty Test

To test the scalability of CHSD the database used in previous experiment is expanded to the size of 5000 documents by adding the database shown in Table 4.5. We take the threshold value of similarity as 0.825 since it was found to be the most optimal value in the previous test.

In this experiment accuracy is calculated as **0.892** in **21922** msec. train

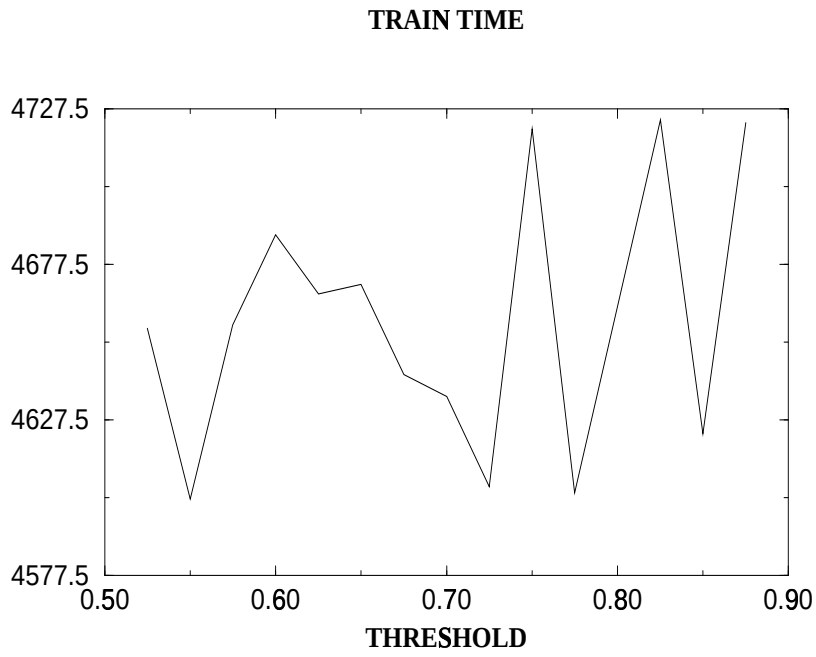


Figure 4.4: Train Time vs. Threshold

time and **48656** msec. test time. Naturally train and test times increased but not beyond the considerable limits. Fortunately accuracy increased by 0.022 as compared to 0.87 in previous test. The main reason of this increase is the effect of larger data. The more learning brings out the better results.

Besides accuracy was expected to be more than this since we expanded the database by adding 3000 more documents to the database in the previuos test as shown in Table 4.5. The main reason of this less than expected increase in accuracy is the approach of Interpolated 11-point Average Precision such that redundant hits of the algorithm are severly punished during the evaluation. That is, if there are wrong ones in a series of answers besides right ones then accuracy is decrased by a great extent. Since the database is 2.5 times bigger than the previous databases redundant hits are inevitably proliferated in this experiment.

We could not apply KNN to this database since it was impossible within means of space limitations. The KNN software that we used takes in an input matrix of $n \times m$ where n is the number of documents and m is the number of all words in the global dictionary of the database. So this matrix is such a sparce matrix that non-zero values are the frequencies of the words. Size of this matrix grows by the number of the documents since each new document adds a new row and a few new columns to it. By the way in this software there

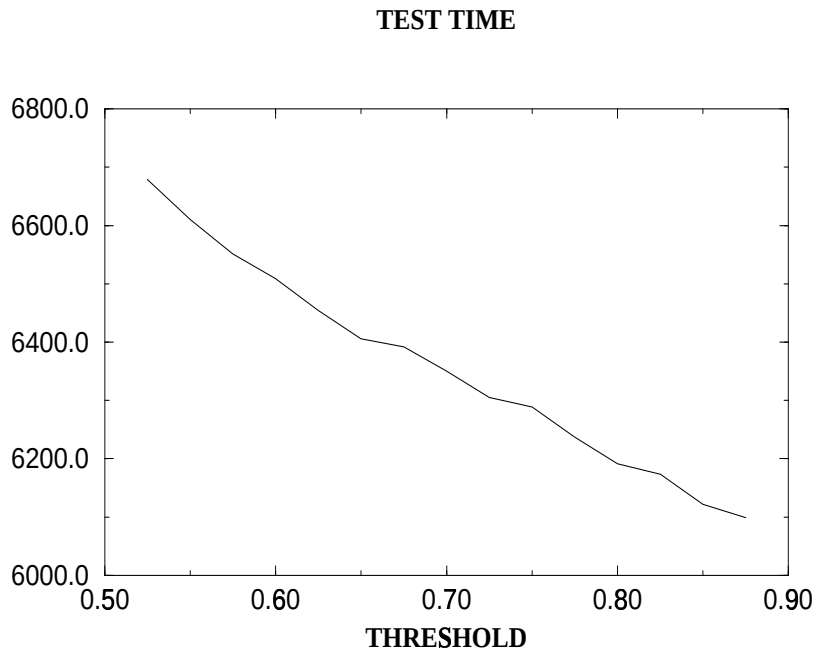


Figure 4.5: Test Time vs. Threshold

is no elimination among the words - other than filtering stopwords - causing many redundant words to be stored in the memory. For example the database with 5000 documents produced 5000×7422 matrix of integers which is stored in a 149 MB file as an input for KNN. It is fairly huge as compared to the 64 MB memory of an average computer.

These experiments prove that CHSD is scalable and outperforms KNN in view of time and space complexity measures. Because CHSD runs over a dense input with no zero values and eliminates redundant words by checking norm-scale values. For instance only 4.5 MB memory space is used by CHSD in this experiment with 5000 documents. At the other hand, as proved by the previous experiment CHSD is faster than KNN and its time complexity does not grow as fast as KNN's does. Because CHSD uses a hierarchical structure and a threshold value of similarity to travel on it. Therefore it neither does redundant calculations nor deals with unnecessary data points.

Name of Newsgroup	Documents	Words
comp.ai.alife	100	3883
comp.ai.edu	100	2858
comp.ai.fuzzy	100	3197
comp.ai.genetic	100	3039
comp.ai.nat-lang	100	5247
comp.ai.neural-nets	100	2463
comp.ai.philosophy	100	4995
comp.databases.object	100	2564
comp.databases.olap	100	2141
comp.databases.paradox	100	1895
comp.databases.pick	100	2113
comp.databases.sybase	100	2381
comp.databases.theory	100	2513
comp.databases.visual-dbase	100	1521
comp.groupware.lotus-notes.admin	100	1440
comp.groupware.lotus-notes.apps	100	1107
comp.groupware.lotus-notes.misc	100	1266
comp.groupware.lotus-notes.programmer	100	1340
comp.os.linux.hardware	100	1031
comp.os.linux.portable	100	1304
comp.os.linux.powerpc	100	1362
comp.os.linux.questions	100	1307
comp.os.linux.security	100	1354
comp.os.ms-windows.misc	100	1875
comp.os.cpm	100	2124
comp.os.geos.misc	100	1324
comp.sys.ibm.pc.hardware	100	1808
comp.sys.mac.hardware	100	2240
comp.security.ssh	100	1046
comp.security.unix	100	1393
Total Number of News	3000	-

Table 4.5: Database Used to Expand Sample Database

Chapter 5

Conclusion

This thesis was started with the question of how to manage the huge database of Usenet newsgroups as a user. There are a lot newsgroups in the system and it is hard to find the a specific newsgroup for a new news to post. In this thesis we have presented a brand-new categorization algorithm named as CHSD for the solution of this problem. It is an appropriate algorithm for hierarchically structured databases such that the database owns a substantial and built-in hierarchy in itself just as Usenet does. The more CHSD learns this hierarchy the better categorization results it brings out.

We succeeded in providing a detailed experimental performance study of both CHSD and KNN algorithms. As compared with KNN, CHSD results in less accuracy even with the most optimal threshold value. But this is not an untolerable difference. However a query takes shorter time by CHSD than it does by KNN. For instance in the experiment of 2000 documents and 6495 words CHSD is 2514 times faster than KNN at the average of total train and test time.

This comparison does not only mean comparing two different algorithms but also means comparing hierarchical approach vs. flat approach, similarity measure vs. distance measure and importance of accuracy vs. importance of speed. CHSD prefers hierarchical approach and similarity measure, and greatly outperforms KNN in speed to the cost of a little loss in accuracy. We conclude that time scalability of an algorithm is so important because amount of data is getting bigger each day thereby CHSD accomplishes its mission.

The advantages of CHSD:

- Robust against the differences among datasizes of groups to some considerable extent.
- Faster than the most of the traditional categorization algorithms.
- Scalable to larger databases in terms of time and space complexities.
- Easy to implement, use and improve.
- It is a combination of components which are open to modifications and improvements.
- Height of the index tree is limited by the depth of the hierarchy in database. Thus no redundant level is constructed.
- It has two useful measures to play with for better results which are called threshold of norm-scaling value and threshold of similarity.

The disadvantages of CHSD:

- It is vulnerable to the spamming in the database.
- Necessarily most of the initial data is replicated in the inner nodes of index tree to construct the hierarchy in the memory.
- The source database that the initial data is produced from must be maintained periodically to keep index-tree updated in long term.
- The optimal threshold value of similarity that controls down-travel of a query vector should be determined beforehand by experimentation since it varies according to the types and sizes of databases.

For more improved versions and better usages of CHSD algorithm there is future work which could be done later:

- The algorithm might have a dynamic structure such that it continues its learning phase while responding to queries and receiving newly posted news from server and updates its index.

- Some of the newsgroups no longer have message traffic. Algorithm can detect these groups and eradicate them from its index. This might be done by checking the *Date* header-lines of posted messages.
- Weigthening words might be helpful to solve biased similarity measures.
- Constructing a different stoplist for each newsgroup will make it more easy to get rid of redundant words and will increase the accuracy to the cost of more time consuming data processing.
- Already constructed tree might be copied to an object file and stored into disk to make later usages faster.

Finally we conclude that CHSD is a worthwhile departure point to implement an efficient and user friendly categorization software. Because it is possible to add new aspects to CHSD by developping new components for better accuracy. In addition, CHSD is applicable to any kind of text database that has or could be arranged into a hierarchical data structure. In short, we believe that CHSD merits further attention in order for it to be put to better use in the future.

Appendix A

Leaf Vector of

bionet.biology.n2-fixation

Word	Frq	N-S	Word	Frq	N-S	Word	Frq	N-S
led	1	1	act	2	1.41421	gem	1	1
pai	1	1	job	8	7.07107	aol	2	1.41421
sai	2	1.41421	map	4	2	bui	2	1.41421
wai	1	1	fit	1	1	gui	4	2.82843
ion	1	1	per	1	1	gmt	2	1.41421
fun	1	1	new	2	2	owe	1	1
how	1	1	ron	1	1	ext	2	2
low	2	1.41421	now	3	1.73205	put	3	1.73205
back	3	2.23607	lead	2	2	abil	2	1.41421
area	1	1	data	1	1	base	5	3
read	2	1.41421	page	5	3.60555	make	11	6.08276
cash	1	1	life	1	1	name	4	2.82843
free	12	5.65685	mail	4	2	real	5	2.64575
sale	4	2.44949	ject	5	2.23607	line	5	3
possibl	1	1	fulltim	1	1	smarter	1	1
continu	2	2	inquiri	1	1	turnkei	1	1
resourc	4	2	context	1	1	softwar	2	1.41421
request	1	1	network	4	2	univers	3	1.73205
california	4	2.82843	background	1	1	vanderbilt	1	1
technologi	1	1	subsidiari	1	1	comprehens	1	1
drosophila	1	1	understand	1	1	permacltur	1	1
scienceweek	4	4	bioinformat	2	2	biologynfix	1	1
permacultur	1	1	contributor	1	1	neurodegener	1	1
selfdisciplin	1	1	lightningfast	1	1	complimentari	1	1

Table A.1: Leaf Vector of *bionet.biology.n2-fixation* (1)

Word	Frq	N-S	Word	Frq	N-S	Word	Frq	N-S
financi	1	1	medicin	1	1	guidanc	1	1
openfac	1	1	briefli	1	1	bernard	1	1
homebas	1	1	special	1	1	nowadai	1	1
hammond	1	1	increas	3	1.73205	address	2	2
compani	4	2.82843	nematod	2	1.41421	central	2	1.41421
descrip	2	2	qualifi	2	2	contact	5	3
account	6	4.24264	privaci	1	1	develop	1	1
announc	2	1.41421	comment	1	1	complet	3	1.73205
forward	2	1.41421	purchas	1	1	atrophi	1	1
success	4	2.82843	absolut	2	2	content	1	1
support	1	1	american	1	1	alphabet	1	1
interfac	2	2	research	6	6	approach	1	1
directli	1	1	merchant	6	4.24264	deliveri	1	1
cashflow	1	1	maryland	2	1.41421	phenomen	1	1
individu	1	1	simplifi	1	1	advertis	1	1
glossari	1	1	transfer	1	1	thousand	1	1
discount	2	1.41421	frequent	1	1	mountain	2	1.41421
interest	3	1.73205	approxim	1	1	whatsoev	1	1
transmit	1	1	thankyou	1	1	position	3	3
thingyou	1	1	question	3	2.23607	opportun	5	3.60555
callpacif	2	1.41421	searchabl	1	1	sheffield	1	1
hierarchi	1	1	highlight	2	2	recommend	2	1.41421
bioscimrc	4	2	chainlett	1	1	sellingdo	1	1
prerecord	1	1	regularli	1	1	inconveni	1	1
rcjohnsen	1	1	millenium	2	2	kumershek	1	1
parkinson	1	1	scientist	1	1	astronomi	1	1
mycorrhiz	1	1	universit	1	1	wbehherjec	1	1
call	8	4.89898	talk	2	2	inalifetim	1	1
oblig	2	2	server	1	1	cell	1	1
help	4	2.44949	relat	3	3	oxidative	1	1
onlin	10	5.83095	collect	1	1	bionet	10	3.16228
polici	3	2.23607	famili	1	1	mlm	2	1.41421
immedi	1	1	comput	2	2	home	5	3
somon	1	1	platinum	3	3	summari	1	1
le	1	1	panel	1	1	center	1	1
mentor	1	1	vender	2	1.41421	ignor	1	1
fine	4	2.82843	kind	3	1.73205	link	3	1.73205
sincer	1	1	consist	1	1	long	3	2.23607
monei	6	6	sw	3	3	function	1	1
resend	1	1	acounti	1	1	phone	4	2.82843
thought	1	1	biologi	6	4.24264	good	2	1.41421
root	2	2	apologi	1	1	bro	1	1
frog	1	1	troubl	2	1.41421	lyon	2	2
happi	1	1	repli	1	1	import	2	1.41421
input	1	1	noprofit	1	1	topic	1	1
applic	4	2.82843	expert	2	1.41421	mailbox	1	1
leav	2	1.41421	bark	1	1	carl	1	1
earn	3	2.23607	rare	2	1.41421	jbrin	2	1.41421
screen	1	1	jersei	2	1.41421	server	1	1
veryifi	1	1	airlin	2	2	more	1	1
biolog	1	1	archiv	2	2	dollar	1	1
abnorm	1	1	attend	1	1	medium	1	1
materi	1	1	market	4	2.44949	effort	2	1.41421
editor	1	1	commis	1	1	commit	1	1
inform	3	1.73205	servic	7	4.3589	primat	1	1
websit	9	5.91608	hubsit	2	2	answer	3	3

Table A.2: Leaf Vector of *bionet.biology.n2-fixation* (2)

Word	Frq	N-S	Word	Frq	N-S	Word	Frq	N-S
normal	3	1.73205	current	2	2	surgic	1	1
easi	1	1	pass	3	1.73205	abstract	1	1
schoo	1	1	cess	6	4.24264	describ	1	1
messag	8	4	respect	1	1	cism	1	1
discus	1	1	list	1	1	wish	1	1
configur	2	1.41421	hms	1	1	insa	1	1
cost	2	1.41421	lose	3	3	custom	4	2.82843
card	14	9.89949	cart	2	1.41421	matt	2	2
rate	2	1.41421	satisfac	4	2.82843	action	1	1
detail	1	1	liter	1	1	nitrogen	1	1
site	1	1	vital	2	1.41421	onthi	1	1
carrer	1	1	lot	1	1	notabl	1	1
extrem	2	1.41421	neuron	1	1	club	2	2
mous	1	1	fruit	1	1	colleagu	2	1.41421
advis	2	1.41421	invit	1	1	recogn	1	1
power	2	1.41421	fungi	2	2	fix	1	1
grateful	1	1	box	1	1	day	3	1.73205
hey	2	1.41421	joyou	2	2	literatur	1	1
connec	1	1	bean	1	1	dear	1	1
featur	2	1.41421	learn	3	2.23607	search	5	3.31662
year	5	2.23607	channel	5	5	find	2	1.41421
place	1	1	email	10	4.47214	small	4	2.82843
sparr	1	1	frame	2	2	grader	1	1
train	1	1	guarante	1	1	await	1	1
tabl	1	1	web	2	2	public	1	1
fact	2	1.41421	access	2	2	receiv	10	5.2915
held	1	1	pfcc	1	1	ticket	2	2
travel	1	1	incom	2	2	locat	1	1
duct	7	4.3589	apprieci	1	1	dealt	2	1.41421
confirm	1	1	radiat	1	1	respond	1	1
respons	2	1.41421	vide	4	2	andor	1	1
index	3	3	bodi	1	1	model	4	4
nodul	1	1	todai	4	2.44949	need	2	1.41421
weekth	2	1.41421	skeptic	1	1	blem	2	1.41421
electron	1	1	pleas	2	1.41421	spent	2	1.41421
credit	20	14.1421	fred	1	1	great	4	2.82843
present	1	1	eastern	2	1.41421	yeast	1	1
eventu	1	1	biomed	1	1	afford	1	1
offer	7	5.19615	differ	2	1.41421	lifetim	1	1
imfo	2	2	info	4	2.82843	degre	1	1
legitim	1	1	high	1	1	engin	3	3
advic	1	1	repres	4	2.82843	organ	6	6
suggest	1	1	school	3	3	recent	2	2
daili	1	1	main	5	5	waiv	2	1.41421
includ	1	1	scienc	2	2	tein	1	1
complaint	1	1	rhizobia	2	2	nitrif	2	2
click	2	2	approv	2	1.41421	join	1	1
voic	2	1.41421	quick	2	1.41421	commun	2	1.41421
easili	1	1	diseas	2	1.41421	direct	1	1
honest	1	1	tomato	2	2	script	1	1
anymor	1	1	revers	1	1	requir	1	1
explor	1	1	explos	1	1	report	11	11
involv	2	2	xenopu	1	1	system	3	1.73205
apolog	1	1	databas	1	1	credibl	2	1.41421

Table A.3: Leaf Vector of *bionet.biology.n2-fixation* (3)

Word	Frq	N-S	Word	Frq	N-S	Word	Frq	N-S
amaz	1	1	send	6	4.24264	meet	1	1
week	10	4.69042	focu	1	1	elit	1	1
time	8	4.47214	sell	1	1	spam	1	1
crit	1	1	busi	29	18.6279	pubm	1	1
menu	3	3	note	4	2	burn	1	1
past	2	1.41421	risk	2	1.41421	shop	2	1.41421
toll	4	2.44949	word	3	1.73205	titl	3	2.23607
hour	3	2.23607	grow	1	1	show	1	1
type	5	3	work	6	3.16228	issu	2	2
post	1	1	turn	1	1	beagl	1	1
chanc	2	1.41421	check	2	1.41421	chang	1	1
medic	3	3	reach	2	1.41421	field	1	1
charg	1	1	death	1	1	direc	2	1.41421
blank	1	1	sagan	1	1	local	4	2
brain	1	1	ethic	2	1.41421	delet	1	1
vacat	2	2	build	2	2	gener	2	2
bring	1	1	genet	1	1	speci	1	1
navig	1	1	genom	4	2	desir	1	1
level	1	1	human	4	2	thing	6	3.16228
centr	4	2	initi	1	1	updat	2	1.41421
spect	2	2	peopl	8	4.69042	tocol	1	1
devot	1	1	coupl	1	1	simpl	2	1.41421
month	4	3.16228	steve	1	1	world	4	2.82843
remov	12	6	error	2	1.41421	north	1	1
sourc	1	1	group	1	1	start	3	1.73205
visit	1	1	short	1	1	futur	2	1.41421
elegan	1	1	aggreg	1	1	availb	1	1
accept	15	9.94987	fitabl	1	1	hawaii	2	1.41421
fascin	1	1	scheme	1	1	harder	1	1
major	10	7.07107	ag	3	2.23607	de	1	1
dh	2	1.41421	jg	2	1.41421	fl	2	2
jm	2	1.41421	is	2	1.41421	uk	1	1
ps	1	1	yo	1	1	tv	1	1
bad	2	1.41421	add	3	1.73205	ibc	1	1
def	1	1	fee	2	1.41421	dan	2	1.41421

Table A.4: Leaf Vector of *bionet.biology.n2-fixation* (4)

Bibliography

[1] Akkuş A. and Güvenir H. A. K nearest neighbor classification on feature projections. In *Proceedings of ICML'96, Saitta L. (Ed.)*, volume ICML'96, pages 12–19, Morgan Kaufmann, Bari, Italy, July 1996.

[2] Akkuş A. and Güvenir H. A. Weighted k-nearest neighbor classification on feature projections. In *Proceedings of the 12th International Symposium on Computer and Information Sciences, Kuru S., Çağlayan M. U., Akın H. L. (Eds.)*, volume ISCIS'12, pages 44–51, Antalya, Turkey, October 1997.

[3] Güvenir H. A. A classification learning algorithm robust to irrelevant features. In *Proceedings of AIMSA'98, Giunchiglia F. (Ed.)*, volume AIMSA'98, pages 281–290, Sozopol, Bulgaria, September 1998.

[4] Frakes W. B. and Baeza-Yates R. *Information Retrieval (Data Structures and Algorithms).* MIS/Information Storage and Retrieval. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[5] Aha D., Kibler D., and Albert M. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[6] Boley D. Principal direction divisive partitioning. Technical Report TR-97-06, Department of Computer Science, University of Minnesota, Minneapolis, USA, June 1997.

[7] Koller D. and Sahami M. Hierarchically calssifying documents using very few words. In *Proceedings of 14th International Conference on Machine Learning(ML-97)*, pages 170–178, Nashville, Tennessee, USA, July 1997.

[8] Lewis D.D. Evaluating text categorization. In *Proceedings of the Speech and Natural Language Workshop*, pages 312–318, Morgan Kaufman, San Mateo, CA, February 1991.

[9] Han E., Boley D., Gomno M., Gross R., Hastings K., Karypis G., Kumar V., Mobasher B., and Moore J. Webace: A web agent for document categorization and exploration. Technical Report TR-97-049, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, USA, 1997.

[10] W. B. Frakes and R. Baeza-Yates. *Information Retrieaval: Data Structures and Algorithms.* Prentice-Hall, 1992.

[11] Demiröz G. and Güvenir H. A. Genetic algorithms to learn feature weights for the nearest neighbor algorithm. In *Proceedings of the 6th Belgian-Dutch Conference on Machine Learning, H. J. van den Herik and T. Weijters (Eds.)*, volume BENELEARN-96, pages 117–126, Universiteit Maastricht, The Netherlands, 1996.

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

[13] H. A. Güvenir and İ. Şirin. Classification by feature partitioning. *Machine Learning*, 23:47–67, 1996.

[14] Witten I. H. and Bell T. C. The zero frequence problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

[15] Uysal İ. Regression by partitioning feature projections. Master's thesis, Bilkent University, Computer Engineering, January 2000.

[16] Williams I. A mcmc approach to hierarchical mixture modeling. *Advances in Neural Information Processing Systems 12*, pages 680–686, 2000.

[17] Fürnkranz J. Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–171, 1997.

[18] Fürnkranz J. Separate and conquer rule learning. *Artificial Intelligence Review*, In press, 1998.

[19] Fürnkranz J. and Widmer G. Incremental reduced error pruning. In *Proceedings of the 11th International Conference on Machine Learning, Cohen W. and Hirsh H. (Eds.)*, volume ML-94, pages 70–77, New Brunswick, NJ: Morgan Kaufmann, 1994.

[20] Karlgren J. Newsgroup clustering based on user behavior - a recommendation algebra. Technical Report SICS-T-94/04-SE, Swedish Institute of Computer Science, March 1994.

[21] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clusterig: A review. In *ACM Computing Surveys*, volume 31-3, pages 264–323, ACM, 1999.

[22] Jain A. K. and Dubes R. C. *Algorithms for Clustering Data*. Advanced Reference. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[23] Lang K. Newsweeder: Learning to filter netnews. In *Proceedings of ICML-95, 12th International Conference on Machine Learning, Armand Prieditis and Stuart J. Russell (Eds.)*, pages 331–339, Lake Tahoe, US, 1995.

[24] Craven M., DiPasquio D., Freitag D., McCallum A., Mitchell T., Nigam K., and Slattery S. Learning to extract symbolic knowledge from the world wide web. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA., 1998.

[25] Devaney M. and Ram A. Efficient feature selection in conceptual clustering. In *Proceedings of 14th International Conference*, pages 164–170, Nashville, Tennessee, USA, July 1997.

[26] Aydın T. Regression by selecting best feature(s). Master's thesis, Bilkent University, Computer Engineering, September 2000.

[27] Domingos P. Occam's two razors: The sharp and the blunt. In *Proceedings of KDD'98*, 1998.

[28] Cutting D. R., Karger R. D., Pederse J. O., and Tukey J. W. Scatter / gather: A cluster based approach to browsing large document collections. In *Proceedings of 15th Ann International ACM Conference (SIGIR'92)*, pages 318–329, Denmark, 1992.

[29] Kohavi R., Langley P., and Yun Y. The utility of feature weighting in nearest-neighbor algorithms, ECML-97.

[30] Quinlan J. R. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[31] Unrau R., Stumm M., and Krieger O. Hierarchical clustering: A structure for scalable multiprocessor operating system design. Technical Report CSRI-268, Computer Systems Research Institute University of Toronto, Toronto, Canada M5S 1A4, March 1992.

[32] Cost S. and Salzberg S. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1):57–78, 1993.

[33] Honkela T., Kaski S., Lagus K., and Kohonen T. Newsgroup exploration with websom method and browsing interface. Technical Report A-32, Helsinky University of Technology, 1996.

[34] Yan T. and Garcia-Molina H. Sift atool for wide-area information dissemination. In *Proceedings 1995 USENIX Technical Conference*, July 1995.

[35] Yavuz T. and Güvenir H. A. Application of k-nearest neighbor on feature projections classifier to text categorization. In *Proceedings of the 13th International Symposium on Computer and Information Sciences, Güdükbay U., Dayar T., Gürsoy A., Gelenbe E. (Eds.)*, volume ISCIS'98, pages 135–142, Antalya, Turkey, October 1998.

[36] Mitchell T.M. *Machine Learning.* McGraw Hill, 1997.

[37] Cohen W. W. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning, Prieditis A. and Russell S. (Eds.)*, volume ML-95, pages 115–123, Lake Tahoe, CA: Morgan Kaufmann, 1995.

[38] Cohen W. W. Learning trees and rules with setvalued features. In *Proceedings of the 13th National Conference on Artificial Intelligene*, volume AAAI-96, pages 709–716, AAAI Press, 1996.

[39] Scott A. Weiss, Simon Kasif, and Eric Brill. Text classification in usenet newsgroups: A progress report. In *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*, pages 11–13, Bulgaria, September 1996.

[40] Yiming Yang. *An Evaluation of Statistical Approaches to Text Categorization.* Kluwer Academic Publishers, 1999.