INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Structural Identification of Unintelligible Documents

by

Martin Fontaine

A M. Sc. Thesis

Submitted to the School of Graduate Studies and Research of the University of Ottawa in partial fulfillment of the requirements for the degree of

Master of Computer Science*

School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario

* The Master of Computer Science program is a joint program with Carleton University, administrated by the Ottawa-Carleton Institute for Computer Science.

Résumé

Cette thèse traite de l'identification de documents non-intelligibles par apprentissage logiciel. Un document non-intelligible est un document qui n'est pas nécessairement exprimé en langage naturel. Nous avons développé une approche à trois niveaux incluant : la représentation de l'alphabet de base, l'extraction des attributs textuels et l'induction de modèles de classification par apprentissage logiciel. Pour la représentation de l'alphabet de base, nous présentons une technique de « clustering » qui permet de réduire la taille de l'alphabet et d'augmenter la densité des données d'entraînement. Pour l'extraction des attributs textuels, nous présentons quatre techniques adaptées pour les documents non-intelligibles. Parmi les techniques d'extraction présentées, nous avons développé deux techniques basées sur le concept de compression de données. Nous avons exploré deux approches pour accomplir l'induction du modèle d'identification : un système basé sur la génération de règles appellé RIPPER et une approche basée sur l'induction grammaticale. Afin de combiner les différentes techniques complémentaires, nous avons développé une architecture orientée-objet qui favorise la réutilisation du code. Les techniques présentées dans cette thèse ont été mises à l'épreuve lors d'une expérience sur trois domaines différents : l'identification des courriels, l'identification des documents BASE64 et l'identification des images GIF et BMP.

Abstract

This thesis is about the identification of unintelligible documents using machine learning techniques. An unintelligible document is a document that is not necessarily expressed in a natural language. We have developed a three level approach including: the representation of a base alphabet, the textual feature extraction and the induction of classification models using machine learning techniques. For the representation of the base alphabet, we have introduced a clustering technique that reduces the size of the alphabet and increases the density of the training documents. For the textual feature extraction, we present four techniques adapted for unintelligible documents. Among the presented feature extraction techniques, we have developed two techniques based on the data compression concept. We have explored two approaches to accomplish the induction of an identification model: a rule-based system called RIPPER and an approach based on grammatical inference. In order to combine different complementary techniques, we have developed an object-oriented framework that encourages code reuse. The presented techniques have been tested in an experiment on three different domains: the identification of e-mail documents, the identification of BASE64 encoded documents and the identification of GIF images versus BMP images.

Dédicace

Ce travail est sincèrement dédié à Sophie mon épouse pour sa patience, sa compréhension et sa générosité. Merci pour ton support moral sans lequel je n'aurrais pu terminer ce travail avec succès.

Table of Contents

List of Figures

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

List of Tables

Chapter 1

Chapter 3

Chapter 5

Chapter 1 – Introduction

1.1 – Overview

Since we entered the telecommunication era, the volume and the variety of information communicated has always increased with the evolution of the different communication media. With the improvement of telecommunication technology and the development of worldwide computer networks like the Internet, a vast volume of information is now available to the community. Because of the variety of software applications and the heterogeneous nature of the Internet, information can be represented in many different formats and could be communicated using various communication protocols. The fact that the information can be presented in any format does not mean that the information is directly intelligible to a human observer. Given a certain format, a series of manipulations has to be performed using different software tools, in order to derive a new format that will be clearly presented to a human end user.

The problem, from which the present work originated, is the identification of a certain piece of information as being associated with a certain well-known format. If a certain unknown document can be identified as being of a certain format, then an appropriate tool could be used to transform the data into a more intelligible format.

The present work focuses on the structural identification of unintelligible documents. Moreover, the system developed will learn its classification task by learning from pre-labeled training documents.

A detailed description of the problem's context and potential usefulness of a system that would satisfy the problem's requirements will be presented in this first chapter.

In chapter 2, a summary of the related work that has been done on similar problems will be presented. Up to now, a lot of research has been done to develop some systems that would classify documents into pre-identified categories, by learning the classification concepts only from pre-labeled training documents. Many of the techniques developed assumed that the training documents would be expressed in a certain natural language. This means that statistical and probabilistic analysis based on words or sequences of words is often the basis for the applied learning systems currently available. Also, previous research on grammatical inference addresses very similar types of problems. Grammatical inference is the field of machine learning that has for its objective to infer a formal grammar from training examples.

A classical problem encountered in machine learning is to identify a set of properties or features that are sufficient to characterize a collection of pre-labeled examples. Chapter 3 addresses the features extraction problem in the specific context of the present work. Four features extraction techniques will be presented with their respective advantages and disadvantages. A clustering technique used to reduce the size of the alphabet observed in the training documents will also be presented.

Having extracted some features and associated them with each training document, a classification model has to be induced from the training examples. The classification model will contain a certain form of knowledge acquired from training that will be used to do further classification of new data. The knowledge contained in the classification model can be expressed in terms of rules that would look for the presence or absence of certain sub-strings in the documents, or it could be expressed

with an acceptor finite state machine representing a grammar. Chapter 4 will present some techniques that allow the induction of a classification model based on the features extracted from the training documents.

In chapter 5, the results of the various techniques applied on some realistic data will be presented and discussed. Moreover, the objet-oriented framework that has been used to run the different experiments of this research project will be described. Such a framework could be very useful to interconnect and reuse components of a sophisticated machine learning system and would allow easy replacement of a part of the system for testing and experimentation.

Chapter 6 concludes this document and presents the lessons learned from the work accomplished. Usefulness and quality of the presented techniques will be discussed. The last section of this document contains a complete enumeration of concepts and idea that were not completely explored in the present work and that might constitute a basis for future consideration.

1.2 – Motivation

This section describes the ideal system that would have to be created to do structural identification of unintelligible documents. The context in which such a system would perform and its possible applications will be discussed.

1.2.1 – Problem's Context

The main objective of this research project is to develop a system that would perform structural identification of unintelligible documents. The ideal system would learn how to do the identification only by learning from pre-labeled training documents. Once trained, the system can classify new documents as being a part of a predefined class (Figure 1.1).

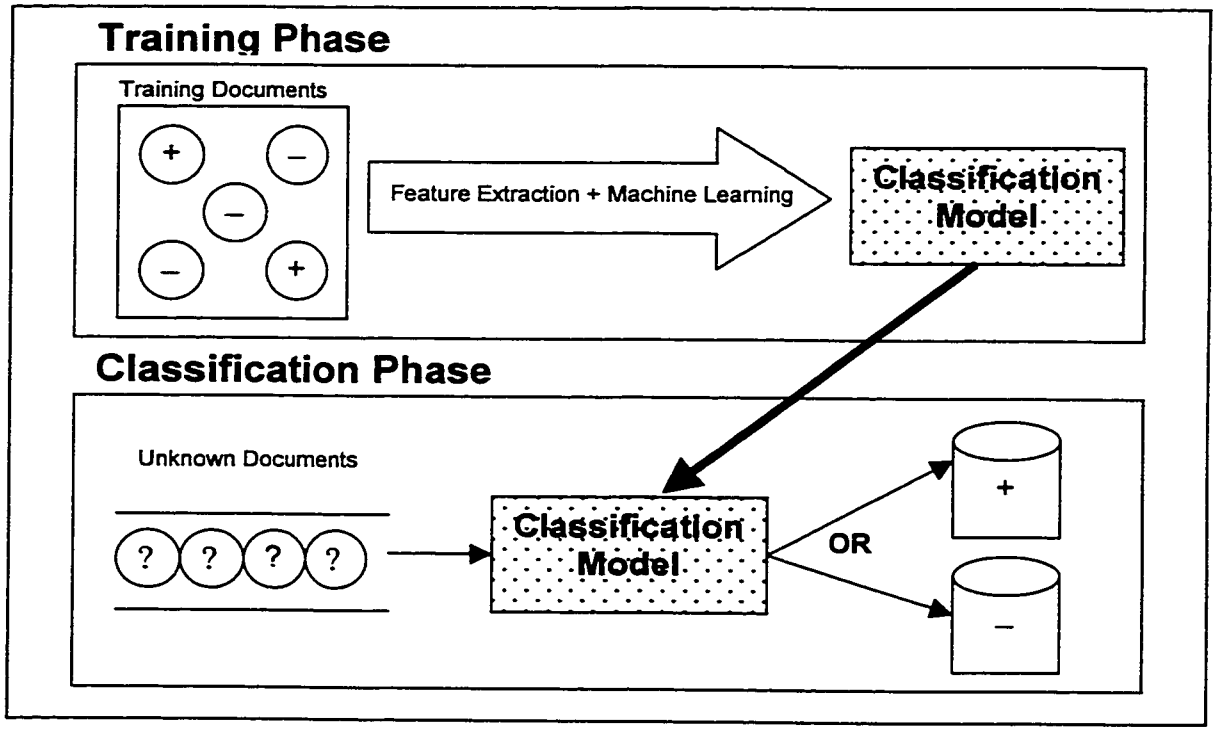


Figure 1.1: Classification System with Two Classes

For the particular scope of this work, the interest is the structural identification of unintelligible documents. This means that it cannot be assumed that the documents are written in a natural language like English or French. Also, the target concept (predefined classes) is based on the "structure" of the documents as opposed to the "content" of the documents. As an example, if the system is trained to classify documents written with a word processor, the identification classes could be MS-Word, Wordperfect, Framemaker, etc. Even if the documents themselves may talk about agriculture, finance or science, the interest of the current work is in their format or structure. Moreover, not assuming the presence of natural language offers the possibility to develop a data classification system for which a classical text classification algorithm could not succeed. Examples of such data would be audio and video files.

Given that the presence of natural language is not assumed, the presence of words or sentences is not guaranteed. However, the key of the classification can be based on the detection of certain sub-strings that appear only in all (or almost all) the training examples of a certain class and do not appear (or appear very rarely) in the training examples of the other classes. As an example, imagine that the two identification classes are: 1) e-mail and 2) not e-mail. The general appearance of an e-mail will contain a collection of sub-strings that are proper to all the e-mail and have a small probability to all appear in the documents that are not e-mails (Figure 1.2).

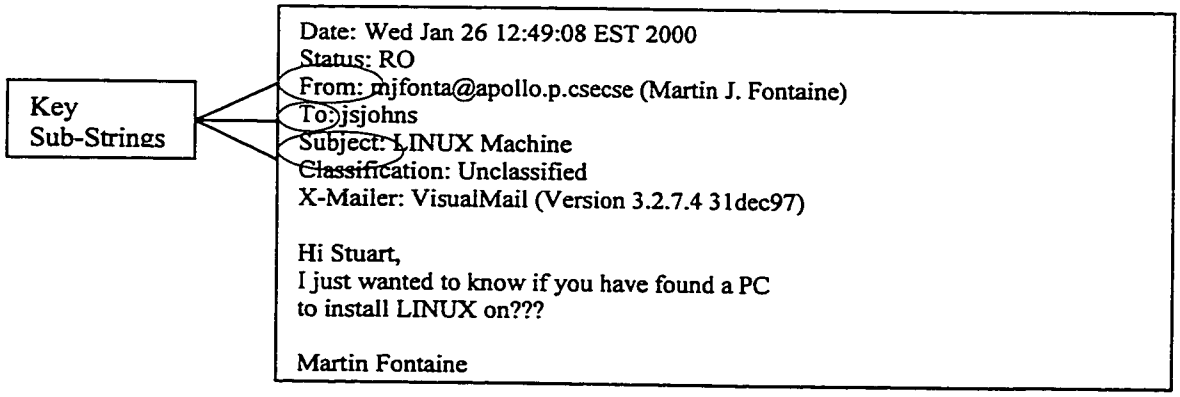


Figure 1.2 : E-mail Example

In this example, the probability that the three sub-strings "From:", "To:" and "Subject:" all appear in a non e-mail document at the beginning of a line is very unlikely. In some circumstances, the classification of a document cannot be done by detecting the presence of a series of sub-strings like in the e-mail case. From the training examples, it is impossible to find a sequence of sub-strings that are common to many examples of a class and absent from the training documents of the other classes. A good example illustrating this reality would be a system that classifies the "base64" encoded documents versus all the other types of documents (Figure 1.3).

```
J9rH0bCiFzq/aWksRHSLPEXXFq4NQu3LaYYhontGtApsiOjWyrwzRHQyon1XZ4jojhPt95QMFd2x
on03yFHRrZVe5eEuqgG78qt8f81D3TXTiI36W0JXuseOjG5h7qYuh7O0UP/S3/0trvXgY0V740bF
OsUlTwFFRrcomVyFIqNb92QKMDq6YOX7bmfYpBJ5fLSZsb0sNE2a8SYn8vpu24xvKKH2kWl/CVfi
+nfNe2jDXfIUXUdBSLey0GeTESXdIlFR44qS7ljRroeipFvec9sBd1291cVVlHSyon1tDeZyMkV9
aCjpVj6ZEhsHMNf9e9Vsimo9+DjRjtcWKZ3hmK9ONKR0htCHCEjpDDmyIuzv2Sm0f/NrmNYSNyUq
+6tQzLWvBmGgKOksFfYJeP0IbKlrHSndcaL9rnmgpTtWtC/yQEt3rGjnQ9HSGamKrm6gpTtGtO3F
RUpnpee2JuZa3vPZX16duDNF2GOiCV3pTkyUdJaUPlE/tBM3P7mJkM6o6wOKkO750CrwQkhnqbBP
wCjpDKkPaBNzDcd83Z+Fks4oQai+hV2/0PQ+mAslnbGmItQQ0hknUxTXENJZck3ZKEK6a0JLW3R0
RglCUHR0RlkfIqCjs6v0UYcdIZ21/Oi8ireWM5X9XeEEnukhRRE5oZl1jpDOOJnCryGkOxa0F93v
L60HHwva+4yiuqbiJjXyRtHRWTo8Gy5n0La3LwtbISAR2r/JNQydiTs+2q44IiF0fhPjTtMbcdc3
1FZBRnfsZ+/rPUwEiKubAIqM7tjPvgEYRUIVmIMpClzR0RkHU3R7aoW1Q6NSXavo6CynKfvJw1p2
fIiLjs6SHHmjDmuvtZvghpDOKECoEA0hneX11H1/tbBk6JGJgJDuGtCCoqOzzzA/UHR0x4D2vZuj
/rtGK475atwR0hlbPqAd1nJ8FhQh3XGg9YcipDMOpoCiozsOtJcEUddVce96ihoPhHR211N0eQM1
3bGgfQNzJ4garpGmvOsYCOmMPR+NieHwlqMpQic6OkvnZ0vUDSPuSoN4oqOzu+cDjI7ueNB+KDo6
s8y6iY7Okh65ROUVBeZoCig6OqOqjyJElXah/UsFw+6cuOP7lIgyAKIBVX3Us4WIzrKzWwkDdQJb
oqghojNL+XM4UStwPpoSPskKjMhHP2do6I4H7dfWIa7nJ+do6LIFbRxtgbc+UgRHQ2ds+hABDd1x
oH1TiqOhOw6076Y5GrrnQHvg8Dc7gf2zyw8UDZ1TfZC2Xbx1V1OKUCR0/ktTSpjtFKH9e4WGPQxx
xzeLhnUJced3yeGsQdz1PYzwfSBu+q4LtwLi+jdVhZReccsv9bcgoTsGtHechAhZcY8Y+U5gIZ01
we2b7ULVSeCe2ByKwyZ45MiI6Pxau9EYFd1xoL1jIhRGRLb0OBoiuuNA+y6vIaLznKWEoEBxb5YC
vMRbz8dn4zS2AnM0RWgv4q3XP5fXEdF5zZfXEdEdC9rvxnVUdMeDNsVARuc1ZWFx3IrIaVKJw0CK
fA6nXOrGWRWN2Jaf30BH5y3NKrH3v4RqYNIWHd0xoR2vrS/iplkldvaIyzMSio7OW/qwi40S4uYV
lVjHJ7B/2V2sMSswXvm0RUjnSZBcYtlOQ/Za5avxanAXF1oFjlL2Qnsm+kJJd2xoE4yUzvOiSnzf
EDm98CIlJzLXp8sOoZpg/+amSGgUmOI+6nJs+ggt37wbLwgFTtYpZ/4SFyJXeffNUdJdG1rFdZR0
x4b265ojpXP2fgiBlO740NYXAindsaF948qR0h0b2sfePS9qeen40PpD0dI55imgHfYmh7cSZ+oV
```

Figure 1.3: Base64 Encoded Document

Even if discriminating sub-strings cannot be found in a "base64" encoded document, the identification of such a document is still feasible. Some characteristic can be found in each "base64" encoded documents:

- A 64 characters alphabet is used.
- There is no space.

- There is a line feed at every 76 characters (typically).

For both examples presented, classification could have been made using a regular expression [AU72]:

E-Mail	`^From:.*\nTo:.*\nSubject:`
Base64	`^[0-9A-Za-z\+\/]{76}[\r\n]{3,}`

Table 1.1: Classification Using Regular Expressions

In the current context, we assume that any document class can be identified with a regular expression. Ideally, a regular expression that matches all examples of a certain class would be what the learning algorithm tries to find.

Given the presented context, some obstacles and difficulties will have to be taken in consideration in order to develop the desired system. First, the feature extraction technique used should concentrate on frequent patterns that are proper to the structure of the documents as opposed to the content of the documents. Often, the structural information is located in a header that occupies a limited proportion of the document. Statistical frequencies of those patterns will be diluted with the contextual information. Second, the training documents can be long. This means that the feature extraction will have to be efficient and the extracted patterns will have to be stored using as little memory as possible. Third, in addition to finding discriminating sub-strings, it would be preferable to also identify critical positional relationships of the different patterns in the document. As an example, the sub-string "To:" is between "From:" and "Subject:" in an e-mail. Fourth, an ideal classification model would have to be noise resistant and easily updatable.

A classification model that is easily updateable should be trained with an "incremental" learning technique. With an incremental learning technique, the classification model can be updated with a new training example at any time. A non-incremental training would have to redo training with the original training documents plus the new training document.

The next chapters of this document will propose potential solutions to the previously described obstacles and difficulties.

1.2.2 – Potential Applications

A system that would learn to do structural identification of unintelligible documents from training examples would be useful in many real life applications. The classification capabilities of the system could be used in the following fields:

- **Intelligent routing.** A document could be routed at a different location and with a different priority depending on its format. As an example, a packet containing video information would have to be routed at a higher priority and/or bandwidth than an e-mail.
- **Data-flow systems.** According to its format, a document could be stored in a specific location or a certain process could be applied to it in order to make it more intelligible.
- **Mobile agent queries.** A certain mobile agent could be trained to retrieve a certain type of documents and could be sent over the Internet to analyze every accessible location and accumulate the desired documents.

In theory, a human analyst could build a regular expression in order to perform the classification task. The analyst would have to analyze many documents in order to find some characteristics that

are proper to the format he is trying to identify. In some cases, the regular expression might be hard to find and is not guaranteed to be perfect. Also, this regular expression would not adapt to slight changes that can occur in the data over time. This phenomenon is called "concept drift" [WK98]. In most of the cases, even if it is hard to come up with a regular expression that matches a category of documents, an analyst can easily determine the category of a document just by looking at it. If a learning system could derive identification rules from training documents, it would facilitate the work of the analyst whose only responsibility would be to provide labeled examples to the system. Also, if the learning system is incremental, the analyst would be able to give additional labeled examples to the learning system in order to improve its performance.

1.1– Contribution

A lot of the techniques and algorithms presented in this document are inspired from previous work that has been accomplished by researchers. Many techniques, that were applied on problems that were similar or different to the one that is the subject of this work, were combined, modified and reused in the presented algorithms. Reusing published knowledge to solve a new problem is the beauty of the collective scientific effort.

The related work will be clearly presented in the next chapter and throughout the document. However, it is still important to clearly identify what is our personal contribution to the machine learning community. In chapter 3, the feature extraction techniques that are based on text compression were specially developed for this specific problem. Also, methods for feature pruning in the compression models were developed by us to avoid the exponentially increasing number of features extracted from the huge training set. The combination of frequent sub-strings with

grammatical inference to reduce the number of states in the initial finite state machines presented in chapter 4 was conceptualized to accelerate the induction. The object-oriented framework of chapter 5 was designed for this specific project and it could constitute a basis for future work.

Chapter 2 – Related Work

2.1 – Related Work on Text Classification

This section presents some references to previous work accomplished in text classification. Most of the literature available puts the emphasis on the classification of documents written in natural language. This is normal and justifiable because most of the research is directed towards some applications that are used to recognize or filter documents on the Internet, which are written in natural language. With the growth of the Internet and the increasing volume of information available, the researchers of the 1990's have seen potential applications for the techniques that would help the user to find the information he/she really wants to see. Traditionally, the search of documents was based on simple query mechanisms based on key words matching. However, it is sometimes difficult to describe the interest in a certain category of documents only with key words. This is why machine learning became a key element in document classification. With machine learning techniques, it is possible to induce a document classifier from training examples. Instead of having to specify some classification rules based on key words, the human analyst just has to train the system to recognize documents with training examples.

David Lewis is one of the first researchers to have used machine learning to do text classification [LE92]. The work accomplished by Lewis has become a reference for the other researchers to explore the possibilities of using machine learning on document classification. Lewis did his experiments on a training set that is now known as the 21,578 REUTERS newswire articles. This constitutes a widely used corpus to compare the accuracy of various machine learning techniques on

document classification. However, this corpus is useful to compare classifiers that have been trained to work in the presence of natural language, which is not the case here.

In all the literature available, the challenges of machine learning in document classification can be divided in two parts. The first part is feature extraction. This is related to extraction of the attributes that represent the textual documents. The second part is the induction of the classifier. This last part uses some machine learning techniques to induce a document classifier based on training examples represented by the features previously extracted.

A classical way to extract features was to use a bag-of-words approach. The idea is to count all the word occurrences in the training documents. Some infrequent words included in the bag-of-words will be pruned and not used as features. Ken Lang presented some techniques in [LA95] to use the features in the bag-of-words to perform classification. Those techniques include stemming, tf-idf weighting and minimum description length.

Stemming is a technique that has the effect of mapping several morphological forms of words to a common feature. For example the words "learner", "learning" and "learned" would all map to the common stem "learn", and this latter string would be placed in the feature set rather than the former three [SM99]. This reduces the size of the bag-of-words.

Tf-idf stands for term frequency and inverse-document-frequency. This metric tends to prefer features that occur often and disadvantages features that appear in too many documents. Features that occurred frequently are definitely more interesting, but features that occurred in too many documents do not have a very good discriminating power. A weight will be associated with each

word in the bag-of-words according to the metric tf-idf. During classification, the words of an unseen document will be compared with those weights. A technique like nearest neighbor could be used to do the comparison between the weights. For more details see [LA95].

The minimum description length (MDL) principle is the basis of an alternative learning technique to tf-idf. The MDL principle provides an information-theoretic framework for balancing the tradeoff between model complexity and training error. The MDL principle expresses the intuitive notion that finding a good model involves finding the right balance between simpler models and models that produce smaller error when explaining the observed data. Ken Lang has applied the MDL principle to bag-of-words in [LA95] to classify the relative importance of various pre-labeled newsgroup documents.

Instead of using the bag-of-words as a feature extraction technique, some researchers have considered using word sequences instead of single words. A sequence of n words is known as n-gram. Using word sequences as features might certainly enrich the feature set. As an example, the word sequence "world wide web" has a very special meaning that would not be detected if the words "world", "wide" and "web" were considered separately. Mladenic and Grobelnik in [MG98] and Furnkranz in [FU98] considered word sequences in their work.

In both approaches in [MG98] and [FU98], a pruning mechanism is applied. First, the concept of a stop-list is used to prune some features. A stop-list contains frequent words that are not necessarily discriminating properties. In English, some articles like "at", "the", "for", etc. might be included in the stop-list. The feature extraction algorithm does not consider words included in the stop-list. Second, a term frequency threshold is used to prune infrequent word sequences. The application of

this pruning mechanism is very simple: every observation that does not occur more often than the threshold will be pruned. Third, when each i-gram is being extracted, if the prefix (i-1)-gram or the suffix (i-1)-gram did not occur more than the term frequency threshold, then the i-gram in question will be excluded. In addition to these pruning mechanisms, Furnkranz introduced a document frequency threshold. In order to be considered a feature, a word sequence has to appear in a number of documents higher than the threshold.

After having pruned the features, Mladenic and Grobelnik used a Naïve Bayesian [MI97] learner to train the classifier. Furnkranz used a different technique to induce the classifier. He used RIPPER, a rule-based learner developed by William Cohen [CO95]. RIPPER is described in chapter 4 of this thesis.

To enrich the bag-of-words and the n-grams feature extraction, Scott and Matwin [SM99] have explored feature extraction techniques based on the syntactic and semantic relationships between words. The main idea was to incorporate knowledge about the language to improve and refine the feature extraction process. They have explored feature extraction techniques based on noun phrases and key phrases. Also, they developed a technique comparable to stemming except that the idea is to map words semantically instead of morphologically. The mapping was based on synonyms and hypernyms ("Hypernym" is a linguistic term for the "is a" relationship – a knife is a weapon, therefore "weapon" is a hypernym of "knife"). The synonyms and hypernyms were obtained from a large on-line thesaurus called "WordNet". Like Furnkranz, Scott and Matwin used RIPPER to induce the identification model.

Since the present work does not assume the presence of natural language in the training documents, we are mainly interested in the concept of "character based" n-gram extraction as opposed to "word based" n-gram extraction. The literature describes systems that use character based n-grams. Often, a distinction is made between positional and non-positional n-grams. The notion of a positional n-gram specifies the respective position of each character of the n-gram within a word. It does not necessarily assume that the characters are contiguous, as opposed to non-positional n-grams. In our case, we will mainly be interested in non-positional n-grams. However, the literature describes interesting techniques to store large quantities of n-grams occurrences using a reduced amount of memory. J. R. Ullmann presents a technique called "random superimposed coding" in [UL77] that reduces the memory usage for storing n-grams. Also, Riseman and Hanson present a technique in [RH74] that has the same memory usage objective to store positional binary n-grams.

At the very end of 1999, Fabrizio Sebastiani published a survey on machine learning in automated text categorization [SE99]. This survey is a very good reference that describes a significant part of the machine learning technique used in text classification.

2.2 – Related Work on Grammatical Inference

A lot of theoretical research has been accomplished to date. However, it does not seem like there is a lot of applications published for these theories. We have not found any previous work that applied grammatical inference to textual document classification like we would like to do. However, we believe that there is definitely a strong potential for the use of grammatical inference in text classification.

A good theoretical document written by Pierre Dupont and Laurent Miclet has been published by the "Institut National de Recherche en Informatique et en Automatique" [DM98]. This document is a survey of grammatical inference theories and algorithms. It contains grammatical inference techniques that require only positive examples and some other techniques that require positive and negative examples. The algorithms RPNI and RPNI2, discussed in chapter 4, are among the algorithms presented in this document.

Pierre Dupont and Lin Chase presented an application of probabilistic grammatical inference applied to speech recognition in [DC98]. The objective of their experiment was to evaluate the effect of symbol clustering on probabilistic grammatical inference. In turn, the objective of symbol clustering is to reduce the alphabet in order to obtain a denser training set, which is very sparse in practice. We describe symbol clustering in chapter 3. The probabilistic grammatical inference technique presented in [DC98] did not use negative examples during learning. It only relies on statistical similarities between positive examples. However, it is not proven that probabilistic inference is better than exact inference (using negative and positive examples). In many applied cases, it is difficult to obtain negative examples.

At the end of 1999, Kevin Lang presented two algorithms called EXBAR and ED-BEAM in [LA99]. EXBAR is a heuristic[1] algorithm that guarantees that the inference of the final automaton will be minimum (i.e. using the fewest states) and compatible with the positive and the negative training sets. ED-BEAM is an inexact algorithm that does not guarantee the minimum automaton. However, ED-BEAM is supposedly designed to work efficiently on larger problems. Those algorithms were champions during the "Abbadingo" competition held in 1999. However, it is important to mention

[1] It is proven that the inference of the minimum final deterministic automaton is a NP-hard problem [GL67].

that these results were obtained on a testing domain that is quite different than ours. 810 problems were tested. Each problem is composed of 20 training strings of length 30. A label is associated with each string. Labels were assigned by a Moore target machine containing 4-21 states. The objective of the test is to find the minimum acceptor automaton that fits the training set of each problem. We will see in chapter 4 that our problem domain is much larger than the problem domains used in the "Abbadingo" competition.

2.3 – Related Work on Symbol Clustering

In the present work, we combined symbol clustering techniques with n-gram extraction and grammatical inference. In our experiments, we used symbol clustering as presented by Dupont and Chase in [DC98] (see chapter 3). The idea of the technique is to minimize the training set perplexity of a class bigram [NK93].

Another category of problem, known as the keyboard optimization problem, has a lot of similarities with what we expect from symbol clustering in text classification. The idea is to distribute the 26 letters of the alphabet on a keyboard that has a restricted number of keys (less than 26). The distribution of the characters has to be made in such a fashion where the ambiguity is minimized according to a certain language. As an example, consider the alphabet distribution on the keyboard of a telephone (figure 2.1).

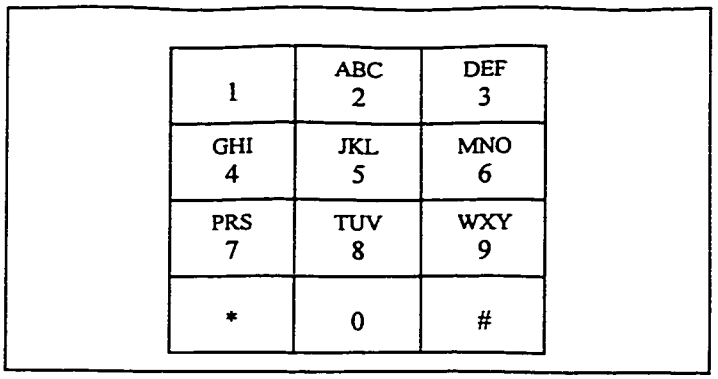


Figure 2.1: Telephone Keyboard

The keyboard of a telephone presents a lot of ambiguities for the English language. For example, the encoding "269" can represent the words "box" or "boy" which are two valid English words. The challenge of the keyboard optimization is to find an appropriate character distribution on the keyboard for the words of a dictionary of a given language. For example, the keyboard presented in figure 2.2 presents fewer ambiguities for the English language.

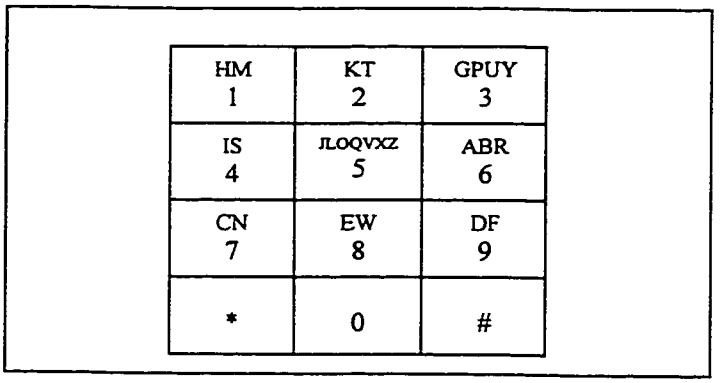


Figure 2.2: Optimal English Keyboard

The keyboard optimization problem addresses the same alphabet reduction issues that interest us in text classification. Solutions to the keyboard optimization problem can be found in [OV92], [LE85], [LM86] and [MI86].

Chapter 3 – Feature Extraction

3.1 – Introduction to Feature Extraction

This chapter presents some techniques and algorithms to extract textual features from documents. The features could be defined as attributes characterizing each document. Basically, a learning algorithm is trained with pre-labeled documents. Each document provided to the learning algorithm is expressed in terms of the values of its features plus a label identifying the class of the document. The role of the learning algorithm is to find the values of the features that will characterize the target concept that has to be learnt.

The main challenge is to identify a set of relevant features that would be appropriate to characterize the training documents. The basic idea exploited in this chapter is to extract frequent sub-strings from document examples of a class. We present four feature extraction techniques. Each technique has a pruning mechanism that attempts to remove infrequent features.

3.2 – N-Grams Extraction

In order to prepare training documents for a classification algorithm, a well-known approach is to associate a feature vector with each training document. In the context of textual documents, the feature vector is composed of binary values representing the presence (feature is TRUE) or absence (feature is FALSE) of a certain sub-string in the training document. Those sub-strings are commonly called n-grams. A 1-gram corresponds to a string of size one, a 2-gram corresponds to a string of size

two and so on. The technique presented in this section is exactly what was presented by Johannes Furnkranz [FU98] with the exception that n-grams are sequences of n characters instead of sequences of n words.

Those n-grams will have to be extracted from the training documents. If we consider a "byte" representation of each character of a document, there are 256 possible distinct 1-grams, $256^2 = 65536$ possible distinct 2-grams, etc. If a document contains N characters, there is exactly N sequences of one character, (N - 1) sequences of two characters, etc. As an example, consider the following document: "Hello".

1-grams for the string "Hello"	"H", "e", "l", "l", "o"
2-grams for the string "Hello"	"He", "el", "ll", "lo"

Table 3.1: N-gram Representation of the String "Hello"

Each document will be expressed with a feature vector representing which n-grams are included in the document in question. Of course, the possible n-grams that could be used as features will have to be *a priori* selected because, relatively to the size of n, there is an exponential number of possible n-grams. In order to determine the relevant features, we describe below an existing algorithm based on the Apriori algorithm presented by Agrawal et al. [AM95].

The idea of the algorithm (Figure 3.1) is to extract frequent n-grams from the training documents of a certain class. The algorithm will be executed for a maximal n-gram size determined by a human user. All the possible n-grams up to the given size will be computed from the training documents. A certain n-gram will be considered a feature if and only if all the following conditions are satisfied:

1. The n-gram occurred a number of times greater than a certain frequency threshold determined by the user (minFrequency).

2. The prefix and the suffix of a n-gram have not been pruned from the (n-1) features set. As an example, the 3-gram "abc" will be considered if it satisfies conditions 1 and 3, and if the 2-grams "ab" and "bc" were previously considered as features.
3. The n-gram appears in at least a certain number of training documents. The user will determine this threshold (minDocFrequency).

```
Procedure GenerateFeatures(Documents, MaxNGramSize, MinFrequency, MinDocFrequency)

    Features[0] = {}
    For n=1..MaxNGramSize
        Candidates = {}
        DocumentOccurence = {}{}
        Features[n] = {}
        For each Doc ∈ Documents
            For each N-gram ∈ Ngrams(Doc,n)
                Prefix = firstCharaters((n-1),N-Gram)
                Suffix = lastCharacters((n-1), N-Gram)
                If (Prefix ∈ Features[n-1] AND Suffix ∈ Features[n-1])
                    Counter{N-Gram} = Counter{N-Gram} + 1
                    Candidates = Candidates ∪ N-Gram
                    DocumentOccurences[Doc][N-Gram] = 1
        For each N-Gram ∈ Candidates
            If (Counter{N-Gram} >= MinFrequency AND
              SumOf(DocumentOccurences[*][N-Gram]) >= MinDocFrequency)
                Features[n] = Features[n] ∪ N-Gram
    Return Features
```

Figure 3.1: Apriori Like Algorithm to Generate N-Gram Features

This algorithm extracts the features that will correspond to each boolean value of the feature vector associated with each document that will be used to train the data classification system. It is important to specify that the documents used to extract the features do not have to be the same documents as those that will be used to train the data classification system. Extracting the features and training the classification system are two distinct operations. Also, it would be useful to be able to keep the extracted features persistently in order to reuse them to build a feature vector for a document that was not necessarily a part of the feature extraction training set.

In order to achieve this requirement, the n-grams will be moved from a hash table to a tree structure. The features will initially be stored in a hash table. Since the features are accessed frequently during the feature extraction process, it is important to store them into hash tables for quick lookups. However, hash tables have to utilize more space than the memory occupied by the data it really contains in order to reduce collisions. On the other hand, the tree structure uses less memory and is suitable for the detection of multiple patterns. In order to take advantage of both data structures, features will be transferred from the hash table to the tree structure as soon as the algorithm does not need to refer to them. In fact, the algorithm does not need to lookup the features of size (n-2) when pruning features of size n. At the end of the algorithm, the tree contains all the features for each n-gram size. This tree will be stored persistently and will be used to build feature vectors on new documents. In the feature tree, each n-gram corresponds to a branch. If many patterns share a common prefix, they will also share the corresponding edges in the tree. This structure reuses memory and accelerates lookups in the tree. Each node ending a branch that corresponds to a feature will be flagged as being a feature. The example of figure 3.2 illustrates the tree structure.

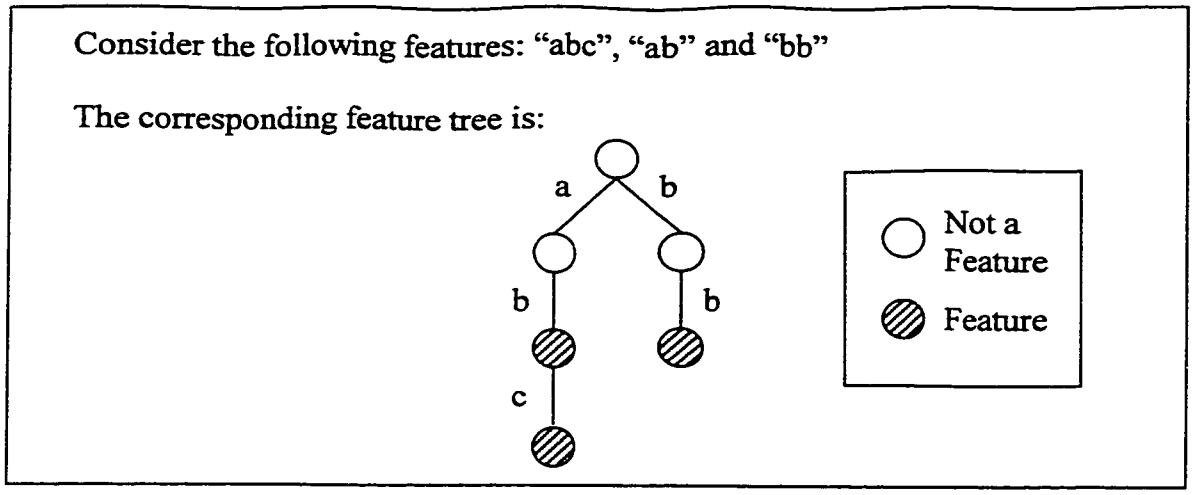


Figure 3.2: An Example of a Tree Structure

This data structure is very suitable for pattern lookup in logarithmic order as described by Donald E. Knuth [KN73].

3.2.1 – Advantages

The main advantage of this feature extraction technique is its exhaustiveness. In fact, all the possible sub-strings of size less or equal to n that appear in the training documents will be considered. A vast variety of features will be associated with each training document leaving a lot of choices to the learning algorithm to find the series of discriminating features. Also, a simple pruning mechanism (in pseudo-code of figure 3.1) allows the user to reduce the number of features produced, which is necessary in many cases.

3.2.2 – Disadvantages

Many aspects of this feature extraction technique could be improved. One of the inconveniences of the presented algorithm is the fact that pruning relies on integer thresholds determined by the user. This pruning mechanism does not necessarily respect theories of probabilities and statistics. As an extreme example, imagine a user fixes a minimum occurrence ratio of ten. The importance of this threshold will vary depending on the size of the training set. If the training set has a million characters, the threshold of ten is a lot more permissive than if the training set would have a hundred characters. Also, the same threshold will be applied to all sub-strings regardless of their size. Statistically speaking, the probability that a long sub-string occurs a certain number of times is lower than a shorter sub-string. The threshold should be adapted relatively to the size of the sub-string being tested for pruning.

Another disadvantage is that the user has to determine the maximal n-gram size. This may seem unimportant, but it affects the performance and the accuracy of feature extraction. The user has to make his best guess for this maximal n-gram size. If the size is longer than the maximal size of the actual discriminating pattern, then the feature extraction process will do useless work extracting sub-strings that are unnecessarily long. On the other hand, if the user fixes a maximal n-gram size that is too short, interesting discriminating patterns can be missed.

Finally, the time and space requirements are demanding. Since we do multiple passes over the training set, the time is in the order of the size of the training set multiplied by the maximal n-gram size. However, an alternative solution to this problem was proposed by Agrawal[AM95] that keeps in memory positions of previously non-pruned features, allowing the algorithm to do only one pass

over the training documents. Of course, the price of this benefit has to be paid with the memory necessary to store all the pointers for each feature. According to Furnkranz [FU98] multiple passes over the training documents payoff if the number of features is large and storing pointers begins to payoff when the feature sets become small. In the context of the presented work, the number of features will be larger since we work with n-grams based on characters instead of words. Also, the memory used by the hash tables during feature extraction is considerable. Using a tree structure to store the features as soon as they do not need to be referred by the algorithm reduces memory usage. However, a certain processing time is required to transfer the features from the hash tables to the tree structure.

3.3 – Benefits of a Reduced Alphabet

When reading a document, a certain convention has to be known in order to represent each symbol read. In modern computers, the smallest unit that could be interpreted is the bit. However, standards exist to represent the alphabetic characters in terms of bits to facilitate the communication of textual documents. These standards are known as character sets. As an example, the widely spread extended ASCII code (or Latin-1) is used to represent most of the characters from Latin languages plus some control characters, various symbols, digits and punctuation. The extended ASCII code maps each sequence of 8 bits to a certain character. Some other character sets may use a different number of bits as the smallest unit. For example, UNICODE maps a sequence of 16 bits to a character.

In the specific context of the current work, the presence of any natural language is not assumed. This implies that the encoding with a character set known in advance is even less guaranteed. Moreover, maybe the intelligible information contained in the document is not even textual! If no character set

can be assumed, what will be the alphabet used to interpret the training documents? An obvious and easy solution would be to interpret each document as a bit stream. Since the bit is the smallest unit understood by a computer, it could be possible to represent all the discriminating features with a sequence of bits. In theory it is possible and logical. However, it is practically a difficult task. The feature extraction technique would have to work on documents that contains a lot more symbols, and the significant features would be a lot longer. As an example, imagine the Apriori algorithm from section 3.1 used with a bit representation instead of an extended ASCII representation. The algorithm would have to extract n-grams on documents that have 8 times more symbols. In addition, if the document is interpreted as a stream of bits, n-grams will not be extracted based on an 8-bit boundary. Consequently, a n-gram could include as its first bits the last bits of an ASCII character, and as its last bits, the initial bits of the next ASCII character. For these reasons, a larger number of n-gram occurrences will have to be counted in the case of a bit representation versus an ASCII representation.

The solution that was adopted for the current problem was to use the extended ASCII alphabet with an alphabet reduction if needed. A reduced alphabet is an alphabet mapping in which the number of symbols of the target alphabet is smaller than the original alphabet. It could be defined as follow:

Let Σ be the original alphabet and Σ' be the target alphabet.

A mapping g is a function that maps a character from Σ to Σ':

$$g : \Sigma \rightarrow \Sigma'$$

We are dealing with a reduced alphabet when:

$$|\Sigma'| < |\Sigma|$$

Similarly Σ^* defines any sequence of characters from Σ.

Consequently, we can define the function g* that maps a sequence of characters from the original alphabet to the target alphabet:

$$g^*: \Sigma^* \rightarrow (\Sigma')^*$$

For example, assume the original alphabet is the extended ASCII alphabet. Then $|\Sigma| = 256$.

The following mapping is defined:

$$g(c) = \begin{cases} U \text{ if c is an upper letter} \\ L \text{ if c is a lower letter} \\ D \text{ if c is a digit} \\ C \text{ if c is a control character} \\ S \text{ if c is a space} \\ T \text{ if c is a tab} \\ P \text{ if c is a punctuation character} \\ E \text{ if c is an end of line character} \\ N \text{ otherwise} \end{cases}$$

The cardinality of the target alphabet is:

$$|\Sigma'| = 9$$

The value of g*("211 Bank Street") is equal to "DDDSULLLSULLLLL" .

Using a reduced alphabet may improve the relevance of the features in many cases. In some circumstances, if a standard alphabet is used, it is almost impossible to find a discriminating pattern that would be included in all the training documents of a certain class. As an example, imagine a document that would be associated with the class: "List of prices" (Figure 3.3).

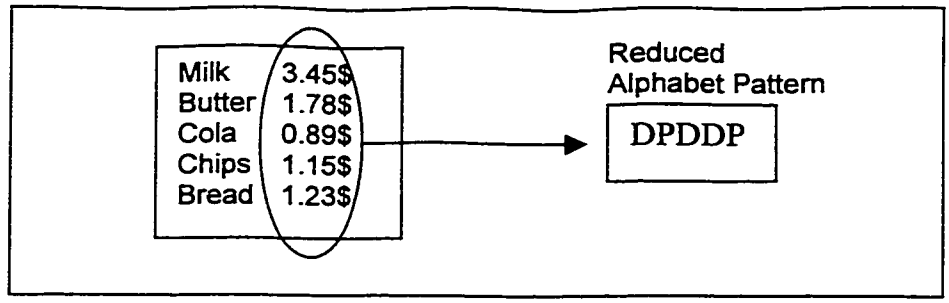


Figure 3.3: Reduced Alphabet Pattern on a List of Prices Example

It is almost impossible to find a characteristic pattern that would be proper to many of the lists of prices, since they are not necessarily composed of the same items and the same prices. However, if a reduced alphabet is used, it would be possible to find patterns that are characteristic of many lists of prices. Like in the example, if the reduced alphabet previously presented is used, it is possible to identify the characteristic pattern "DPDDP" that corresponds to the standard price notation expressed with a dollar sign.

In general, a reduced alphabet tends to improve feature extraction on "sparse" data. However, precision can be lost with a reduced alphabet. In some applications, it would be desirable to take the "best of both worlds" and generate a feature vector with a standard alphabet, and another one with a reduced alphabet.

Another advantage in favor of a reduced alphabet is the effect of generalization that it provides. The fact that a subset of characters is mapped to a single character creates the effect of generalization. A reduced alphabet will be more resistant to noise since a character that was not even observed during feature extraction can be valid when mapped in a n-gram of the reduced alphabet.

So far, the benefits of a reduced alphabet have been described. However, an important question remains unanswered: "How do we build the mapping function without previous knowledge of the data?" A user that has a certain prior knowledge of the data could define a mapping function. The mapping function could also be a general intuitive mapping like the one presented in the previous example. However, the ideal system would have to depend only on training documents. The next section presents a symbol clustering technique that attempts to automatically define a new reduced alphabet based on training examples.

3.3.1 – Symbol Clustering

This section describes a symbol clustering algorithm that infers a certain alphabet reduction from training documents. The user determines the desired number of target symbols of the reduced alphabet. The presented algorithm was previously applied by Pierre Dupont and Lin Chase [DC98] and was originally created by H. Ney *et al.* [NK93]. This particular technique is one of many published approaches for symbol clustering. It worked well when applied on the current problem. The idea of the algorithm is to minimize the training set perplexity of a class bigram. In order to do a clear description of this approach, the concept of a class bigram and the perplexity measure need to be defined.

A class bigram is defined assuming that a certain symbol to symbol mapping (noted $g : x_i \rightarrow g(x_i)$) exists. The class bigram is in fact a conditional probability defined as:

$$p(x_i \mid x_{i-1}) \stackrel{def}{=} p(x_i \mid g(x_i)) \cdot p(g(x_i) \mid g(x_{i-1}))$$

This notion illustrates the probability to predict the next character (x_i) knowing a certain preceding character (x_{i-1}) and assuming a certain mapping g. One of the objective of the algorithm will be to

maximize this probability. In detail, maximizing $p(x_i \mid g(x_i))$ encourages small clusters and maximizing $p(g(x_i) \mid g(x_{i-1}))$ encourages the prediction power of the mapping.

The perplexity is a measure that helps evaluate the relative quality of several hypotheses. In our case, different hypotheses correspond to the different possible mappings. The sample perplexity will be evaluated according to the training set S. The sample perplexity is defined as follows:

$$2^{\left(\frac{-1}{\|S\|}\sum_{i=0}^{\|S\|}\log_2 P_A(x_i \mid q^i)\right)}$$

In this generic statement, $\|S\|$ corresponds to the number of symbols in the training set. Initially, this formula was developed in a finite state machine context. The expression $P_A(x_i|q^i)$ represents the probability of observing x_i knowing that the current state of the state machine A is q^i. In our case, the state machine is not used but we could translate the concepts to our problem. The current state of the state machine is equivalent to the last symbol read from a document. Knowing that a certain symbol has been read, $P_A(x_i|q^i)$ represents the probability of predicting the next symbol in the document.

This general definition of the sample perplexity will be adapted to represent the training set perplexity of a class bigram as follow:

$$2^{\left(\frac{-1}{\|S\|}\sum_{i=0}^{\|S\|}\log_2 (p(x_i \mid g(x_i)) \cdot p(g(x_i) \mid g(x_{i-1})))\right)}$$

Figure 3.4 presents an algorithm that attempts to find a mapping that will minimize the training set perplexity of a class bigram. The idea of the algorithm is to gradually explore the possible mappings by moving the characters from cluster to cluster. The algorithm is separated in many iterations. After

each iteration, the mapping that gives the lowest perplexity over the training set will be selected and used for the next iteration. The algorithm stops (statement notEnd in figure 3.4) after a certain predefined number of iterations (e.g. 100) or when the gain between two consecutive perplexities is relatively small (e.g. less than 0.01).

```
Procedure Cluster(TrainingDocuments, NumberOfTargetSymbols)

    G = initializeMapping(TrainingDocuments, NumberOfTargetSymbols)
    While (notEnd)
        For all character a ∈ original alphabet
            PP = {∞,∞,...,∞}          // size of PP = NumberOfTargetSymbols
            For j = 1 to NumberOfTargetSymbols
                G(a) = cluster-j
                PP[j] = calculatePerplexity(TrainingDocuments, G)
            k = argmin PP
            G(a) = cluster-k
    Return G
```

Figure 3.4: Clustering Algorithm

Assuming that k is the number of target symbols, initializing the mapping is done by assigning the (k-1) most frequent characters of the training set to the (k-1) first clusters. The remaining characters are assigned to the last cluster.

The execution of this algorithm provides a reduced alphabet in which the predictability of the symbols is maximized. As an example, consider an execution of this algorithm on an English text with two target symbols. The expected results would probably be composed of two clusters (mapping of many characters to a target symbol) containing respectively vowels and consonants. In the English language, a vowel is often followed by consonant and vice-versa.

3.4 – Feature Extraction Based on Text Compression

A text compression category called "adaptive dictionary encoders" is based on the idea of replacing a repeated string by a reference to a previous occurrence [BC90]. Jacob Ziv and Abraham Lempel first published a text compression technique based on this idea in 1977 [ZL77]. The goal of their algorithm is to compress the memory space occupied by a text by replacing redundant strings by a pointer to the position were the string was first seen in the text (Figure 3.5).

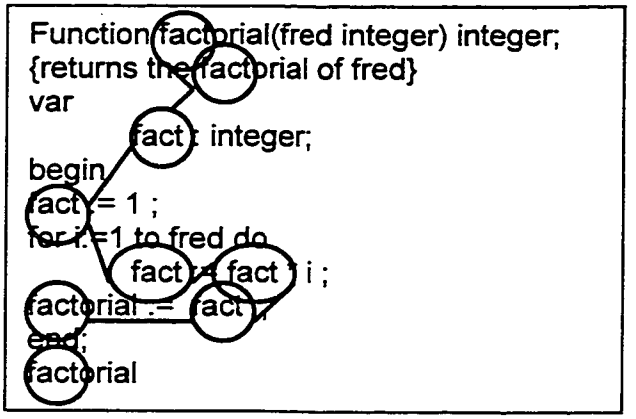


Figure 3.5: Principle of the Ziv-Lempel Text Compression

Of course, the objective of our work is not to compress text. However, some important characteristics of this family of algorithms can be very interesting from a "feature extraction" point of view. First of all, the adaptive dictionary encoders have the objective to "eliminate" redundancy in order to compress text. In our case, we want to "detect" what is redundant for the purpose of identifying the recurring and discriminating features. After having built the compression model as described by Ziv and Lempel, counting the number of pointers that refer back to a certain sub-string indicates the occurrence frequency of this particular sub-string. Another property of text compression that is important for feature extraction is the fact that the compression model contains the same amount of information as the original text. It is always possible to reconstruct the original

text from the compression model. This means that all the "pieces" of a text are somehow contained in the compression model. It implies that no part of the text will be excluded from the feature extraction analysis. Finally, compression model occupies, by definition, a reduced amount of memory. This characteristic will become very useful when extracting features over enormous training sets.

The next section will present in detail the text compression technique known as LZ78 that was created by Ziv and Lempel [ZL78]. This technique is based on the construction of a tree structure containing unique sub-strings that are the atoms of the text compression (i.e. sub-strings that compose the original text). As it was discussed in section 3.1, the tree structure has many useful properties for feature extraction.

3.4.1 – The LZ78 Text Compression Technique

The LZ78 text compression algorithm sequentially reads all the characters of an entire document. As the algorithm reads a file, the sequence of characters read from a starting position is a potential candidate for the compression model. The algorithm tries to express this sequence of characters in terms of a sub-string that previously occurred (i.e. a member of the compression model). When it is not possible to express the sequence of characters read in terms of a previous sub-string, the sequence in question will become a new member of the compression model. A new member of the compression model is always created from another member plus one character. After a new member is created, the algorithm resets its starting position from which it will read the document and repeats the operations until the end of the file is reached (Figure 3.6 for Pseudo-Code).

```
Procedure LZ78Compression(aDocument)

        aTree := Tree.new();
        evaluationState := aTree.getRoot();
        For each char ∈ (sequential characters from aDocument)
                aChild := evaluationState.findChildForCharacter(char);
                if (aChild == nil)
                        evaluationState.addChildForCharacter(char);
                        evaluationState := aTree.getRoot();
                else
                        evaluationState := aChild;
        Return aTree
```

Figure 3.6: LZ78 Compression

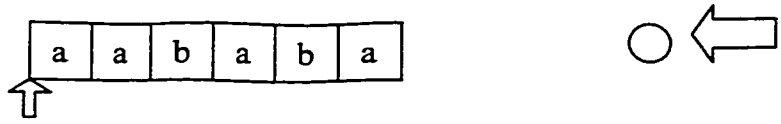
For the LZ78 algorithm, the sub-strings of the compression model correspond to the nodes in the tree structure. Every time a new member is added a node is added at the end of a branch. The following example illustrates the idea of the LZ78 compression algorithm:

Example:

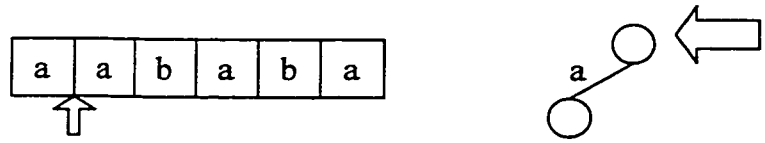
Consider the original document: "aababa".

In the following figures, the upward arrow represents the reading position in the document. The initial position is equal to zero, i.e. before the first character. The leftward arrow represents the evaluation-state.

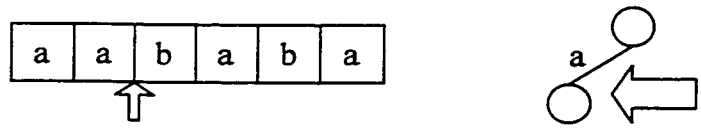
The original compression model is a tree with a single root:



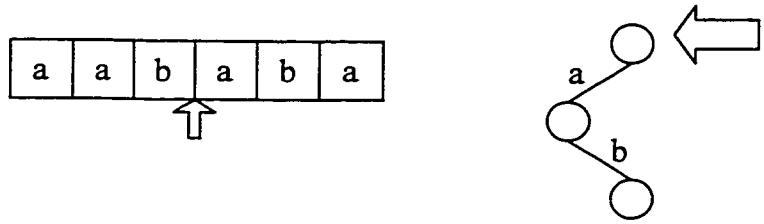
The first character read from the document will be "a". Since the compression model is empty, it is impossible to express the sub-string "a" in terms of a previous sub-string. Therefore, the root of the compression model will be extended with the character "a" and the tree will be reevaluated from the root for the next iteration:



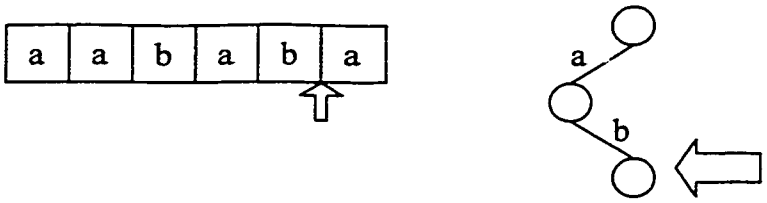
The next character read is "a". Since an "a" occurred before, the next evaluation-state will be the state that represents the string "a":



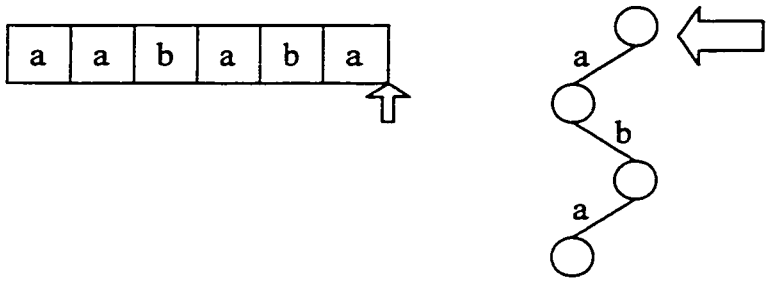
The next character read is "b". Since no node represents the string "ab", the evaluation-state will be extended with the new character "b". Because we created a new node, the evaluation-state will be reset to the root.



The next two characters read will be "ab". Because they occurred before in the compression model, the evaluation-state will be set to the node corresponding to "ab":



The final character "a" will be added to the evaluation-state since the sub-string "aba" is not included in the compression model:



When the end of the document is reached, the final tree structure contains the sub-strings "a", "ab" and "aba". The original document can be expressed in terms of those sub-strings.

In the feature extraction perspective, each node of the final compression model will be a candidate feature. However, a pruning mechanism has to be available to keep only the features that are significantly frequent. The general procedure to extract features from training documents using a text compression algorithm will be the following:

1. Build a compression model over the training documents.
2. Count the occurrences of the sub-strings of the compression model occuring in the training documents.
3. Prune features that are not considered frequent enough from the compression model.

A question that might be asked at this point is: "Why counting the sub-strings after having built the compression model?" It is true that operations one and two could be combined. While doing the text compression, a counter could be incremented every time a node is used in the tree structure. However, this counting method would miss some of the sub-strings included in the final model (Figure 3.7).

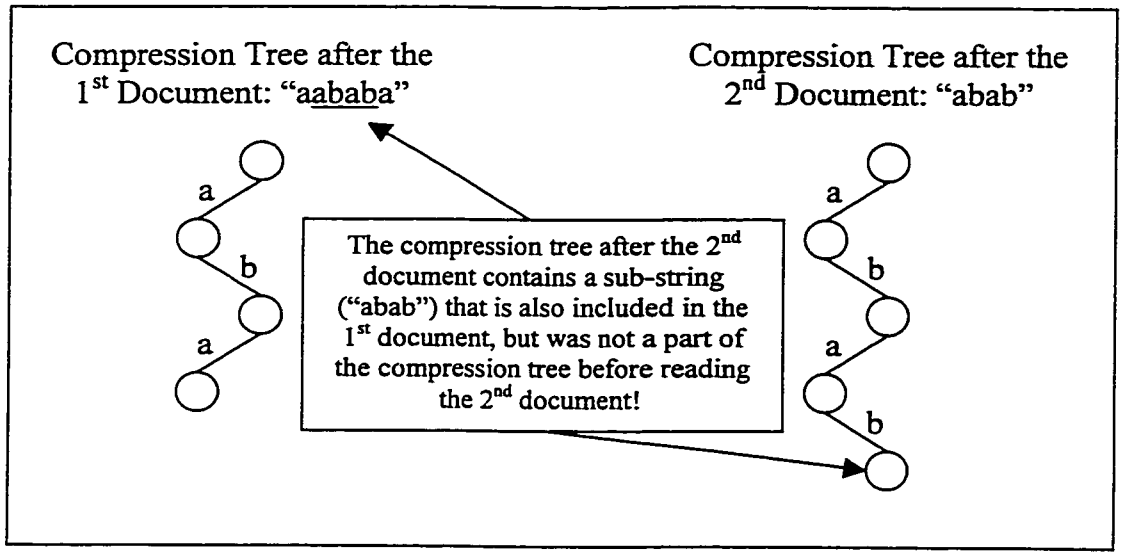


Figure 3.7: Why Counting Sub-Strings when the Compression Model is Complete.

Because of the way the compression algorithm works, the compression model is extended with new and longer sub-strings according to the amount of text compressed. Consequently, at the end of the compression over all the training documents, the compression model will contain a lot more and longer sub-strings compared to what it had after having compressed the few initial characters. It is possible that sub-strings that were lately added to the compression also appeared in the initial documents. This is why it is preferable to do step one and two one after the other. The next two sections present two different approaches to count and prune the sub-strings from the compression model.

3.4.2 – The Complete Covering Approach

The first step of this approach will be to express the training documents in terms of the sub-strings contained in the compression model. More precisely, we want to find a sequence of sub-strings that will be exactly equal to the training documents when concatenated (i.e. "covering" the training documents). Consider the following example:

Assume that the compression model contains the sub-strings "a", "ab" and "aba", and consider the original string "aababa". There are many possible ways to express the original string in terms of the compression model:

6 sub-strings	"a", "a", "b", "a", "b" and "a"
5 sub-strings	"a", "ab", "a", "b" and "a"
4 sub-stings	"a", "ab", "ab" and "a"
3 sub-stings (optimal)	"a", "ab" and "aba"

Table 3.2: Possible Covering of the String "aababa"

The complexity of this task increases relatively with the size of the training documents and the size of the compression model. First of all, each training document has to be expressed as a concatenation of the sub-strings from the compression model. Just to find a possible solution on a large document is quite a challenge. However, it is guaranteed that a solution exists because the compression model was built from the same training documents as the ones that are used for the covering task. Once a solution is found, we have to count the sub-strings that were used when covering the training documents. Moreover, for each sub-string, the number of documents in which it appears will be counted. To perform the covering task, a depth-first state exploration strategy will be used. Each state is described as follows (Figure 3.8): 1) A pointer to a training document. 2) A position within the training document. This position is updated after having applied a transition that leads to the state

3) A pointer to the parent state 4) A reference to the compression model node that represents the sub-string used to reach the state (called "treeNode")

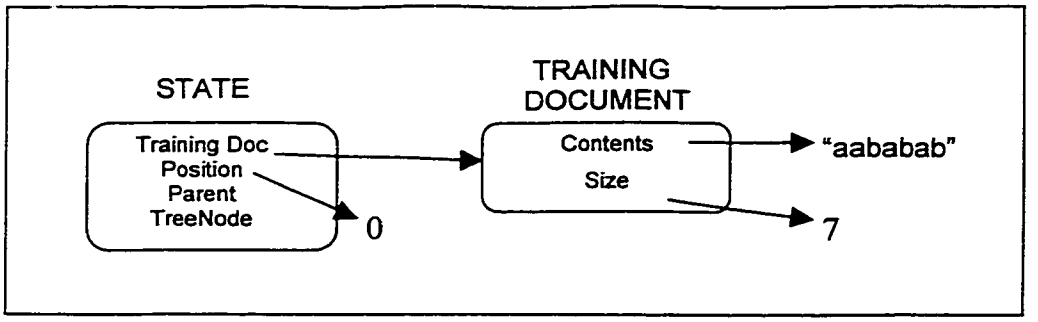


Figure 3.8: State for Depth-First Exploration

The goal-state has a position equal to the size of the training document. The initial state has a position equal to zero. The exploration strategy starts with the initial state, generates successor states and explores them in a depth-first manner. The search stops when the goal-state is reached. When a successor state is created, it still points to the same training document as its predecessor, but the position will be updated. The position of each state depends on the sub-string previously chosen by the operator that lead to the state in question. All the operators correspond to the choice of a certain sub-string from the compression model. An operator applied to a certain state will generate another state whose position is the position of its predecessor plus the size of the sub-string associated with the operator (Figure 3.9). A state is considered equal to another state if they have the same position. This property will be used to avoid cycles and to prune the search space.

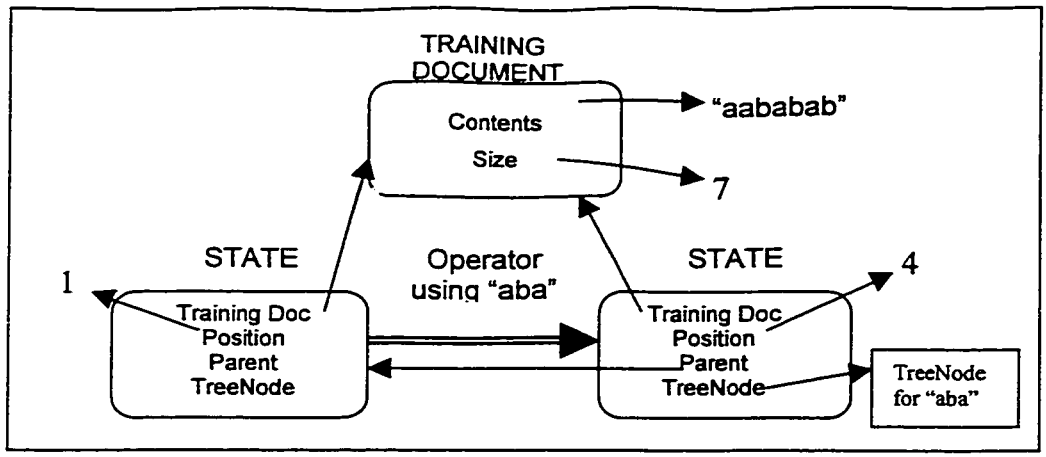


Figure 3.9: Example of a State Transition

The operators that could be applied to a certain state depend on the compression model and on the position of the state. Only the sub-strings that match the original training document starting from the position of the state will be legal operators for the state in question.

The following example (Figure 3.10) describes a complete state space exploration. Note that the number in each state represents its position:

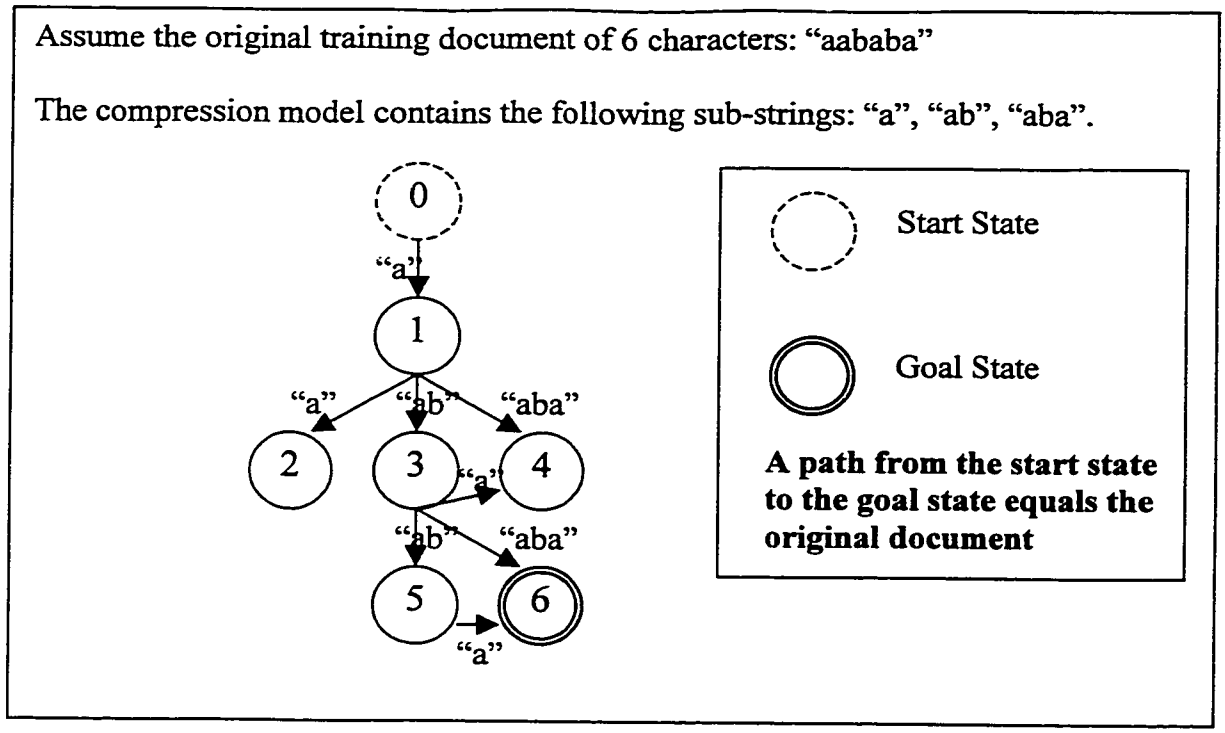


Figure 3.10: An Example of a State Exploration

The preceding example showed a complete exploration strategy in which there are two possible solutions. For the current project, the depth-first search will be used to reach the goal-state. When there are many possible candidates for further exploration, the preference will be given to the state with the highest position. Heuristically, this choice will lead the exploration strategy to the goal more rapidly. Also, this will encourage the generation of a solution that uses longer sub-strings. When the goal-state is reached, the solution is obtained by backtracking the sequence of state transitions up to the root. Each sub-string used as a state transition corresponds to a node in the compression model. For each of those nodes, a counter will be incremented signifying that the sub-string it represents has been used. In addition, each node contains a set of references to training documents (Note that a set cannot contain two identical elements). When the covering is performed

on a training document, the document in question will be added to the set of each node that is part of the solution. This set will be used to determine the number of documents in which a sub-string appears (Figure 3.11). Figure 3.12 describes the complete pseudo-code of the covering approach.

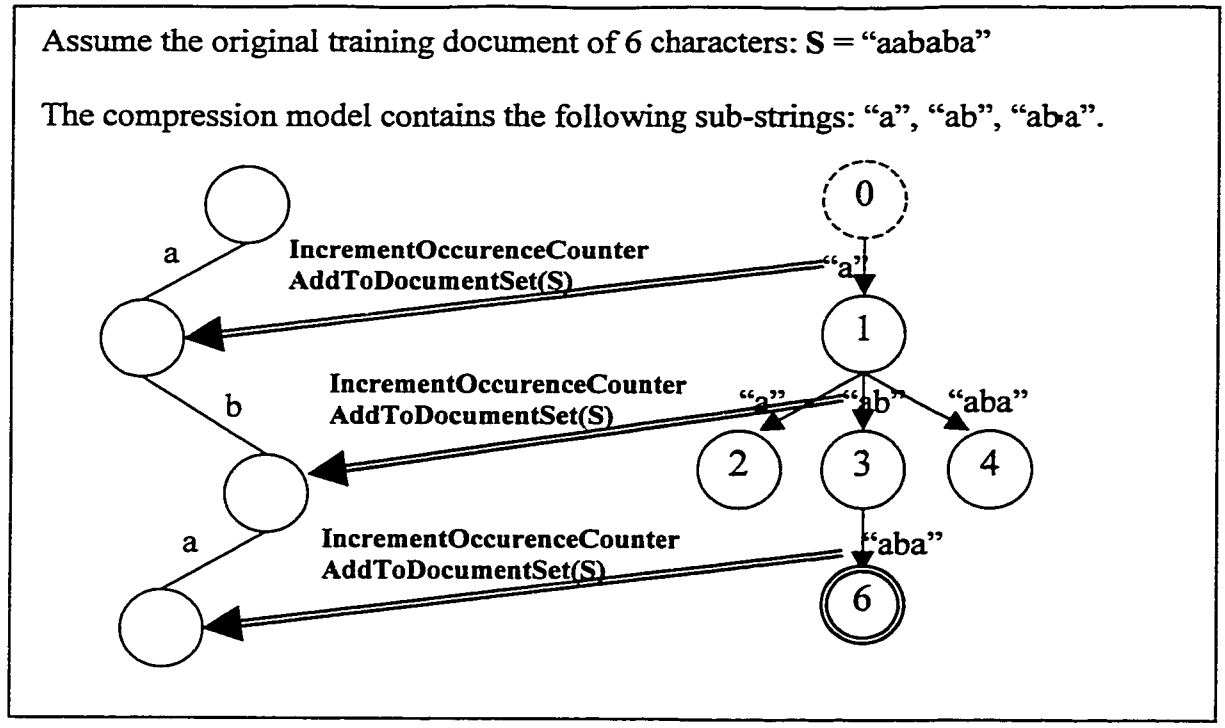


Figure 3.11: Derivation of the Solution

```
Procedure Covering(aDocument, aCompressionTree)

        initialState := State.newOn(aDocument);
        Return (DepthFirstCovering({initialState}, aDocument, aCompressionTree))

Procedure DepthFirstCovering(statesToExplore,aDocument, aCompressionTree)

        While (statesToExplore.notEmpty())
                aState := statesToExplore.removeLast();
                If (aState.isGoal())
                        Return (generateSolution(aState,aDocument))
                successors := aState.generateSuccessors(aCompressionTree);
                For each aSuccessor ∈ successors
                        statesToExplore.addLast(aSuccessor);
        Return "Error No Solution!"


Method State >> generateSuccessors(aCompressionTree)

        successors := {};
        /* possibleNodes contains a collection of nodes from the compression tree
        representing sub-strings that matches the current document at the current position,
        ordered from the  shortest to the longest sub-string*/
        possibleNodes := aCompressionTree.findPossibleSubStrings(this.getDocument(),
                                                                this.getPosition());
        For each aNode ∈ possibleNodes
                newState := State.newOn(this.getDocument());
                newState.position := this.getPosition() + aNode.getDepth();
                newState.correspondingTreeNode := aNode;
                newState.parent := this;
                successors.addLast(newState);
        Return successors

Procedure GenerateSolution(aState, aDocument)
        tempState := aState;
        do
                tempState.getCorrespondingTreeNode().incrementCount();
                tempState.getCorrespondingTreeNode().addDocument(aDocument);
                tempState := tempState.getParent()
        until (tempState != nil)
```

Figure 3.12: Pseudo-Code of the Covering Approach

After having performed the LZ78 compression and run the depth-first search exploration strategy, each node of the compression model contains the number of occurrences of the string it represents and a set of documents in which it appeared. The information contained in each node will be used during the pruning process.

The pruning algorithm will select some nodes as being features and will prune the nodes that are not part of a branch containing a feature. A node will be marked as being a feature if the sub-string it represents respects the following criteria:

1. The occurrence of the sub-string of length "l" divided by the total number of characters in the training set is greater or equal to the value of a certain pruning function p(l) (described in the next paragraph).
2. The sub-string has a length "l" greater or equal to a minimum length determined by the user, and, occurred in at least a certain number of documents. A user will determine the minimum number of documents in which a sub-string must occur.

The pruning function (noted p(l)) has the length of the sub-string as parameter (Figure 3.13). The value of this function represents a minimum acceptance threshold. This acceptance threshold will be lower for a longer sub-string, since the probability of observing many occurrences of a longer sub-string is lower than the probability of observing the same number of shorter sub-string.

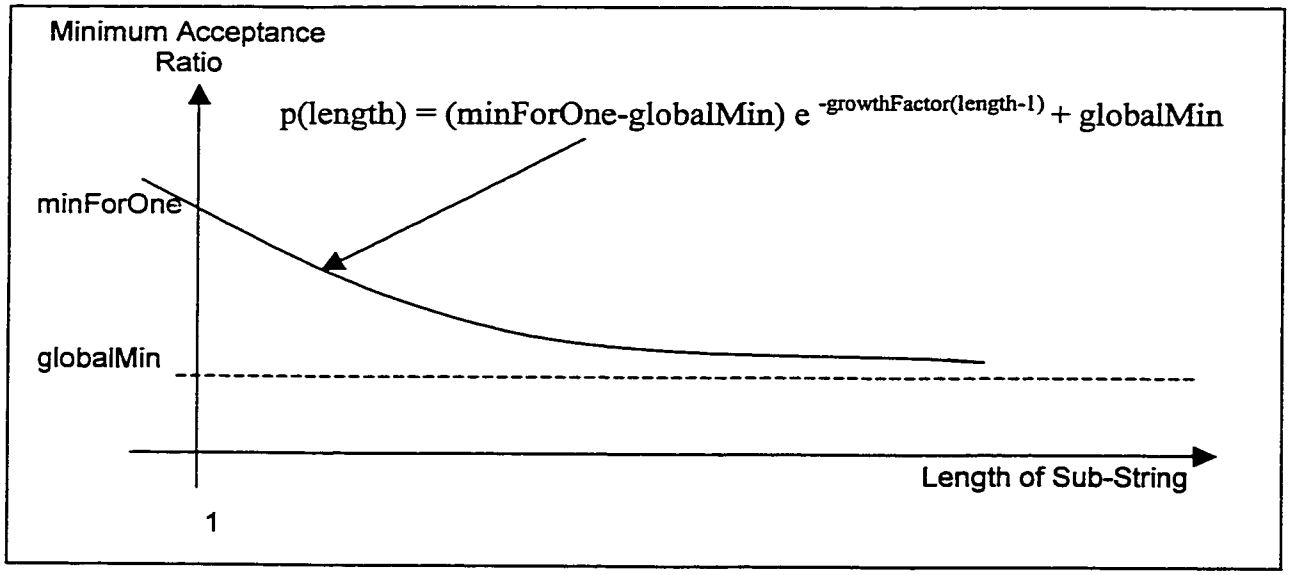


Figure 3.13: Pruning Function

The shape of the pruning function will depend on the following parameters:

MinForOne: This is the minimum acceptance ratio for strings of size 1. If MinForOne = 8%, a single character will be considered a feature if its frequency in the character distribution is at least 8%.

GlobalMin: The minimum acceptance ratio will never be lower than GlobalMin regardless of the size of the sub-string.

GrowthFactor: This value represents the importance given to longer sub-strings versus shorter sub-strings. The higher this value, lower will be the minimum acceptance ratio for longer sub-strings.

So far, the criteria to accept a node as a feature have been described. Now, a method to apply the pruning to the compression model has to be defined. At the end of pruning, the compression model will be a tree containing only the features as described in section 3.1. The idea of the pruning algorithm is to read the tree in post-order (starting from the leave) and prune progressively the nodes that do not satisfy the criteria. After having read a node in post-order, its statistical information (i.e. occurrence counter and document set) will be passed on to its parent. Also, the statistical information of the node in question will be added as a child of the root. The statistical information of a certain sub-string of size n is simply distributed to its prefix sub-string of size (n-1) and to its last character. As an example, if the sub-string "bio" is not considered frequent enough, the sub-string can be decomposed in the sub-strings "bi" and "o" that might be frequent enough. It is possible to do that since the sub-strings that were counted by the exploration did not overlap (this is why this method is called "complete covering"). By distributing the statistical information, the ratio (sub-string occurrence / number of characters) is always valid for the nodes that have not been visited. In fact, if you calculate the sum of all the counters of the node that have not been visited multiplied by the sub-string size they represent the result will be equal to the total number of characters of the training set.

A node distributes its statistical information (Figure 3.14) to the nodes representing its prefix sub-string and its suffix character as follow: 1) its occurrence counter is added to the receiver nodes, 2) its set of documents is combined with the receiver nodes.

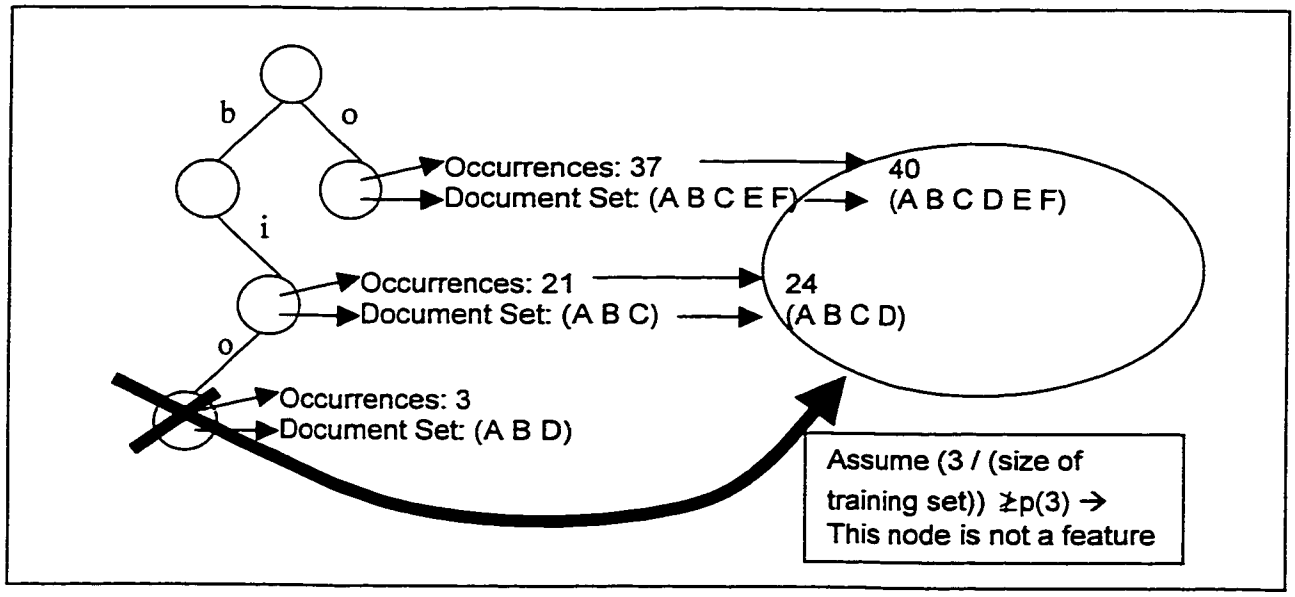


Figure 3.14: Distribution of the Statistical Information

A node will be removed if and only if it is not considered a feature and it does not have any descendants that are considered a feature. This rule will be applied to avoid pruning a branch that contains features. As an example, if the sub-string "bio" is considered a feature, pruning the node corresponding to "b" would also eliminate the sub-string "bio". Also, an important detail needs to be mentioned: when reading the tree in post-order, the reading has to be done in two steps. First, read the tree in post-order up to level 1, and then read the level 1 of the tree. This has to be respected since sub-strings of size 1 can be added to the root during the pruning of the deeper levels of the tree. The pseudo-code of the pruning algorithm is presented in figure 3.15.

```
Procedure Pruning(aCompressionTree, minimalLength, minDocOccurence, pruningFunction)

        For each aChild ∈ aCompressionTree.getRoot().getChildren()
                RecursivePruning(aChild, FALSE, aCompressionTree, minimalLength,
                                minDocOccurence, pruningFunction);
        For each child ∈ aCompressionTree.getRoot().getChildren()
                /* For all children of level 1, prune or mark as feature*/

Procedure RecursivePruning(aNode, & thereIsAFeatureDeeper, aCompressionTree,
                        minimalLength, minDocOccurence, pruningFunction)

        thereIsAFeatureDeeper := FALSE;
        For each (character,aChild) ∈ aNode.getChildren()
                boolean := FALSE;
                (childCount,childDocumentSet) := RecursivePruning(aChild, boolean, aCompressionTree,
                                                minimalLength, minDocOccurence, pruningFunction);
                thereIsAFeatureDeeper := thereIsAFeatureDeeper OR boolean;
                aNode.count := aNode.count + childCount;
                aNode.documentSet.addAll(childDocumentSet);
                aCompressionTree.getRoot().addChild(character,childCount,
                                                childDocumentSet);
                if(!boolean)
                        aNode.removeChildForCharacter(character);
        if(
        (aNode.count / aCompressionTree.totalNumberOfCharacters) >= pruningFunction(aNode.depth) AND
        ( aNode.depth >= minimalLength AND aNode.getDocumentSet().getSize() >= minDocOccurence))
                thereIsAFeatureDeeper := TRUE;
                aNode.markAsFeature(TRUE);
                Return nil;
        Else
                ANode.markAsFeature(FALSE);
                Return (aNode.count, aNode.documentSet)
```

Figure 3.15: Pseudo-Code of the Pruning Algorithm

To recapitulate, the following summarizes the complete covering approach for feature extraction. First of all, a compression model is build over the training documents of a certain class. Then, each training document is expressed as a concatenation of sub-strings contained in the compression model. The total number of occurrence of each sub-string will be counted. In addition, each string will contain a set of references to the documents in which it occurred. A pruning algorithm keeps only the sub-strings that are frequent enough and, those have a certain minimum size and occurred in at least a given number of documents. The pruning is based on a pruning function that determines the minimum acceptance ratio of a sub-string. The pruning function gives lower acceptance ratio for longer sub-strings. At the end of the process, the compression model is a tree that contains the desired features.

3.4.2.1 – Advantages

The first advantage of the complete covering approach is the reduced memory space occupied by the features at any time of the process. Since the algorithm is based on a compression model using a tree structure, all the common prefixes share the same memory space by using the same edges in the tree. Moreover, as mentioned before, the tree structure is an appropriate structure for searching textual patterns in a text.

A second advantage is the pruning mechanism made possible by this method. As opposed to the Apriori N-Gram extraction that relies on an integer threshold for pruning, the complete covering approach uses a minimum acceptation ratio for pruning. The pruning based on a ratio is now possible since we use a tree structure with redistribution of the statistical information as described in figure 3.14. As the numerator of the ratio is the number of occurrences and the denominator is the total number of characters of the training set, the size of the training set should not affect pruning. Also, the minimum acceptation ratio is reduced according to the size of the sub-string being tested for pruning because the probability of observing longer sub-strings frequently is less than a shorter sub-string. The pruning function (Figure 3.13) dictates this behavior. The pruning algorithm and the way of counting the sub-strings by covering the training documents will generate a reduced number of features compare to the number of features generated by the Apriori algorithm. This can be desirable in some applications.

Another advantage is the fact that the maximal feature length does not have to be specified. If repetition occurs, the compression tree will get deeper. Generally, extracting features from documents with a lot of redundancy (like natural language) will result in a deeper compression tree. On the other hand, extracting features on documents that contain pseudo-random characters (like

base64 encoding) will result in a shallow compression tree. Also, since the maximal feature length is not specified, there is no need to do multiple passes over the documents. In the present case, two passes will be executed: one to build the compression model and another to perform the covering of the documents and the counting of the sub-strings used.

3.4.2.2 – Disadvantages

This feature extraction technique is less exhaustive than the Apriori N-gram extraction algorithm. Since only one pass is done over the training documents to build the compression model, overlapping sub-strings will not be considered. More training documents will be required to detect a discriminating sub-string. This approach will be good if the discriminating features appear at a reasonable distance from each other. With this approach, a sub-string of a certain maximal size is not guaranteed to be considered. However, it is hoped that with many training documents, the compression algorithm will detect frequent sub-strings that are common to many documents or that are frequent within a document.

The parameters of the pruning function have to be set accordingly to the best knowledge of an external user. It might be complicated for a user to tune the parameters appropriately. These parameters will have to be adjusted relatively to the size of the alphabet used. As an example, if an alphabet with 256 symbols is used, applying a minimum acceptance ratio of 8% for single characters over the total characters of the training set might eliminate a lot of characters. But the same ratio would not be as restrictive for an alphabet of size 10. Also, a ratio composed of the occurrence of a sub-string divided by the total number of characters respects the proportions of occurrence versus the size of the training set. However, the confidence of applying a pruning method based on a ratio depends on the size of the sample observed. As an extreme example, assume that a minimum

acceptance ratio for a single character is 10%. If a training set of size 10 is observed, all the single characters that appear in the training set will be accepted as a feature. However, if a training set has 100 000 characters, a character will have to appear at least 10 000 times to be accepted as a feature.

3.4.3 – The Probabilistic Pruning Approach

The idea at the basis of this approach is to use the concept of a binomial distribution to select frequent n-grams to become features characterizing the training set of a certain class. The n-grams could be naively extracted from document examples of a certain class considering sub-strings overlapping[2]. The frequency of each n-gram observed in the document examples would be kept in a table. We will see later in this section that this n-gram gathering method poses a problem when approximating a binomial distribution by a normal distribution.

In order to summarize classical probability notions [BA89], a binomial distribution corresponds to a sequence of n independent Bernoulli tests. A Bernoulli test is an experiment for which the outcome can have two possible values (FAILURE or SUCCESS) and each experiment has an equal probability of success. Also, each experiment has to be independent from each other. A binomial distribution will have the following properties:

If X is a variable distributed according to a binomial distribution, n is the number of observations and p the probability of success of a Bernoulli test, we can observe the following:

$$\textbf{Expected Value (X)} = np$$
$$\textbf{Variance (X)} = np(1-p)$$
$$\textbf{Standard-Deviation (X)} = \sqrt{(np(1-p))}$$

Figure 3.16: Properties of a Binomial Distribution

[2] Example: Overlapping 2-grams: "abcd" → "ab","bc"and "cd". Non-overlapping 2-grams: "abcd" → "ab", "cd".

For the feature extraction problem, the Bernoulli test will be: "Correctly predicting the next n-gram". The probability of such a test depends on the size of the n-gram and on the size of the alphabet. As an example, if we consider 1-gram with an alphabet of 256 characters, the probability of predicting the next 1-gram will be (1/256) assuming that the character distribution is uniform and the characters are independent from each other. Under the same assumptions, the probability of predicting the next 2-gram is $(1/256^2)$ and so forth for longer n-grams. Theoretically, it is not perfectly exact because the 2-gram distribution depends on the 1-gram distribution. As an extreme example, imagine we first gather the 1-gram distribution and realize that characters "a" and "b" occur 90% of the time. Then it becomes false to assume that all 2-grams have an equivalent probability of being observed. In fact a 2-gram containing "a" and/or "b" will have a greater probability of being observed. Also, the independence of all n-grams from each other depends on the way they are extracted from the text. If the n-gram extraction consider overlapping sub-strings, than the assumption of independence is not totally respected. In addition, if the n-gram extraction is performed using a clustered alphabet, the property of independence is even less respected. In fact, symbol clustering attempts to define a mapping that maximizes the predictability of a sequence of characters.

It is agreed that the assumptions of a binomial distribution are not respected rigorously, but at the practical level this approach may still be an excellent approximation for selecting features. In order to select the n-grams that are particularly frequent compared to the average distribution, we will approximate the binomial distribution by a normal distribution. According to the DeMoivre-Laplace theorem, a binomial distribution can be approximated by a normal distribution if the following criteria are respected (Figure 3.17):

1) $n \geq 30$
2) if $p \leq 0.5$ then $np \geq 5$
   if $p > 0.5$ then $n(1-p) \geq 5$

Figure 3.17: Criteria for Approximating a Binomial Distribution by a Normal Distribution

Assuming that it is possible to approximate a binomial distribution by a normal distribution, it is now possible to use the central limit theorem to perform the pruning of infrequent features. According to this theorem the occurrence distribution of the n-grams of a certain size is expressed by the classical "bell curve" (Figure 3.18).

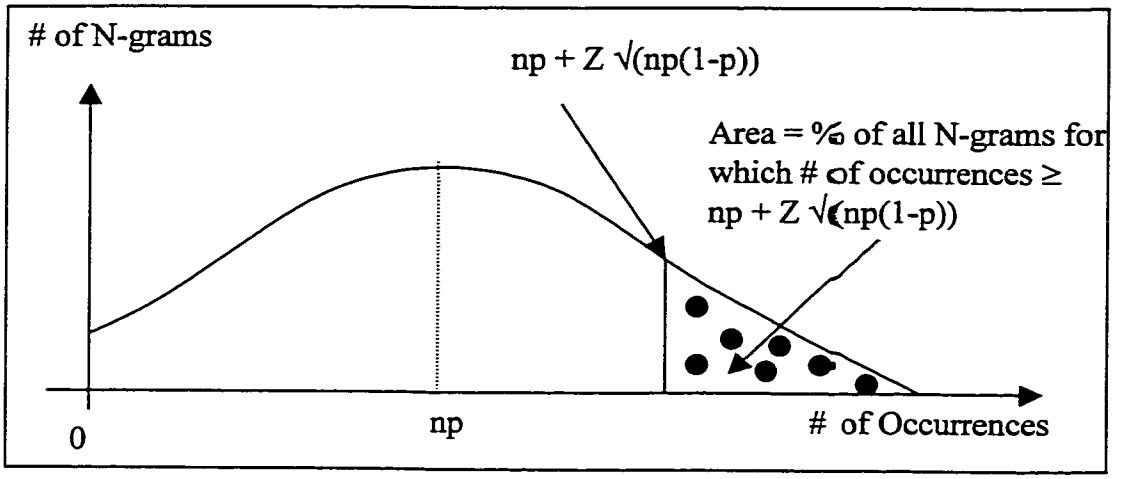


Figure 3.18: Central Limit Theorem Applied to N-Grams

Using the central limit theorem, a user could decide on a certain percentage representing the proportion of sub-strings that are at the far right end of the normal distribution. By determining this value, the user decides the proportion of the most frequent sub-strings he wants to keep as features. Having the desired percentage, the Z[3] value will be determined with the normal distribution formula

[3] This value could be determined from a normal distribution table given a certain percentage.

or a normal distribution table in order to fix the acceptance threshold for selecting n-grams of a certain size. The minimum acceptance threshold will be $(np + Z \sqrt{(np(1-p))})$ for a certain number of observations (n) and a certain probability of observing a n-gram of a certain size (p).

This approach reasonably respects the theories of probabilities and statistics and should provide a powerful and accurate tool to prune the features. However, it is almost impossible to apply this approach in practice by gathering all the overlapping n-grams of a text. The technique could work for short n-grams of size 1 or 2. But things get more complicated for longer n-grams. In fact, to approximate a binomial distribution with a normal distribution, the DeMoivre-Laplace (Figure 3.17) criteria have to be respected. The first constraint of observing at least 30 characters should not be a problem in practice. However, it is harder to respect the second constraint in practice. The probability of correctly predicting the occurrence of a n-gram of any size will always be less than 0.5. This implies that the minimum number of observations required to approximate a binomial distribution by a normal distribution is $(n = 0.5 / p)$. The following table illustrates the minimum number of observations required with an alphabet of 256 characters:

1-gram	$0.5 * 256 = 128$
2-gram	$0.5 * 256^2 = 32\ 768$
3-gram	$0.5 * 256^3 = 8\ 388\ 610$
4-gram	$0.5 * 256^4 = 2\ 147\ 480\ 000$
n-gram	$0.5 * 256^n$

Table 3.3: Minimum Number of Characters to approximate a Binomial Distribution by a Normal

The numbers in this table indicate that an enormous quantity of training documents would be required to apply this pruning mechanism on n-grams of a reasonable size. We propose a heuristic

approach based on the LZ78 compression technique that will allow the utilization of the central limit theorem for pruning.

First of all, a compression tree will be built from the training documents. Then, every occurrence of each sub-string included in the compression tree will be found and counted in the training documents. This counting mechanism will consider overlapping sub-strings. When counting the sub-strings, the number of distinct sub-strings per length will be calculated. Also, the total number of observations per length will be calculated as showed in figure 3.19.

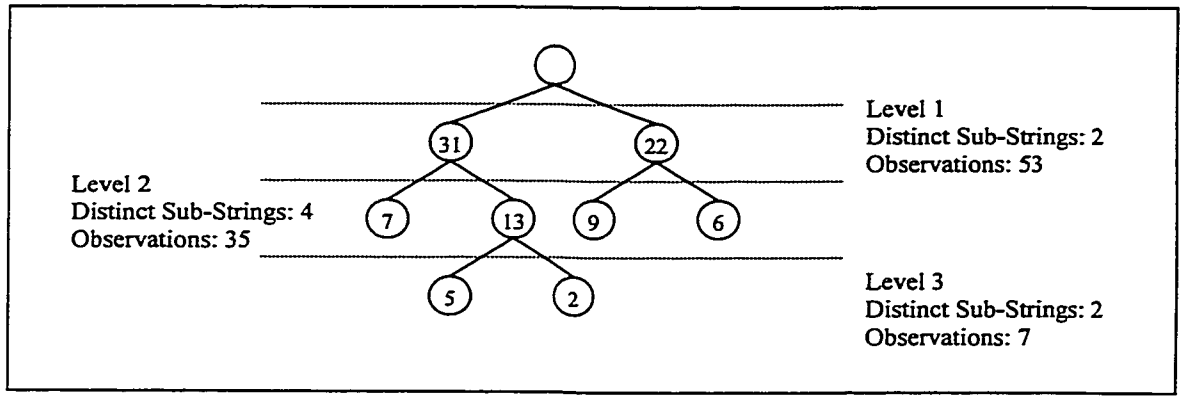


Figure 3.19: Distinct Sub-Strings and Observations per Level

Having the compression tree, the number of distinct nodes per level and the total number of occurrences per level, an acceptance threshold will be calculated for each level with the central limit theorem. In fact, the probability of observing a certain n-gram depends on the number of observations at "depth n" in the compression model. The acceptance threshold for a level i will be calculated from the definition of the central-limit theorem (figure 3.18) as follows:

$$ThresholdForLevel-i = ObservationForLevel-i * \frac{1}{DistinctNodesForLevel-i} + \sqrt{\frac{ObservationsForLevel-i}{DistinctNodesForLevel-i} * (1 - \frac{1}{DistinctNodesForLevel-i})}$$

By considering only the compression model as a sub-set of all the possible n-grams, it is possible in practice to approximate the binomial distribution with a normal distribution and therefore, it is possible to use the central limit theorem to prune features. In addition to the pruning mechanism using the central limit theorem, a pruning based on document occurrence, similar to what is described in section 3.3.2, will be integrated in the pruning algorithm. Assuming that a LZ78 compression is already built on the training documents, the pseudo-code of figure 3.20 describes the "counting" algorithm over a training document.

```
Procedure CountSubStrings(aDocument, aCompressionTree)

        observationsPerLevel := Array.newWithSize(aCompressionTree.maxDepth());
        observationsPerLevel.addAll(0);
        pointerCollection := {};

        For each char ∈ (sequential characters from aDocument)
                nextNode := aTree.getRoot().getChildrenForCharacter(char);
                For i = 0 to (pointerCollection.getSize() – 1)
                        pointerNextNode := pointerCollection[i].getChildrenForCharacter(char);
                        if (pointerNextNode == nil)
                                pointerCollection.removeElementAt(i);
                        else
                                pointerCollection[i] := pointerNextNode;
                if (nextNode != nil)
                        pointerCollection.addElement(nextNode);
                For each aNode ∈ pointerCollection
                        observationsPerLevel[aNode.getDepth()]++;
                        aNode.incrementCount;
                        aNode.addDocument(aDocument)
        Return observationsPerLevel
```

Figure 3.20: Pseudo-Code for Counting all the Sub-Strings of a Compression Tree

This "counting" algorithm will count all the sub-strings contained in the compression tree that occur in the training documents. At this point, all the nodes of the compression tree contain an occurrence counter and a number of documents in which it appeared. This information will be used to select features from the compression tree and to prune the nodes that are not a part of a branch representing a feature. At the end of the pruning algorithm, the compression tree will contain all the sub-strings

that have been selected to be features of the training documents. Pseudo-Code of figure 3.21 describes the pruning algorithm. This algorithm takes a compression tree as parameter and also an array corresponding to the pruning threshold for each level calculated with the central limit theorem.

```
Procedure Pruning(aCompressionTree, thresholdsPerLevel, minimalLength, minDocOccurence)

        Return recursivePrining(aCompressionTree.getRoot(),thresholdsPerLevel);

Procedure RecursivePruning(aNode, thresholdsPerLevel, minimalLength, minDocOccurence)

        thereIsAFeatureDeeper := FALSE;
        For each (character,aChild) ∈ aNode.getChildren()
                childWantsToStay := RecursivePruning(aChild, thresholdsPerLevel,
                                        minimalLength, minDocOccurence);
                if (!childWantsToStay)
                        aNode.removeChildForCharacter(character);
                thereIsAFeatureDeeper := thereIsAFeatureDeeper OR childWantsToStay;
        if (aNode.depth > 0)
                if (aNode.count >= thresholdPerLevel[aNode.depth] AND
                  (aNode.depth >= minimalLength AND
                  aNode.numberOfDocuments >= minDocOccurence))
                        aNode.markAsFeature(TRUE);
                        thereIsAFeatureDeeper := TRUE;
                else
                        aNode.markAsFeature(FALSE);
        Return thereIsAFeatureDeeper;
```

Figure 3.21: Pseudo-Code for Pruning a Compression Tree

In order to recapitulate this section, the following summarizes the probabilistic pruning approach. The first thing to do is to build a LZ78 compression model over the training documents. After having built the compression model, the number of distinct nodes per level is kept in an array. Then the occurrence of all the sub-strings from the compression model and the number of documents in which they appear will be counted over the training documents. After having performed this operation, the number of observations per level will be kept in an array. With the number of distinct nodes per level and the number of observations per level[4], a pruning threshold will be determined for each level. The idea behind the calculation of the thresholds is to use the concepts of a binomial

[4] As illustrated in Figure 3.19.

distribution approximated with a normal distribution. The threshold is calculated with the central limit theorem for a certain value "Z" representing the desired portion that the user wants to select from the n-gram distribution.

3.4.3.1 – Advantages

The advantages that the probabilistic pruning approach provides are very similar to those associated with the complete covering approach. Since the probabilistic pruning approach is also based on the generation of a compression model, the economy of memory is still an advantage. Also, the end result is a tree, which is appropriate for quick pattern searches. Moreover, there is no need to fix a maximal feature size like in the Apriori n-gram extraction.

The counting mechanism is more exhaustive than the one presented in the complete covering approach. In the probabilistic pruning approach, overlapping is considered when counting the occurrence of the sub-strings of the compression model, as opposed to the complete covering approach. The pruning mechanism is still heuristic, but is closer to the theories of probabilities and statistics. The pruning algorithm uses a different threshold for all the different feature lengths. Also, the threshold is calculated according to the size of the sample, which gives a higher confidence in the pruning.

On a user perspective, configuring the pruning algorithm is now very simple. The only things that needs to be provided are a "Z" value representing the proportion of sub-strings selected as features, a minimum length and a minimum number of documents in which a sub-string must appear. These parameters are independent of the size of the alphabet or the size of the training set.

3.4.3.2 – Disadvantages

Even if this technique is more exhaustive than the complete covering approach, it is still less exhaustive than the Apriori n-gram extraction. In the Apriori n-gram extraction, all the sub-strings up to a certain size are considered. In the probabilistic pruning approach, only the sub-strings from the compression model are potential candidates. Therefore, more training data is required to form longer sub-string in the compression model.

3.5 – Feature Extraction Based on a Hierarchical Grammar

In 1997, Craig G. Nevill-Manning and Ian H. Witten published an algorithm called SEQUITUR that has for objective to identify hierarchical structure in sequences [NW97]. This section describes the possibility for using the rules generated by this algorithm as potential features for unintelligible document classification.

The algorithm SEQUITUR reads sequentially the symbols from a set of training documents. The sequence of symbols read will be expressed with a set of rules (a grammar). Initially, all the characters will be appended to an initial rule. At any time the two following conditions must be respected:

1. No pair of adjacent symbols can appear more that once in the grammar.
2. Every rule must be used more than once.

Rules will be created or removed from the grammar in order to respect those conditions. When a rule is created, the rule is identified with a new unique symbol and replaces the occurrences of repeated character pairs. A rule will be removed when it is used only once. Its content will replace the only

occurrence of the rule. The following example (Table 3.4) gives an idea of the behavior of the algorithm:

Symbol Number	The String so Far	Resulting Grammar	Remarks
1	A	0 → A	
2	AB	0 → AB	
3	ABC	0 → ABC	
4	ABCD	0 → ABCD	
5	ABCDB	0 → ABCDB	
6	ABCDBC	0 → ABCDBC	BC appears twice
		0 → A1D1 1 → BC	Enforce digram uniqueness
7	ABCDBCA	0 → A1D1A 1 → BC	
8	ABCDBCAB	0 → A1D1AB 1 → BC	
9	ABCDBCABC	0 → A1D1ABC 1 → BC	BC appears twice
		0 → A1D1A1 1 → BC	Enforce digram uniqueness, A1 appears twice
		0 → 2D12 1 → BC 2 → A1	Enforce digram uniqueness
10	ABCDBCABCD	0 → 2D12D 1 → BC 2 → A1	2D appears twice
		0 → 313 1 → BC 2 → A1 3 → 2D	Enforce digram uniqueness, rule 2 is used only once
		0 → 313 1 → BC 3 → A1D	Enforce rule utility

Table 3.4: Example of a SEQUITUR Execution on the String "ABCDBCABCD"

The algorithm SEQUITUR was used almost in its entirety for the present work. It is a very efficient linear algorithm. We have added a pruning mechanism to SEQUITUR that restricts the number of rules that will be used as features. The complete description of the implementation of SEQUITUR was presented in [NM97] and will not be showed in this document.

According to Nevill-Manning and Witten, the number of rules in the grammar increases almost linearly with the number of characters read in the training documents. A training set of 200 000 characters can generate a number of rules in the order of 50 000. In order to select the rules from the grammar as features, we decided to implement a pruning mechanism on top of SEQUITUR. For the pruning mechanism, we assume that a counter is associated with each rule and represents the number of times a rule has been used. Moreover, we assume that the "pure size" of each rule can be easily calculated. The pure size of a rule is its actual length in terms of symbols from the original alphabet. As an example imagine the grammar is composed of two rules: 0 → 11 and 1 → ab. The pure size of rule 0 is 4 and the pure size of rule 1 is 2.

The idea of the pruning algorithm is to keep, as features, the following rules:

- Rules that occur frequently.
- Rules that are long.
- Rules that are frequent and long.

First it is obvious that the rules that occur more frequently compared to the average should be kept as features. Second, the rules that are significantly longer than the average length should be considered. The fact that a long string appears more than once may have some significance in the training set. Third, rules that are reasonably frequent and reasonably long will be considered as features. This third pruning criterion is the intersection of the first two. In order to do this pruning the mean and the standard deviation will be calculated for the occurrence and pure size of the rules contained in the grammar. With the mean and the standard deviation, we will be in position to use the central limit theorem to prune the features exactly like it was described in the previous section. With normal distribution of occurrence and pure size, a user will determine four thresholds. Figure 3.22 shows graphically how the pruning thresholds are calculated given four parameters:

Z1 : Number of occurrences (pruning threshold parameter).
Z2 : Pure size (pruning threshold parameter).
Z3 : Number of occurrences (pruning threshold parameter used in conjunction with Z4).
Z4 : Pure size (pruning threshold parameter used in conjunction with Z3).

Generally, Z1 will be higher than Z3 because Z1 determines the threshold for the number of occurrences only as opposed to Z3 that determines the number of occurrences threshold that is in conjunction with the pure size threshold determined by Z4. For the same reason, Z2 should be higher than Z4.

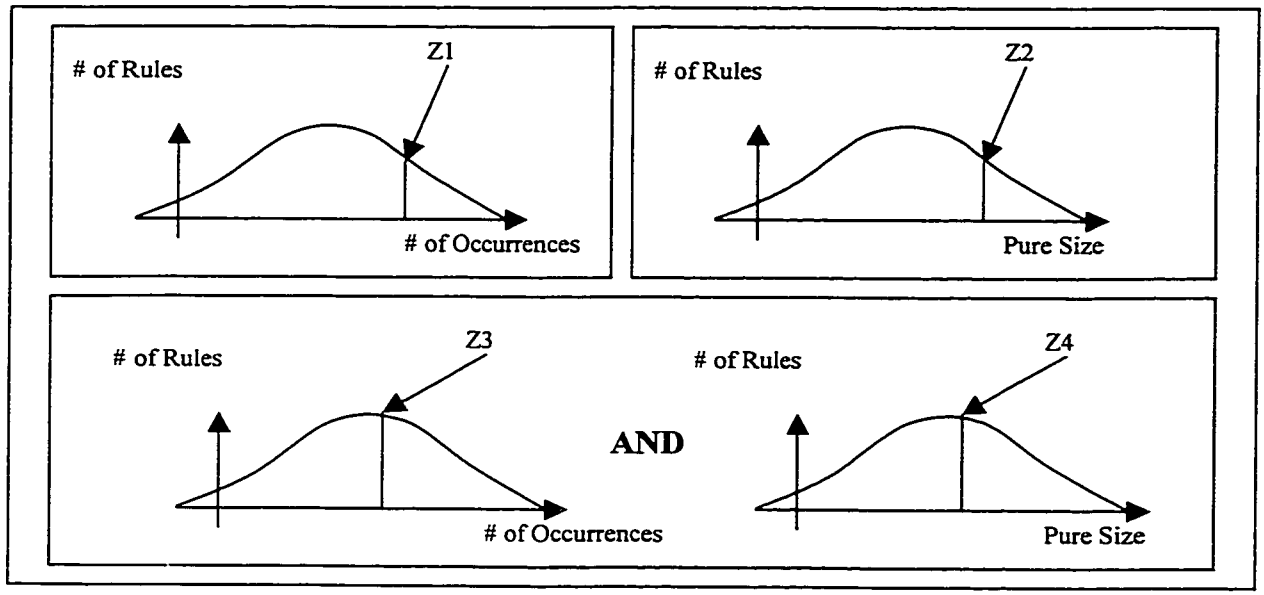


Figure 3.22: Pruning Thresholds for SEQUITUR

3.5.1 – Advantages

Using SEQUITUR as a feature extraction mechanism offers many advantages. There is no need to specify a maximal n-gram size like in the Apriori n-gram extraction technique. Every pattern that has more than one character and occurs more than twice will be included in the grammar built by SEQUITUR. This last property of SEQUITUR is in fact, exactly what we expect a grammar would

include if we want to extract features from it. Generally, 1-grams are not very interesting because they are not discriminating properties and sub-strings that occurs only once are not interesting either. As mentioned in the previous section, SEQUITUR can generate an enormous number of rules. To extract the features from the SEQUITUR grammar, the pruning mechanism that we have implemented seems to be appropriate to control the number of features extracted. Because SEQUITUR is an efficient algorithm, it is feasible to extract features on a very large training set in a reasonable period of time.

3.5.2 – Disadvantages

We have not found an efficient way to implement a pruning based on fact that a rule must occur in a minimum number of documents. Without this type of pruning, a single document with a lot of repeated characters can affect the generation of features that do not necessarily reflect the entire training set. However, we do not say that it is not possible to improve the pruning mechanism associated with SEQUITUR. We believe that there is room for improvement and that further research is required to develop an accurate pruning mechanism that would improve feature extraction with SEQUITUR.

Chapter 4 – Classification Model Induction

4.1 – Introduction to Classification Model Induction

The task of the classification model is to identify an unseen document as being a part of one or many pre-defined classes. The behavior of an identification model could be implemented in many different ways. It is possible to imagine an identification model that would identify a document as being a part of a class by searching for some key words or by searching for a regular expression that is proper to the documents of the class. In this section, we are mainly interested in the techniques for inducing a classification model from pre-labeled training documents. In the previous section, we introduced some techniques to extract features and associate them with the training documents. The induction mechanism will rely on those features to determine the rules that will dictate the behavior of the classification model. We use the term "induction" because the goal is to learn a classification that is proper to all the documents of a certain class only by observing a limited number of training documents. The rules induced from the training documents will have to be more general than the rules that fit exactly the training set, otherwise only the training examples would be properly classified by the classification model (This phenomenon is called "overfitting"). On the other hand, the rules should not be too general for a specific class because they could accept documents from the other classes.

Once trained with a machine learning algorithm, the induction model is ready to accomplish its task. For some systems, it is possible to refine the classification model with additional labeled training documents without having to re-train the system on the entire training set. Such systems are

categorized as incremental learning systems. Systems that must be re-trained with the entire initial training set plus the additional documents are non-incremental learning systems. Ideally, in practice, an incremental learning system would be desirable to allow users to correct and improve the classification model. This might be required to adapt to periodic changes in the incoming data. This phenomenon is known as concept drift [WK98].

The next sections of this chapter present two approaches to induce classification models from training examples. The first approach is to use a rule-based learning system to identify classification rules. Each rule is composed of a conjunction of presence or absence of sub-strings. A rule has for role to determine the class of an unseen document. The second approach presented is based on grammatical inference techniques. Grammatical inference has for its objective to identify a grammar that will "accept" all the training examples of a certain class and "refuse" examples of the other classes.

4.2 – Rule-Based Learner

In chapter 3, we have seen some techniques to determine the features that correspond to each value of a binary feature vector. A feature vector is associated with each document. Each feature is a sub-string and each boolean entry of the feature vector represents the absence or presence of this sub-string within the document (Figure 4.1). It is also possible that each document has more than one feature vector in the cases where many different alphabets were used when extracting the features. These feature vectors usually represent an enormous number of sub-strings. The goal of a rule-based learner is to select a minimum number of features that are necessary to discriminate a class versus the other classes. The selected features will be expressed in terms of classification rules. A

classification rule is a conjunction of boolean features, i.e. presence or absence of certain sub-strings, associated with a class. If a document contains (or does not contain) each sub-string represented by the rule, then it is classified as a member of the class associated with the rule.

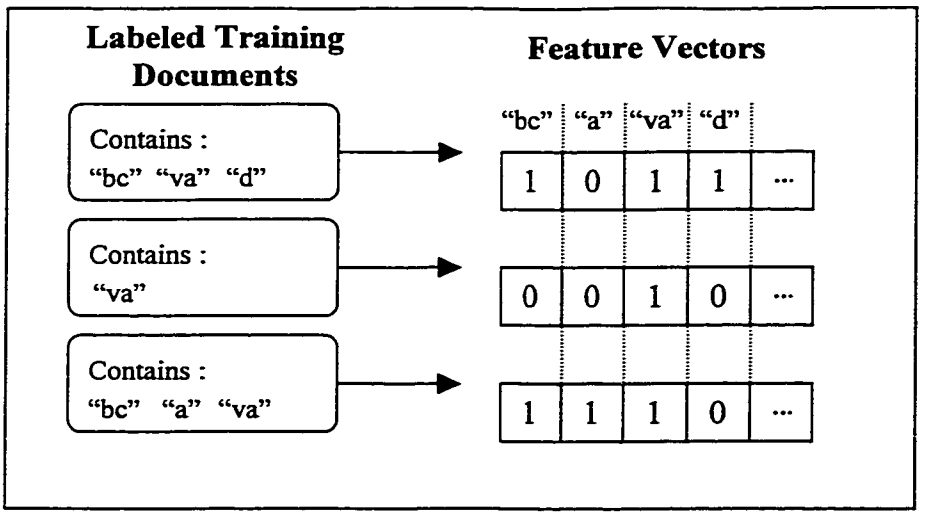


Figure 4.1: Training Documents with Feature Vectors

The rule-based learning system used for the experiments presented in this document was RIPPER. RIPPER was developed by William Cohen [CO95] and is based on the incremental reduced-error pruning algorithm developed by Furnkranz and Widmer [FW94]. The learning system RIPPER is optimized and very efficient for binary feature vectors. It has the ability to accept set-valued attributes of binary features. This last property is highly desirable for our application because the attributes are represented as a set of sub-strings. The representation of a binary feature vector is conceptual but in fact, each document is associated with the set of sub-strings that it contains. With the set-valued attribute capability of RIPPER, all that is required is to provide the set of features from each document and RIPPER will infer the binary vector representation. The length of the vector will be equal to the number of distinct features of the entire set of training documents. Implicitly, a binary vector will be built and associated with each document. The binary values of the

vector are determined by the document including or not the feature associated with each position of the vector.

The result of the algorithm is a set of classification rules. Each rule looks for the presence or absence of a set of sub-strings in order to associate a previously unseen document with a class. These rules correspond to a subset of the binary features associated with the training document. A rule is built by selecting a minimum sequence of binary features that is common to a majority of documents of a target class and does not appear in the documents of the other classes (Figure 4.2).

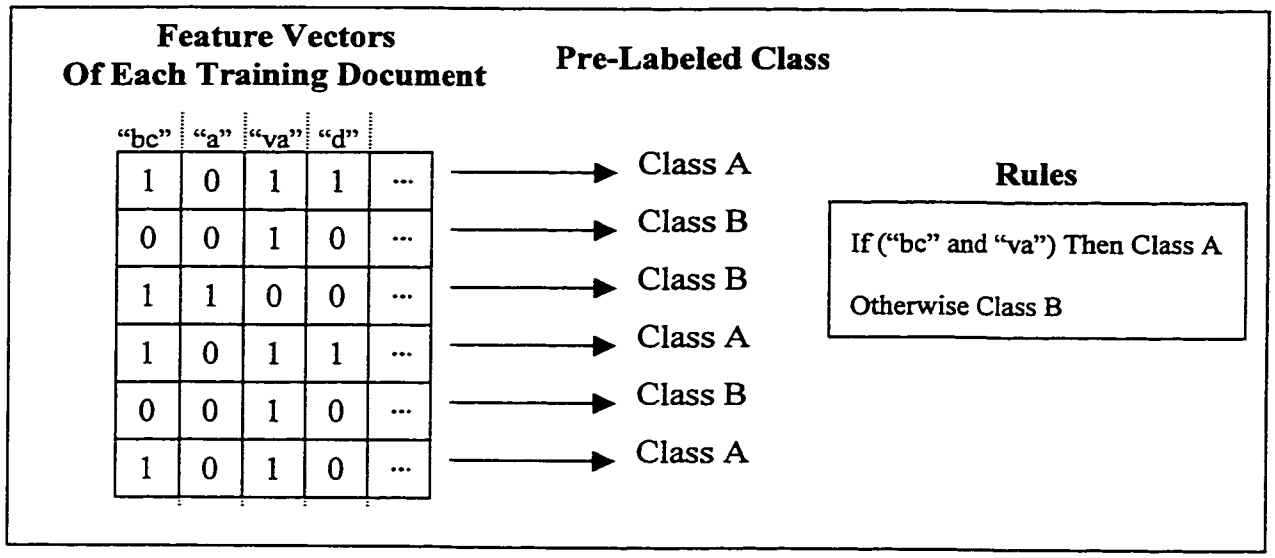


Figure 4.2: RIPPER's Parameters and Results

For the current project, we decided to use RIPPER since it implements set-valued attributes and it is optimized for considerably large number of binary features. However, some other techniques or other products might be used to accomplish the same task.

4.2.1 – Advantages

One of the main advantages of rule-based learners, in particular RIPPER, is their efficiency of induction. For a very large number of features and a large number of training documents, RIPPER induces identification rules surprisingly fast. It generates classification rules with a level of confidence in the rule. If a rule classifies more documents, it will have a higher level of confidence. The rules are easily interpretable by a human analyst. Moreover, RIPPER offers the possibility to adjust some settings or to include initial rules, which gives a better control to a human analyst over the training. In general, RIPPER is a good algorithm when the discriminating features are a set of distinct sub-strings that appear in a certain class and in no other classes.

4.2.2 – Disadvantages

When using a rule-based system like RIPPER, the training data has to be presented as a set of attributes. In our case, we modeled the training documents as a set of sub-strings that were extracted with a feature extraction process. With RIPPER, the sub-strings sets are mapped to binary vectors representing the fact that a document contains or does not contain a certain sub-string. Basically, the only property that can be captured is the absence or presence of a sequence of sub-strings in a document. There is nothing that takes in consideration the position where a sub-string occurred within a document. This implies that there is no way to observe that a certain sub-string occurs before or after another sub-string. The "structural" aspect of the sub-strings is not considered during learning. Maybe, these "structural" characteristics reveal an important discriminating factor. As an example, in an e-mail, the sub-sting "From:" is often observed before the sub-string "To:".

Another disadvantage of an algorithm like RIPPER is the fact that learning is not incremental. There is no way to improve the classification model (in this case the classification rules) by providing additional pre-labeled documents. If additional pre-labeled documents are available, the system will have to be re-trained with the original training documents plus the additional documents.

4.3 – Regular Grammatical Inference

4.3.1 – Introduction and Overview of Existing Algorithms

The objective of regular grammatical inference is to learn a regular language from a finite set of examples. The examples might be strings that belong to the language or strings that do not belong to the language. These examples will be called respectively positive and negative examples. Some grammatical inference techniques have been created to learn only from positive examples when it is not possible to obtain negative examples. A very good description of these algorithms is included in previous work by Pierre Dupont and Laurent Miclet [DM98]. However, it has been proven that any class of languages over an alphabet Σ that contains every finite language together with at least one infinite language over Σ cannot be exactly identifiable only from positive examples [GL67]. On the other hand, it has been shown in [AN80] that correct inference from positive data only is possible for some classes of "nonempty recursive languages". For the present work, we will focus on grammatical inference techniques that use positive and negative examples since it is possible to obtain negative examples in our case.

In this section, we will describe the concepts of regular grammatical inference in an intuitive manner. For a formal and rigorous description of various grammatical inference techniques see

[DM98]. The main concept at the basis of most of the grammatical inference techniques is the generalization of a deterministic finite state machine (DFSM), initially built from the positive examples. A finite state machine can be defined formally as a 5-tuple (Q,Σ,δ,q_0,F) where Q is a finite set of states, Σ an alphabet, δ a transition function, q_0 is the initial state and F is a subset of Q identifying the final or accepting states [DP96]. The state transition function is defined as $\delta(q_1, a) \rightarrow q_2$ where $q_1, q_2 \in Q$ and $a \in \Sigma$. A finite state machine is deterministic if and only if for any q in Q and any a in Σ, $\delta(q, a)$ has at most one member.

The first step is to build the prefix tree acceptor (PTA) from the positive examples. Each transition between states in the tree corresponds to a symbol read from a positive example. The final state corresponding to the last character of an example will be an acceptor state $\in$ F. All the common prefixes of the examples share the same transitions in the PTA. An illustrative example of a PTA is showed in figure 4.3.

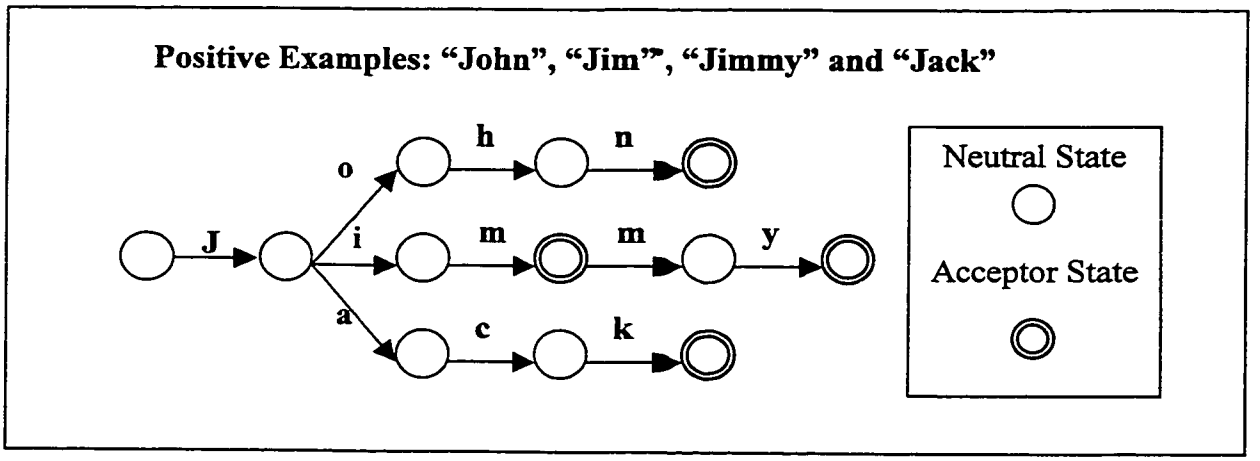


Figure 4.3: Example of a PTA

The PTA is a DFSM that accepts only the positive examples. Keep in mind that the objective of grammatical inference is to infer the language that covers more than just the positive examples.

Consequently, the next task will be to generalize the PTA in order to accept more documents than only those presented in the positive training set. It is important to mention that generalization will be controlled by the negative examples. In any case, the generalized DFSM should never accept a negative example. The generalization will be performed to the limit. The deterministic finite state machines that are "at the limit of generalization" belong to the border set (Figure 4.4).

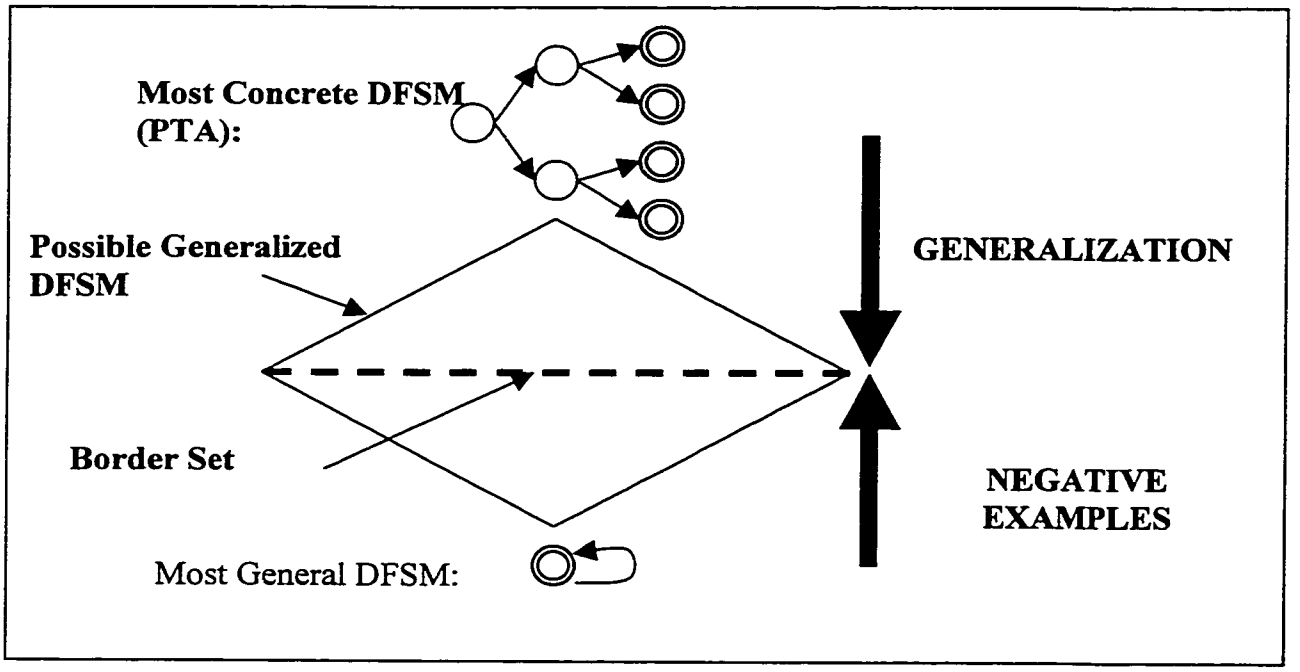
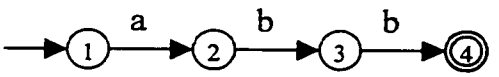


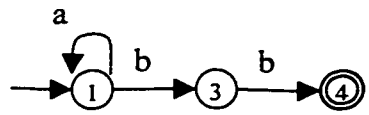
Figure 4.4: Generalization of DFSM

The generalization of a DFSM is done by merging nodes. While merging the nodes, the DFSM has to remain deterministic at any time. The next example illustrates these concepts:

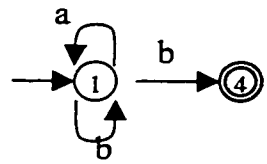
This DFSM accepts only the sub-string "abb":



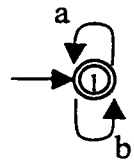
If nodes 1 and 2 are merged, the DFSM is more general because it accepts "abb", "bb", "aabb", "aaabb", etc. It accepts now any number of occurrences of the character "a" followed by "bb".



The next generalization step would be to merge nodes 1 and 3. However this fusion introduces non-determinism because there are two transitions with character "b" going out of state 1:



In order to correct this situation, the fusion will be performed recursively until the determinism is respected. In this case, states 1 and 4 will be merged:



So far, we have presented the main ideas and concepts from regular grammatical inference using positive and negative training examples. The algorithm RPNI[5] proposed by Oncina and Garcia [OG92] and, independently by Lang [LA92] works under the principles previously described. RPNI takes a set of positive examples and a set of negative examples as parameters. RPNI finds the minimum DFSM that accepts the elements of the target language if the training set is "characteristic". The formal description of a "characteristic" training set is defined in Appendix 1 at the end of this document.

[5] RPNI stands for Regular Positive and Negative Inference

Finding the minimum DFSM compatible with the positive and negative samples is a problem that has been proven to be NP-Hard [AN78, GL78]. However, because it assumed that the training set is characteristic, RPNI has a complexity of $O((||I_+|| + ||I_-||) \cdot ||I_+||^2)$ where $||I_+||$ and $||I_-||$ are the number of symbols in the positive and negative training sets [DM98]. If RPNI is not provided with a characteristic sample, the final DFSM will still be compatible with the positive and negative samples, but the final DFSM will not necessary be minimum i.e. containing the fewest states. Pseudo code of figure 4.5 illustrates the RPNI algorithm.

```
Procedure RPNI(I+, I-)

    /* N is the number of states in the PTA(I+) */
    /* Note that a block is a fusion of many nodes */
    π := {{0}, {1}, ..., {N-1}}        /* Each block is initially a node of PTA(I+) */
    A := PTA(I+)

    For i=1 to (|π| - 1) do:                          /* Loop on the blocks of partition π
                                                        Bi is the i-th block of π*/
        For j=0 to (i-1) do:                          /* Loop on the blocks of lower rank */
            π' := π\{Bj,Bi} ∪ {Bj∪Bi}                 /* Merge Block Bj and Bi */
            A/π' := derive(A, π')                     /* Express A in terms of the blocks in π' */
            π'' := determ_merge (A/π')                /* Merge the states that cause
                                                        non-determinism */
            if (compatible(A/π'', I-))                /* If A/π'' accepts no examples from I- */
                A := A/π''
                π := π''
                break loop on j

    Return A
```

Figure 4.5: Pseudo-Code of the Algorithm RPNI

RPNI is a non-incremental learner, which implies that we have to give an entire training set to induce the DFSM. Pierre Dupont has developed an incremental version of RPNI that he called RPNI2 [DP96]. RPNI2 has a comparable complexity to RPNI in theory, but in practice RPNI2 can even perform better. RPNI2 accepts training data one by one. It refines the DFSM according to the

new examples without discarding the previous knowledge, which consist of accepting or refusing the previous examples.

Kevin Lang [LA99] has recently developed an exact grammatical inference algorithm. The algorithm of Lang (called EXBAR) induces a minimum DFSM that is compatible with the training set without assuming a "characteristic" training set like RPNI. Instead of starting with a PTA, which only contains the positive examples, EXBAR starts with an APTA that contains the positive and the negative training examples. The final states (states that belong to the set F previously described) have a label specifying if they have been issued from a positive or a negative example. The main idea of this algorithm is to fix an upper bound on the number of states of the target DFSM and try to come up with a DFSM that satisfies the training set. If it is impossible, than the upper bound is raised by one unit. The algorithm is based on the red-blue framework. Red nodes are nodes that will be included in the final DFSM. The blue nodes are potential candidates to become a red node or to be assimilated by a red node. The color of a node characterizes its position in the DFSM. The blue nodes are always the roots of a tree and adjacent to a red node (Figure 4.6). The number of red nodes can never be over the upper bound. When a node assimilates another node, it takes its label (positive or negative). A node that has a positive label can never assimilate a node with a negative label and vice-versa. The DFSM has to remain deterministic at any time. The algorithm terminates when there is no more blue nodes.

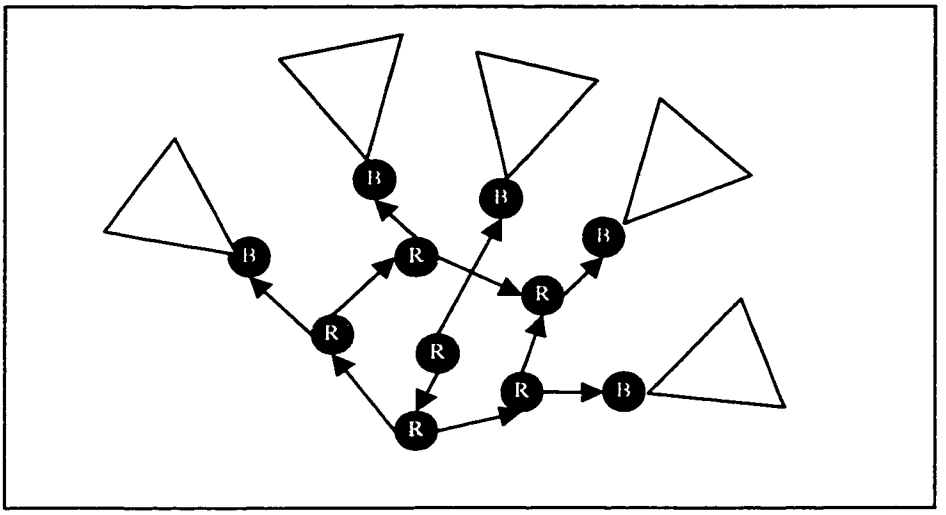


Figure 4.6: Red-Blue Framework used in EXBAR

EXBAR selects heuristically among the blue nodes, the next node to be promoted to red node or assimilated by a red node. A blue node will be promoted to red node if no red node can assimilate it. Each node has a label that could be positive, negative or null. Also, each node can have the color blue, red or none. A fusion always occurs between a red node and a blue node. A fusion will be refused if a node with a positive label has to be merged with a node with a negative label and vice-versa. Figure 4.7 illustrates a situation were a blue node cannot be merged with any red node. In this case, the blue node in question can be promoted to red node, if the total number of red nodes does not exceed the upper bound.

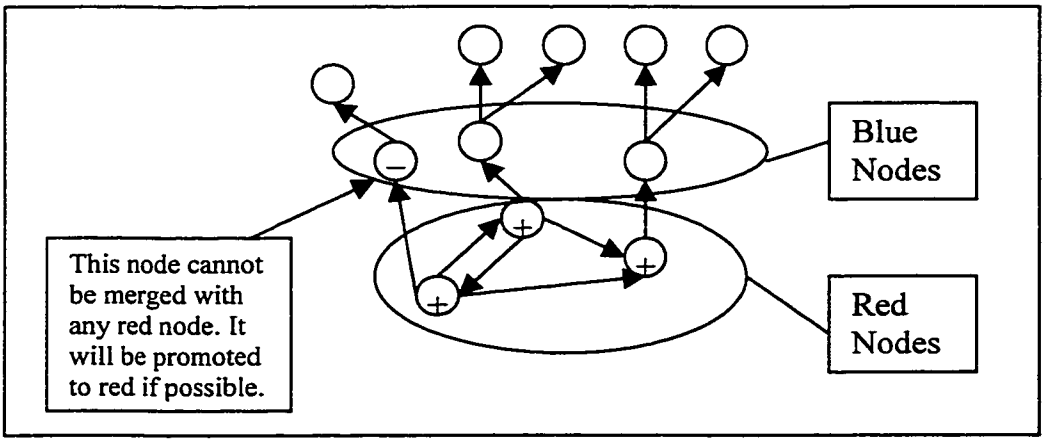


Figure 4.7: Red-Blue Fusion in EXBAR

4.3.2 – Regular Grammatical Inference applied to Data Classification

In this section, we will discuss how regular grammatical inference could be applied to the current problem, which is structural identification of unintelligible documents. In the introduction of this document, we mentioned that the only assumption that we could make about the documents is that they could be classified using a regular expression. We have seen some techniques that allow the induction of an acceptor finite state machine. The concept of acceptor finite state machine and regular expressions are related. In fact, the well-known regular expression interpreters, like the one used in PERL and the one used in UNIX and C, transform the regular expression into an acceptor finite state machine before finding the regular pattern in a document.

In theory, it would be possible to apply the regular grammatical inference techniques on the training documents exactly like it is presented in the literature. As an example imagine that we want to classify e-mail documents versus non e-mail documents. We have 50 examples of e-mails and 50 examples of non e-mail documents. The training documents have, in average, 10 000 bytes each. Initially, the first thing to do is to build the initial PTA with the e-mail documents, exactly like illustrated in figure 4.3 except that instead of "John", "Jim", "Jimmy" and "Jack" we use each e-mail document. The documents are read using the extended ASCII (8 bits per character). Then a grammatical inference algorithm like RPNI or RPNI2 is used to generalize the DFSM under the control of the negative examples.

Applying regular grammatical inference like presented in the previous paragraph poses some major problems. First, the number of states in the PTA will be enormous. In our examples, the PTA will

contain approximately 50 * 10 000 = 500 000 states[6]. The principle of the RPNI algorithm is to explore the lattice of the possible DFSMs obtained by performing successive state merges. In this case, the lattice in question, or the search space, is huge. If there are 500 000 states, there are $\binom{500000}{2} \approx 1.25E11$ possible state merges. In reality, an algorithm like RPNI will not try all those merges. A lot of merges will be performed recursively as a consequence of the non-deterministic situation. In addition, the inner loop in the RPNI algorithm (Figure 4.5) will break when a merge has been successfully performed. Nevertheless, it is obvious that the amount of work necessary to generalize a PTA containing a large number of states is considerable. Moreover, after each merge attempt, the negative examples have to be applied to the DFSM to verify its compatibility, i.e. rejecting the negative training set. As soon as one negative example is accepted by the DFSM, the other negative examples do not need to be verified for this specific merge tentative. These observations reveal that the larger the training set (in terms of the number of symbols), the larger induction processing time and memory. On one hand, we need a lot of training examples to increase the accuracy of the final classification model. On the other hand, the more we have training examples, the longer is the induction process.

Also, if RPNI is used, we are guaranteed to obtain a minimum DFSM that is compatible with the training set only if the sample is "characteristic". In practice, it is impossible to guarantee this assumption. As stipulated in the definition of a characteristic sample (Appendix 1), the size of the characteristic sample depends on the size of the alphabet. This means that the probability to provide elements from various documents that are considered "characteristic" is lower if the size of the alphabet is higher. Without a characteristic training set, a minimum DFSM would be obtained with an algorithm like EXBAR, but the processing time for training would be a lot higher. Ideally a

[6] This number is approximate because common prefixes will share the same states.

minimum acceptor DFSM would be desired, but an acceptor DFSM that is not necessarily minimum could be acceptable for most of the situation.

In order to use grammatical inference as a mechanism to train classifiers the following issues would have to be taken in consideration:

- A PTA with a large number of states takes a longer time to induce.
- If the number of symbols in the negative examples is high, then the effort to verify the compatibility of the DFSM after a merge increases.
- Using an alphabet with fewer symbols increases the probability of providing training elements that are considered members of the characteristic training set. Having a characteristic training set guarantees that RPNI will induce the minimum DFSM compatible with the training set.

The next sections of this chapter will present some potential solutions addressing those issues.

4.3.2.1 – Limiting the Number of Symbols Read from Training Documents

A very simple approach to improve time performances would be to limit the size of the training documents. This would reduce the induction time for two reasons. First, because there are fewer states in the PTA. Second, since the negative examples are shorter, it will take less time to verify the compatibility of the DFSM with the negative examples. For a specific maximal document size M, a training document would be formed as follow: If the document has a size n less or equal to M, then the document is taken as it is. If the document has a size n greater than M, then only the first M characters are considered. In many applied cases, only looking at the initial characters of a document

is enough to distinguish the discriminating information proper to the class of this document. As an example, looking at the header of an e-mail is enough to guess that it is an e-mail.

4.3.2.2 – Reducing the Alphabet with a Clustering Algorithm before Grammatical Induction

Reducing the alphabet has some advantages as described previously in section 3.2. In addition, using a reduced alphabet will increase the probability of having elements that constitute the characteristic training set. This means that using a reduced alphabet increases the "relevance" of the training set, which is very sparse in practice if the extended ASCII code is used. Pierre Dupont and Lin Chase [DC98] previously applied this solution with success using a symbolic clustering algorithm (Section 3.2.1 of this thesis).

For the grammatical inference itself, the number of states in the PTA will not change because we are using a reduced alphabet. However, there are fewer possible transitions between states. This implies that fewer transitions can go out of a state if we want to keep the determinism of the finite state machine. Because of this last constraint, state merges will frequently introduce non-deterministic situations that would have to be handled by recursive state merges.

4.3.2.3 – Using Frequent Sub-Strings as State Transitions

Instead of using individual symbols as elements of an alphabet, it is possible to redefine an alphabet that would be based on sub-strings instead. This would have for effect the reduction of the number of states in the PTA and the reduction of the length of the training examples. As an example, consider a set of n sub-strings with an average size of g characters. Then each document composed of m

extended ASCII characters would be translated in a training example using an average of (m/g) sub-strings.

If we want to define an alphabet based on sub-strings, we first have to identify which sub-strings will be part of the alphabet. Ideally, we would like to select sub-strings that are frequent in the documents of the class that we are trying to identify. In chapter 3, we presented four different feature extraction techniques that all include a pruning mechanism to exclude infrequent sub-strings. These techniques could be used to determine the initial alphabet based on sub-strings. After having selected the sub-strings that compose the alphabet, we have to associate a unique identifier with each sub-string (e.g. a number or a unique symbol) (Figure 4.8). Now, we have to translate the training documents from a standard alphabet like the extended ASCII code to the new alphabet composed of sub-strings. After the translation, each document will be expressed as a sequence of sub-string identifiers. The order of these sub-strings identifier has to be exactly the same as the order of occurrence of each sub-string in the document (Figure 4.8).

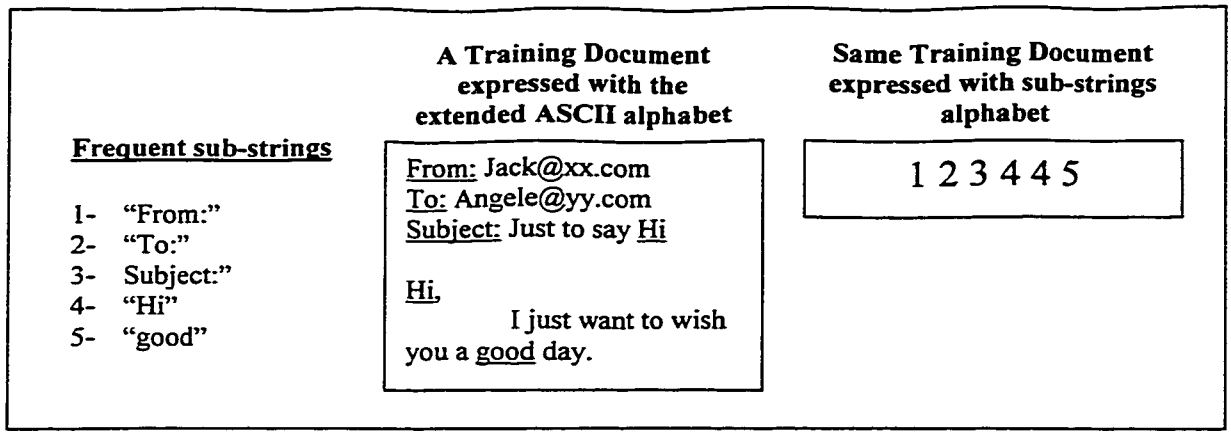


Figure 4.8: Training Document Expressed with a Sub-String Alphabet

It is important to mention that the sub-strings cannot overlap when expressing a document with the sub-string alphabet. As an example, assume that the sub-string alphabet contains the sub-strings 1-"abc" and 2-"bcd". The document "... abcd..." cannot be express as " ... 1 2 ...". In this case, the first complete sub-string to be encountered ("abc") will be selected and "bcd" will be ignored. Another exception case occurs when two sub-strings share a common prefix. As an example, assume that the alphabet contains 1-"Nervous" and 2-"Nervously". The document "... Nervously ..." will be expressed as "... 2 ..." because the long sub-string takes over the short sub-string when they share the same prefix. The algorithm to express a document in terms of a sub-string alphabet is described in figure 4.9. In this algorithm we use a stream structure that encapsulates the training document. A stream keeps a position pointer in the document. The position can be accessed and can be set to a certain value. The initial position of a stream is zero. The stream understands messages such as "next" which returns the next character and advances the position pointer. Also, because we determine the sub-string alphabet with a feature extraction technique, the sub-strings will be provided in a tree structure like described in section 3.1.

```
Procedure ExpressInTermsOfSubStrings(streamOverADocument, featuresTree)

        newDocument := {};

        while (!streamOverADocument atEnd())
                nextFeatureNode := featuresTree.getDeepestFeatureNode(streamOverADocument)
                If (nextFeatureNode != NULL)
                        newDocument.addLast(nextFeatureNode.getUniqueIdentifier())
        Return newDocument

Method Tree >> getDeepestFeatureNode (aStream)
        nextNode := this.getRoot()
        nextFeatureNode := NULL
        position := min(aStream.getPosition() + 1, aStream.getSize())

        while (aStream notAtEnd)
                nextNode := this.getChildrenForCharacter(aStream.next())
                if (nextNode == NULL)
                        aStream.setPosition(position)
                        Return nextFeatureNode
                Else if (nextNode.isFeature())
                        position := aStream.getPosition()
                        nextFeatureNode := nextNode
        aStream.setPosition(position)
        Return nextFeatureNode
```

Figure 4.9: Pseudo-code to Express a Document in Terms of Sub-Strings

In summary, the idea of the technique presented in this section is to express each document with a sub-string alphabet determined by a feature extraction technique. Then each document expressed as a sequence of unique identifiers representing the sub-strings will constitute the training set for a grammatical inference algorithm like RPNI. This will reduce considerably the number of states in the PTA and will accelerate the induction significantly.

4.3.2.4 – Interpretation of the DFSM

It is important to describe the way we have decided to interpret an acceptor DFSM when applied on a document. The traditional way to interpret a DFSM is to apply a state transition for each character read in the document. When the end of the document is reached, the current state of the DFSM determines the end result. If the state is an acceptor state, the DFSM accepts the document. If it is not an acceptor state, the DFSM refuses the document.

We have modified the way that we interpret an acceptor DFSM. We apply a state transition for every character read in the document. As soon as the current state of the DFSM is an acceptor state, the document is accepted. If the end of the document is reached without having visited an acceptor state, then the document is rejected. Also, if a certain character is read from the document and there is no transition going out of the current state of the DFSM, than the current state remains unchanged.

We have decided to use this mode of interpretation because we wanted to exit the evaluation as soon as a discriminating pattern has been found without having to go through all the documents every time. Also, we had better performances when using this interpretation mode in our experiments. The choice of the interpretation mode does not really affects a grammatical inference algorithm like RPNI. This is true as long as the same interpretation mode is used all the time during and after the induction. After a state fusion, the negative examples are all applied on the DFSM to validate the compatibility with the negative training set.

4.3.2.5 – Probabilistic Automaton Inference

Some grammatical inference algorithms have been developed to work without negative examples. This is the case of ALERGIA developed by Carrasco and Oncina [CO94]. The algorithm is very similar to RPNI. The only difference is that it performs a different compatibility test before merging two nodes. Also, the DFSM contains a counter in each state and in each transition. A state merge will be accepted if the DFSM satisfies a certain probabilistic test derived from the Hoeffding bound [HO63]. If this test is respected, we say that the states are α-compatible. Appendix 2 describes the details of the Hoeffding bound.

We suggest that it might sometimes be convenient to combine the probabilistic inference techniques with some exact inference techniques. In our case, it would be possible to combine RPNI with an α-compatibility test (see Appendix 2).

4.3.2.6 – Advantages

Regular grammatical inference presents many conceptual advantages over the other text classification techniques. First of all, this method identifies a regular grammar, which is a lot richer than a set of sub-strings for textual document classification. A regular grammar can contain static sequence of characters as well, but it has the flexibility to accept generic patterns including wildcards and repetitions. In theory, if grammatical inference is used with an alphabet based on individual symbols, the general structure of the documents could be encapsulated in the acceptor DFSM.

If grammatical inference is used with a sub-string alphabet, it could achieve the same goal as a rule-based classification system like RIPPER. The parameters are documents expressed as a sequence of sub-strings associated with a class, similarly to RIPPER. The end result is a classifier that will classify a document based on a sequence of sub-strings. In addition, the discriminating positional aspect of the sub-strings is included in the DFSM. If a certain sub-string must occur before another to reflect the structure of a certain class, it will be indicated in the transitions of the DFSM.

One of the most important advantages is the fact that the learning process can be incremental. An algorithm like RPNI2 could be used to induce the DFSM and to refine it with additional training examples. This property can be exploited to develop learning systems that are in constant evolution.

4.3.2.7 – Disadvantages

One of the main disadvantages of this technique is its learning time complexity. Even if research in regular grammatical inference progresses and the algorithms are improving, the time required to train on a relatively large training set is a lot more than a rule-based learner like RIPPER. Possible improvements, like reducing the alphabet or using a sub-string alphabet are possible. However, these additional generalizations may affect the precision of the final model.

Second, the final DFSM is not necessarily as easy to interpret by a human analyst as the rules derived by RIPPER. This might consitute a disadvantage in systems were the human users assist the automated learner.

Also, there is no mechanism to consider the frequency of some transitions that occur in many positive examples. These transitions may have a discriminating power over the training set. If two states separated by a transition are merged, the discriminating effect of this transition is cancelled. Even if a series of transitions are common to many positive examples, nothing will prevent the merging of the states between these transitions just because they are frequent. The only criterion that prevents state merging is the refutation of the negative examples when applied to the resulting DSFM. Chapter 5 presents a discussion of this characteristic supported with some experimental results.

Chapter 5 – Experiment Setup and Results

5.1 – Introduction

This chapter is divided in two main parts. The first part is a description of the "document classification learning" object-oriented framework used for this project. The document classification object-oriented framework is an extensible object-oriented design built to easily incorporate various feature extraction techniques and various learning algorithms with little change to the code. With this framework, it is possible to test many possible combination of clustering, feature extraction and machine learning algorithms. The framework has been built and adapted for all the techniques presented in chapters 3 and 4. The second part of this chapter presents and discusses the experimental results of the techniques presented in chapters 3 and 4. Three training tests have been selected for the experiments: 1) Classification of e-mail documents versus non-e-mail documents. 2) Classification of BASE64 encoded documents versus non-BASE64 encoded documents. 3) Classification of images in the GIF format versus images in the BITMAP format.

5.2 – Document Classification Learning Framework

An object-oriented framework is, by definition, a set of cooperating classes that make up a reusable design for a specific class of software [JF88] [DE89]. The framework presented in this section was designed to allow multiple combinations of complementary algorithms that have for a task, as a whole, to train classification model from training examples. The framework includes all the techniques presented in chapters 3 and 4. The framework is divided in three parts: alphabet

streaming, feature extraction and classification model induction. We used the unified modeling language (UML) notation to describe the framework [LR98].

5.2.1 – Alphabet Streaming

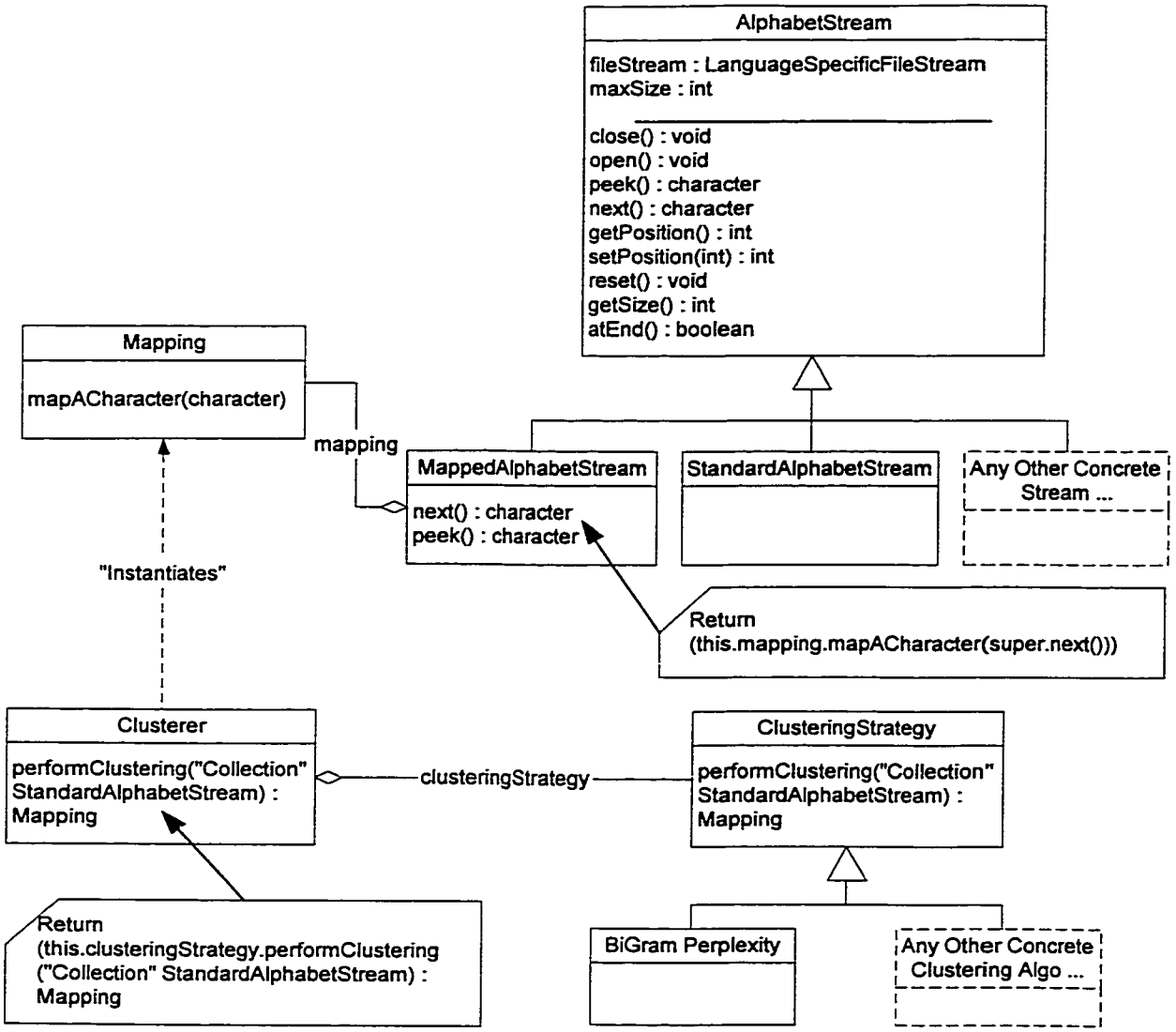


Figure 5.1 : Alphabet Streaming Design

This part of the framework has for its objective to define the objects that will read the files representing the training documents. As it was mentioned in section 3.3, a redefinition of the alphabet used to interpret the training documents might be very useful. However, we want the

alphabet definition to be transparent to the rest of the system. Also, some classes are needed to implement the clustering algorithms. These classes are responsible for defining an alphabet reduction from document examples. The following describes the classes that address those issues:

- **Alphabet Stream.** This class is an abstract class. This means it cannot have any instances. It provides some generic functionality and dictates the methods that the sub-classes should understand. All the external applications that require file accessing will have to use the methods defined by this class. The methods implement the following functionality:
    - Open and close a file.
    - Get the file size.
    - Read the next character.
    - Peek at the next character without changing the file's position.
    - Get, set or reset the file's position.
    - Verify if the file's position is at the end.

  Each sub-class of the Alphabet Stream object will have a pointer to an external file. Also, a variable "maxSize" could be defined to indicate the maximal size to truncate the file. This class could be extended to implement any new streaming strategy.
- **Standard Alphabet Stream.** This is a concrete class extending the super-class "Alphabet Stream". It implements all the methods specified by its super-class. All the characters accessed by this class will use the extended ASCII alphabet of 256 characters.
- **Mapped Alphabet Stream.** This is also a concrete class extending the super-class "Alphabet Stream". It implements all the methods specified in its super-class. However the alphabet considered is not necessarily the extended ASCII alphabet. Each instance of a "Mapped Alphabet Stream" object will point to an instance of a "Mapping" object.

Every time a character is read, it will first be read using the extended ASCII code. Then the "Mapping" object will be used to return the mapped symbol of the ASCII character.

- **Mapping.** This object represents a character mapping. It implements one method that takes an ASCII character in parameter and returns the mapped symbol of this character. The implementation of the mapping is hidden to the other objects. The "Mapping" object could be instantiated using an Abstract Factory pattern [GH95]. In our case, the "Mapping" objects are instantiated with the "Clusterer" object.
- **Clusterer.** This object has for role to create "Mapping" objects using a clustering algorithm. It implements a method that takes a collection of "Standard Alphabet Stream" objects as parameter and returns a "Mapping" object. The clustering is performed on the streams passed as parameter. We have implemented a Strategy pattern [GH95]. This means that we have separated the objects that implement the functionality from the object that provide the functionality to the other objects. Consequently, the method "performClustering" simply delegates the clustering task to a certain "Clustering Strategy" object. Every "Clusterer" object has a pointer to a "Clustering Strategy" object.
- **Clustering Strategy.** This object is an abstract class. It specifies the method that all its sub-classes should implement. The only method specified is "performClustering". This class can be extended to implement any clustering algorithm.
- **Bi-Gram Perplexity.** This is a concrete implementation of the super-class "Clustering Strategy". It performs the symbol clustering algorithm described in section 3.3.1.

5.2.2 – Feature Extraction

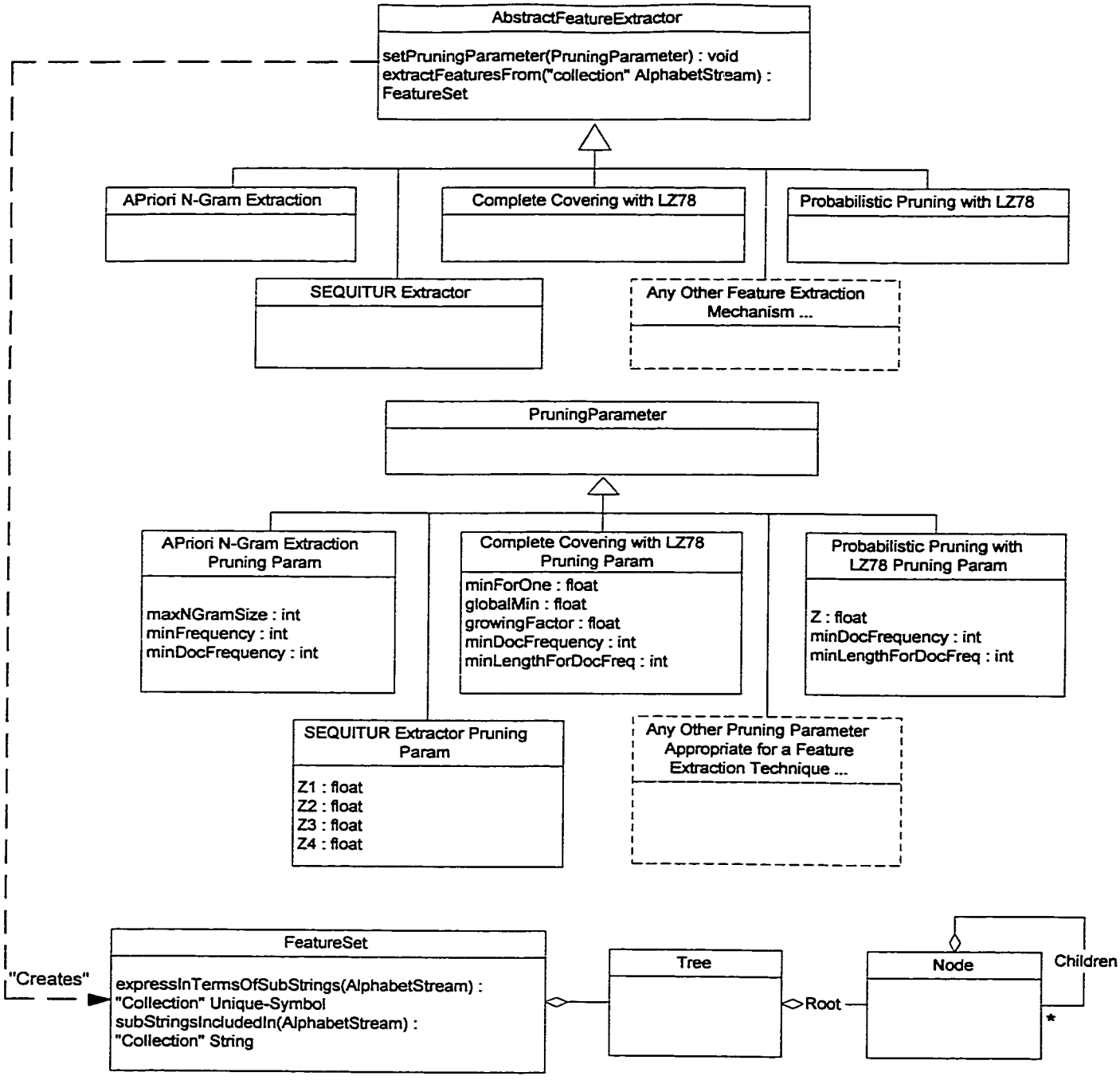


Figure 5.2 : Feature Extraction Design

The objective of this part of the framework is to define a common interface to a set of classes that represent different feature extraction techniques. All the feature extraction techniques should be

interchangeable in the system without having to modify any code. The end result of a feature extraction algorithm is a "Feature Set" object. This object contains all the extracted features and will be reused by the objects that belong to the classification model induction category. The following describes the objects of this part of the framework:

- **Abstract Feature Extractor**. This is an abstract class. It cannot be instantiated. This class defines the methods that all the feature extractors (i.e. the sub-classes) should implement. The first method defined is "setPruningParameter". This method takes an object of the type "Pruning Parameter". In fact, all the sub-classes will have to cast this object into their own pruning parameter format. The second method defined is "extractFeaturesFrom". This method takes a collection of "Alphabet Stream" objects as parameter. Features will be extracted from the streams passed in parameter. The returned value of this method is a "Feature Set" object that contains the extracted feature. Any new feature extractor will have to extend this object and implement its methods.
- **Apriori N-Gram Extraction, Complete Covering with LZ78, Probabilistic Pruning with LZ78 and SEQUITUR Extractor**. These classes are concrete feature extractors extending the super class "Abstract Feature Extractor". They implement all the methods specified in the super-class "Abstract Feature Extractor". It is important to mention that their implementation of the method "setPruningParameter" will have to cast the parameter of type "Pruning Parameter" into their own appropriate type. As an example, the "SEQUITUR Extractor" object will cast the parameter of type "Pruning Parameter" into its own type, which is "SEQUITUR Extractor Pruning Param".
- **Pruning Parameter.** This is the super-class of all the possible pruning parameters. It is used as an interface for type compatibility and polymorphism.

- **Apriori N-Gram Extraction Pruning Param, Complete Covering with LZ78 Pruning Param, Probabilistic Pruning with LZ78 Pruning Param and SEQUITUR Extractor Pruning Param**. These classes represent the pruning parameters of all the different feature extractors. They will be passed as parameter to the method "setPruningParameter" defined by the "Abstract Feature Extractor" object.
- **Feature Set.** This object contains all the features extracted by a feature extractor. The data structure used to store the features is independent from the "Feature Set" object. In our case, we used a tree structure but it could be something else. The "Feature Set" object points to the data structure that contains its features. This is an example of the Bridge pattern [GH95]. The "Feature Set" object provides some utility methods. First, we implemented the method "expressInTermsOfSubStrings" which takes an "Alphabet Stream" object as parameter and returns an ordered sequence of the sub-strings unique identifiers included in the stream without considering overlapping. Second, we implemented the method "subStringsIncludedIn" which takes an "Alphabet Stream" object as parameter and returns a collection of the sub-strings included in the stream considering overlapping.
- **Tree and Node**. These objects represent the tree structure used to store the features. An implementation of a tree structure is proposed by the Composite pattern [GH95].

5.2.3 – Classifier Induction

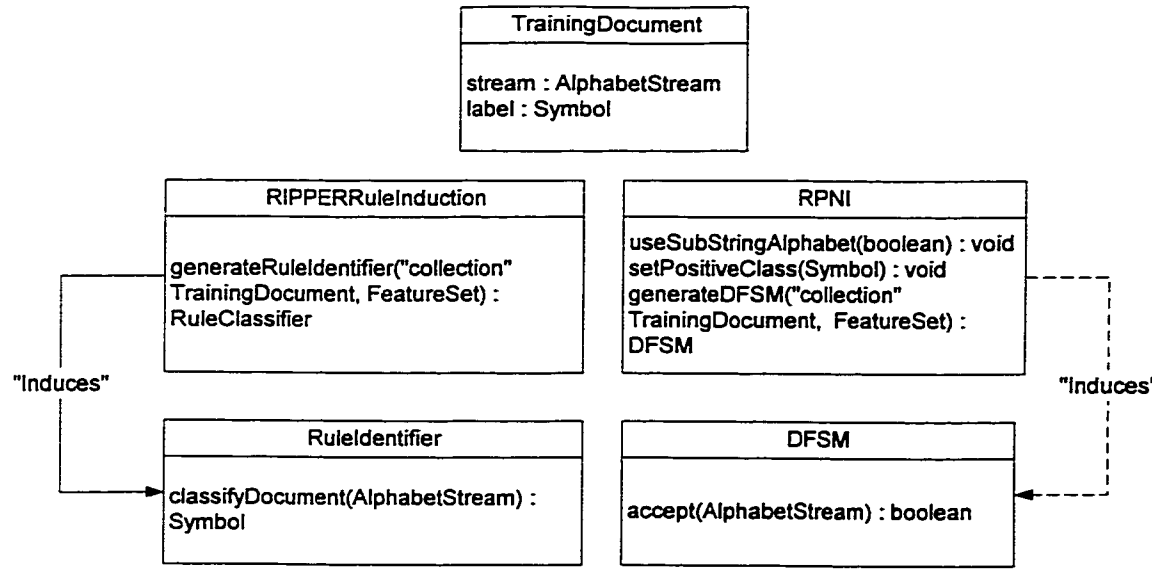


Figure 5.3 : Classifier Induction Design

This last part of the framework has for objective to induce a final classification model. The induction of the classification model can be done using various techniques. Generally, these techniques require training examples. We implemented a "Training Document" object that would be accepted by any classifier induction. Also, a "Feature Set" object can be provided to any classification induction algorithm. These features will be associated with each training document with their proper values. The end result object produced by classification induction is the classification model itself that will be used to do future classification. The following description explains the objects used in this part of the framework:

- **Training Document.** This object is simply an association of an "Alphabet Stream" object representing a training document with a label identifying the class of the document.

- **RIPPER Rule Induction.** This is a wrapper around the RIPPER algorithm. It takes as parameter a collection of "Training Document" objects and a "Feature Set" object. The training documents are provided to RIPPER using the features. The resulting rules provided by RIPPER are parsed in and returned as a "Rule Identifier" object.
- **RPNI.** This object implements the RPNI grammatical inference algorithm. It requires a collection of "Training Documents" objects. Optionally, it can accept a "Feature Set" object if the state transitions use sub-strings instead of single characters. The result of the inference is an acceptor deterministic finite state machine represented by the object "DFSM".
- **Rule Identifier.** It contains the rules generated by RIPPER. Given a certain "Alphabet Stream" it will return the class that corresponds to the first matching rule.
- **DFSM.** It is an acceptor deterministic finite state machine. Given a certain "Alphabet Stream" it returns true if it is accepted by the DFSM or false otherwise.

5.2.4 – Cooperation of the Objects

The framework is divided in three parts: alphabet streaming, feature extraction and classifier induction. Each object of a certain part could be combined with one or many objects from the two other parts to build a different system. Figure 5.4 illustrates various possibilities:

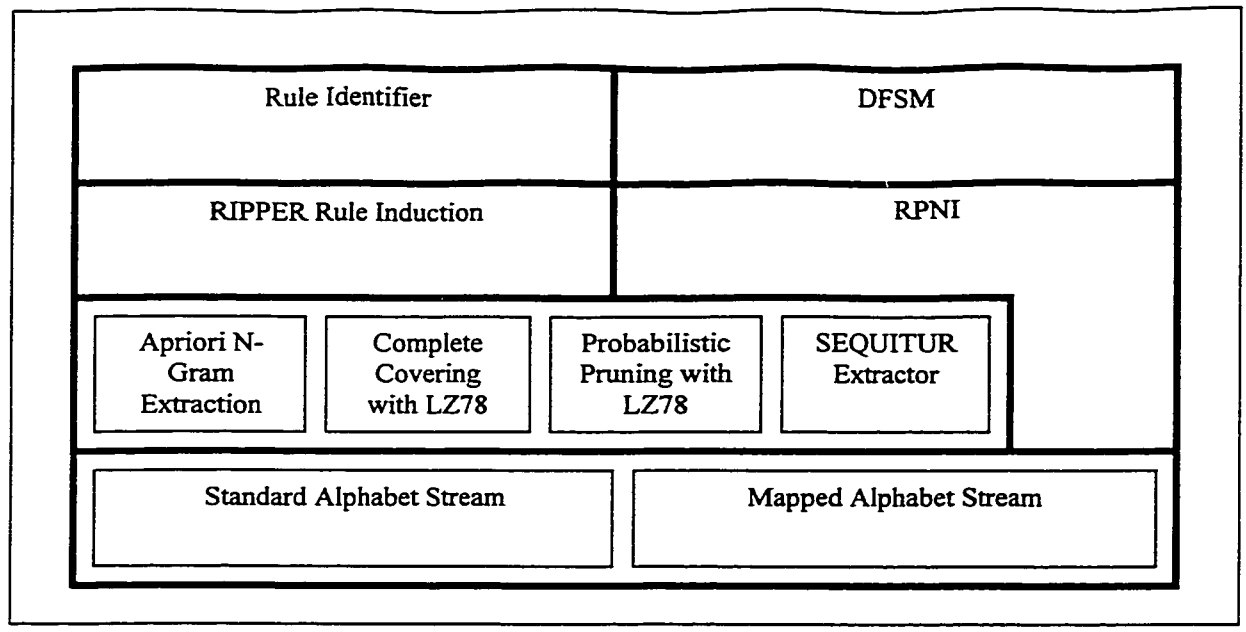


Figure 5.4 : Cooperation of the Framework Objects

5.3 – Results and Discussion

The objective of this section is to present some experimental results in order to demonstrate the strengths and weaknesses of the techniques presented in chapter 3 and chapter 4. We have chosen three test domains to apply our feature extraction and learning algorithms. Those three domains are:

1. Classify e-mail documents versus documents that are not considered e-mail. Damaged or incomplete e-mails are not considered e-mails.
2. Classify BASE64 encoded documents versus documents that do not contain any BASE64 encoded data.
3. Classify GIF images versus bitmap images.

The following figures illustrate the structure of the documents.

```
Date: Wed Jan 26 12:49:08 EST 2000
Status: RO
From: mjfonta@apollo.p.csecse (Martin J. Fontaine)
To: jsjohns
Subject: LINUX Machine
Classification: Unclassified
X-Mailer: VisualMail (Version 3.2.7.4 31dec97)

Hi Stuart,
I just wanted to know if you have found a PC
to install LINUX on???

Martin Fontaine
```

Figure 5.5 : E-mail Example

```
J9rH0bCiFzq/aWksRHSLPEXXFq4NQu3LaYYhontGtApsiOjWyrwzRHQyon1XZ4jojhPt95QMFd2x
on03yFHRrZVe5eEuqgG78qt8f81D3TXTiI36W0JXuseOjG5h7qYuh7O0UP/S3/0trvXgY0V740bF
OsUlTwFFRrcomVyFIqNb92QKMDq6Y0X7bmfYpBJ5fLSZsb0sNE2a8SYn8vpu24xvKKH2kWl/CVfi
+nfNe2jDXfIUXUdBSLey0GeTESXdIlFR44qS7ljRroeipFvec9sBdl291cVVlHSyon1tDeZyMkV9
aCjpVj6ZEhsHMNf9e9Vsimo9+DjRjtcWKZ3hmK9ONKR0htCHCEjpDDmyIuzv2Sm0f/NrmNYSNyUq
+6tQzLWvBmGgKOksFfYJeP0IbK1rHSndcaL9rnmgpTtWtC/yQEt3rGjnQ9HSGamKrm6gpTtGtO3F
RUpnpee2JuZa3vPZX16duDNF2GOiCV3pTkyUdJaUPlE/tBM3P7mJkM6o6wOKkO750CrwQkhnqbBP
wCjpDKkPaBNzDcd83Z+Fks4oQai+hV2/0PQ+mAslnbGmItQQ0hknUxTXENJZck3ZKEK6a0JLW3R0
```

Figure 5.6 : BASE64 Encoded Document Example

```
GIF89aŒ □ €ÿ ÿÿÿÀÀÀ!ù□□ □ ,  Œ □  □ÎŒ Ší□£œÚ‹ÏÚ
,û□†"Å•cÕ□éŠN€cŠé°□3vCo'$õkÈá€4"qh;êv<f#□íï¤ÁZÂyEX□Ë
(•SõŠ□_ì4«%W×O`çÍèÁÙíÇ□□=ãÝj}úÊƒ6µu□xwÈÇ&˜¸ç‡H7"□§DY
'        æ¯        iÈh×X8□Ú7ø×d¦HªÇÉªÄðÊ)8Ù(ø§Q›âš›Ú»Ië
□é□J:êg□ÔY£{□‡Y&□»ðùÜ÷□fÝ1□|#Ãò
-.q*^n~>è□¾ÎNQ ;
```

Figure 5.7 : GIF Image Example

```
BMB□    B ( -    □ □    □ □

□
 □  □   €€ ÀÀÀ ÿÿÿ !□□!□□!□□!□□!□□□
!□□!□□!□□!□□!□□ !□□!□□!□□!□□!□□□!□□!□□!□
□!□□!□□□!□□!□□!□□!□□ !□□! □□!□□!□□!□□□
!□□!□□!□□!□□!□□□!□□!□□!□□!□□!□□ !□□!□□!□
□!□□!□□□!□□!□□!□□!□□!□□□!□□ !□□!□□!□□!□□
!□□!□□!□□!□□!□□□!□□!□□!□□!□□!□□□!□□!□□!□
```

Figure 5.8 : Bitmap Image Example

For each testing domain, we have built a training set containing and equal number of positive examples and negative examples. A positive example is a document that belongs to the class that we want to identify. In our case the classes that we want to identify are e-mail, BASE64 and GIF images. A negative example is any kind of document except the type of document that we want to identify. Consequently, we have built a positive example set and a negative example set for each testing domain. The negative training sets have been built with various kinds of documents like ms-word documents, damaged e-mails, uu-encoded documents, etc. However, for the GIF versus bitmap domain, we used only bitmaps in the negative training set.

The e-mail classification problem is a problem that could apparently be done by identifying discriminating patterns that belong to e-mail documents and do not belong to other documents. Generally, a series of patterns containing 8 characters or less each should be sufficient to obtain good results. The BASE64 classification problem is different. There is no apparent patterns of a fixed size that belongs to BASE64 encoded documents and does not belong to other documents. The characters appear in a pseudo-random order. However, we can recognize, in an undamaged document, that a strict alphabet of 64 characters is used exclusively and that all the lines have the same number of characters. The last problem domain, classifying GIF images over bitmap images, should be an easy task. In fact, all the GIF images begin with the string "GIF" and all the bitmap images start with the string "BMB".

All the observations of the previous paragraph are observations that a human would notice about these data. We will see in the next paragraph if the machine learning techniques will be able to find some properties to classify accurately those three data types as well as a human.

5.3.1 – Experiment Setup and Results

In chapter 3, we presented four feature extraction techniques:

- Apriori n-gram extraction.
- Complete covering approach with LZ78 compression.
- Probabilistic pruning with LZ78 compression.
- The hierarchical grammar with SEQUITUR.

Among the classification model induction presented in chapter 4, we have chosen the following for our experiment:

- Rule-based learner with RIPPER.
- Grammatical inference with RPNI.

We have divided our experiment in two distinct parts. First we have tested all the feature extraction techniques on the three test domains and induced the classification model with RIPPER. Second, we have tested many aspects of grammatical inference on the three testing domains.

5.3.1.1 – Different Feature Extraction Techniques Tested with RIPPER

The basis of our experiment is to try each feature extraction technique with different pruning parameters for each testing domain. The features are extracted on all the training documents of the class that we are trying to identify (E-Mail, BASE64 and GIF). Also, we tested the effect of clustering with the BASE64 classification problem when extracting the features. The clustering is performed using the algorithm presented in section 3.3.1. The clustering algorithm was performed on a BASE64 encoded document of 3000 characters. The training documents examples and counter-examples are expressed in terms of the extracted features and provided to RIPPER. Then we use RIPPER to generate classification rules. For our experiment, we used 5-fold cross validation to

measure the accuracy of the classifier. This means, that we separated the data in 5 subsets. The algorithm will be trained using 4 subsets and the rules generated will be tested on the remaining subset. This test is preformed 5 times in order to test all 5 subsets. The global accuracy of the test is the average accuracy of the 5 tests. The accuracy is given with an error rate percentage and a certain confidence interval. The following tables present the experiment parameters and the results obtained without any explanations. The following section will discuss in detail the results presented in the tables.

Domain	Number of Positive Examples	Number of Negative Examples
E-Mail	100	100
Base64	66	66
GIF vs. Bitmaps (BMP)	150	150
Note: All the training documents will be truncated if over 3000 characters.		

Table 5.1 : Global Set-up

Apriori Feature Extraction Results

The pruning mechanism of the Apriori feature extraction technique (see section 3.2) requires three parameters:

- **Minimum Frequency**. It specifies the minimum number of occurrences of an n-gram in the union of all documents. This value is expressed with an integer.
- **Minimum Document Frequency.** This value is an occurrence threshold for the number of documents in which a n-gram must occurs. This value is expressed as a percentage of the size of the positive training set.
- **Maximal N-gram Size.** This corresponds to the maximal number of characters in a n-gram. We decided to fix this value to 8 for our experiment.

Table 5.2 shows results from a 5-fold cross validation using RIPPER.

Min. Freq.	Min. Doc. Freq. (%)	# Features	Confidence Interval from RIPPER (%)	Accuracy (%)
E-Mails				
3	30	1759	1.90	**96.00**
5	50	816	1.63	**98.00**
7	65	559	1.63	**97.00**
10	75	456	1.05	**98.00**
Base64				
3	30	3139	2.54	**91.59**
5	50	273	4.14	**88.02**
7	65	84	2.13	**92.53**
10	75	69	0.90	**91.65**
GIF vs. BMP				
3	30	298	0.00	**100.00**
5	50	268	0.00	**100.00**
7	65	168	0.00	**100.00**
10	75	65	0.00	**100.00**
Base64 with 2 clusters				
7	65	51	1.37	**42.42**
Base64 with 3 clusters				
7	65	52	5.51	**68.79**
Base64 with 10 clusters				
7	65	1174	3.32	**94.07**
Base64 with 3 clusters + ASCII alphabet				
7	65	136	2.13	**92.53**

Table 5.2 : Apriori Feature Extraction Results

Feature Extraction Results with Complete Covering Approach and LZ78

The pruning mechanism of the complete covering approach (see section 3.4.2) requires five parameters:

- **Growth Factor**. This value represents the importance given to long sub-strings over shorter sub-strings. It is a parameter of the pruning function.
- **Minimum Ratio for Sub-String of Size 1.** The minimum acceptance ratio for a sub-string of size one determines the value of the pruning function when the size of the sub-string equals one.
- **Global Minimum Ratio.** Minimum acceptance ratio for sub-strings of any size. We have decided to fix this value to 6×10^{-6}.

- **Minimum Length for Document Occurrence Pruning.** Minimum sub-string size to apply the pruning based on document occurrences. We have decided to fix this value to 3.
- **Minimum Document Occurrences.** Minimum number of documents in which a sub-string must appear. This value is expressed as a percentage of the size of the positive training set.

Growth Factor	Min. for One (%)	Min. Doc. Occurrence (%)	# Features	Confidence Interval from RIPPER (%)	Accuracy (%)
E-Mails					
3	6.5	30	82	1.05	**98.00**
2.5	7.0	50	15	1.43	**96.50**
2	7.5	65	7	1.63	**93.00**
1.5	8.0	75	6	1.63	**93.00**
Base64					
4	7.5	23	57	4.46	**83.30**
3.5	8.0	26	34	2.99	**74.29**
3	8.5	30	16	4.12	**71.37**
2.5	9.0	32	5	3.44	**74.23**
GIF vs. BMP					
3	6.5	30	104	0.00	**100.00**
2.5	7.0	50	24	0.00	**100.00**
2	7.5	65	9	0.00	**100.00**
1.5	8.0	75	4	0.00	**100.00**
Base64 with 2 clusters					
4	7.5	23	479	0.86	**99.23**
Base64 with 3 clusters					
4	7.5	23	651	1.02	**98.52**
Base64 with 10 clusters					
4	7.5	23	960	1.94	**95.55**

Table 5.3 : Complete Covering Feature Extraction Results

Feature Extraction Results with Probabilistic Pruning Approach and LZ78

The pruning mechanism of the probabilistic approach (see section 3.4.3) requires five parameters:

- **Z.** This value determines the acceptance threshold according to the central limit theorem. It is in fact a multiplier of the standard deviation to determine the acceptance threshold from the mean.

- **Minimum Length for Document Occurrence Pruning.** Minimum sub-string size to apply the pruning based on document occurrences. We have decided to fix this value to 3.
- **Minimum Document Occurrences.** Minimum number of documents in which a sub-string must appear. This value is expressed as a percentage of the size of the positive training set.

Z	Min. Doc. Occurrence (%)	# Features	Confidence Interval from RIPPER (%)	Accuracy (%)
E-Mails				
1.645	30	1191	1.37	**97.00**
2	50	461	1.05	**98.00**
2.25	65	311	1.12	**98.50**
2.5	75	245	1.05	**98.00**
Base64				
1.645	30	177	4.30	**85.66**
2	50	8	1.43	**76.54**
2.25	65	7	7.09	**77.58**
2.5	75	2	1.89	**75.05**
GIF vs. BMP				
1.645	30	160	0.00	**100.00**
2	50	54	0.00	**100.00**
2.25	65	32	0.00	**100.00**
2.5	75	19	0.00	**100.00**
Base64 with 2 clusters				
1.645	30	282	0.86	**99.23**
Base64 with 3 clusters				
1.645	30	209	0.86	**99.23**
Base64 with 10 clusters				
1.645	30	2785	1.56	**94.01**

Table 5.4 : Probabilistic Pruning Feature Extraction Results

Feature Extraction with SEQUITUR

The pruning mechanism of SEQUITUR (see section 3.5) requires four parameters:

- **Z1.** This value determines the frequency occurrence threshold for the rules according to the central limit theorem.

- **Z2.** This value determines the "pure size" threshold for the rules according to the central limit theorem.
- **Z3 and Z4.** Like Z1 and Z2 but they are applied in conjunction.

Z1 and Z2	Z3 and Z4	# Features	Confidence Interval from RIPPER (%)	Accuracy (%)
E-Mails				
1	-0.25	4314	1.63	**95.50**
1.25	-0.15	2050	1.68	**96.50**
1.5	0	526	1.05	**98.00**
1.645	0.15	446	0.88	**97.50**
Base64				
1	-0.25	3789	1.63	**91.65**
1.25	-0.15	3627	4.45	**87.09**
1.5	0	474	2.53	**93.13**
1.645	0.15	394	0.85	**96.98**
GIF vs. BMP				
1	-0.25	3500	0.37	**99.67**
1.25	-0.15	3478	0.37	**99.67**
1.5	0	655	0.00	**100.00**
1.645	0.15	402	0.00	**100.00**
Base64 with 2 clusters				
1	-0.25	22	1.69	**93.96**
Base64 with 3 clusters				
1	-0.25	21	2.21	**93.90**
Base64 with 10 clusters				
1	-0.25	246	4.23	**89.51**

Table 5.5 : SEQUITUR Feature Extraction Results

5.3.1.2 – Tests with Grammatical Inference

We have performed many experiments using the RPNI algorithm. The idea of the experiments is to change the alphabet of the DFSM used for induction. We have seen many ways to change this alphabet in chapter 3. First, the extended ASCII alphabet could be used. Second, we can reduce the extended ASCII alphabet by applying a symbol clustering with a certain number of clusters as described in section 3.3.1. Third, it is possible to use a collection of sub-strings as an alphabet for the

DFSM as presented in section 4.3.2.3. For this experiment, we used the feature probabilistic pruning with LZ78 extraction technique to select the features used as an alphabet.

For the experiment using a character-based alphabet, we used a reduced number of training examples, because each test takes a considerable amount of resources to run. We did not use cross validation in these cases. We simply tested the resulting acceptor DFSM on the remaining examples. For the sake of comparison, we also ran RIPPER on the same smaller training set using the probabilistic pruning approach for the feature extraction. Results are presented in table 5.8.

For the experiments using a sub-string-based alphabet, we used the same number of training examples as the experiment with RIPPER. We also used a 5-fold cross validation in these cases.

Domain	# Positive Training Data	# Negative Training Data	% False Positives	% False Negative	# States in PTA	Accuracy (%)
Character by Character with Extended ASCII Alphabet						
E-Mail	20	20	36.25	8.75	39006	**55.00**
BASE64	20	20	4.35	18.48	58447	**77.17**
GIF vs. BMP	20	20	0.00	8.46	20902	**91.54**
Character by Character with a Reduced Alphabet with 2 Clusters						
BASE64	20	20	19.57	2.17	54603	**78.26**
Character by Character with a Reduced Alphabet with 3 Clusters						
BASE64	20	20	0.00	1.09	54603	**98.91**
Character by Character with a Reduced Alphabet with 10 Clusters						
BASE64	20	20	8.70	26.09	58424	**65.21**

Table 5.6 : Character by Character RPNI Induction Results

Domain	# Positive Training Data (5-fold cross validation)	# Negative Training Data (5-fold cross validation)	Average % False Positives	Average % False Negative	Average # States in PTA	Average Accuracy (%)
Sub-String Alphabet Obtained with Probabilistic Pruning (Z = 1.645, minLengthDoc = 3, minDoc = 50%)						
E-Mail	80	80	15.50	12.00	11041.4	**72.5**
BASE64	53	53	18.94	33.33	532.24	**47.73**
GIF vs. BMP	120	120	0.00	0.00	558.4	**100.00**
Sub-String Alphabet Obtained with Probabilistic Pruning (Z = 1.645, minLengthDoc = 3, minDoc = 50%) With a Reduced Alphabet of 3 Clusters						
BASE64	53	53	0.00	0.76	1409.0	**99.24**

Table 5.7 : Sub-String Alphabet RPNI Induction Results

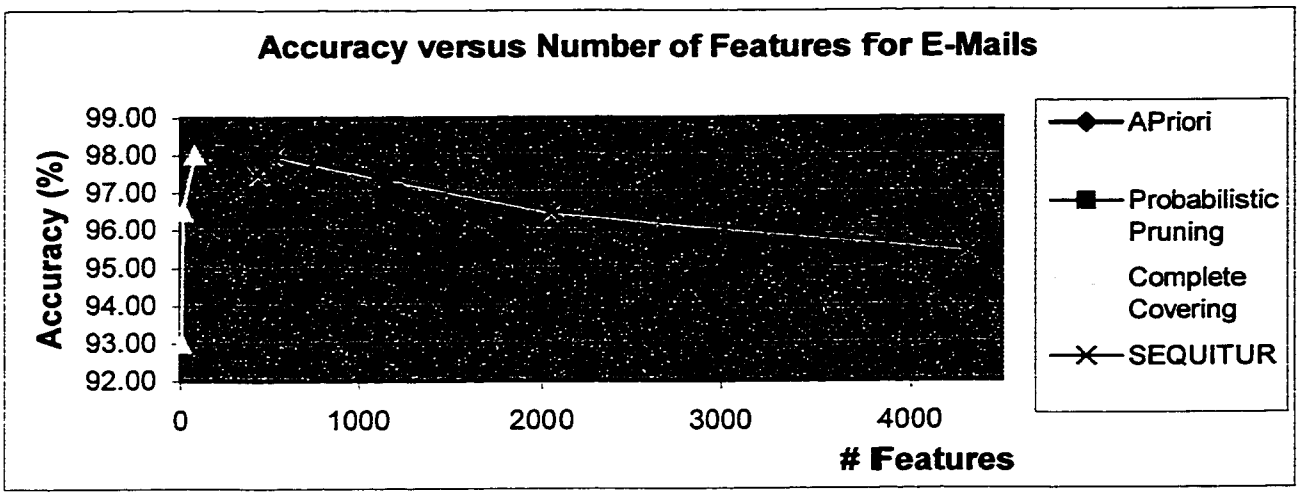
Z	Min. Doc. Occurrence (%)	# Features	Confidence Interval from RIPPER (%)	Accuracy (%)
E-Mails				
2.25	65	294	1.92	**93.75**
Base64				
1.645	30	113	5.03	**64.13**
GIF vs. BMP				
1.645	30	91	0.85	**98.08**
Base64 with 3 clusters				
1.645	30	169	1.09	**98.91**

Table 5.8 : Probabilistic Pruning Feature Extraction Results with RIPPER and a Smaller Training Set

5.3.2 – Effect of the Number of Features on the Classifier Accuracy

This section has for its objective to analyze the effect of feature pruning on each feature extraction technique used. We will see that the number of features used not only affects the time required during the learning phase, but it also affects the accuracy of the classifier. The analysis of the effect of the number of features on the accuracy of the classifier has been done separately for the three domains: e-mails, BASE64 and GIF versus bitmaps.

Graph 5.1 compares the effect of the number of features on the accuracy of the classifier for the four feature extraction techniques used on the e-mail problem domain. We can observe that the pruning of the complete covering approach is more severe than the other techniques. Nevertheless, this feature extraction technique obtained fairly good results considering the small number of features generated. For the case of the complete covering approach, the accuracy has increased with the number of features.

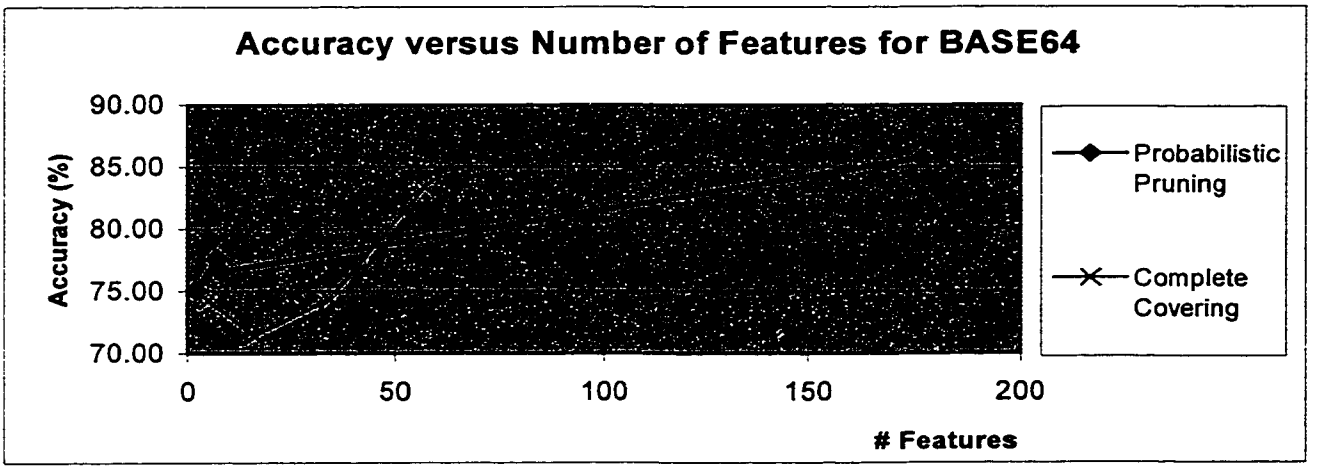


Graph 5.1

The three other techniques have generated a greater number of features, which allow us to do a better analysis. Intuitively, one would be tempted to believe that the accuracy of the classifier increases with the number of features simply because there are more possible attributes that could be used by the machine learning algorithm. On the other hand, if a very small number of features were used, it would be difficult for the learning algorithm to find a discriminating set of attributes that identify the target concept. In fact, we have observed that the accuracy of the classifier increases with the number of features but only up to a certain point. After that point, the accuracy of the classifier decreases. In this specific case, all the learning techniques did worse when generating more than 1000 features compared to generating less than 1000 features. Obviously, we believe that a certain minimum number of features are required to have an acceptable accuracy. As an extreme example, if only one feature were used, the accuracy would be perfect if and only if this feature appears in all the positive examples and no negative examples. On the other hand, if too many features are generated, the risk of over-fitting [MI97] the positive training set increases. Again, as an extreme example, if a million features are available, it is certainly possible to find a sequence of features that would fit all the positive examples and no negative examples because there are so many possible combinations.

However, it is not guaranteed that this feature combination will be general enough to classify an unseen document accurately because the selected sequence of features is too specific to the training set. Even if RIPPER implements a pruning algorithm that reduces the over-fitting effect, we think that an excessive number of features still affects the accuracy of the classifier.

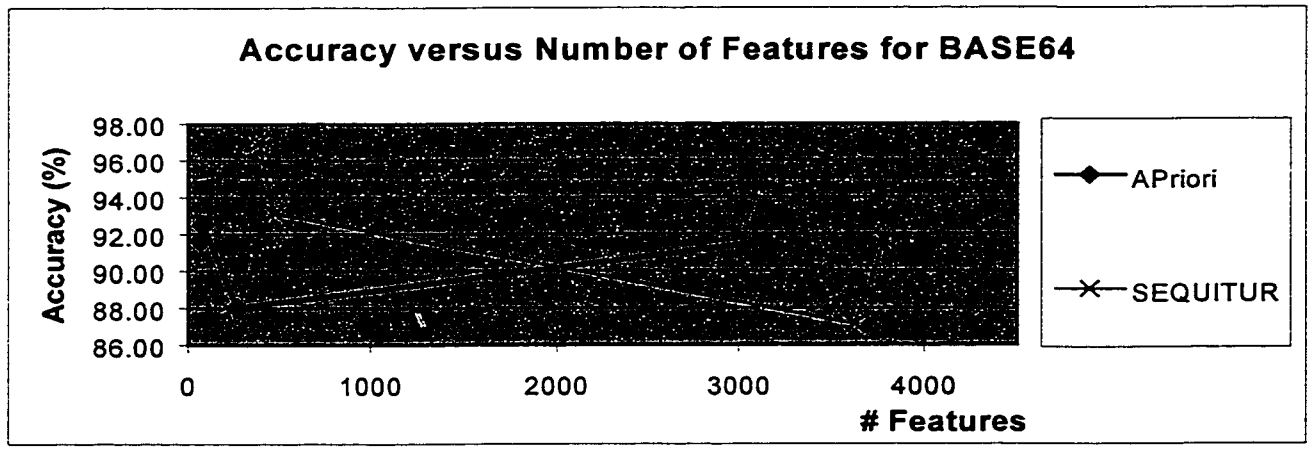
Graphs 5.2 and 5.3 illustrate the effect of the number of features on the accuracy of the classifier for the BASE64 problem domain. We displayed the results on two graphs because the number of features generated was significantly different for two of the feature extraction techniques. The results presented for the BASE64 problem are different than those obtained for the e-mail problem. As opposed to e-mail documents, it is important to re-mention that the characters of a BASE64 encoded document appear in a pseudo-random order. This means that there is no single clear set of sub-strings that are proper to all the BASE64 encoded documents and do not appear in other documents. It is harder to find a regular tendency in those results.



Graph 5.2

The complete covering approach and the probabilistic pruning approach extracted fewer features simply because the probability of finding a recurrent and frequent pattern among the documents is

very low. Those two techniques extract features based on the frequency across documents and within documents, in proportion with the size of the training set. Also, they tend to prefer longer sub-strings. There are not a lot of frequent and redundant sub-strings of a reasonable size in the BASE64 training set. The Apriori feature extractor (graph 5.3) extracted more features because its acceptance threshold is only based on an integer value without considering the size of the sub-strings. By chance, it is possible to consider redundant small sub-strings. SEQUITUR has also accumulated a large quantity of features. We know that SEQUITUR produces a lot of small sub-strings. In fact it produces all the possible 2-grams observed in the documents. SEQUITUR does not contain a pruning parameter acting across documents like the three other techniques. We think that it is possible that over all the characters observed, some short sub-strings have been considered frequent by SEQUITUR.



Graph 5.3

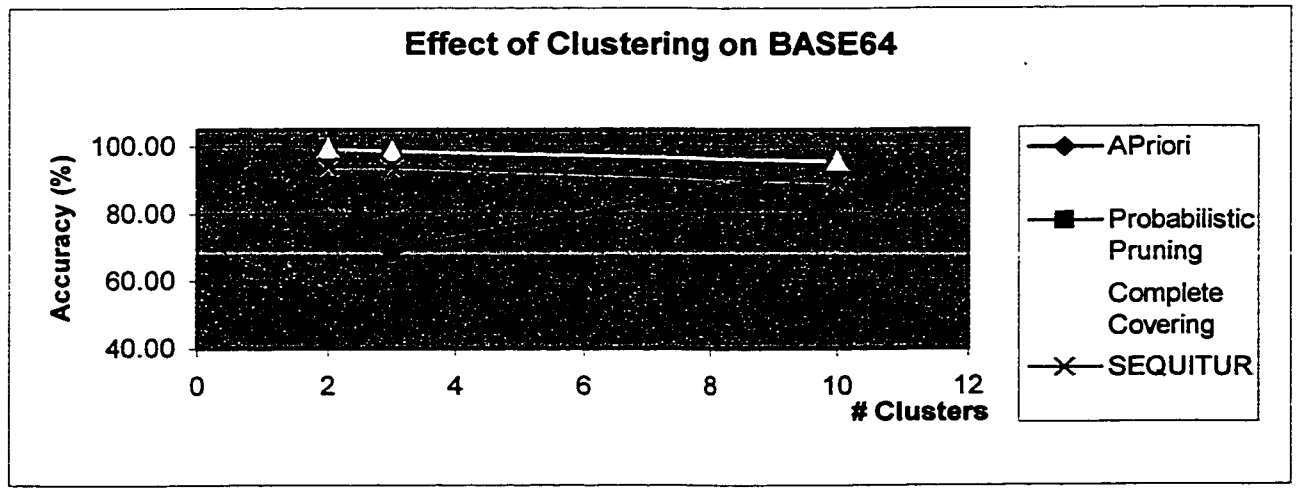
A greater number of features seems to have improved the accuracy for the probabilistic pruning approach and the complete covering approach. However, their accuracy is definitely below the accuracy provided by Apriori and SEQUITUR feature extraction techniques. We think that the accuracy is lower because these techniques did not extract a sufficient number of short sub-strings.

It is harder to isolate the effect of the number of features on the accuracy for the Apriori extractor and SEQUITUR. The best accuracy is observed for a low number of features. When the number of features increases, the accuracy drops below 88% and raises back to 92%. It is difficult to draw a conclusion from these results considering the confidence interval of those results.

For the GIF images versus bitmap images, the accuracy was always 100% except for two cases with SEQUITUR. The accuracy was good because there is a clear discriminating pattern that separates GIF images from bitmaps images. SEQUITUR did not reach perfection in two cases because the number of features were 3478 and 3500. As it was mentioned before, an excessive number of features increases the risk of over-fitting.

5.3.3 – Effect of Clustering on the Classifier Accuracy

In this section, we analyze the effect of clustering on the performance of the classifier for the BASE64 problem. We have tested the feature extraction techniques on a clustered alphabet containing 2, 3 and 10 symbols (or clusters).



Graph 5.4

Using 2 or 3 clusters gave approximately the same results and using 10 clusters decreases the accuracy for all the feature extraction techniques except for the Apriori feature extractor. The difference between the Apriori feature extractor and the other techniques is the fact that it only considers sub-strings up to a certain maximal size. The other techniques will build new sub-strings when repetition occurs without any maximal sub-string size.

Running the clustering algorithm on a BASE64 document with 2 clusters returned an alphabet mapping that maps the 64 characters plus the line feed into one cluster and all the other extended ASCII characters into the other cluster. If this clustering is performed, a BASE64 document is seen as a long string containing identical symbols. If three clusters are used, the 64 characters are mapped into the first cluster (symbol 1), the line feed into the second cluster (symbol 2) and all the other extended ASCII characters are mapped into the third cluster (symbol 3). If a clustering with 3 symbols is used, then a BASE64 document is seen as a collection of equal sequences containing the symbol 1, each sequence being separated by the symbol 2. Consequently, the algorithms that create longer sub-strings based on repetition had the opportunity to build longer sub-strings.

On the other hand, the Apriori feature extractor was not able to consider sub-strings that are longer than 8 characters. This characteristic explains the poor performance of the Apriori feature extractor versus the other techniques. The only way to increase the accuracy of the Apriori feature extractor was to increase the number of clusters. By increasing the number of clusters, some generality is lost. This is why the three techniques based on sub-strings repetition lost some accuracy when using 10 clusters. It was more difficult to identify sub-strings repetition with a higher number of clusters. However, using 10 clusters increased the accuracy of the Apriori feature extractor. For this feature

extraction technique, using 10 clusters instead of 3 increased the number of features from 52 to 1174. It is probable that 52 features shorter than 8 characters was not enough to induce an accurate classifier. Even if the Apriori feature extractor did not performed as well as the other techniques when using a clustered alphabet, it did better than without clustering.

5.3.4 – Comparison of the Feature Extraction Techniques

In this section, we compare the performance of the feature extraction techniques in terms of accuracy and features generated. We do this comparison for the three problem domains. We separated the BASE64 domain in two parts: 1) when considering the extended ASCII alphabet, 2) when considering a clustered alphabet.

Chart 5.5 illustrates graphically this comparison. The values in the chart represent the best results obtained with each feature extraction technique applied on each problem domain. Chart 5.6 illustrates the number of features used to obtain the best results for each technique on each problem domain.

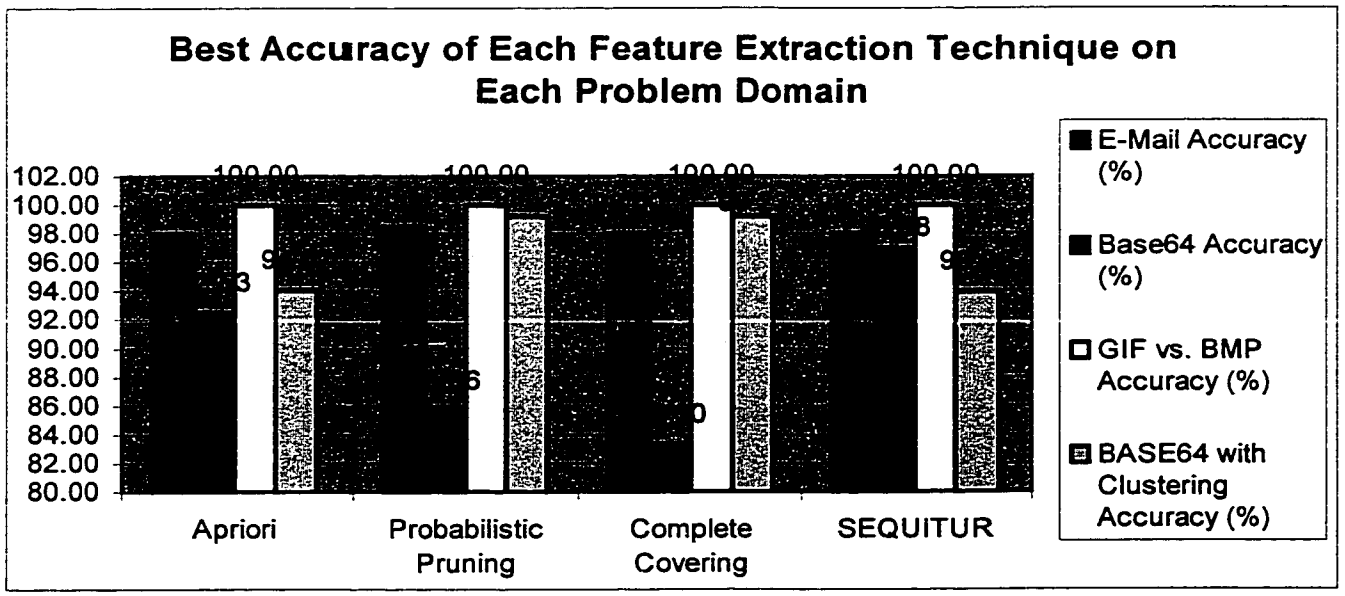


Chart 5.5

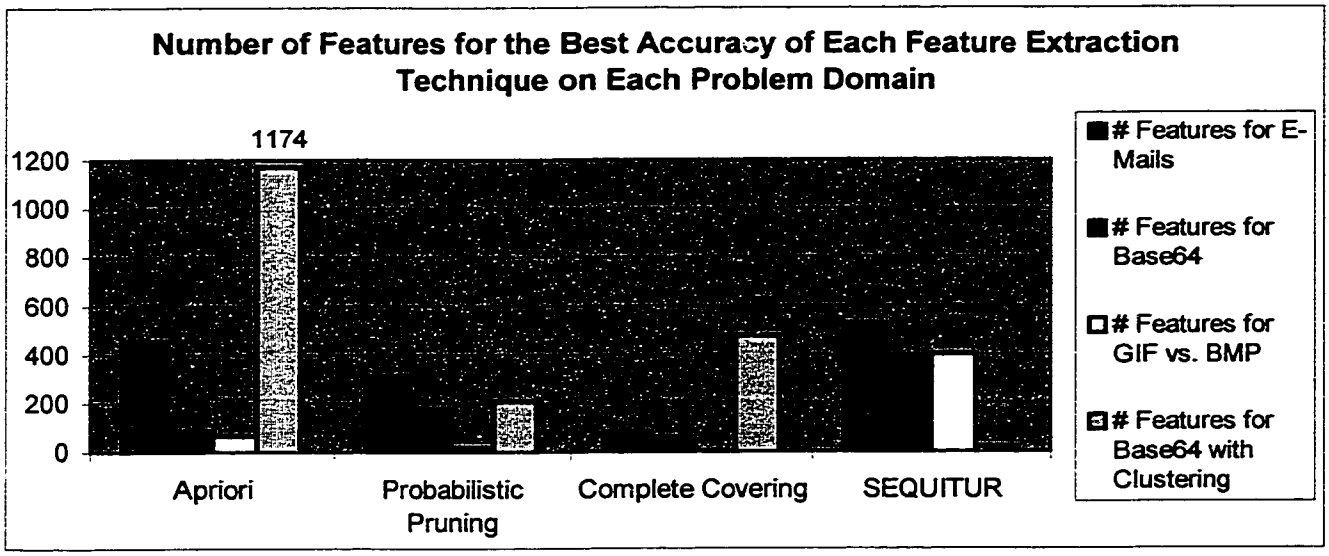


Chart 5.6

For the e-mail problem, all the techniques are reasonably equivalent in terms of accuracy. The complete covering approach generated fewer features to obtain the same results as the other techniques. This demonstrates that all the techniques have captured and considered the important patterns contained in the e-mail documents. Even if approaches like complete covering and probabilistic pruning are less exhaustive then the Apriori feature extraction, we have demonstrated that it did not affect accuracy for this problem domain considering the size of the training set.

All the techniques classified the GIF images versus the bitmap images without any problem. Again, the complete covering approach generated fewer features to obtain an accuracy equivalent to the other techniques. This demonstrates that all the techniques extracted the key feature for a relatively easy problem.

For the BASE64 problem domain, SEQUITUR gave the best accuracy if no clustering is used. It is probably because it generated considerably more features than the other techniques. Like we said in

the previous section, the better accuracy of SEQUITUR for this problem is probably due to the fact that it does not include a cross document pruning like the other techniques.

When clustering is used, the complete covering approach and the probabilistic pruning approach performed better than the other techniques in terms of accuracy. These techniques generated a reasonable quantity of features because repetition occurred when the clustered alphabet was used. Also, these techniques were able to form sub-strings indefinitely long if repetitions occurred. On the chart 5.6, the Apriori feature extractor has a higher number of features than the others simply because it obtained its best result with 10 clusters instead of 2 or 3. We believe that SEQUITUR generated fewer features because most of the features were around the average in terms of frequency. For the BASE64 problem with a clustered alphabet, it was difficult for SEQUITUR to determine some features that occurred more frequently than the others with the central limit theorem.

The feature extraction times are not illustrated in this document. However, the Apriori feature extractor is slower than the three other techniques. The complete covering approach tends to be slower when the compression tree becomes deeper. This is due to the fact that the depth-first search has to do more work because the branching factor becomes higher.

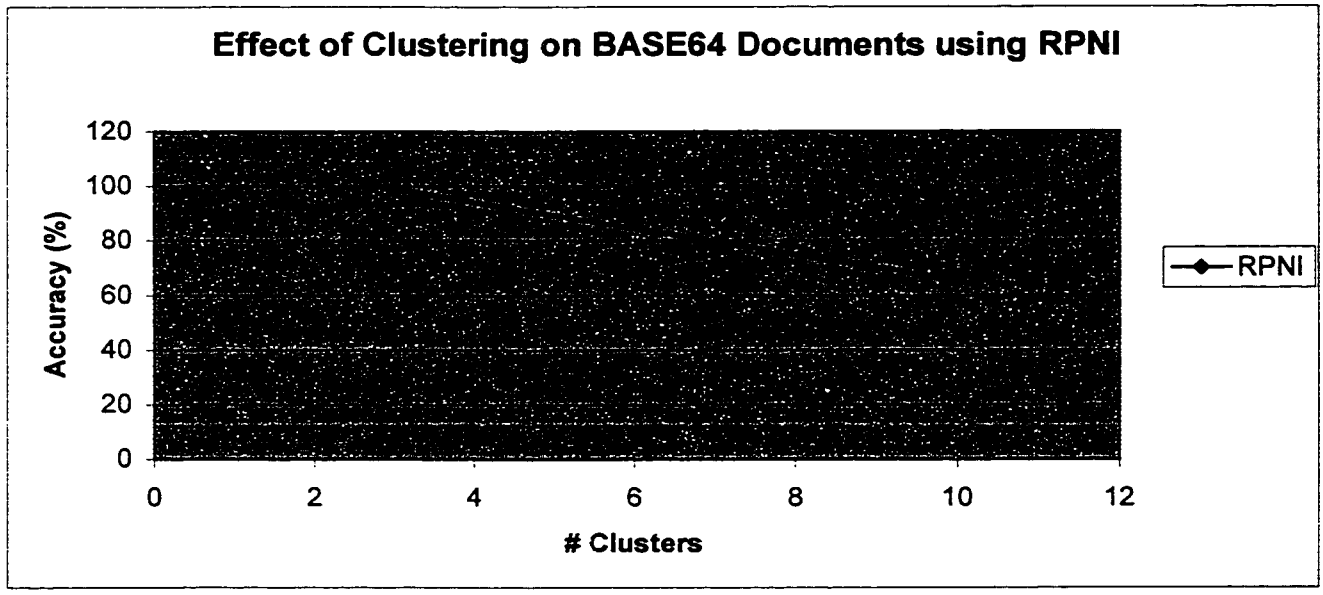
5.3.5 – Performances of Grammatical Inference

First of all, we have experimented with grammatical inference (RPNI) considering an extended ASCII alphabet. We observed that the accuracy of the classifiers obtained with RPNI is not as good as the accuracy obtained with RIPPER. Many reasons can explain the difference between the accuracy of the two induction methods using an extended ASCII alphabet. First, we have not used the same training set size for the two experiments because the amount of resources (time and memory) required by RPNI is a lot higher than RIPPER (if the induction uses a character-based alphabet). As it is shown in table 5.8, using a smaller training set with RIPPER does affect the classifier accuracy. The negative effect of using a smaller training set was minimal for the e-mail, GIF versus BMP and Base64 with clustering. However, the negative effect of reducing the training set was worse for the base64 problem domain. Second, there is a major difference in the learning techniques of the two learners. RPNI tries to identify a DFSM that would encapsulate the sequence of character transitions required to identify a document. RIPPER simply selects features that occur in the positive examples and does not appear in any negative example. The concept of sequencing during learning is absent from RIPPER. The learning task is consequently more difficult for RPNI because it considers sequencing. Third, the examples provided to RPNI are entire documents, i.e. each character of the document is a branch of the PTA. In the case of RIPPER, we first extract some features that correspond to the attributes used by RIPPER during learning. The features constitute a pre-selected subset of the potential key attributes for identification.

We obtained the best accuracy for the GIF images versus bitmap images problem domain. It is because RPNI has been able to identify a discriminating sequence of the extended ASCII characters number 0 and number 255 in the GIF images. The accuracy of the BASE64 problem domain was

better than the e-mail problem because the observed alphabet of the BASE64 documents is reduced compared to the observed alphabet of the e-mails. With a reduced alphabet, it is easier to observe discriminating sequences.

In fact, we improved the accuracy of RPNI by considering a clustered alphabet of 2 and 3 symbols with the BASE64 problem domain (Graph 5.7). We have obtained the best accuracy with 3 clusters. Using 10 clusters did not improve accuracy over a non-clustered alphabet.



Graph 5.7

We have also considered using frequent sub-strings as an alphabet for RPNI. We used a feature extraction technique (probabilistic pruning in this case) to obtain this alphabet. When using a sub-string based alphabet for RPNI, the parameters become more similar to the parameters provided to RIPPER. The only exception is that RPNI considers repetitions and sequencing in its initial PTA and RIPPER only considers an unordered feature set for each training example. We have used the same

training set and the same validation method for RPNI and RIPPER. Chart 5.8 presents the comparison of RPNI and RIPPER.

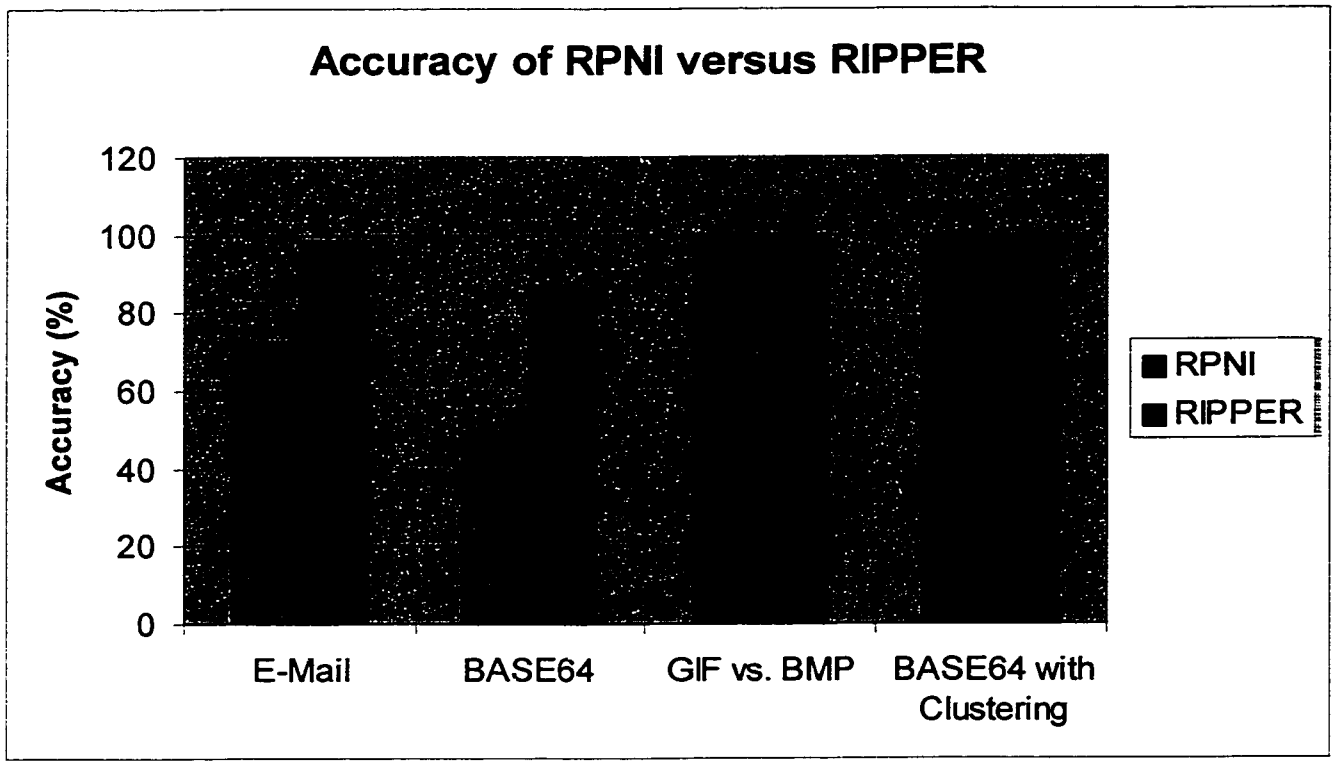


Chart: 5.8

RPNI performed as well as RIPPER for the GIF versus bitmaps domain and for the BASE64 domain with a clustered alphabet. However, RPNI did worse than RIPPER for the e-mail and the BASE64 without clustering problem domains. These observations could be explained by a characteristic of regular grammatical inference. For algorithms like RPNI, the objective is to find a minimum acceptor DFSM that accepts all the positive examples and no negative examples. The generalization leading to a minimum acceptor DFSM is done by merging nodes. At the end of induction, the DFSM accepts all the positive examples and refuses all the negative examples with a relatively small number of states, which makes the DFSM more general than the PTA. However, there is no

consideration for the fact that a certain sequence of transitions might be frequent in many of the positive examples while merging the nodes. In RPNI, if two nodes can be merged without accepting a negative example, nothing will stop the fusion even if the transition between the two nodes is a discriminating pattern. Figure 5.9 describes this problem with an APTA (a PTA containing positive and negative examples).

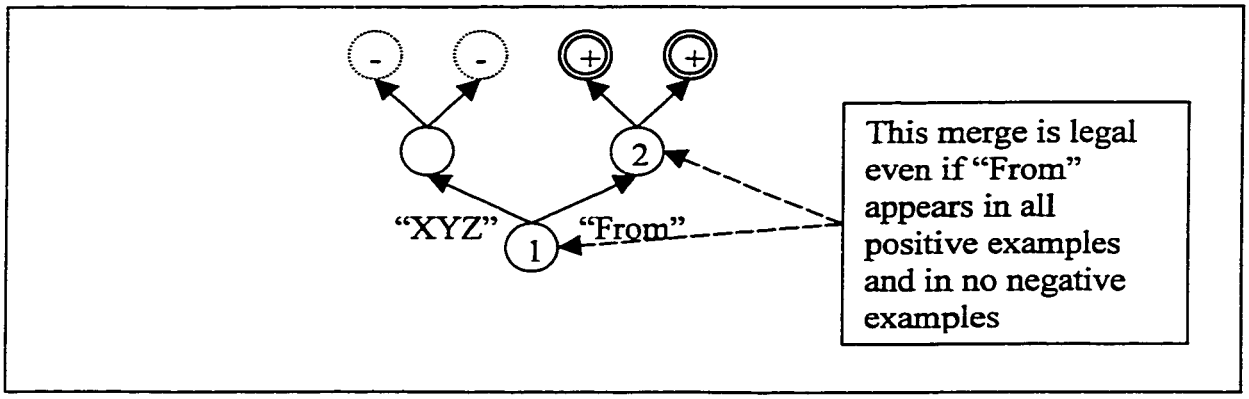


Figure 5.9: Fusion Mechanism of RPNI

Even if the sub-string "From" belongs to the positive examples only, nothing will stop the fusion of state 1 and state 2. The fusion of state 1 and 2 cancels the discriminating effect of the sub-string "From". By merging state 1 and 2, the transition representing the sub-string "From:" becomes optional because, if the current state is "state 1" applying the transition will not change the current state of the DFSM. The only objective of RPNI is to generalize the DFSM by doing state fusions to reduce the DFSM and staying compatible with the training set. We would have liked to incorporate a mechanism to refuse state merges when a discriminating transition is involved. We tried to use the Hoeffding bound mechanism presented in chapter 4. Unfortunately, we did not obtain better results.

We think that the property described in figure 5.9 explains why the results are worse for the e-mail and BASE64 domains. Early in the induction, discriminating patterns are merged into neutral states.

Consequently, identification relies on a series of transitions that are not necessarily statistically relevant. The classification accuracy is better when the documents have a general structure observable throughout the entire documents. It is the case for GIF versus bitmaps. There are discriminating patterns located at regular intervals in the documents. It is the same case for BASE64 when clustering is used. When the number of possible transitions is reduced, and the patterns associated with those transitions appear regularly in the documents, it is possible to obtain very good results with RPNI. This is why we conclude that the accuracy of grammatical inference in document classification can be greatly improved with a symbol clustering algorithm.

5.3.6 – Considering Character Frequencies as a Classification Mechanism

Using character frequencies as a mechanism for text classification might be a simple and accurate method in many cases. For example, the character distribution of base64 documents is a discriminating property if compared with the character distribution of other document types. However, the concept of character distribution excludes the notion of characteristic sub-strings that might differentiate documents. For example, assume that we want to classify e-mails versus plain texts. Both document formats would have similar character distributions, which is proper to the English language. The fact that discriminating sub-strings are proper to e-mail will not be considered if the only classification attribute is the character distribution.

To confirm the above statements, we have performed an experiment described in table 5.9. In this experiment, we calculated the occurrence ratio of each character for each document. Each training document has for property, a vector of n occurrence ratio (n being the size of the alphabet). We used

the k-nearest neighbors algorithm [MI97] to classify the documents. In this case, we decided to select the 5 nearest neighbors to classify documents.

# Positive Training Examples	# Negative Training Examples	# Positive Testing Examples	# Negative Testing Examples	Accuracy (%)
E-Mail				
66	66	34	34	**89.71**
Base64				
44	44	22	22	**95.45**
GIF vs. BMP				
100	100	50	50	**100**
Base64 with 3 clusters				
44	44	22	22	**100**

Table 5.9: K-Nearest Neighbors Classification Result

We can see that document classes that have discriminating character distributions (Base64 with or without clustering and GIF versus BMP) produced better or equivalent classification accuracy with the k-nearest neighbors algorithm than accuracies obtained with RIPPER and n-gram extraction. On the other hand, the accuracy of e-mail classification was 8.79% worse with the k-nearest neighbors algorithm. This is probably due to the fact that e-mails are identifiable with discriminating sub-strings. In summary, we observe that character frequencies might sometimes be an accurate and simple mechanism for text classification, but in other cases the use of sub-strings will provide better results.

5.3.7 – Conclusions on the Results

According to the results presented in this section we can conclude that the number of features generated influences the accuracy of the classifier. If there are not enough features, it is more difficult for the learner to identify discriminating attributes. On the other hand, if there are too many features, the risk of over fitting the training data increases. According to graph 5.1, it seems that

there is an optimal number of features that would give the best accuracy for the classifier. How do we determine this optimal number of features is a question that, unfortunately, remains unanswered.

Another important point came out of the results about the benefits of clustering. Clustering the alphabet improved the accuracy of the classifier in the case of documents that have a regular structure without having any distinct discriminating patterns (like BASE64 encoded documents). In this task, the best results were obtained using the complete covering and the probabilistic pruning approaches based on LZ78 and a clustered alphabet. Clustering has also been proven to be essential to obtain acceptable results with grammatical inference.

The techniques based on LZ78 compression gave classifying accuracy comparable to the previously known Apriori algorithm. In addition, techniques based on LZ78 compression are faster and take less memory, which is ideal for persistent storage of the features.

Grammatical inference provided worse or equal results compared to RIPPER. Results were comparable for the documents that had a general regular structure (GIF vs. BMP or BASE64 with clustering), but the results were worse for the cases in which identifying discriminating patterns does the classification (e-mail problem).

Chapter 6 – Conclusion and Future Work

6.1 – Conclusion

In conclusion to this work, it is important to evaluate the quality of the solution to the problem that was presented in the introduction of this document. The main challenge was to start from the previous work that was mainly developed to do classification of document written in natural language, and adapt and develop some new techniques to classify unintelligible documents.

Considering various feature lengths, gathering overlapping features and pruning features of different lengths are new aspects that were not considered in the classical bag-of-words approach. Also, because feature extraction based on single characters generates more features than feature extraction based on words, we had to develop some representation that would save memory and facilitate feature pruning.

In addition, we had to develop some techniques that would improve the classification accuracy of documents that do not contain discriminating patterns, but that are rather identifiable by their regular structure.

Originally, we would have liked to come up with a single solution, completely automated, that would solve all the classification problems for any kind of problem domain. Maybe it was a bit optimistic considering the time allowed for this project. We came up with many techniques that could be applied to many different problem domains. Some techniques are better in certain cases and

some other techniques are better in other cases. Also, each technique requires a certain number of parameters. Basically, we developed three layers of tools including streaming, feature extraction and classification model induction.

We realized that allowing various combinations of the tools of the three layers were possible and necessary to handle problem of different types. This is why we have developed a flexible object-oriented framework that facilitate the combination of complementary algorithms in order to adapt to any problem types with very little code changes.

The advantages of the new feature extraction techniques based on LZ78 have been demonstrated theoretically and practically. The adaptation of SEQUITUR to extract features has also demonstrated a great potential. These feature extraction techniques can perform well in detecting discriminating patterns or identifying recurrent regular structures. The importance of feature pruning has been observed in our experiments. The quality of the feature pruning mechanisms that we have implemented has been demonstrated. Pruning did not exclude important features, which could have affected the learner. We have simplified the pruning parameters in the probabilistic pruning approach using LZ78. The simplicity of the pruning mechanism parameters is a key factor in real situations where the user has to assist the learner.

It has been demonstrated that an alphabet reduction with clustering can improve the quality of the features and the accuracy of the classifier. The feature extraction techniques based on LZ78 that we have developed were very appropriate when a clustered alphabet is used.

Conceptually, grammatical inference provides some significant advantages over the other inference techniques like RIPPER. Grammatical inference considers the "structural" property of the

documents. Also, grammatical inference could be used in an incremental manner that would facilitate the reinforcement of the classification model. However, the experimental results obtained with RPNI were not better than the results obtained by RIPPER. We still believe that the idea of using grammatical inference as an alternative to a rule-based learning system should be explored in future research.

6.2 – Future Work

This section presents some research ideas that we did not have time to explore. We believe that research in the domains proposed in this section would bring additional value to the work presented in this thesis.

6.2.1 – The Wrapper Approach

All the feature extraction techniques that we have presented included a pruning mechanism to allow the control over the number of features produced. We have seen that the number of features influenced the accuracy of the classifier. Each pruning mechanism requires a certain number of parameters to adjust the pruning level. In order to have a completely automated system, it seems like we would have to try many pruning settings and choose the one that gives the better classifier in terms of accuracy. Ron Kohavi and George H. John have worked on a very similar problem in [KJ98]. The solution that they presented was the called: "The Wrapper Approach". The idea is to successively select a feature subset, induce a classifier and evaluate the quality of the feature subset on the classifier's accuracy. The algorithm finally selects the classification model that obtained the best accuracy using the optimal feature subset. The Wrapper Approach can be adapted easily to the

feature extraction techniques presented in this thesis. Each subsequent iteration could modify the pruning parameters, induce the classifier and verify the accuracy of the classifier. The goal would be to find the optimal pruning parameter values that give the best classifier accuracy.

6.2.2 – Improving the Pruning Mechanism of SEQUITUR

We have developed a pruning mechanism to select the most significant rules from SEQUITUR (see section 3.5). We believe that this technique has a greater potential in terms of feature extraction. We would have liked to incorporate a pruning parameter that would incorporate a cross document frequency test for each rule. We have implemented a pruning mechanism based on the frequency over all the symbols of the training set without considering that certain rules are applied in many documents. It would be possible to modify the algorithm (without dramatically affecting its good performance) to count the number of document in which a rule appear. Cross document pruning would definitively improve SEQUITUR as a feature extraction technique.

6.2.3 – Improving Grammatical Inference Techniques for Text Classification

In chapter 5, we have identified the fact that grammatical inference does not consider that certain sequences of transitions are common to many positive examples and to few negative examples while merging nodes. We think that it would be possible to develop a grammatical inference algorithm that would consider statistic similarities when merging nodes. This could improve the quality of the classifier especially in a document classification context.

Also, Daniel Fredouille and Laurent Miclet have started the exploration of grammatical inference by specialization in [FM00]. As opposed to grammatical inference by generalization, grammatical inference by specialization starts with the most general automaton and tries to concretize the automaton by un-merging nodes and adding state transitions. Even if the results are not yet better than the induction by generalization, the idea has great conceptual advantages.

6.2.4 – Human Assistance in Regular Grammatical Inference

Up to know, regular expressions have been widely used in filtering, detection and classification systems. Regular expressions are very powerful and flexible. However, it is difficult for a user to come up with a regular expression that would match the exact document that he/she is looking for. Moreover, the structure of the data can change over time and the regular expression may become obsolete. We think that grammatical inference can be used as a regular expression refinement mechanism. First, the human analyst comes up with a regular expression that he thinks will recognize the appropriate documents. Then, the regular expression could be transformed in an acceptor DFSM. When the classifier is in production, the human analyst could periodically look for documents that have been misidentified by the classifier. Those documents could be used as training documents to refine the acceptor DFSM that acts as a classifier. The algorithm RPNI2 would be appropriate for this kind of task.

Appendix 1 – Characteristic Training Set

This appendix has for role to define formally what is a characteristic training set. The formalism used has been taken from [DP96]. In order to introduce the definition of a characteristic training set, we first need to introduce some preliminary notions.

A language L is any subset of Σ^*. Σ^* corresponds to all the possible sequence of characters from the alphabet Σ.

The prefixes of L are defined as follows:

$$Pr(L) = \{u \mid \exists v, uv \in L\}$$

The right quotient of L by u is defined as follows:

$$L/u = \{v \mid uv \in L\}$$

The set of short prefixes of a language L is defined as follows:

$$Sp(L) = \{x \in Pr(L) \mid \nexists u \in \Sigma^* \text{ with } L/u = L/x \text{ and } u < x\}$$

The relation "<" is defined as the standard order. As an example, if Σ = {a,b} the standard order would be λ, a, b, aa, ab, ba, bb, aaa, ...

The kernel of the language L is defined as follows:

$$N(L) = \{xa \mid x \in Sp(L), a \in \Sigma, xa \in Pr(L)\}$$

Having defined those preliminary notions, we can now define the notion of a characteristic training set as follow:

A sample $D^c = (I_+^C, I_-^C)$ is characteristic for the language L and the algorithm RPNI if it satisfies the following conditions:

1. $\forall x \in N(L)$, if $x \in L$ then $x \in I_+^C$ else $\exists u \in \Sigma^*$ such that $xu \in I_+^C$
2. $\forall x \in Sp(L)$, $\forall y \in N(L)$ if $L/x \neq L/y$ then $\exists u \in \Sigma^*$ such that ($xu \in I_+^C$ and $yu \in I_-^C$) or ($xu \in I_-^C$ and $yu \in I_+^C$)

For a finite set of states Q of the target DFSM and an alphabet Σ the size of the characteristic training set is in the order of:

$$| I_+^C | = O(| Q |^2 . | \Sigma |) \text{ and } | I_-^C | = O(| Q |^2 . | \Sigma |)$$

Appendix 2 – The Hoeffding Bound

This appendix describes the application of the Hoeffding bound [HO63] in the probabilistic grammatical inference algorithm ALERGIA. A fusion between two states will be authorized if the two states are α-compatible. A merge between two states q_1 and q_2 is α-compatible if the two following conditions are satisfied:

1. $\left| \frac{C(q_1,a)}{C(q_1)} - \frac{C(q_2,a)}{C(q_2)} \right| < \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}} \left(\frac{1}{\sqrt{C(q_1)}} + \frac{1}{\sqrt{C(q_2)}}\right), \forall a \in \Sigma \cup \lambda$.
2. $\delta(q_1,a)$ **and** $\delta(q_2,a)$ **are** α**-compatible,** $\forall a \in \Sigma$.

The notation used:

$C(q_i)$ = Counter of state q_i.

$C(q_i,a)$ = Counter of the transition going out of the state q_i using the symbol "a".

Σ = Alphabet.

λ = Empty string.

$\delta(q_i,a)$ = Subsequent state using transition with symbol "a" from state q_i.

α = Confidence interval $\in]0,1[$. The confidence interval is used to determine if two probability approximations are considered equals in the Hoeffding bound.

The idea is to merge states that have the same probability to be visited during one interpretation of the DFSM.

Bibliography

[AM95] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen & A. I. Verkamo (1995). Fast discovery of association rules. In U. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), Advances in Knowledge Discovery and Data Mining, pp. 307-328. AAAIPress. 1995.

[AN78] D. Angluin. On the complexity of minimum inference of regular sets. Information and Control, pages 337-350, 1978.

[AN80] D. Angluin. Inductive Inference of Formal Languages from Positive Data. Information and Control 45, pages 117-135, 1980.

[AU72] A. Aho and J. Ullman. The Theory of Parsing, Translation and Compiling, Vol.1: Parsing. Prentice-Hall, Englewood Cliffs, 1972.

[BA89] Gérald Baillargeon. Probabilités, statistiques et techniques de régression. Les Éditions SMG. ISBN : 2-89094-035-7. 1989.

[BC90] Timothy C. Bell, John G. Clearly and Ian H. Witten. Text Compression. Prentice-Hall. ISBN: 0-13-911991-4. (1990).

[CO94] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In Grammatical Inference and Applications, ICGI'94 number 862 in Lecture Notes in Artificial Intelligence, pages 139-150. Springer Verlag, 1994.

[CO95] William W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell (Eds.), Proceedings of the 12th International Conference on Machine Learning (ML-95), Lake Tahoe, CA, pp. 115-123. Morgan Kaufmann. (1995).

[DC98] Pierre Dupont and Lin Chase. Using symbol clustering to improve probabilistic automaton inference. Lecture Notes in Artificial Intelligence, No. 1433, Springer-Verlag, Grammatical Inference, ICGI'98, pp. 232--243, 1998.

[DE89] L. Peter Deutsch. Design reuse and framework in the Smalltalk-80 system. In Ted J. Biggerstaff and Alan J. Perlis, editors, Software Reusability, Volume III : Application and Experience, pages 57-71. Addison Wesley, Reading, MA, 1989.

[DM98] Pierre Dupont and Laurent Miclet. Inférence Grammaticale Régulière : Fondements Théoriques et Principaux Algorithmes. Institut National de Recherche en Informatique et Automatique. No. 3449. France. 1998.

[DP96] Pierre Dupont. Incremental Regular Inference. In Grammatical Inference : Learning Syntax from Sentences, ICGI'96, number 1147 in Lecture Notes in Artificial Intelligence, pages 222-237. Springer Verlag, 1996.

[FM00] Daniel Fredouille and Laurent Miclet. Expériences sur l'inférence de langage par spécialisation. To appear, in proceedings of CAP 2000, St-Ethienne. 2000.

[FU98] Johannes Furnkranz. A Study Using N-grams Features for Text Categorization. Technical Report OEFAI-TR-98-30. 1998.

[FW94] J. Furnkranz and G. Widmer (1994). Incremental Reduced Error Pruning. In W. Cohen and H. Hirsh (Eds.), Proceedings of the 11th International Conference on Machine Learning (ML-94), New-Brunswick, NJ, pp. 70-77. Morgan Kaufmann.

[GH95] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns, Element of Reusable Object-Oriented Software. Addison Wesley. ISBN 0-201-63361-2

[GL67] E.M. Gold. Language identification in the limit. Information and Control, 10(5): 447-474, 1967.

[GL78] E.M. Gold. Complexity of automaton identification from given data. Information and Control, 37:302-320, 1978.

[HO63] W. Hoeffding. Probability irregularities for sums of bounded random variables. Journal of the American Statistical Association, 58(301):13-30, 1963.

[JF88] Ralph E. Johnson and Brian Foote. Designing reusable classes. Journal of Object-Oriented Programming, 1(2): 22-35, June/July 1988.

[KJ98] Ron Kohavi and George H. John. The Wrapper Approach. Book chapter in Feature Extraction, Construction and Selection : A Data Mining Perspective, edited by Huan Liu and Hiroshi Motoda. 1998.

[KN73] Donald E. Knuth. The Art of Computer Programming, Vol.3 Sorting and Searching. Addison-Wesley. ISBN: 0-201-03803-X pp.: 481-499. (1973).

[LA92] Kevin J. Lang. Random DFAs can be approximately learned from sparse uniform examples. In 5th ACM workshop on Computational Learning Theory, pages 45-52, 1992.

[LA95] Ken Lang. NewsWeeder: Learning to Filter Netnews. Proceedings of the 12th International Conference on Machine Learning ICML-95. 1995

[LA99] Kevin Lang. Faster Algorithms for Finding Minimal Consistent DFAs. (1999). www.neci.nj.nec.com/homepages/kevin/.

[LE85] S. H. Levine, An Adaptive Approach to Optimal Keyboard Design for Nonvocal Communication, Proc. Int. Conf. And Soc., pp. 334-337, 1985.

[LE92] David D. Lewis. Representation and Learning in Information Retrieval. Ph.D. thesis, University of Massachusetts at Amherst. Technical Report 91-93. February 1992.

[LM86] S. H. Levine, S. L. Minneman, C. O. Getschow and C. Goodenough-Trepagnier, Computer Disambiguation of Multi-Character Test Entry: An Adaptive Design Approach, Proc. IEEE Int. Conf. Syst., Man, Cyber., pp. 298-301, 1986.

[LR98] Craig Larman. Applying UML and patterns: an introduction to object-oriented. Prentice-Hall. ISBN: 0-13-748880-7. 1998.

[MG98] Dunja Mladenic and Marko Grobelnik. Word sequences as features in text-learning. (1998). www-ai.ijs.si/DunjaMladenic.

[MI86] S. L. Minneman, Keyboard Optimization Technique to Improve the Output Rate of Disabled Individuals, Proc. Ninth Annu. RESNA Conf., Minneapolis, MN, pp. 402-404, 1986.

[MI97] Tom M. Mitchell. Machine Learning. WCB/McGraw-Hill. ISBN: 0-07-042807-7. (1997).

[NK93] H. Ney and R. Knesser. Improved clustering techniques for class-based statistical language modeling. In European Conference on Speech Communication and Technology, pages 973-976, Berlin, 1993.

[NM97] Craig G. Nevill-Manning and Ian Witten. Identifying Hierarchical Structure in Sequences : A linear-time algorithm. Journal of Artificial Intelligence Research 7 (1997) 67-82, Published 9/97.

[OG92] J.Oncina and P.Garcia. Inferring regular languages in polynomial update time. In Perez de la Blanca, A.Sanfeliu and E.Vidal, editors, Pattern Recognition and Image Analysis, volume 1 of Series in Machine Perception and Artificial Intelligence, pages 49-61. World Scientific, 1992.

[OV92] B. J. Oommen, R. S. Valiveti and J. R. Zgierski. An Adaptive Solution to the Keyboard Optimization Problem. IEEE Transactions on Systems, Man and Cybernetics, vol. 22, no. 5, September/October 1992.

[RH74] E. M. Riseman and A. R. Hanson, A Contextual Post Processing System for Error Correction Using Binary N-Grams, IEEE Trans. Comput. 23, pp. 480-493, 1974.

[SE99] Fabrizio Sebastiani. Machine Learning in Automated Test Categorization. Consiglio Nazionale delle Ricerche, Italy. 1999.

[SM99] Sam Scott and Stan Matwin. Feature Enfineering for Text Classification. Proceedings of ICML99 Bled, Slovenia. Pp 379-385. (1999).

[UL77] J. R. Ullmann, A Binary N-Gram Technique for Automatic Correction of Substitutions, Deletion, Insertion and Reversal Errors in Words, The Computer Journal 20(2), pp. 141-147, 1977.

[WK98] Gerhard Widmer and Miroslav Kubat (Eds.) (1998). Special Issue on Context Sensitivity and Concept Drift. Machine Learning 32(2).

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. IEEE Trans. Information Theory, IT-23(3), 337-343, May 1977.

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. IEEE Trans. Information Theory, IT-24(5), 530-536, September 1978.