01 Jan 2002

Adaptive Information Filtering: Concepts and Algorithms

Daniel R. Tauritz
Missouri University of Science and Technology, tauritzd@mst.edu

Adaptive Information Filtering
concepts and algorithms

Adaptive Information Filtering
concepts and algorithms

Proefschrift
ter verkrijging van
de graad van Doctor
aan de Universiteit Leiden,
op gezag van de Rector Magnificus Dr. D.D. Breimer,
hoogleraar in de faculteit der Wiskunde en
Natuurwetenschappen en die der Geneeskunde,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 25 juni 2002
te klokke 14:15 uur

door

Daniel Remy Tauritz

geboren te Leiden in 1973

Promotiecommissie

ISBN 90-9015926-6

The work described in this doctoral dissertation has been carried out in assignment of the Dutch National Science Foundation (NWO), at the Leiden Institute of Advanced Computer Science (LIACS), under the auspices of the research school IPA (Institute for Programming research and Algorithms). IPA Dissertation Series 2002-10.

Abstract

Adaptive information filtering is concerned with filtering information streams in dynamic (changing) environments. The changes may occur both on the transmission side — the nature of the streams can change — and on the reception side — the interests of the user (or group of users) can change. While information filtering and information retrieval have a lot in common, this dissertation's primary concern is with the differences. The temporal nature of information filtering necessitates more flexible document representation methods than does information retrieval where all the occurring terms are known in advance. Also, information filtering typically maintains user interest profiles requiring a learning system capable of coping with dynamic environments in place of the static queries characteristic of information retrieval.

The research described in this dissertation investigates the employment of two distinct machine learning approaches, namely evolutionary computation (evolutionary algorithms) and neural computation (neural networks), for the intelligent optimization of incremental classification of information streams. The document representation employed in this research is weighted n-gram frequency distributions. The weights associated with the n-grams are the attributes being optimized.

The results indicate the feasibility of the machine learning approach described in the previous paragraph. Written documents as well as spoken documents were succesfully classified, within the constraints posed by adaptive information filtering. The scalability issue requires further investigation: the classification results dropped from above 95% correct for two topics to below 85% correct for ten topics, although the drop in classification results seemed to level off above eight topics.

Preface

The research described in this dissertation was instigated by a personal need of the author to deal with an overflow of data. Unsatisfied with the available information filtering systems — which tended to be very domain and media specific — he set out on a crusade to develop a general purpose information filtering system. Inspired by machine learning courses on neural networks (classification / clustering) and evolutionary algorithms (optimization), the idea took form to combine classification and optimization to build the sought after information filtering system. Once an appropriate document representation had been established, namely letter trigram distributions, a first feasibility study was conducted, the results of which were reported in [Tau96c]. These results were encouraging, and the outline of how such a system might be constructed was presented in [Tau96b]. The logical next step, namely the working out of the outline, the implementing of a first prototype of such a system, and an experimental analysis of the said system, constituted the author's Master's research, conducted at the NATO C3 Agency (formerly known as SHAPE Technical Center). This research was published in the author's Master's thesis [Tau96a], a short abstract of which was published in the JCIS'97/FEA'97 conference proceedings [TKSK97] with a longer abstract being accepted for publication in Elsevier's Information Sciences journal [TKSK00].

After having obtained his Master's degree, the author decided to pursue a Doctoral degree, continuing the same line of inquiry. The theoretical derivation of the two-pool evolutionary algorithm developed during the Master's research and refined during the Doctoral research, was first published in a technical report in 1999 [TSK99a], followed by a paper in the proceedings of the 1999 Intelligent Data Analysis conference [TSK99b]; a more definitive version of the derivation appears in this dissertation. The letter trigram distributions of the Master's research have been generalized to arbitrary alphabet n-gram distributions; in particular, the experiments reported on in this dissertation employ letter n-grams to represent textual documents and phoneme n-grams to represent spoken documents. Some preliminary results of the generalized system were reported in [TSK00], but only for letter trigram distributions; the results in this dissertation cover multiple values of n as well as phoneme n-gram distribution results. The investigation of neural networks instead of evolutionary algorithms as the learning component of our adaptive information filtering system has not previously been reported on. The neural network approach is detailed in this dissertation, and the two different machine learning approaches compared with each other as well as with published results of researchers other than the author.

The overflow of data which motivated the author many years ago to commence on this particular line of research, continues unabated. The treasure on the other side of the rainbow (read: the perfect information filtering system) is still on the drawing board, the research still in full swing. The author has, however, learned about a great many things while conducting this research, not the least of which has been *how* to conduct research, and has had a jolly good time doing so. For every question answered, two new questions have popped up, but that, the author has been told, is the hallmark of a healthy research program. On that note the author wishes you an entertaining journey into the arcane realms of information systems, document representations, evolutionary algorithms, and neural networks. And remember, if you end your journey with more new questions than questions answered, the author has done his job well!

This document was prepared and typeset using the MiKTEX[1] distribution of LATEX 2$_\varepsilon$ and BIBTEX. For literature research *The Collection of Computer Science Bibliographies*[2] administered by Alf-Christian Achilles was indispensable, as was the *ResearchIndex*[3] provided by the NEC Research Institute.

I owe a debt of gratitude to many people who have supported me during my dissertation research. First and foremost I wish to acknowledge Dr. Ida G. Sprinkhuizen-Kuyper for being my daily supervisor all these years. She inspired me from the very beginning to boldly pursue my research interests, which resulted first in my Master's thesis and now has culminated in my doctoral dissertation. She has gone far beyond the call of duty (even for an academic advisor) in supporting and encouraging me; her door was never closed to me, which I also mean quite literally as I was a guest in her house on multiple occasions during visits to Maastricht University. If I can become half the advisor to my students, that she was to me, I will consider that one of the main accomplishments of my life.

Secondly I acknowledge Prof. dr. Joost N. Kok for being my dissertation advisor. He gave the greatest gift a dissertation advisor can give: the freedom to follow the road along which my research ideas led me, wherever that would

[1]`http://www.miktex.org/`
[2]`http://liinwww.ira.uka.de/bibliography/`
[3]`http://www.researchindex.com`

take me. This is a dangerous gift, both for the giver and the receiver, and I hope that I have lived up to his expectations.

Further I acknowledge all my colleagues and teachers who gave me advice; in particular Dr. Thomas Bäck for instructing me in some of the fine points of Evolution Strategies and Dr. Walter Kosters for critiquing my dissertation. A special acknowledgement also to Marloes van der Nat, who with unwavering equanimity and always with a smile, took care of my every administrative need.

Finally I wish to acknowledge my family. My parents have proofread my dissertation to the point where they probably know it better than I do myself. My father provided many insightful comments and my mother labored endlessly to perfect the spelling and grammar, and beautify my prose. I saved for last the person closest to my heart: my wife Sharon, who has encouraged me every minute of every hour for five years while I conducted my research and wrote my dissertation. Nobody should have to be patient that long and I am eternally thankful that she was. I therefore gratefully dedicate this dissertation to my wife. This one is for you Sharon!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Synopsis

The principal objectives of this chapter are to state the problem of *Information Overload*, giving an overview of the fields within the computer and information sciences that deal with it and to introduce in particular the subfield of *Adaptive Information Filtering* as a crucial part of the solution, outlining the contributions this dissertation research has made to it.

Section 1.1 provides the background and motivation for the research reported in this dissertation. Section 1.2 introduces adaptive information filtering, lists some of the special challenges it poses, and indicates through the formulation of research questions how the dissertation research attempts to address these specific issues.

1.1 Background

Subsection 1.1.1 outlines the problem of *Information Overload*, motivating the research reported in this dissertation and introduces traditional subdisciplines within the computer and information sciences. The two fields of prime concern to us, namely information retrieval and information filtering, are further discussed in Subsection 1.1.2 and Subsection 1.1.3 respectively.

1.1.1 Information overload

The following is a quote from the August 2001 issue of "Communications of the ACM" [Kei01]:

> Computer systems today store vast amounts of data. Researchers, including those working on the "How Much Information?" project at the University of California, Berkeley, recently estimated, about 1 exabyte (1 million terabytes) of data is generated annually worldwide, including 99.997% available only in digital form. This worldwide data deluge means that in the next three years, more data will be generated than during all previous human history.

One of the great problems of our time is our inability to deal with the amount of data we generate. The above quote emphasizes the significance of this problem. Nonetheless, the age we live in is often termed the "information age" and the problem is commonly known under such labels as "Infoglut" [BYT92] and "Information Overload" [Mae94]. While this confusion of "data" with "information" is widespread, distinguishing between the two is, in fact, the foundation on which the field of information sciences is built. The distinction can be precisely formulated in terms of the following definition:

Definition 1.1 *Information is defined to be relevant data.*

The transformation from data to information, or, in other words, the separation of relevant data from irrelevant, is the central concern of the information sciences. This chapter will investigate several aspects of this transformation and various approaches to performing it. One approach in particular, Adaptive Information Filtering (AIF), will be singled out for attention. The remainder of the dissertation focuses on developing algorithms in support of AIF.

Following is an overview of some of the subfields of the information sciences dealing with the above mentioned transformation:

Information Retrieval IR attempts to satisfy short-term information needs by indicating the availability or non-availability of data relevant to particular requests (called queries in IR terminology). When the response includes the actual data deemed relevant (instead of merely a reference to it) information disclosure would be a more suitable term than IR.

However, as the term IR is widely accepted for both cases, we will follow that custom in this document.

Information Filtering IF attempts to satisfy long-term information needs by maintaining profiles describing those needs and filtering data streams, returning data which matches one or more profiles.

While the emphasis of this dissertation is on IF, most of the research is also to a certain extent applicable to IR. We will therefore first present an overview of IR before moving on to IF.

1.1.2 Information retrieval

There is no commonly accepted definition of IR, and over the years it has come to encompass an increasingly broad field. A few decades ago researchers in this field, such as Lancaster and van Rijsbergen, adopted the view that IR systems satisfy queries, put to them by users with a specific information need, by returning references to material likely to be of interest. For instance, in [Rij79] we find the following definition for IR as stated by Lancaster: "Information Retrieval is the term conventionally, though somewhat inaccurately, applied to the type of activity discussed in this volume. An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request." Now-a-days, however, users expect their information needs to be satisfied by immediately being supplied with the material likely to be of interest. In other words, today's users expect an information disclosure system, a view adopted for instance in [Bruza93]. As the term IR is more widely accepted, we will employ it throughout this thesis, although in fact we will be referring to an information disclosure system.

An IR system is the complete system necessary to perform the process of IR. A librarian exemplifies such a system. Someone (e.g., the user) makes a request for information which the librarian then tries to satisfy. An IR system consists basically of two components: a user interface for making the requests (called queries in IR terminology) and presenting the retrieved information, and a retrieval system which tries to satisfy the query by returning relevant references. The queries can be formulated either in natural language, or in a so-called query language. The way the retrieval system works and the method for formulating the queries are interdependent, and can be described by a retrieval model.

Retrieval models

Here we will describe two classic retrieval models, namely the Boolean retrieval model [Rij79] and Salton's Vector Space Model (VSM) [Sal89]. A more recent model of interest is the Probabilistic model, see [BC92, Coo94]. In the Boolean retrieval model the user has to formulate queries based on Boolean logic. Those documents which are "true" for the query are retrieved. The queries have to be expressed in terms of index terms and combined by the usual logical connectives *and*, *or*, and *negation*. The following example will illustrate the Boolean retrieval model.

> Assume the following query:
>
> Q = (apple AND pie) OR ((not cheese) AND sandwich)
>
> This will retrieve all documents containing the terms 'apple' and 'pie', as well as all documents containing the term 'sandwich', but not the term 'cheese'. Suppose for a moment that this query was submitted to a recipe database. One can then imagine that the person submitting the query is hungry for apple pie and any sandwich not containing cheese.

The great advantage of the Boolean retrieval model is that searching can be done very efficiently by using an inverted index, a matrix indicating which documents contain which terms. There are, however, two disadvantages. The first is that Boolean retrieval is an all-or-nothing decision, which does not correspond to the intuitive notion that relevance of a document to a query is a matter of degree. This makes it difficult to deal with the often vague intentions of users, and makes the system very sensitive to spelling errors in both the queries and the stored documents. The second disadvantage is that humans, especially untrained ones, often find it very difficult to formulate complex Boolean queries. In the VSM both documents and queries are represented as vectors in a common vector space. The dimensions of this space are the indexing terms. Similarity of meaning is assumed proportional to the distances between the vectors in the space. As vector spaces are only properly spanned by orthogonal vectors, the term vectors must be assumed to be uncorrelated. This is, however, not usually the case. The issue is discussed in depth in [Sal89]. The advantages of the VSM are that the queries can be formulated in natural language and that retrieval yields a ranking of the available documents according to relevance. The disadvantage is, however, that the search is not efficient, making the use of very large document collections and large numbers of indexing terms impractical. A possible solution is to cluster all the documents, and during retrieval find, for instance, the closest cluster and then compute only the distances between the query and the documents within that cluster. This is called cluster based retrieval and is discussed in both [Rij79] and [Sal89].

Relevance feedback

A powerful enhancement to the user interface presented so far is to allow the user to provide feedback to the system by indicating how relevant the retrieved documents were. This information can then be used by the system in various ways. The classic way is to use this information to refine the search [Sal89], but it can also be used to optimize certain parameters associated with the search process, creating a learning system.

Information disclosure systems

Information disclosure systems are composed of the previously mentioned components as well as a full-document retrieval system. So-called Internet search engines are one example of the sort of information disclosure systems which are currently receiving a lot of attention. The most popular of these search engines maintain huge keyword indexes to billions of World Wide Web pages.

Evaluation

The classic method for evaluating the effectiveness of an IR system employs the concepts of recall and precision [Zav95]. Precision is the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is the ratio of the number of relevant documens retrieved to the total number of relevant documents. So for both, zero represents worst and one best performance. The goal of an IR system is, thus, to maximize both recall and precision. However, there is a trade-off between these two measures. To increase recall one can relax the conditions to be met for a document to be retrieved, but this will inevitably lead to a decrease in precision, with more irrelevant documents being retrieved. To express the effectiveness of an IR system by one number, recall (r) and precision (p) can be combined in van Rijsbergen's E-measure [Rij79]:

$$E = 1 - \frac{(1 + B^2) \cdot p \cdot r}{B^2 \cdot p + r} \tag{1.1}$$

The parameter B reflects the relative importance attached to recall and precision. A value of 0.5 (or 2.0) for B corresponds to attaching twice (or half) as much importance to precision as to recall. With a value of 1.0 for B, the two factors are weighted equally. The range of E is between 0 and 1. Lower values of E indicate better performance, with 0 being best and 1 being worst.

For more information on information retrieval see, for instance, [Che95, SJW97, Wie98, vdP00].

1.1.3 Information filtering

The following is a quote from the November 2001 issue of "Communications of the ACM" [Fox01]:

> An overload of information pouring into the U.S. National Security Agency's most secret information-gathering program might have slowed the investigation into September's terrorist attacks in New York and Washington, reports the New York Post. Echelon, a computerized interception program so powerful authorities don't officially acknowledge its existence, has spewed forth so much information analysts can't keep pace with the growing mountains of clues that may reveal terrorist tactics and hideouts. Intelligence experts say the backlog of intercepted communiqués from Middle East, Africa, and even the U.S. is drowning NSA agents in information as they sift through the past year's data. Echelon, operating within Europe, is reportedly able to intercept, record, and translate any electronic communication — telephone, data, cellular, fax, email, telex — sent anywhere in the world.

The above quote demonstrates an obvious lack in the Information Filtering (IF) capabilities of the NSA. IF is the process of filtering incoming data streams in such a way that only particular data are preserved, dependent on given information needs. In the case of IF being employed as preprocessor for an IR system, these needs are typically predefined and rather static. When IF is being used as a replacement for IR to satisfy the needs of a particular person (or group of persons), these needs are, generally speaking, apt to change considerably over time. The remainder of this section will discuss IF systems in general, while those aspects specific to adaptive IF systems will be discussed in Section 1.2. For a comprehensive introduction to IF, see, for instance, [FD92].

An IF system is the complete system necessary to perform the process of IF. One example is a censure department. The incoming data stream is filtered according to certain guidelines, leaving only approved documents in the output data stream. This can be done manually, or automatically through the use of a software system. The type of system discussed in this dissertation will be a software system. There are two main components common to all IF systems, namely the data stream and the filtering transformation. In the case of a censure department the data stream might consist of movies and the filtering transformation be performed by a computer program censuring certain films based on content analysis.

The data stream

In the case of a software system the data stream can consist of any object which can be digitally represented. Besides the traditional medium of textual documents, this also includes such modern media as audio, video, and multimedia in general. Some examples are:

E-mail E-mail messages are typically textual documents with little structure. Sender and subject can be indicated, but predominantly the body of the message is free form text. Recent extensions allow the inclusion of multimedia in the messages, but do not generally add semantic structure to the text.

Internet newsgroups Messages posted to Internet newsgroups are partially structured textual documents. While the body of the message is free form text, just like an E-mail message, it typically includes information such as sender, subject, keywords, and implicit information with regard to the topic of the message. This implicit information can be derived by examining to which particular newsgroup the message was posted.

News streams More and more news is becoming available via the Internet, for example from Reuters (see `http://www.reuters.com`). This usually involves textual documents ordered by topic.

Radio The traditional medium of the radio is (still) an important supplier of data; by digitizing the (often analog) broadcasts it can be processed by software systems. Recent years have seen an increasing number of direct radio broadcasts via the Internet.

Television Television, arguably the most important medium of the present day, can also be digitized and then processed by software systems. This is even more complicated than processing straight audio data, both because of the much wider bandwidth, and the necessity of having to deal with the video component, for which no obvious transformation to text exists. However, most — if not all — television broadcasting corporations are offering multimedia enhanced news reports via the World Wide Web.

The filtering transformation

The basic concept here is that any object which differs "too much" from the stated information need will be filtered out, thus leaving only relevant objects in the output stream. There are, however, a number of reasons for desiring a more sophisticated approach to the filtering process. First of all, the distinction between relevant and non-relevant data is not a sharp one; rather the degree to which an object corresponds to an information need gradually declines as the "distance" between the two increases. And, secondly, the actual information needs are often not well understood, so the danger exists that relevant data will be filtered out because of an incorrect formulation of the information need. One way to deal with these problems is to cluster the incoming data stream and represent the information need as a set of points in the space of clusters.

Differences between IR and IF

There are two major differences between IR and IF. First of all, in IR the information need is a short-term need, to be satisfied immediately, while in IF the information need is typically a medium to long-term need. This has various implications. For instance, in IR a query can be refined, as opposed to IF, where user interests change over time. And secondly, in IR a static data storage is assumed during the processing of a query, while in IF the objective is the filtering of dynamic data streams. For an in depth review of the differences between IR and IF, see [BC92].

Collaborative filtering

An interesting extension of IF is collaborative IF, as proposed in [GNOT92] in which it is described as follows:

> "Collaborative filtering simply means that people collaborate to help one another perform filtering by recording their reactions to documents they read. Such reactions may be that a document was particularly interesting (or particularly uninteresting). These reactions, more generally called annotations, can be accessed by others' filters."

1.2 Dissertation research

Adaptive Information Filtering (AIF) is the process of filtering incoming data streams in such a way that only relevant data (information) is preserved. The relevancy of the data is dependent on the changing (adaptive) needs of a particular person or group of persons with a shared interest. Think, for example, of a newspaper. From all the news in the world available to the reporters a selection is made based on what is deemed to be of interest (relevant) to the readers of the newspaper, the rest is filtered out. As the information needs of the readers change, the reporters must adapt their selection criteria correspondingly or the newspaper will fail.

An AIF system is the complete system necessary to perform the process of AIF. One example is the system employed by our newspaper; radio and television news reports employ similar systems. The type of system discussed in this dissertation will be a software system. An AIF system consists of three main components: the data stream, the transformation from data to information through filtering, and the adaptive behavior. In the case of a software system the data stream might, for example, consist of Internet news, the transformation be performed by a computer program, and the adaptive behavior by the interaction between the user and the computer program. For more information on AIF see, for example, [Bac91, HKW94].

1.2.1 Representation

In the case of a software system with as incoming data stream Internet news, the data stream consists of textual documents with a specific classification. For example, if a document was posted to newsgroup x, then one may assume that the document can be classified corresponding to the topic of newsgroup x. This is of course not always the case, but except for groups with a very low S/N (signal to noise) ratio it will hold for the majority of the posted documents. Another common incoming data stream is E-mail which also consists of textual documents but which lacks a specific classification. In the rest of this dissertation textual and phonetic documents will be assumed. To deal with the data stream it is necessary to be able to compare the documents with the interests of the user at a given time. This implies the necessity of storing the interests of the user and the need for a transformation of documents and user interests to a common space where they can be compared. This issue will be examined in more detail in Section 2.1, which covers estimating document similarity by applying a metric to document vector representations. The classification algorithm employed in the dissertation research is introduced at this point. The basic idea is that any object which differs "too much" from the user's interests will be filtered out, leaving only relevant objects for the user to peruse. There are, however, a number of reasons for desiring a more sophisticated approach to the filtering process. In addition to the reasons mentioned in Subsection 1.1.3, there is also the fact that users will often have varied interests and in such cases will prefer to have objects pertaining to each particular interest grouped together. Again the solution is to cluster the incoming data stream and represent the user interests as points in the space of clusters.

A special challenge posed by AIF is the dynamic data environment. In a static document collection, characteristics such as word occurences are known a priori, facilitating the employment of word (term) indexing methods. In AIF, on the other hand, this is not the case. New words may occur frequently, resulting in diminished performance unless new indexes are built, a process which can be very time consuming. One way to prevent this would be to use complete dictionaries for indexing, but this would result in huge indexes. Another drawback of using word based document representations is their sensitivity to spelling variations/errors. An alternative representation method is based on so-called *n*-grams. This brings us to our first research question:

> Do n-grams have the potential to provide document representations more suited to adaptive information filtering than traditional methods of representation?

Chapter 2 provides an in-depth introduction to n-grams. Various ways in which they can be used for obtaining document representations are listed as are a number of other applications. A set of conditions are introduced that need to be satisfied in order for n-grams to have the potential to outperform traditional methods of representation. Extensive bibliographic references on the subject of n-grams are provided. Next, a detailed review of the employment of n-grams for textual representation is presented. The Latin alphabet plus space delimiter is shown to satisfy the above mentioned conditions. Advantages of using n-grams over this alphabet are provided and the Reuters-21578 text categorization collection is introduced as the data set employed in the dissertation research. Some preprocessing methods for textual documents are indicated and a classification experiment is described illustrating the baseline performance of letter trigrams. This section is followed by a look at employing n-grams for phonetic representation. Phonetic alphabets are shown to satisfy the n-gram usability conditions mentioned earlier. Details on obtaining the topical phoneme encoded spoken document dataset required for our experiments are furnished, as are advantages of using n-grams over a phonetic alphabet. A classification experiment is described illustrating the baseline performance of phoneme n-grams for varying values of n.

1.2.2 Adaptation

In order to adapt to the changing information needs of the user, interaction with that user is necessary. Input from the user is needed with respect to two important issues. First of all, input is required concerning the user's interests, and, secondly, the user must indicate which objects have been clustered correctly and which incorrectly. One possibility is that the clusters themselves represent the user's interests.

In Chapter 2 it is argued that weighted n-gram representations are necessary to improve the baseline performance of letter and phoneme n-gram classification. The weights indicate relative importance of particular n-grams in the classification process. As the interests of a user change, or changes occur in the data streams being filtered, the optimal set of weights changes too. Therefore, an incremental weight optimization method is required to "learn" the optimal weights for each user and continuously adapt them. This leads us to a number of research questions:

> Do machine learning techniques have the potential to satisfy the specific requirements of adaptive information filtering systems?
>
> How can machine learning techniques be employed to optimize the weights associated with n-grams in n-gram document representations?
>
> What type of evolutionary algorithm is capable of addressing the specific requirements of adaptive information filtering?
>
> What degree of improvement in classification results can be achieved through the use of weighted n-gram document representations?

We attempt to address these questions by exploring n-gram document representations in combination with two machine learning techniques, namely evolutionary computation and neural computation.

Chapter 3 provides an introduction to evolutionary computation: how it works, what some of its advantages and disadvantages are, and from which computational models the field originated. The core of the chapter is the derivation of the two-pool Classification Evolutionary Algorithm (CEA) developed in the course of the dissertation research. It has been specifically designed to optimize incremental classification algorithms suitable for employment in AIF systems. Their suitability rests on their incremental nature which addresses the temporal aspect of AIF systems. The chapter continues with a description of the evolution component as employed in our research, as well as implementation details. This is followed by a description of an experiment illustrating the two-pool CEA applied to simulated documents.

A historic overview of evolutionary computation in the information sciences is set out in Chapter 4. The two-pool CEA introduced in Chapter 3 is employed by the AIF simulation system constructed as part of the dissertation research. After presenting a schematic overview and a more detailed algorithmic overview, the parameters of the AIF simulation system are explained in-depth. Some experimental results are then provided and plots are shown comparing the results. This is followed by a comparison of the two-pool CEA and related research employing the *k nearest neighbours* classifier and a genetic algorithm.

Chapter 5 provides an introduction to neural computation by describing some common types of neural network architectures and their dynamics. A number of learning rules are listed and the *stability-plasticity dilemma* is formulated. This is followed by a historic overview of neural networks in the information sciences. Next, the neural network simulation system constructed as part of the dissertation research is explained in-depth. The neural network and its dynamics are then described, followed by an algorithmic overview of the system. A section containing some experimental results for letter and phoneme bigrams and trigrams comes next. Plots are displayed which reveal that performance improves over time as the neural networks learn; these plots also permit comparisons to be made among the results. The effect of the parameter n and the learning rate are measured in separate experiments. Finally, the neural network results are compared with the evolutionary computation results of Chapter 4.

Chapter 6 consists of three parts. The first part presents a summary of the most important results as reported in this dissertation. The second part answers the research questions posed in this section. The third part makes some suggestions with respect to future research. Appendix A describes the C++ n-gram class library developed in the course of the dissertation research. Appendix B explains what Alphabet Index Lists (AIL) are and provides the source code of a *Text to Latin AIL* converter and of a *Text to Phoneme AIL* converter. Finally, Appendix C provides the mathematical proof of a statement made in Chapter 2.

Chapter 2

Document representations employing n-grams

Synopsis

This chapter introduces n-grams. They can be employed to obtain document representations for a wide variety of document types and have advantages over traditional document representations that are especially pertinent for the application of adaptive information filtering.

A short introduction to representing documents in vector spaces is given in Section 2.1, followed by a detailed introduction to n-grams, how they can be employed to represent documents in vector spaces, and a summary of their many other applications, in Section 2.2. The advantages of n-gram document representations over more traditional document representations are expounded and criteria provided for evaluating how well suited a particular n-gram representation is for a given document type.

This dissertation covers adaptive information filtering of textual and phonetic document types. Section 2.3 and Section 2.4 describe in detail the n-gram representations of these respective document types and demonstrate their suitability to such representation utilizing the criteria earlier provided.

While the n-gram document representations introduced in Section 2.3 and Section 2.4 are highly effective in certain restricted domains, for the application of adaptive information filtering they fall short of what is required for practical utilization. The premise of this dissertation is that this limitation can be overcome by associating weights with the n-grams, contingent on finding the right weights. Weighted n-gram document representations are presented in Section 2.5; finding the right weights will be the subject matter of the entire remainder of the dissertation!

The storage of n-gram document representations is not at all trivial, and is the focus of Section 2.6. This section also introduces the C++ n-gram class library that was developed in the course of the dissertation research. It is described in detail along with its source code in Appendix A. Finally, Section 2.7 contains the chapter conclusions.

2.1 Representing documents in vector spaces

Documents are sequences of symbols conveying information. The nature of the information conveyed determines the topic, or topics, the document is about. In information filtering a profile representing a user's long-term information requirements is maintained. As documents appear in certain targeted data streams, they are compared with this profile, filtering those of potential interest out and presenting them to the user. Some kind of similarity measure is required in order to compare profiles and documents. Optimally, we would like to measure conceptual similarity, because this would be closest to our own perception of measuring similarity. An understanding of the semantics of the documents is required in order to measure conceptual similarity. While this is not yet feasible ([Sch93a], page 140), we can estimate it by applying a metric to syntactic representations of profiles and documents. The availability of transformations from profile and document space to one particular vector space would permit using a metric in that vector space to compute this estimate.

One way to represent profiles is by a set of topics. Documents are classified and if they are found to match one of the topics they are filtered out and presented to the user as belonging to one of the user's topics of interest. A topic can be represented as an average of the vector space representations of documents known to belong to that topic. This has the great advantage that profile and document space are one and the same, requiring only a single transformation to a vector space instead of two. Traditionally, computing term frequency distributions has been used for this transformation [Rij79]. In this dissertation we will examine the use of n-gram frequency distributions instead.

Once one has determined a transformation from document space to vector space, the similarity of documents can be estimated by applying a metric to the vector representations of the documents. Consider two documents, d and e. Let the vectors $\vec{d} = (d_1, d_2, \ldots, d_h)$ and $\vec{e} = (e_1, e_2, \ldots, e_h)$ be their respective vector representations, with h being the dimension of the vector space. The Minkowski 1-norm is defined as follows:

$$|\vec{d}| = \sum_{i=1}^{h} |d_i| \tag{2.1}$$

Let $\overline{\vec{d}} = (\overline{d_1}, \overline{d_2}, \ldots, \overline{d_h})$ with $\overline{d_i} = d_i/|\vec{d}|$ for $i = 1, 2, \ldots, h$ and $\overline{\vec{e}} = (\overline{e_1}, \overline{e_2}, \ldots, \overline{e_h})$ with $\overline{e_i} = e_i/|\vec{e}|$ for $i = 1, 2, \ldots, h$, be their respective normalized corresponding vectors. The vector representations employed in the dissertation research are based on occurrence frequencies and are therefore not length independent. Normalizing ensures that the length of a document does not influence its vector representation. The similarity between d and e can then be estimated by applying a metric to $\overline{\vec{d}}$ and $\overline{\vec{e}}$. In this dissertation the Minkowski ℓ_p-metric will be employed:

$$\ell_p(\overline{\vec{d}}, \overline{\vec{e}}) = \sqrt[p]{\sum_{j=1}^{h} (\overline{d_j} - \overline{e_j})^p} \tag{2.2}$$

In particular two special cases are used, namely for p equal to one which is called the Manhattan metric, and for p equal to two which is called the Euclidean metric. For an in depth overview of measuring in the information sciences see [BMK94].

We can employ a classification algorithm to determine which topic belonging to a user profile represents a particular document best. In the dissertation research the *nearest cluster algorithm* (Algorithm 1) was used for classifying. Given a document vector and the cluster prototype vectors which represent the centers of the clusters, the closest cluster is returned by computing the distance of the document vector to each cluster prototype vector.

2.2 Introduction to n-grams

Our definition of n-grams is a slightly modified version of the one given in [Coh97]:

Algorithm 1 Nearest cluster

```
$k \leftarrow$ number of clusters
$C = (\vec{c_1}, \vec{c_2}, \ldots, \vec{c_k}) \leftarrow$ cluster prototype vectors
$\vec{d} \leftarrow$ document vector
smallest dist $\leftarrow \infty$
for $i = 1, 2, \ldots, k$ do
    dist $\leftarrow \ell_p(\vec{c_i}, \vec{d})$
    if dist $<$ smallest dist then
        smallest dist $\leftarrow$ dist
        closest cluster $\leftarrow i$
    end if
end for
result $\leftarrow$ closest cluster
```

Definition 2.1 *Given a sequence of tokens $S = (s_1, s_2, \ldots, s_{N+(n-1)})$ over the token alphabet $\mathcal{A}$ with N and n positive integers, an n-gram of the sequence is an n-long subsequence of consecutive tokens. The i^{th} n-gram of S is the sequence $(s_i, s_{i+1}, \ldots, s_{i+n-1})$. Note that there are N such n-grams in S.*

The main difference between our definition and Cohen's definition is that Cohen employs the term *symbols* without specifying a symbol alphabet, while we employ the term *tokens* and specify the token alphabet $\mathcal{A}$.

Let $|\mathcal{A}|$ be the size of $\mathcal{A}$ and $\mathcal{A}(n)$ the number of unique n-grams over $\mathcal{A}$. $\mathcal{A}(n)$ can be computed as follows:

$$\mathcal{A}(n) = |\mathcal{A}|^n \tag{2.3}$$

The use of n-grams offers an intuitive, yet powerful, method of representing many different kinds of documents in vector space. There are various methods for representing documents using n-grams. An *n-gram occurrence array* (also called a *non-positional binary n-gram array*) is an n-dimensional array of size $\mathcal{A}(n)$ whose elements represent all possible n-grams. The value of each element in the array is set to either 1 or 0 depending on whether the represented n-gram occurs or not. Such an array does not indicate the position of n-grams within words. Conversely, a set of *positional binary n-gram arrays* captures this information by maintaining for each n-gram an n-dimensional array in which the value of $(i_1, i_2, \ldots, i_n)$ is set to 1 if and only if the tokens appear in at least one word in positions $i_1, i_2, \ldots, i_n$ respectively. While these two methods have been used to represent predefined lexicons for the use of spelling error detection ([Kuk92b], page 381), they do not capture sufficient information for document comparison. A third method is to compute *n-gram frequency distributions* by counting the number of occurrences of all n-grams in a document and storing those values in a vector. This method produces representations suitable for document comparison and is employed in this dissertation. The similarity between documents can then be estimated using the Minkowski ℓ_p-metric (see Equation 2.2 on page 17). Let $\vec{d}$ and $\vec{e}$ be n-gram frequency distribution vectors with $d_i \geq 0$ and $e_i \geq 0$ for $i = 1, 2, \ldots, h$, then the dimension of the vector space is $\mathcal{A}(n)$. This yields:

$$\ell_p(\vec{\overline{d}}, \vec{\overline{e}}) = \sqrt[p]{\sum_{j=1}^{\mathcal{A}(n)} (\overline{d_j} - \overline{e_j})^p} \qquad (2.4)$$

Computing the traditional term frequency distribution is a special case of determining the n-gram frequency distribution, namely where the token alphabet contains all the terms occurring in all the documents that need to be classified, and then computing the 1-gram distribution. In 1-gram analysis the occurrence of single tokens is determined, in 2-gram analysis that of pairs of tokens, in 3-gram analysis that of triplets, etc. When talking about a specific value of n, especially for lower values of n, its Latin name is often used instead of the numeric value; for example, 2-grams are often called bigrams, 3-grams trigrams, 4-grams quadgrams, but 5-grams usually just 5-grams.

Documents need not be restricted to text. Speech can be transformed employing an alphabet of phonemes, pictures employing an alphabet of 3x3 pixel combinations, musical scores employing an alphabet of musical symbols, etc. One of the criteria for composing a token alphabet is the requirement that the object to be vectorized can be described as a sequence of tokens contained in the alphabet. So, for instance, if your object happens to be a musical score it would make sense to select musical notes and symbols for inclusion in your alphabet, but it would probably not make sense to include words (unless, of course, the musical score contained lyrics!). Because traditionally n-grams have been used to vectorize textual documents using the Latin alphabet (plus space delimeter) as token alphabet, many research papers use a narrower definition which does not mention tokens nor token alphabets, but is equivalent to the above definition employing the Latin alphabet as token alphabet. Examples of the use of n-grams for representing music can be found in [UZ98, Dow99, UZ99].

When the following two conditions are met, n-grams have the potential to outperform the traditional term frequency distribution:

1. The token alphabet is relatively small.

2. The token alphabet is representative.

The first condition is relative to the number of terms used in the traditional term frequency distribution, which is in the rule quite large. The second condition is to ensure that the original document is sufficiently represented by the tokens of the token alphabet to make comparison based on n-grams of those tokens feasible.

Document representation is but one of the many varied applications of n-grams. In [BS01] n-grams are listed as a method for matching medical records by applying n-gram analysis to record fields. And in [Mar00] n-grams are employed for computer security, in specific for computational immunology. It is stated in [Mar00] that:

> Some recent advances in intrusion detection are based on detecting anomalies in program behavior, as characterized by the sequence of kernel calls the program makes.

Tokens are assigned to the different kernel calls. The sequences of kernel calls can then be expressed as strings of tokens. The n-grams occurring in those token strings are determined during a training phase. In operational mode the then occurring kernel-call n-grams are compared to the ones generated during training. A few mismatches may simply be due to the training not having been exhaustive of all possible normal behavior n-grams, but a large number of mismatches is a good indicator of an attack.

The scientific literature contains many more applications of n-grams, such as spelling error detection and correction [ME62, MC75, Ull77, ZPZ81, DeH82, AFW83, CVBW92, Kuk92a, Kuk92b, ZD95], optical character recognition [Rav67, RE71, RH74, Neu75, HRF76, HS82, HSV87, Sen94], text compression [Rub76, Wis87, Krz95], language identification [Sch91a, SR96], automated text categorization [CT94, HD94, Huf95, Dam95, Tau96c, Lan00a, Lan00b, CMS01], information retrieval [SH73, Wil79, DeH82, TS88, Com90, Ada91, AP92, Sch93a, Cav94, Dam95, ELW95, LA96, PPF96, MM97, LCP99, MSL$^+$99, MSLN00] and information filtering [Cav93, Tau96b, Tau96a, TKSK97, TSKK97, TSK99a, TSK99b, TKSK00, TSK00, Lan00a, Lan00b]. Other areas of application for n-grams with corresponding references may be found in [Kuk92b, Coh97, RW98]. For an extensive n-gram bibliography see the *n-gram clearinghouse*[1] or *The Collection of Computer Science Bibliographies*[2].

2.3 Textual representation

The employment of n-grams for textual representation is not a novel concept. In 1979, for instance, such was proposed in [Wil79]. And as early as 1973 the use of n-grams had already been suggested for information retrieval, for example in [SH73]. As far back as 1962 n-grams had already been applied to spelling error detection and correction in [ME62]. And in ([Coh97], page 292) a reference is given for employing n-grams for text compression dating back to 1951.

Representing textual documents with n-gram frequency distributions is, however, a very computational and memory intensive technique — especially for higher values of n — and it was not until the early 1990's that sufficient computing power was available for making this a practical approach. The reason for this is the exponential explosion associated with increasing the value of n. For instance, if one employs a compact token alphabet, say let $\mathcal{A}$ be the Latin alphabet plus the space delimiter, then $|\mathcal{A}| = 27$. If one distinguishes between upper and lower case letters, and also places significance in numerical digits, then $|\mathcal{A}| = 63$. Table 2.1 gives an idea of how fast $\mathcal{A}(n)$ increases with n. To deal with this exponential increase in size it is necessary to employ special techniques such as sparse vectors and hash tables. However, employing the Latin alphabet plus the space delimiter fulfills both the conditions set in Section 2.2:

1. $|\mathcal{A}| = 27$ versus many thousands or even tens of thousands of terms in a traditional term frequency distribution
2. the vast majority of text is composed of letters and spaces

Therefore it has the potential to outperform traditional term frequency distribution based methods. In the rest of this dissertation the token alphabet

[1]`http://www.liacs.nl/home/dtauritz/ngram/`
[2]`http://liinwww.ira.uka.de/bibliography/`

Table 2.1: Exponential n-gram explosion

n	27^n	63^n
1	27	63
2	729	3,969
3	19,683	250,047
4	531,441	15,752,961
5	14,348,907	992,436,543
6	387,420,489	62,523,502,209

consisting of the Latin alphabet and delimiter will be denoted by $\mathcal{T}$. Here are some of the advantages to using n-grams over $\mathcal{T}$:

- Robustness: Relatively insensitive to spelling variations/errors
- Completeness: Token alphabet known in advance
- Domain independency: Language and topic independent
- Efficiency: One pass processing
- Simplicity: No linguistic knowledge is required

We experimented using the Reuters-21578 text categorization collection. The documents in this collection appeared on the Reuters newswire in 1987. The collection is downloadable from David D. Lewis' professional home page[3]. The documents are in SGML format and tagged for the purpose of splitting into training and test sets as used in published studies concerning text classification. A subset of the collection suited our purposes. First of all it was necessary that a document be indexed with only one topic. This limited the subset to 9494 documents. And, secondly, it was required that the document be a regular text document. This further limited the subset to 8654 documents. From that subset only those documents belonging to the ten most frequently occurring topics in the subset, as listed in Table 2.2, were employed.

Preprocessing textual documents can improve the classification results. A number of methods are:

- Stop word filtering: filters out common words such as *the*.

[3]`http://www.research.att.com/~lewis/`

Table 2.2: Subset of Reuters-21578 used in experiments

tag	topic	size
acq	Mergers/Acquisitions	2125
coffee	Coffee	114
crude	Crude Oil	355
earn	Earnings and Earnings Forecasts	3735
interest	Interest Rates	211
money-fx	Money/Foreign Exchange	259
money-supply	Money Supply	97
ship	Shipping	156
sugar	Sugar	135
trade	Trade	333

- Stemming — also called base form reduction — is the process of reducing words to their stems, typically accomplished through suffix stripping [Pai90].

- Conflation is the process of grouping together non-identical words which refer to the same principal concept [Pai90].

We employed stop word filtering, but neither stemming nor conflation, in our experiments. See Table 2.3 for the stop words used. All non-letters were treated as spaces and all sequences of multiple spaces were replaced by a single space.

The following experiment was conducted to illustrate the performance of n-gram representations for classifying textual documents. Given a set of classes, the cluster prototype vectors were initialized by averaging a certain number of normalized document vectors. Note that the resulting cluster prototype vectors are automatically normalized; since we employed the Minkowski 1-norm for normalization (Equation 2.1), all the elements of a weighted frequency distribution vector were larger or equal to zero, and at least one element was larger than zero. The proof is given in Appendix C. Documents from those classes were chosen at random and the *nearest cluster algorithm* (Algorithm 1 on page 18) was executed to determine which class a document had been chosen from. The number of times this was done correctly divided by the total number of tries, constituted the performance. This *n-gram cluster experiment algorithm* is shown in Algorithm 2.

Table 2.3: Stop words removed during preprocessing

about	after	again	all	also	always	am	an
and	any	are	as	at	be	been	before
between	beyond	both	but	by	can	come	could
did	do	does	each	every	first	for	from
get	go	had	has	have	he	here	how
if	in	into	is	it	its	last	let
like	made	make	makes	many	may	me	mine
more	most	much	my	never	no	not	now
of	off	on	one	only	or	other	our
ours	out	over	reuter	reuters	said	same	say
she	should	so	some	such	than	that	the
their	them	then	there	these	they	this	to
too	try	until	use	very	was	we	went
were	what	when	where	which	who	whose	why
will	with	without	would	yes	yet	you	your
yours							

Algorithm 2 n-gram classification experiment

```
$k \leftarrow$ number of classes
$l \leftarrow$ number of documents to average for cluster initialization
$m \leftarrow$ number of documents to classify for each class
for all $k$ classes do
  $\vec{c_i} \leftarrow$ normalized average of $l$ documents belonging to class $i$
end for
for all $k$ classes do
  for all $m$ documents belonging to current class do
    let $d$ be the document
    compute normalized $n$-gram frequency distribution vector $\vec{d}$
    $cluster \leftarrow nearest\ cluster$ (k,C,$\vec{d}$)
    if class of $d$ equals $cluster$ then
      $score \leftarrow score + 1$
    end if
  end for
end for
$result \leftarrow score/(k * m)$
```

Table 2.4: Comparison of trigram classification with and without stop word filtering. The results are in the form of percentage correctly classified.

Topics	Unfiltered	Keywords removed
Coffee, trade	99.0	98.0
+ crude	95.3	94.7
+ money-fx	89.5	90.0
+ sugar	88.4	89.6
+ money-supply	84.7	85.3
+ ship	82.0	82.3
+ interest	78.3	79.8
+ acq	79.1	79.3
+ earn	78.6	79.0

The results of running the *n-gram cluster experiment algorithm* (Algorithm 2) for the number of classes ranging from 2 to 10, the number of documents to average for cluster initialization set to 30, the number of documents to classify set to 50, $n = 3$ and $p = 1$ (Manhattan metric) on this dataset, both without any preprocessing and with stop word filtering, are shown in Table 2.4. The second row displays the results of the 2-topic experiments, the topics being coffee and trade. Each following row adds one topic, indicated with $+ < topic >$. Thus the third row shows the results of the 3-topic experiments, the topics being coffee, trade, and crude.

Stop-word filtered n-gram clustering has a small overall advantage over unfiltered n-gram clustering, therefore filtering will be employed in this research.

2.4 Phonetic representation

Although there has been a substantial amount of research into word and letter n-grams going back for several decades, the use of phoneme n-grams is a more recent development. Sequences of phoneme symbols have been analyzed employing n-grams to guess missing phonemes in [YH92]. And phoneme n-grams have been employed in spoken document retrieval in [NWZ00, Ng00].

In most languages written text does not correspond to its pronunciation, so that in order to describe correct pronunciation some kind of symbolic representation is needed. Every language has a different phonetic alphabet, though the International Phonetic Association[4] has attempted to create a universal phonetic alphabet, aptly named the International Phonetic Alphabet. This alphabet is depicted in Figure 2.1[5]. It consists of the following seven sections: *consonants (pulmonic)*, *consonants (non-pulmonic)*, *vowels*, *diacritics*, *suprasegmentals*, *tones & word accents*, and *other symbols*. For a detailed description of this alphabet, see [Ass99].

All human languages can be represented by a phonetic alphabet ranging in size from about 13 to 85 phonemes. The number of phonemes in English and most other languages cannot be defined exactly, due to differences in dialect and ambiguity as to how precisely to define a phoneme. However, most linguists put the number of phonemes in English at around 40.

Phonetic alphabets satisfy the conditions set in Section 2.2:

1. $13 \leq |\mathcal{A}| \leq 85$ versus many thousands or even tens of thousands of terms in a traditional term frequency distribution

2. per definition all speech can be represented by phonemes

In order to experiment with classifying spoken documents through the use of phoneme n-grams, a topical phoneme encoded spoken document dataset is required. Not finding this readily available we considered two options, namely:

1. obtain a topical spoken document dataset and software for transforming it to phoneme space

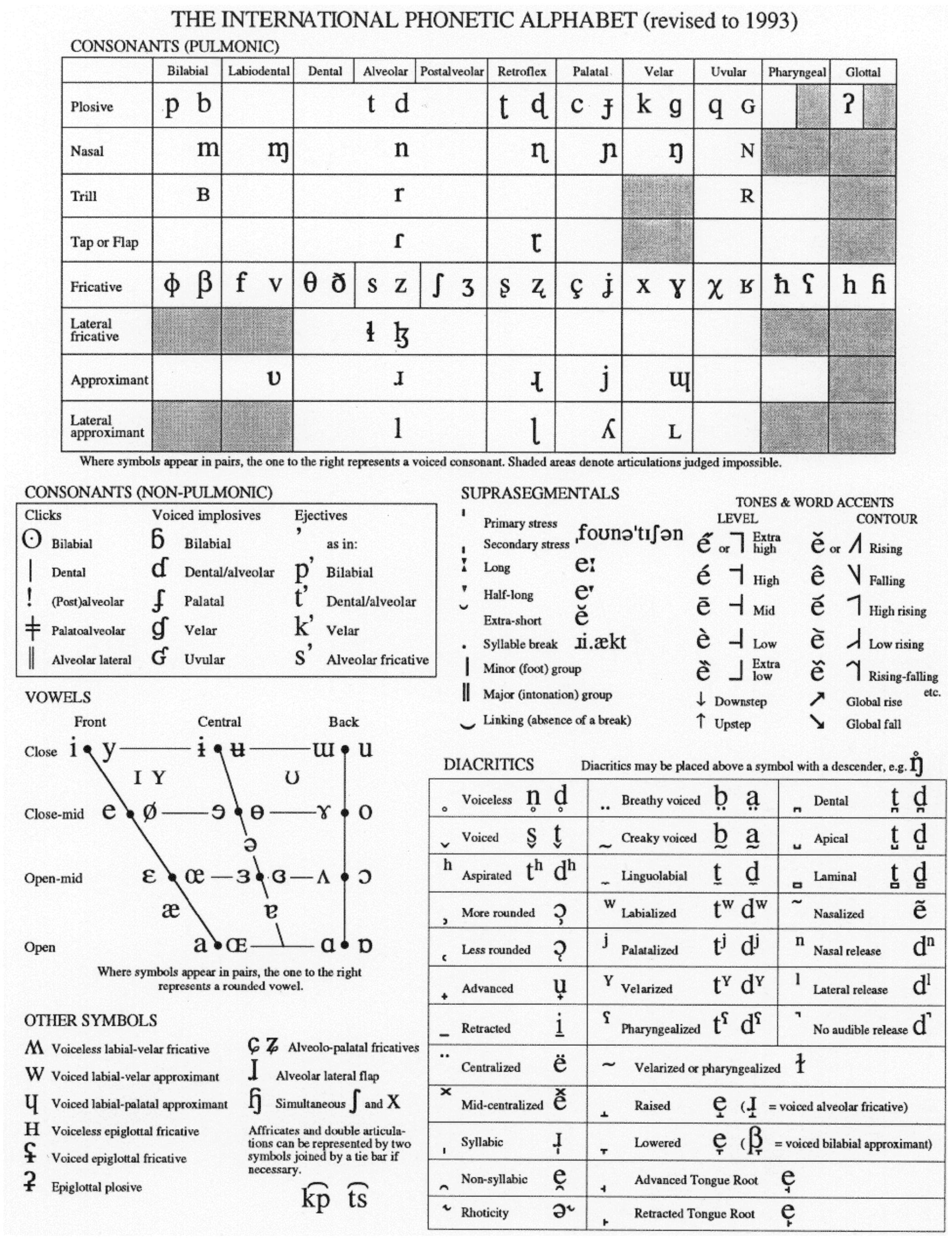
2. simulate the previous option by transforming a topical textual document dataset to phoneme space

We decided on the latter option for two reasons. First of all, we already had a topical textual document dataset, namely the Reuters-21578 text categorization collection introduced in the previous section. And secondly, given a word-to-phoneme dictionary this transformation can be easily implemented, therefore making it unnecessary to obtain transformation software. Employing the Reuters-21578 dataset has the added advantage of facilitating comparisons between the classification of the original textual documents and the classification of their phonetic transformations. The transformation was

[4]http://www.arts.gla.ac.uk/IPA/ipa.html

[5]Image provided by the International Phonetic Association (c/o Department of Linguistics, University of Victoria, Victoria, British Columbia, Canada)

Figure 2.1: International Phonetic Alphabet



THE INTERNATIONAL PHONETIC ALPHABET (revised to 1993)

CONSONANTS (PULMONIC)

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ	▨	▨
Trill	ʙ			r				▨	ʀ		▨
Tap or Flap				ɾ		ɽ		▨			▨
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative	▨	▨		ɬ ɮ						▨	▨
Approximant		ʋ		ɹ		ɻ	j	ɰ			▨
Lateral approximant	▨	▨		l		ɭ	ʎ	ʟ		▨	▨

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

CONSONANTS (NON-PULMONIC)

Clicks	Voiced implosives	Ejectives
ʘ Bilabial	ɓ Bilabial	ʼ as in:
ǀ Dental	ɗ Dental/alveolar	pʼ Bilabial
ǃ (Post)alveolar	ʄ Palatal	tʼ Dental/alveolar
ǂ Palatoalveolar	ɠ Velar	kʼ Velar
ǁ Alveolar lateral	ʛ Uvular	sʼ Alveolar fricative

SUPRASEGMENTALS

ˈ	Primary stress	ˌfoʊnəˈtɪʃən
ˌ	Secondary stress	
ː	Long	eː
ˑ	Half-long	eˑ
˘	Extra-short	ĕ
.	Syllable break	ɹi.ækt
|	Minor (foot) group	
‖	Major (intonation) group	
‿	Linking (absence of a break)	

TONES & WORD ACCENTS

LEVEL		CONTOUR	
e̋ or ˥	Extra high	ě or ˩˥	Rising
é ˦	High	ê ˥˩	Falling
ē ˧	Mid	e᷄ ˦˥	High rising
è ˨	Low	e᷅ ˩˨	Low rising
ȅ ˩	Extra low	e᷈ ˧˦˧	Rising-falling etc.
↓	Downstep	↗	Global rise
↑	Upstep	↘	Global fall

VOWELS

	Front	Central	Back
Close	i y	ɨ ʉ	ɯ u
	ɪ ʏ		ʊ
Close-mid	e ø	ɘ ɵ	ɤ o
		ə	
Open-mid	ɛ œ	ɜ ɞ	ʌ ɔ
	æ	ɐ	
Open	a ɶ		ɑ ɒ

Where symbols appear in pairs, the one to the right represents a rounded vowel.

OTHER SYMBOLS

ʍ Voiceless labial-velar fricative
w Voiced labial-velar approximant
ɥ Voiced labial-palatal approximant
ʜ Voiceless epiglottal fricative
ʢ Voiced epiglottal fricative
ʡ Epiglottal plosive
ɕ ʑ Alveolo-palatal fricatives
ɺ Alveolar lateral flap
ɧ Simultaneous ʃ and x

Affricates and double articulations can be represented by two symbols joined by a tie bar if necessary. k͡p t͡s

DIACRITICS Diacritics may be placed above a symbol with a descender, e.g. ŋ̊

◌̥ Voiceless n̥ d̥	◌̤ Breathy voiced b̤ a̤	◌̪ Dental t̪ d̪
◌̬ Voiced s̬ t̬	◌̰ Creaky voiced b̰ a̰	◌̺ Apical t̺ d̺
ʰ Aspirated tʰ dʰ	◌̼ Linguolabial t̼ d̼	◌̻ Laminal t̻ d̻
◌̹ More rounded ɔ̹	ʷ Labialized tʷ dʷ	◌̃ Nasalized ẽ
◌̜ Less rounded ɔ̜	ʲ Palatalized tʲ dʲ	ⁿ Nasal release dⁿ
◌̟ Advanced u̟	ˠ Velarized tˠ dˠ	ˡ Lateral release dˡ
◌̠ Retracted i̠	ˤ Pharyngealized tˤ dˤ	̚ No audible release d̚
◌̈ Centralized ë	◌̴ Velarized or pharyngealized ɫ	
◌̽ Mid-centralized e̽	◌̝ Raised e̝ (ɹ̝ = voiced alveolar fricative)	
◌̩ Syllabic ɹ̩	◌̞ Lowered e̞ (β̞ = voiced bilabial approximant)	
◌̯ Non-syllabic e̯	◌̘ Advanced Tongue Root e̘	
˞ Rhoticity ə˞	◌̙ Retracted Tongue Root e̙	

accomplished by processing the stop-list filtered documents one at a time, replacing all the words with their phonetic counterparts (unknown words were skipped). The Carnegie Mellon University Pronouncing Dictionary[6] was employed to this end. This is a machine-readable pronunciation dictionary for North American English that contains over 125,000 words and their transcriptions. Version 0.6d was stripped of all entries containing symbols other than letters, leaving 112,129 words. It employs 39 phonemes and three types of lexical stress, namely no stress, primary stress and secondary stress. We retained all 39 phonemes, but discarded the lexical stress.

In the rest of this dissertation the token alphabet consisting of the above indicated 39 phonemes and delimeter will be denoted by $\mathcal{P}$. The advantages of employing n-grams over $\mathcal{P}$ are similar to the advantages of employing n-grams over $\mathcal{T}$ (see Section 2.3), namely:

- Robustness: Relatively insensitive to pronunciation variations/errors
- Completeness: Token alphabet known in advance
- Domain independency: Language[7] and topic independent
- Efficiency: One pass processing
- Simplicity: No linguistic knowledge required beyond speech-to-phoneme transformation

To illustrate the performance of n-gram representations for classifying phonetic documents, the experiment described in Section 2.3 was repeated, this time using $\mathcal{P}$ and the phonetically transcribed Reuters dataset. Results for varying values of n are shown in Table 2.5.

The results indicate that phoneme unigrams are not representative enough for classification purposes, but that n-grams for higher values of n are. While trigrams have an advantage over bigrams, quadgrams offer no further improvement, sometimes even constituting a disadvantage. Considering that the computational time increases exponentially with n, it would seem that bigrams and trigrams clearly offer the best performance and will therefore be the main focus of the research reported in this dissertation.

[6]http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[7]language independent under the condition of a shared alphabet

Table 2.5: Comparison of phoneme n-gram clustering for $n = 1, 2, 3, 4$. The results are in the form of percentage correctly classified.

Topics	$n = 1$	$n = 2$	$n = 3$	$n = 4$
Coffee, trade	82.0	99.0	100.0	100.0
+ crude	72.7	92.0	94.7	94.0
+ money-fx	69.5	87.5	92.0	92.0
+ sugar	71.6	87.6	90.0	90.0
+ money-supply	70.3	83.3	85.3	85.0
+ ship	63.1	79.4	82.3	80.9
+ interest	58.0	75.5	79.0	77.3
+ acq	54.2	74.9	79.3	78.2
+ earn	53.8	74.0	78.6	78.0

2.5 Weighted representation

As shown in Table 2.4, classification accuracy quickly drops as more classes must be distinguished. For real world use we need to have higher accuracy rates. This can be accomplished by associating weights with the n-grams.

Let $\vec{w} = (w_1, w_2, \ldots, w_{\mathcal{A}(n)})$ with $0 \leq w_i \leq 1$ for $i = 1, 2, \ldots, \mathcal{A}(n)$ be the weight vector representing the relative importance of the different n-gram frequencies. The weighted n-gram frequency vectors $\overrightarrow{d^w} = (d_1^w, d_2^w, \ldots, d_{\mathcal{A}(n)}^w)$ and $\overrightarrow{e^w} = (e_1^w, e_2^w, \ldots, e_{\mathcal{A}(n)}^w)$ are defined as follows: $d_i^w = d_i w_i$ and $e_i^w = e_i w_i$ for $i = 1, 2, \ldots, \mathcal{A}(n)$. The normalized weighted n-gram frequency vectors $\overrightarrow{\overline{d^w}} = (\overline{d_1^w}, \overline{d_2^w}, \ldots, \overline{d_{\mathcal{A}(n)}^w})$ and $\overrightarrow{\overline{e^w}} = (\overline{e_1^w}, \overline{e_2^w}, \ldots, \overline{e_{\mathcal{A}(n)}^w})$ are defined as follows: $\overline{d_i^w} = d_i^w / |\overrightarrow{d^w}|$ and $\overline{e_i^w} = e_i^w / |\overrightarrow{e^w}|$ for $i = 1, 2, \ldots, \mathcal{A}(n)$. The similarity between d and e may then be estimated by applying the Minkowski ℓ_p-metric to $\overrightarrow{\overline{d^w}}$ and $\overrightarrow{\overline{e^w}}$:

$$\ell_p(\overrightarrow{\overline{d^w}}, \overrightarrow{\overline{e^w}}) = \sqrt[p]{\sum_{j=1}^{\mathcal{A}(n)} (\overline{d_j^w} - \overline{e_j^w})^p} \tag{2.5}$$

The central research theme of this dissertation will be the search for an algorithm that optimizes $\vec{w}$ under the constraints imposed by the application area of Adaptive Information Filtering.

Table 2.6: Letter n-grams occurring in the part of the Reuters dataset employed.

n	27^n	# occurring	% occurring
1	27	27	100.0000
2	729	676	92.7298
3	19,683	8,013	40.7103
4	531,441	40,197	7.5638
5	14,348,907	124,225	0.8657
6	387,420,489	290,569	0.0750

Table 2.7: Phoneme n-grams occurring in the part of the Reuters dataset employed.

n	40^n	# occurring	% occurring
1	40	40	100.0000
2	1,600	1,095	68.4375
3	64,000	10,303	16.0984
4	2,560,000	45,958	1.7952
5	102,400,000	156,200	0.1525
6	4,096,000,000	360,979	0.0088

2.6 Storage and retrieval

All n-gram frequency distributions are vectors of size $\mathcal{A}(n)$ and can, therefore, be stored in arrays of the same size in a straightforward manner. This is by far the easiest way to implement their storage and for $1 \leq n \leq 2$ also computationally efficient both in terms of processing speed and space requirements. For $n \geq 3$, however, the vectors become ever more sparse in typical application domains such as text or phonetics representation. This is illustrated by Table 2.6 and Table 2.7. Processing speed and space requirements will, therefore, be severely and adversely affected unless more sophisticated data structures are employed for $n \geq 3$.

There are various methods for efficiently storing sparse vectors, differing in complexity, speed and space requirements, dependent on the particular application demands. The simplest method is to store the non-zero values of an array together with their indices in a list. For example, the vector (0,0,0,0,9,0,0,3,0) can be stored as {(5,9),(8,3)}. While this greatly reduces the storage space requirements, it also introduces quite a bit of overhead, both in terms of the storage space required for the indices, as well as the computational overhead associated with accessing a specific n-gram frequency value. This can be overcome by employing hash tables, and in particular for n-gram vectors, recursive hash tables [Coh97]. Although recursive hash tables offer very high performance and require minimal storage space, they do still involve computational overhead, namely executing a hash function and resolving collisions. An alternative approach that avoids such computational overhead at the cost of allowing a certain amount of sparseness is the *master index method*. In this method a master index stores n-gram indices for all n-grams encountered and all vectors are stored as arrays of the same size as the master index, their elements associated with the corresponding master index element. For example, if the master index at a particular time is (23,57,89,139,254), then the array (0,4,0,0,13) represents a document which consists of 4 n-grams with index 57 and 13 n-grams with index 254. The great advantage of this method is that operations on the vectors, for instance adding, incur no overhead. Encountering a not yet indexed n-gram involves updating the master index and inserting zeros in the corresponding location of all stored arrays. It is therefore important to minimize such occurrences by proper initialization.

For the research described in this dissertation a custom C++ n-gram class library[8] was developed employing the *master index method*. This library defines the class *ngramdistribution* and some supporting functions, types and constants. It supports the use of n-gram distributions for various applications. The value of n as well as the token alphabet is user definable. There is full support for exception handling through the use of custom exceptions of the type *ngram_exception*. A detailed description is given in Appendix A.

2.7 Chapter conclusions

This chapter provides an introduction to representing documents in vector space and measuring similarity between documents and user profiles employing the Minkowski ℓ_p-metric (Equation 2.2 on page 17). It further introduces the *nearest cluster algorithm* (Algorithm 1 on page 18) as the classification algorithm employed in the dissertation research. This is followed by an introduction to n-grams and the various ways they can be employed to represent

[8]`http://www.liacs.nl/home/dtauritz/ngram/`

documents. In particular, n-gram frequency distributions are discussed as the method of representation employed in the dissertation research. The following criteria are provided for evaluating the effectiveness of an n-gram token alphabet: it should be small and it should be complete. Examples of many different applications of n-grams are provided along with an extensive list of references.

The next part of the chapter examines in detail the employment of n-grams with the Latin alphabet as token alphabet for representing textual documents. This type of n-gram will be further referred to as a *letter n-gram.* Advantages of employing letter n-grams for this purpose are listed and the Reuters-21578 text categorization collection is introduced as the dataset employed in our experiments. An experiment demonstrating the potential of letter n-grams for the classification of textual documents is described. The algorithm (Algorithm 2 on page 24) is presented and the results displayed in Table 2.4 (page 25). The results show that when few classes are involved the classification accuracy is very high, but that as classes are added this accuracy drops considerably. They also show that, excepting experiments with few classes, filtering out common words such as *the* and *and* provides a small improvement in accuracy.

This section is followed by a section on the employment of n-grams with a token alphabet consisting of phonemes. This type of n-gram will be further referred to as a *phoneme n-gram.* The International Phonetic Alphabet is introduced and the Carnegie Mellon University Pronouncing Dictionary is presented as the key to obtaining our phoneme encoded dataset. The advantages of employing phoneme n-grams are indicated and the results of an experiment demonstrating the classification of spoken documents for various values of n are displayed in Table 2.5 (page 29). The best results are obtained for trigrams, followed by bigrams. Unigrams provide insufficient discriminating power and quadgrams perform slightly worse than trigrams while requiring exponentially more computational resources. Just as with the letter n-gram experiment, classification accuracy drops considerably as the number of classes is increased.

The next part of the chapter introduces weighted n-grams as a potential solution to the problem of insufficient classification accuracy. Optimizing these weights is the focus of the entire remainder of the dissertation. The last part of the chapter is concerned with the storage and retrieval of n-grams. For larger values of n this is not a trivial problem as it involves huge sparse vectors. Various methods such as *indexed lists*, *hash tables* and the method employed in our research, *master indexed lists*, are described. This section concludes with the introduction of the C++ n-gram class library.

Chapter 3

Two-pool evolutionary algorithm

Synopsis

It is our aim to optimize the weight vector $\vec{w}$ in order to improve classification accuracy for the application area of Adaptive Information Filtering (AIF). We do not have an a priori solution to this optimization problem. And to complicate matters further, the optimal value of $\vec{w}$ changes over time due to the nature of AIF. Therefore, classic optimization techniques are inadequate due to the size of the search space. This chapter introduces Evolutionary Computation (EC) as a method capable of efficiently approximating optimal values of $\vec{w}$.

A short introduction to evolutionary computation is given in Section 3.1, and some advantages and disadvantages are noted. In Section 3.2 some background to the field is offered by describing the four main streams from which the field originated: Genetic Algorithms, Evolutionary Programming, Evolution Strategies, and Genetic Programming. This is followed in Section 3.3 by a detailed derivation of the general purpose two-pool classification evolutionary algorithm developed in the dissertation research. The component of the algorithm responsible for performing an actual step of the evolutionary process (selection, reproduction, mutation and competition) is left undefined as it is specific to the particular problem domain in which the algorithm is applied. Section 3.4 describes the specification of this component as employed in the domain of AIF. Details concerning representation and related matters are also provided. Section 3.5 illustrates the working of the two-pool classification evolutionary algorithm with the evolutionary component described in Section 3.4. This is done in the form of an experiment employing objects generated on-the-fly. Section 3.6 contains the chapter conclusions.

3.1 Introduction to evolutionary computation

EC is the field that studies Evolutionary Algorithms (EAs). EAs are a class of optimization algorithms which come in handy when no a-priori solutions to a specific optimization problem are available. They work by evolving a population of trial solutions employing techniques inspired by evolutionary biology [Mic96, Fog95]. Some of their advantages are:

General purpose EAs are not problem specific; they show acceptable performance for acceptable costs over a wide range of problems.

Complex problems EAs are capable of solving complex problems for which no classical analytical method is available; they have the ability to deal with large amounts of data, many free parameters, complex relationships between the parameters, and many local optima.

Solution availability EAs are iterative optimizing algorithms; the best solution found so far is always available, in contrast to the usual case where no solution is available till the algorithm has terminated.

There are also some disadvantages, including:

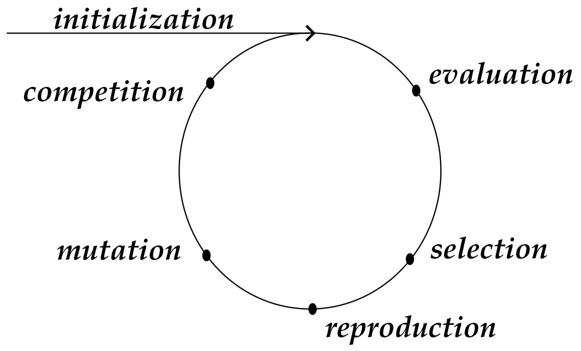
Premature convergence EAs are prone to getting stuck in local optima; for many EAs there is no guarantee that the optimal solution will be found in finite time.

Computationally intensive EAs tend to be computationally intensive, especially when many free parameters are involved and the required population is large.

Parameter optimization Determining optimal values for the EA parameters — those are the parameters that control the EA, not the problem parameters being optimized — is very difficult and problem dependent.

The main components of an EA are initialization of the start population plus those components associated with the evolutionary cycle: evaluation, selection, reproduction, mutation and competition. See Figure 3.1 for a diagram of the evolutionary cycle. Each member of a population consists of a

Figure 3.1: Evolutionary cycle



trial solution to a particular problem and, possibly, attributes such as age. A trial solution is encoded in genes. The genes are represented by, for instance, bits, integers or floating point numbers. There are various methods for initializing the start population. It can be done, for example, randomly or homogeneously. Sometimes specific knowledge is available allowing more directed initialization, for instance knowledge gleaned from previous optimization attempts. There are in general no rules for this, the best method for a particular application must often be determined experimentally. To evaluate the members of a population, it is necessary to determine their individual fitness. This is dependent on their genotype and their environment and is computed using the so-called fitness function. In the next step of the cycle, parents are selected to generate offspring. In most EAs population members with a higher fitness have a correspondingly greater chance to be selected. This causes selective pressure. The higher the selective pressure, the faster the EA converges, but also the greater the chance that it converges to a local optimum instead of to a global one. In the reproductive phase offspring are created by applying so-called genetic operators to the parents. A genetic operator can be as simple as the duplication function, creating a duplicate of a parent, or a more complex operator such as performing crossover on two (or more) parents. There are various ways of performing crossover, but they all have in common that the genes of the offspring are obtained by applying a function to the genes of the parents. In one-point crossover all the genes up to a specific point called the crossover point are duplicated from the one parent, all the genes after the crossover point from the other parent. In uniform crossover each gene has an equal chance to be selected from each of the parents. Which types of genetic operators are best depends on the specific application and invariably needs to be determined experimentally. After reproduction the genes of the offspring are mutated with a certain probability. This introduces new genetic material into the population pool, which allows new regions of the search space to be entered and reduces the chance of premature convergence. In the last phase of the cycle the newly expanded population is reduced in order to limit the population size. This can be done in various ways, such as randomly, based on fitness or dependent on an attribute such as age. For a comprehensive hands-on introduction to EAs see [Mic96], for a more philosophical approach see [Fog95].

3.2 Background

The field of Evolutionary Computation has its origins in a number of different models that, until recently, were studied independently and were not seen to be instances of a generic model unifying the field. To this day the various models maintain their own identities and supporters, although increasingly the lines separating the models are fading. Some of the better known models are:

Genetic Algorithms (GAs) Classical GAs were developed by John Holland in the 1960's and 1970's. They operate on fixed-length binary strings which encode the optimization problem and employ binary crossover and mutation. Holland laid the theoretical foundations for classical GAs in [Hol75].

Evolutionary Programming (EP) was originally developed by Lawrence J. Fogel in the 1960's. He co-authored the 1966 landmark publication for the field [FOW66], in which finite state automata were evolved to predict symbol strings generated from Markov processes and non-stationary time series. Unlike classical GAs, classical EP employs no prescribed type of representation. Instead of operating on encoded problems, EP operates directly on the problem representation. Typically the only genetic operator employed is mutation. The mutation operator makes changes to the representation according to a statistical distribution in which minor changes are far more prevalent than major changes.

Evolution Strategies (ES) were originally developed by Ingo Rechenberg and Hans-Paul Schwefel in 1963. They were intended for real valued function optimization employing multi-variate zero-mean Gaussian mutations. While they are similar to EP, they differ in a number of ways. For instance, EP typically employs stochastic selection while ES employ deterministic selection. And ES employ recombination, unlike EP. Landmark publications in the field of ES include [Rec73] and [Sch81].

Genetic Programming (GP) was introduced in 1990 by John R. Koza [Koz90]. GP is an automated method for creating a computer program to solve a particular problem. This is done by evolving a population of computer programs. These programs are expressed as parse trees, rather than as lines of code. Because of this, Lisp is often used as the programming language of choice. In GP the most important genetic operator is recombination, implemented by exchanging randomly selected subtrees in the individuals.

For an in-depth comparison of the first three models see [Bäc96]. The two-pool classification algorithm derived in Section 3.3 and employed in the dissertation research, as specified in Section 3.4, is inspired by the Evolutionary Strategies approach. As with ES, the aim is to optimize problems encoded in real-valued vectors. The same type of genetic operators are employed: Gaussian zero-mean mutation and uniform crossover. Some of the differences are in the selection process and the self-adaption component of ES that provides local fine-tuning of the mutation rate. The use of separate child and adult pools is also specific to our two-pool classification algorithm.

3.3 Derivation of two-pool EA

In this section we will consider the development of classification EAs (CEAs) from a general perspective. However, in our case the members of a population are weight vectors, the score of a member is the number of correctly classified documents and its age is the total number of documents it has classified.

The set of objects to be classified will be denoted by S and the number of objects in S with $|S|$. An object in S will be denoted by σ and $c(\sigma)$ will represent the class that σ maps to. The set $P = \{P_1, P_2, \ldots, P_{pop\_size}\}$ is the population of trial solutions, with *pop_size* a positive integer. For the purpose of indexing the population members we define i as an integer between 1 and *pop_size*. Two essential components of any CEA are the evaluation of all the population members and, based on that, the evolvement of the population. The evolvement component will be denoted with $EVOLVE(P)$ and will remain undefined as it is dependent on the application domain. The evaluation component will be denoted with $EVAL(S, P)$ and is defined as follows:

$$EVAL(S,P): \quad \forall P_i \in P \text{ determine } FITNESS(S, P_i) \tag{3.1}$$

The result of classifying an object given a trial solution is either zero (incorrect) or one (correct). The result function is defined as follows:

$$RESULT(\sigma, P_i) = \begin{cases} 0 & \text{if } CLASSIFY(\sigma, P_i) \neq c(\sigma) \\ 1 & \text{if } CLASSIFY(\sigma, P_i) = c(\sigma) \end{cases} \tag{3.2}$$

The result function works by comparing the actual mapping of an object to the mapping of that object computed using a trial solution. The function which performs that computation is defined as:

$$CLASSIFY(\sigma, P_i) = \text{the class } \sigma \text{ maps to using } P_i \tag{3.3}$$

The fitness of a trial solution given an object set is the average score of that trial solution on classifying all the objects in the object set. The range of the fitness is from zero to one with zero being the worst (all classifications incorrect) and one the best (all classifications correct). The fitness function is defined as follows:

$$FITNESS(S, P_i) = \frac{\sum_{\sigma \in S} RESULT(\sigma, P_i)}{|S|} \tag{3.4}$$

The *static object set CEA* can then be defined as given in Algorithm 3. First P and S are initialized and the initial population evaluated. This is followed by the main loop in which the population is consecutively evolved, then evaluated. The loop terminates as soon as the termination condition is true.

Note that the efficiency of this algorithm can be enhanced by restricting the evaluation of population members to newly added members only.

Algorithm 3 Static object set CEA

```
initialize $P, S$
EVAL(S,P)
while not termination condition do
    EVOLVE(P)
    EVAL(S,P)
end while
```

The initialization of P is performed by creating the start population and initializing each individual in it. S is initialized by creating the set of objects to be classified. The definition of the termination condition depends on various factors, such as time and solution requirements. A fixed number of loop executions puts an upper boundary on the required computational time, but does not guarantee convergence to an optimal solution. On the other hand, a termination condition based solely on the best solution found so far leaves the computational time unbounded. The property of evolutionary computation that at all times the best solution found so far is available, as opposed to many classical algorithms which need to complete their entire execution in order to compute a solution, is a key advantage.

3.3.1 Expanding object set

If S expands over time we can simply execute the *static object set CEA* (Algorithm 3) after each expansion to find a mapping from object space to class space at any given time. If the set of objects is smaller than the object space and represents it better as it expands, then the mapping found by the CEA will better approximate the mapping from object space to class space as time progresses. In this case it is likely that the mapping found at any particular time is a good approximation of the mapping to be found the following time and therefore would make a good starting point for the next search. Time will be denoted with τ and the object added to S at $\tau = \hat{\tau}$ with $\sigma_{\hat{\tau}}$. The *expanding object set CEA* is given in Algorithm 4. First P and S are initialized, the initial population is evaluated, and τ is set to 0. This is followed by an infinite loop in which first τ is increased by one indicating a single time step, then an object is added to S, and finally evolution takes place. Evolution works the same as with the static object set CEA: consecutively the population is evolved, then evaluated, till the termination condition is true. The comments with respect to the termination condition are identical to those in the previous section.

Algorithm 4 Expanding object set CEA

```
initialize $P, S$
EVAL(S,P)
$\tau \leftarrow 0$
loop
  $\tau \leftarrow \tau + 1$
  add $\sigma_\tau$ to $S$
  while not termination condition do
    EVOLVE(P)
    EVAL(S,P)
  end while
end loop
```

3.3.2 Shifting window

There are a number of reasons why we may not want to use an ever expanding set of objects to find a mapping from object space to class space. For one, this requires an ever increasing amount of computational resources, both in terms of memory and in CPU cycles. And secondly, the mapping may change over time so that obtaining $c(\sigma)$'s might prove to be an expensive operation or it is even possible that old $c(\sigma)$'s are not obtainable at all. In this case we can impose a shifting window on S limiting the number of objects to be used in the evolutionary process at any given time. The size of the shifting window will be indicated with ω. Once $|S| = \omega$ the window starts to shift. For each new object added to S, the oldest one is removed. The *shifting window CEA* is given in Algorithm 5. First P is initialized, S is set to the empty set, and τ is set to 0. This is followed by an infinite loop in which first τ is increased by one indicating a single time step, an object is added to S, and if now $|S| > \omega$ (true if $\tau > \omega$) the first added object to S is removed, the population is evaluated, and then evolution takes place. Evolution and its termination condition are again identical to the case of the static object set CEA.

Algorithm 5 Shifting window CEA

```
initialize $P$
$S \leftarrow \emptyset$
$\tau \leftarrow 0$
**loop**
  $\tau \leftarrow \tau + 1$
  add $\sigma_\tau$ to $S$
  **if** $(\tau > \omega)$ **then**
    remove $\sigma_{\tau-\omega}$ from $S$
  **end if**
  *EVAL(S,P)*
  **while** not termination condition **do**
    *EVOLVE(P)*
    *EVAL(S,P)*
  **end while**
**end loop**
```

3.3.3 Age

One thing we lose by employing a shifting window is the information with respect to how well trial solutions performed on objects no longer contained in S. And the smaller ω is, the greater this loss. We introduce the concepts of member age and member score to preserve this information in our *shifting window CEA*. The age of a member is defined as the number of population generations since the creation of that member and is denoted with P_i^{age}. The score of a member is defined as the number of correct classifications it has made since its creation and is denoted with P_i^{score}. The fitness function is now defined as:

$$FITNESS(P_i) = \frac{P_i^{score}}{P_i^{age}} \tag{3.5}$$

And the evaluation component becomes:

$$\begin{aligned}EVAL(S,P) &: \forall P_i \in P : \forall \sigma \in S : if\ \sigma = \sigma_\tau\ or\ P_i\ changed : \\ &P_i^{age} \leftarrow P_i^{age} + 1, P_i^{score} \leftarrow P_i^{score} + RESULT(\sigma, P_i) \\ &\text{and compute } FITNESS(P_i)\end{aligned} \tag{3.6}$$

3.3.4 Two pool

One of the consequences of the new way of determining fitness is that, as the age of a member increases, so does its statistical reliability in approximating the *true* fitness of a member. That is, its fitness if computed using S equal to the entire object space. If, when producing offspring, the new member's score and age are set to zero, as opposed to basing them on those of its parent(s), its statistical reliability plunges and time is needed to recover some measure of reliability. In that case it is necessary to prevent the new member from participating in the evolution process until it *matures*. This can be accomplished by splitting the population into two pools, namely a child pool P^c and an adult pool P^a with $P = P^c \cup P^a$, $|P^c|$ the number of members in P^c , $|P^a|$ the number of members in P^a and *maturity age* the age at which members are moved from P^c to P^a. Note that this requires a modified $EVOLVE(P)$ function which restricts the selection and reproduction phases to P^a and adds the offspring generated from P^a to P^c. The *two-pool CEA* is given in Algorithm 6. First P is initialized such that the population is equal to the child pool and the adult pool is empty, S is set to the empty set, and τ is set to 0. This is followed by an infinite loop in which first τ is increased by one indicating a single time step, an object is added to S, and if now $|S| > \omega$ (true if $\tau > \omega$) the first added object to S is removed, then evolution takes place. While the termination condition of the evolution loop is identical to that employed with the static object set CEA, the evolution loop has changed. Instead of the entire population evolving, this is now restricted to the adult pool. Thus, if the adult pool is empty, no evolution takes place. And after evaluating the population, any children that have reached maturity are moved to the adult pool.

3.3.5 Simplified two pool

A special case of the two-pool CEA (Algorithm 6) can be distinguished when the shifting window size is set to one and the termination condition is set such that the inner loop is executed only once for each outer loop. This results in a considerable increase in the speed of the algorithm. This *simplified two-pool CEA* is given in Algorithm 7. First P is initialized such that the population is equal to the child pool and the adult pool is empty, and τ is set to 0. This is followed by an infinite loop in which first τ is increased by one, indicating a single time step, and then evolution takes place. Evolution

Algorithm 6 Two-pool CEA

```
initialize $P$ such that $P = P^c$ and $P^a = \emptyset$
$S \leftarrow \emptyset$
$\tau \leftarrow 0$
loop
  $\tau \leftarrow \tau + 1$
  add $\sigma_\tau$ to $S$
  if $(\tau > \omega)$ then
    remove $\sigma_{\tau-\omega}$ from $S$
  end if
  while not termination condition do
    if $|P^a| > 0$ then
      EVOLVE(P)
    end if
    EVAL(S,P)
    for all $P_i \in P^c$ do
      if $P_i^{age} \geq$ maturity age then
        move $P_i$ from $P^c$ to $P^a$
      end if
    end for
  end while
end loop
```

is now implemented as a single instance of evolving the adult pool (if it is not empty), evaluating the population on the current object, and moving any children that have reached maturity to the adult pool. Note that the evaluation component (Equation 3.6 on page 43) can be simplified; because $S = \{\sigma_\tau\}$ the condition $if\ \sigma = \sigma_\tau$ *or* P_i *changed* is redundant.

Algorithm 7 Simplified two-pool CEA

```
initialize $P$ such that $P = P^c$ and $P^a = \emptyset$
$\tau \leftarrow 0$
loop
    $\tau \leftarrow \tau + 1$
    if $|P^a| > 0$ then
        EVOLVE(P)
    end if
    EVAL($\{\sigma_\tau\}, P$)
    for all $P_i \in P^c$ do
        if $P_i^{age} = maturity\ age$ then
            move $P_i$ from $P^c$ to $P^a$
        end if
    end for
end loop
```

3.4 Evolution

The algorithms presented in the previous section are generic in the sense that they do not specify the *EVOLVE*(P) function, nor the function *CLASSIFY*(σ, P_i) (Equation 3.3). In our research we employed the *simplified two-pool CEA* (Algorithm 7). This section will describe the specific *EVOLVE*(P) function we used with it. But first we will discuss representation and some related matters. Each population member has two attributes, namely age and score, represented by positive integers. The age and score of the members of the start population are initialized to 0. Taking Equation 3.5 and Equation 3.6 together we obtain $0 \leq FITNESS(P_i) \leq 1$, 0 indicating that all documents were classified incorrectly and 1 indicating that all documents were classified correctly. The genes hold the weights employed in the weighted

n-gram representations of the AIF system. They are implemented as doubles ranging from 0 to 1.

The *evolve algorithm* is shown in Algorithm 8. It is implemented as a loop to be executed the number of times indicated by the number of offspring variable. However, since for each offspring created an individual is removed from the adult pool, the number of times the loop can be executed is also bounded by the size of the adult pool. Two kinds of reproduction were investigated, namely single parent duplication (crossover disabled) and uniform crossover (crossover enabled). When crossover is enabled two parents are selected and uniform crossover is applied to obtain a child. When crossover is disabled a parent is selected and a child is obtained through duplication. The child, independent of the way in which it was created, is next mutated and then added to the child pool. Finally, the weakest adult is removed from the adult pool in order to maintain the same population size.

Algorithm 8 Evolve algorithm

```
for $i = 1$ to min(*number of offspring*,$|P^a|$) do
  if *evolution with crossover* then
    *first parent* $\leftarrow$ SELECTION($P^a$)
    *second parent* $\leftarrow$ SELECTION($P^a$)
    *child* $\leftarrow$ CROSSOVER(*first parent*,*second parent*)
  else
    *parent* $\leftarrow$ SELECTION($P^a$)
    *child* $\leftarrow$ COPY(*parent*)
  end if
  MUTATE(*child*)
  add *child* to $P^c$
  remove adult with lowest fitness from $P^a$
end for
```

Selection is carried out by choosing with a certain probability the fittest adult, otherwise by selecting with the same probability the next fittest, and so on till either a parent has been selected or the adult pool is exhausted, in which case the fittest adult gets selected by default. This means that the probability of selection decreases exponentially from the fittest adult to the least fit adult, creating considerable selective pressure. The amount of selective pressure can be regulated via the *selective pressure rate*, implemented by a floating point number ranging from 0 to 1. For example, a *selective* *pressure rate* of 0.1 means that we begin with a 0.1 chance of selecting the fittest member, then a 0.9×0.1 chance of selecting the second fittest member, a $0.9^2 \times 0.1$ chance of selecting the third fittest member, etc. The chance that none of the members is selected is $0.9^{|P^a|}$. Therefore, the total chance of selecting the fittest member is $0.1 + 0.9^{|P^a|}$.

The age and score attributes of the offspring were initialized to 0. Mutation was performed by adding with a certain probability zero-mean Gaussian noise to the genes of a member. Note that after two *age threshold* generations the size of the child pool is *maturity age* $\times$ *number of offspring*, under the condition that that is not larger than the population size. So, for example, if the size of the population is 100, the *maturity age* is 10 and the *number of offspring* is 4, then after 20 generations the child pool will stabilize at size 40 and the adult pool at size 60. If, however, *maturity age* $\times$ *number of offspring* is larger than the population size, the population will never stabilize and individuals will never get the chance to become mature adults. Therefore, we impose the following restriction: $|P| >$ *maturity age* $\times$ *number of offspring*.

3.5 Experiment

In order to illustrate the *simplified two-pool CEA* (Algorithm 7 on page 46) employing the *evolve algorithm* (Algorithm 8 on page 47), the following experiment was conducted. Given a set of classes, initialize the cluster prototype vectors by averaging a certain number of object vectors and normalizing the averages. Then execute the *simplified two-pool CEA* with the infinite loop modified so as to terminate after a certain specified number of loopings. Objects were generated on-the-fly in the following vector format (noise| information). Noise is encoded as a vector of noise elements $(a_1, a_2, \ldots, a_{noise})$ with *noise* indicating the amount of noise called the *noise factor* and $0 \leq a_i \leq 1$. Information is encoded as a vector $(b_1, b_2, \ldots, b_{classes})$ with *classes* indicating the number of classes and $b_i = 0.8$ if and only if the class the object maps to is class i, otherwise $b_i = 0.2$. So, for example, in an experiment with 4 classes an object belonging to class 3 could be represented as follows:

$$(\underbrace{0.1, 0.9, \ldots, 0.6}_{noise}, \underbrace{0.2, 0.2, 0.8, 0.2}_{information})$$

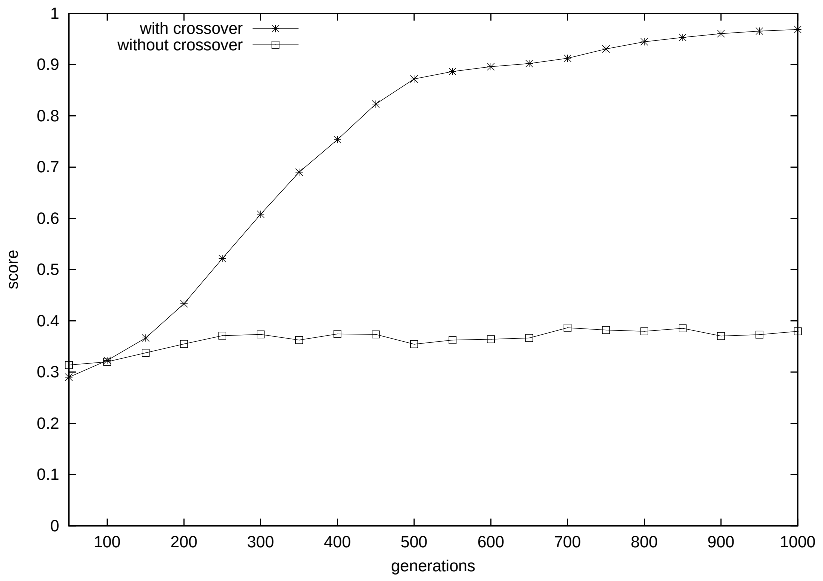
Table 3.1: Parameter values

parameter	value
noise factor	1000
number of runs	20
plot step size	50
number of generations	1000
population size	40
maturity age	10
number of offspring	2
gene initialization	uniform random
selective pressure	0.1
gene mutation chance	0.1
deviation	0.1
averaging number	30
system score array size	100

The genes of a population member encode a weight vector $\vec{w}$ of size *noise* + *classes*. The function $CLASSIFY(\sigma, P_i)$ (Equation 3.3) employs the *nearest cluster algorithm* (Algorithm 1 on page 18). The elements of σ are multiplied by the elements of $\vec{w}$ (the genes) and the resulting vector is normalized. Each of the cluster prototype vectors is multiplied in the same way and then normalized. The performance of a CEA is defined as the number of correct classifications made by the fittest member at the time of classification divided by the total number of loopings. For this particular experiment a global optimum is known, namely zeros to cancel out the noise and ones to maximize the information. Therefore the upper bound for the fitness is 1: perfect classification. The experiment was run with the parameter values as shown in Table 3.1.

A noise factor of 1000 means that an optimal solution will contain 1000 genes/variables with (near) zero values. The number of runs (20) indicates how many times the experiment was repeated to obtain a reliable average result. The plot step size (50) is the number of generations between each plot point in the result graph. The number of generations (1000) indicates how many documents will be processed in one run of the experiment. A gene initialization set to uniform random means that the genes of the members of the start population as well as the genes of offspring are uniform randomly set to between 0 and 1. A gene mutation chance of 0.1 indicates that in the mutation phase each gene has a chance of 0.1 of being mutated. A deviation of 0.1 means that a gene mutation is performed by adding Gaussian noise with zero mean and 0.1 standard deviation. The averaging number (30) is the number of objects that will be averaged during initialization to create the cluster prototype vectors. A system score array size of 100 indicates that the last 100 classifications (or the total number of classifications if there have been fewer than 100) are employed for determining performance. This limit is employed to ensure plasticity of the performance measure; otherwise new classifications have a decreasing impact while the performance measure is intended to measure current performance, not past performance. The results for experiments with and without crossover are shown in Figure 3.2. The result graph clearly shows the importance of the crossover operator in

Figure 3.2: The 8 class experiment



this particular experiment. While this does not necessarily mean that this operator will be as effective in our other experiments, it does warrant investigation. The crossover graph bears resemblance to an S-curve: during the first 200 generations the noise elements outweigh the information elements, causing a lot of poorly performing offspring to be created. As the weights associated with the noise elements decrease and the offspring improve in quality, the results improve at an accelerating pace. Between roughly 200 and 500 generations the results improve at a constant rate, improvements becoming more difficult as the amount of search space containing better solutions decreases in size. After about 500 generations the performance level obtained makes finding better solutions so difficult that the increase in performance starts to bog down. While the global optimum of 100% accuracy is not reached within the alotted number of generations, the experiment with crossover enabled surpasses the 95% mark without any evidence of premature convergence. After 1000 generations a typical population member with high fitness encodes a weight vector such as the following:

$$(\underbrace{\ldots, 0.00415, 0.00000, 0.00025, 0.00065, 0.00000, 0.00022, 0.00000, 0.00000}_{noise},$$
$$\underbrace{1.00000, 0.79215, 0.79295, 0.24354, 0.80772, 0.34838, 0.16689, 0.83221}_{information})$$

As mentioned earlier, one known global optimum has zero valued noise elements and one valued information elements. The weight vector shown here is comparable: the noise elements are close to zero, while the information elements are relatively much larger.

3.6 Chapter conclusions

This chapter covers the application of Evolutionary Computation (EC) for the optimization of the weight vectors introduced in Section 2.5 and is particularly concerned with the development of Classification EAs (CEAs). Some common components are defined, such as the evaluation component and its subcomponents: the result function, the classify function, and the fitness function. The first CEA introduced is the *static object set CEA* (Algorithm 3 on page 41). Next the concept of *time* is introduced along with its accompanying *expanding object set CEA* (Algorithm 4 on page 42). In order to limit the size of the expanding object set the *shifting window CEA* is introduced (Algorithm 5 on page 43). To prevent all information on past performance being lost as the window shifts, we next introduce the concepts of *age* and *score*. This necessitates the concept of *maturity* and results in the last step of the derivation of the *two-pool CEA* (Algorithm 6 on page 45). The CEA employed in the dissertation research is a special case of the two-pool CEA called the *simplified two-pool CEA* (Algorithm 7 on page 46).

The next section details the particular evolvement component employed in our research (Algorithm 8 on page 47), including the details of the selection, crossover, and mutation mechanisms, as well as a number of implementation details. This is followed by an experiment illustrating the simplified two-pool CEA employing the above mentioned evolve component. The experiment consists of classifying randomly generated objects which contain a lot of noise. The objects are vectors of the format (noise| information). The weights associated with the noise part of the vector should optimally be zero and the weights associated with the information part of the vector should optimally be one. The results when employing crossover demonstrate the effectiveness of the simplified two-pool CEA: within 500 generations the classification accuracy is over 85% and within 800 generations the accuracy surpasses 95%.

Chapter 4

Evolutionary classification

Synopsis

To measure the real-world performance of the two-pool Classification Evolutionary Algorithm (CEA) introduced in Chapter 3 as applied to Adaptive Information Filtering (AIF) would require running a trial for a prolonged period with real users and a live data stream. Though this would of course be desirable, it is not practicable until the AIF system has proven itself in experimental setups. Simulations must therefore be employed to gauge the potential real-world performance. In the dissertation research the user feedback and data stream are simulated as follows: documents are selected from the Reuters-21578 text categorization collection (see Section 2.3) and presented for classification to the AIF simulation system. The classification function does not have access to information concerning from which topic the document was selected. This information is, however, employed to simulate user feedback.

Section 4.1 introduces a number of selected publications concerning previous research on evolutionary computation in the information sciences. Section 4.2 begins with a conceptual overview of the AIF simulation system, and then goes on to take a closer look at its algorithmic description. A detailed investigation of its parameters is presented in Section 4.3. In Section 4.4 some experimental results are shown and discussed. This is followed in Section 4.5 by a comparison between our experimental results and those of William B. Langdon, who's research was based on ours. Finally, in Section 4.6 some conclusions are drawn as to the effectiveness of the two pool CEA for AIF and with respect to the present limitations of the AIF simulation system and the dataset employed.

4.1 Evolutionary computation in the information sciences

In [JRSW95] it is stated that Raghavan and Birchard in their 1979 publication [RB79] seem to have been the first to suggest that it would be possible to employ a Genetic Algorithm (GA) for document clustering, although none of their experiments used actual documents and queries. One of the earliest "real" works on the application of Evolutionary Computation (EC) in the information sciences was Michael Gordon's research on employing GAs for document retrieval in 1985 [Gor85], followed up in 1988 by [Gor88]. He describes therein a document retrieval system where competing document descriptions are associated with a document and altered over time by a GA according to the queries used and relevance judgements made during retrieval. Blair published related research in 1990 [Bla90]. Gordon took a similar approach in his research reported in 1991 on document clustering [Gor91]. For another early work on the application of GAs to document clustering, see Raghavan and Agarwal's 1987 publication [RA87]. Baclace employed a GA as part of a hybrid learning algorithm within an information filtering system in his 1991 work [Bac91]. The GA evolves a population of *feature agents* employing crossover as sole genetic operator. Instead of encoding the genes as bit strings, the GA acts directly upon the features.

A different approach was taken by Yang, Korfhage and Rasmussen in their 1993 publication [YKR93]. While previous research had mainly focused on employing EC for document redescription and document clustering, they developed an adaptive method using GAs to improve query formulation through direct modification of the user queries based on relevance judgements. Kraft, Buckles, Petry, and Prabhu also employed EC for query improvement, but instead of modifying queries they optimized weights associated with Boolean queries [KBPP93, KPBP93, KPB$^+$94]. Their research was continued by Kraft, Buckles, Petry, and Sadasivan [KPBS95, KPBS97]. In 1997 they, together with Prabhu, published their research in the *Handbook for Evolutionary Computation* [PBK$^+$97].

While most of the early research on employing EC in the information sciences was directed at information retrieval and document clustering — with the notable exception of Baclace's work — 1994 saw the appearance of a number of information filtering systems employing EAs. Höfferer, Knaus and Winiwarter published a paper on the development of an AIF system they called *Cognitive Information Filtering System (CIFS)* [HKW94]. It provides prioritized E-mail based on content analysis and employs EAs to adapt descriptions of E-mails in response to user feedback and observed user behavior. Another AIF system was described in Sheth's doctoral dissertation [She94]. It provides filtered Internet news by maintaining a dynamic set of information filtering interface agents. Each agent is responsible for satisfying a specific information need of the user. An agent consists of a population of profile individuals. When processing Internet news, articles are matched against the profiles, and top scoring documents are presented to the user. The scores are based on how well the documents matched a profile, and on that profile's fitness. The user can provide positive or negative feedback with respect to an article. The profile which presented the document is then modified based on the relevance feedback received from the user. At the same time, the fitness of that profile will increase or decrease based on positive or negative feedback, respectively. The populations of profile individuals are all evolved separately. The system adapts to the user's changing information needs and minimizes the amount of required user feedback for the system to perform adequately, while allowing fine-tuning at the user's convenience. Keyword feature extraction is employed to represent the articles. The performance of the system is quite good, though it shares a weakness of many IF systems: it is not adept at dealing with information streams (Internet news groups, in this case) that exhibit a low signal-to-noise ratio.

Jones, Robertson, and Willett published an introduction to GAs and their use for information retrieval in their 1994 paper [JRW94]. Together with Santimetvirul they describe their research on the application of GAs for document clustering in their 1995 paper [JRSW95]. Their GA employed mutation and one-point crossover as genetic operators, steady-state replacement, and conventional roulette-wheel selection. In their experiments they used the Cranfield (1400 documents and 225 queries on the subject of aerodynamics), Harding (2472 documents from the INSPEC database and 65 queries), and LISA (6004 documents from the Library and Information Science Abstracts database and 35 queries) test collections. For performance measure they employed van Rijsbergen's E-measure (see Equation 1.1 on page 6). These experiments revealed slightly worse results for their GA clustering method versus standard nearest-neighbour clustering. Robertson and Willett published further research on the application of GAs for information retrieval in their 1995 report [RW95]. And in 1996 they published an article which describes the development of a GA for the assignment of weights to query terms in a ranked-output document retrieval system [RW96]. Their experimental results indicate a slight overall superiority for their GA over the deterministic weighting scheme they had compared it with.

In 1995 Chen published a long article in the *Journal of the American Society for Information Science* on machine learning for information retrieval [Che95]. In it he compares three machine learning methods applied to information retrieval, namely Hopfield networks to represent neural networks, ID3/ID5R to represent symbolic learning, and GAs to represent EC. His article begins with a detailed introduction to information retrieval and machine learning, followed by a historic overview of machine learning techniques applied to information retrieval. The article concludes with some experimental comparisons.

A 1996 paper by Zhang, Kwak, and Lee [ZKL96] describes a genetic programming approach to building software agents for information filtering. Another genetic programming paper is Smith's 1997 article [Smi97] on the employment of genetic programming to construct Boolean queries for text retrieval through relevance feedback. Terms from relevant documents are used to randomly create Boolean queries. Boolean queries are thought of as genetic programming organisms and, as such, are used for breeding to produce new organisms. The aim is to develop the best Boolean query for a particular information need, given a small corpus of test documents, and then to use that query on the full collection to retrieve yet more relevant documents.

The concepts and first results of the research on employing EC for optimizing weights associated with n-grams for AIF, which has culminated in this dissertation, were also published in 1996 [Tau96c, Tau96b, Tau96a]. Since 1996 several more publications have appeared [TKSK97, TSKK97, TSK99a, TSK99b, TKSK00, TSK00].

Vrajitoru published research on a custom GA crossover operator optimized for information retrieval in an article which appeared in 1998 [Vra98]. And in an article from 1999 [Win99], Winiwarter presented PEA (Personal Email Assistant with Evolutionary Adaptation). PEA filters incoming E-mails and ranks them according to their relevance. Pathak, Gordon, and Fan published a paper in 2000 on employing GAs to adapt matching functions that are used to match document descriptions with queries for information retrieval [PGF00].

To summarize, while the idea of employing evolutionary computation in the information sciences dates back to 1979, the first published results did not appear till the second half of the eighties, and the field only started to blossom in the mid-nineties.

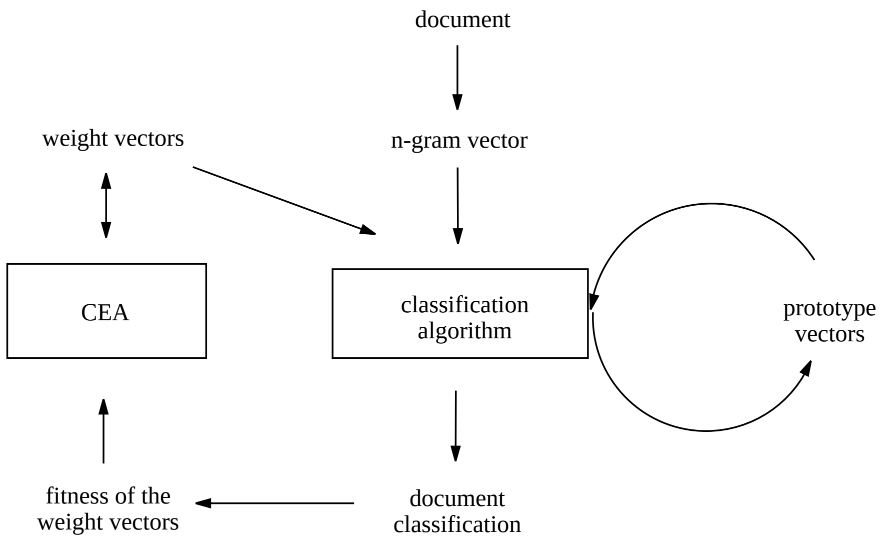
4.2 AIF simulation system

Our AIF simulation system employs the *simplified two-pool CEA* (Algorithm 7 on page 46) and the *evolve algorithm* (Algorithm 8 on page 47). The *simplified two-pool CEA* has been slightly modified: the infinite loop is replaced by a user-specified number of iterations, and the performance of the system is measured periodically using a test set. The function *CLASSIFY*(σ, P_i) (Equation 3.3 on page 40) is implemented by the *nearest cluster algorithm* (Algorithm 1 on page 18). For details with respect to representation, selection, crossover, and mutation, see Section 3.4. Below is a schematic overview of the AIF system, followed by a more detailed algorithmic overview.

4.2.1 Schematic overview

The core of the AIF simulation system is the classification cycle, depicted in Figure 4.1. The cycle starts with the selection of a document for classification. Both the document vector and the cluster prototype vectors are weighted by associating a weight with each n-gram as proposed in Section 2.5. In each cycle this weighted classification is performed for the weight vectors encoded in each member of the population. The fitness of each member is adjusted based on the classification result obtained employing the weight vector encoded in it. Note that the correctness of a classification is based on simulated user feedback. In our experiments the correct classification is provided immediately, in lieu of a user providing such feedback at a later time. At this point evolution takes place and the cycle starts over. At the end of each cycle the member with the highest fitness provides the weight vector used to make the *system classification*. In a real-world application this would be presented to the user. Optionally the cluster prototype vectors can be modified after each cycle, for example by moving the prototype vector of the class the document is assigned to closer to the document vector. In our

Figure 4.1: Schematic overview of the classification cycle of the AIF evolutionary simulation system



experiments we did not do this in order to reduce the number of parameters we had to deal with.

4.2.2 Algorithmic overview

An algorithmic overview of the AIF 2-pool CEA simulation system is shown in Algorithm 9. First the parameter file is processed and a data structure is created containing the following information for each topic employed in a particular experiment:

- topic label
- directory containing AIL files[1]
- file listing the names of the individual AIL files
- cluster averaging set size
- training set size

[1]AIL stands for Alphabet Index List and is explained in depth in Appendix B

Algorithm 9 AIF evolutionary simulation system

```
read parameter file
load dataset file indexes
initialize random number generators
build $n$-gram index
create cluster prototype vectors
for $run = 1, 2, \ldots, number\ of\ runs$ do
   initialize start population
   for $generation = 1, 2, \ldots, number\ of\ generations$ do
      training phase
      if $generation$ is an integer multiple of $plot\ step\ size$ then
         test phase
      end if
   end for
end for
write averaged test results to plot file
```

- test set size

This is followed by initializing the random number generators as specified in the parameter file: the seed can be user specified or provided by the hardware timer. Next a pass is made through all the AIL files employed in the current experiment, indexing every occurring n-gram. The result is a list of all the n-grams occurring in the experiment data set. By employing this preconstructed list as the master index in the experiment, the time required to run the experiment is considerably decreased. This is because by using this method the need for time-consuming incremental updating of the master index is avoided. In a dynamic non-simulated environment, in which the data set is not known beforehand and documents arrive in a temporal fashion, this would obviously not be possible; instead, however, the master index could be initialized with a list of frequently occurring n-grams, reducing the frequency of a non-listed n-gram occurring and thus minimizing the overhead of incremental updating of the master index without losing any of the benefits of employing the full set of possible n-grams.

In the next step the cluster prototype vector of each topic is created by averaging the set of documents belonging to each topic as specified in the parameter file. This is followed by the outer loop of the system which provides averaging of the experimental results by running the experiment multiple times and averaging the results of the different runs. The first operation for each new run is to initialize the start population. Then the inner loop is executed, each execution being one generation of the system. Each new generation begins with a training phase. For every *plot step size* generations the document vectors belonging to the test set are presented and the score is stored for later averaging. When all the generations of all the runs have been executed, the stored test results of all the runs are averaged and the averages are written away to a log file for later inspection and plotting. The averaged results can be expressed as pairs (g,p), with g indicating the amount of training measured in generations, and p indicating the averaged performance of the fittest population member applied to the test set after g training generations.

4.3 Parameters

This section describes in detail all the parameters that can be set for an experiment.

4.3.1 Experiment parameters

These parameters describe the classification problem which this experiment attempts to solve.

n

type	integer
range	≥ 1

n is the variable in n-grams. For an in-depth examination of n-grams see Chapter 2. Computational time increases exponentially with n; in Section 2.4 it was noted that the results did not improve for $n > 3$. Therefore this research focuses primarily on the use of trigrams.

alphabet size

type	integer
range	≥ 2

The alphabet size $|\mathcal{A}|$ is equal to the number of tokens in the corresponding alphabet. For example, the two alphabets employed in this research are $\mathcal{T}$

and $\mathcal{P}$ with corresponding alphabet sizes of 27 and 40 respectively (see Section 2.3 and Section 2.4 respectively). The bare minimum value for alphabet size is 2. If there was only a single token in the alphabet the power to discriminate between n-gram representations of documents would be zero. The larger the alphabet, the larger the discriminating power. However, larger alphabets also require more storage space, more memory, more computational time, and enlarge the search space, making optimization more difficult.

number of topics

type	integer
range	≥ 1

The number of topics determines from how many topics documents will be presented for classification. In general, the more topics, the more difficult the classification problem. The only exception is that when an "easy" topic is added — easy in the sense that the n-gram distribution document representations have on average a larger metric distance to those of other topics than is on average the case at that moment — the increase of the average inter-topic document distance can on some occasions more than compensate for the added complexity of increasing the number of topics. In such cases the classification results are improved by the addition of a topic.

number of runs

type	integer
range	≥ 1

A consequence of the randomness inherent in evolutionary algorithms is that each time an experiment is run the results may differ from the last time it was run, even when all the parameters are identical. It is therefore necessary to run the same experiment multiple times and average the results. The larger the number of runs, the more compensation one obtains for the variation in results. The increase in computational time is linear with the number of runs.

4.3.2 Set sizes

The data set employed is split into three parts for each experiment: for initialization, for training and for testing. This may necessitate a compromise when the sum of the desired set sizes exceeds the available data.

number to average

type	integer
range	≥ 1

During initialization *number to average* documents are averaged, resulting in prototype vectors representing the topic cluster centers. The more documents employed in averaging, the more representative the prototype vectors are of the true topic cluster center. However, the more documents employed in averaging, the fewer the documents available for training and testing.

training set size

type	integer
range	≥ 1

The aim of the training phase is to optimize the weights for the test phase. As there is no access to the test set during training, the only way the optimization of the test set can be accomplished is through generalization. Therefore the size of the training set should be optimized for generalization. The larger the training set, the more representative it is of the entire data set and thus the better it is suited for obtaining generalization. However, the larger the training set, the fewer the documents available for initialization and testing.

test set size

type	integer
range	≥ 1

In the test phase the performance of the AIF simulation is measured by presenting documents the system has not been trained on. The performance is defined as the number of correctly classified documents divided by the total number of documents presented. The larger the test set, the more representative it is of the entire data set and thus the closer the performance of the test set resembles the performance of the data set as a whole. However, the larger the test set, the fewer the documents available for initialization and training.

4.3.3 Evolution parameters

These parameters are all associated with the evolution process and most are common to all evolutionary algorithms, the exception being *maturity age* which is specific to the 2-pool EA.

population size

type	integer
range	≥ 1

The *population size* is one of the principal parameters determining the amount of genetic diversity. The larger it is, the more potential for genetic diversity is present in the population. And the more genetic diversity is present, the smaller the chance of premature convergence to a suboptimal point of the search space. However, the larger the population size, the longer it takes to evaluate all the members of the population for each generation. Also, the more memory is required to contain the population members. It is, therefore, important to find a good balance by experimenting with a variety of population sizes. The optimum size will then be the smallest size that does not (too often) result in premature convergence.

maturity age

type	integer
range	≥ 0

Maturity age was introduced in Section 3.3.4 (page 44) as the age (number of evaluations) of a member at which it is moved from the child pool to the adult pool. A higher maturity age means the member will have obtained a more reliable fitness value before it is allowed to participate in the selection and reproduction cycles. However, it also means that there will be a larger child pool and thus a smaller adult pool. To compensate for this, a larger population size is required.

number of offspring

type	integer
range	≥ 1

The *number of offspring* determines the number of new members added to the child pool. The same number of members will be removed from the adult pool. As noted in Section 3.4, after 2 $\times$ *maturity age* generations the size of the child pool is *maturity age* $\times$ *number of offspring*, under the condition that that is not larger than the *population size*. Therefore, the larger the *number of offspring*, the larger the child pool and thus the smaller the adult pool.

selective pressure rate

type	floating point
range	[0,1]

The *selective pressure rate* determines the amount of *selective pressure*. This is discussed in Section 3.4 (page 46).

gene initialization

type	floating point
range	[0,1]

The genes (weights) can be either initialized to a constant value between 0 and 1, or uniform randomly distributed.

gene mutation chance

type	floating point
range	[0,1]

The *gene mutation chance* is the chance that a particular gene will be mutated. The greater this chance, the more new genetic material is introduced during mutation, thus lowering the chance of premature convergence. However, raising the gene mutation chance also makes the retention of good genetic material more difficult, thereby causing instability in the search process.

standard deviation

type	floating point
range	> 0

The *standard deviation* determines the amount of Gaussian noise added to a gene during mutation. The mean employed in this research is 0, so for a *standard deviation* of 1 we have the standard normal (Gaussian) distribution. A bound check is performed after each gene mutation. If the new value is smaller than 0 it is set to zero and if the new value is larger than 1 it is set to 1. Because of this, appropriate values for the *standard deviation* should be small enough to prevent frequent boundary overruns.

crossover

type	Boolean
range	enabled,disabled

Crossover can either be enabled or disabled. The type of crossover employed is uniform crossover (see Section 3.1). Note that the size of the adult pool has to be larger than or equal to 2 for crossover to be possible.

number of generations

type	integer
range	≥ 0

Each generation all the members are increased one in age and are evaluated by performing a classification. If the classification is correct their score is also increased one. The *number of generations* indicates the duration of the training. The larger the *number of generations*, the more training will be performed; computation time increases linearly with the number of generations. Besides the restriction of computational time, there is another reason for limiting the *number of generations*. Namely, it is possible that too large a *number of generations* could result in *overtraining*. This is an effect due to overfitting for the training set resulting in a decrease in generalization.

4.4 Experimental results

A number of experiments were run with the aim of answering the following questions:

- How well does the two-pool CEA perform for our classification task?
- Is there a difference in performance depending on whether letter n-grams or phoneme n-grams are employed?
- What are the optimal parameter values?
- How does the two-pool CEA compare to other EAs when applied to the same problem?

To this end experiments were performed both for letter trigrams and for phoneme trigrams employing identical parameter settings in order to permit direct comparison. The only parameter that was found to make a non-trivial

Table 4.1: Experiment 1: parameter values

parameter	value
n	3
alphabet size	27
number of topics	2,3,. . . ,10
number of runs	20
initialization set size	30
training set size	30
test set size	30
population size	50
maturity age	10
number of offspring	2
selective pressure	0.5
gene initialization	uniform random in [0,1]
gene mutation chance	0.5
standard deviation	0.2
crossover	disabled
number of generations	2000

difference in the results was the training set size; therefore we will present here both letter trigram and phoneme trigram experiments for differing training set sizes.

The parameter settings for Experiment 1 are shown in Table 4.1; the parameter settings for Experiments 2, 3, and 4 are very similar, the differences being noted in the text. The results are shown in the form of system scores. The *system score* is defined as the number of correct system classifications divided by the total number of system classifications.

Experiment 1 employed letter trigrams ($n = 3$, alphabet size $= 27$) and was consecutively executed for the number of topics ranging from 2 through 10. The cluster prototype vectors were computed by averaging the first 30 document vectors of each topic, the training set was comprised of the next 30 document vectors of each topic, and, finally, the test set consisted of the following 30 documents of each topic. The population size was set to 50, the maturity age to 10, and the number of offspring to 2. Note that this means that the child pool stabilized at 20 individuals, while the adult pool stabilized at 30 individuals. The selective pressure rate was set at

Table 4.2: Experiments 1 & 2: results

	letter trigrams		phoneme trigrams	
Topics	unweighted	weighted	unweighted	weighted
Coffee, trade	0.983	0.988	0.983	0.978
+ crude	0.944	0.964	0.911	0.953
+ money-fx	0.925	0.928	0.892	0.924
+ sugar	0.927	0.941	0.893	0.934
+ money-supply	0.867	0.878	0.850	0.886
+ ship	0.833	0.849	0.810	0.839
+ interest	0.788	0.795	0.767	0.789
+ acq	0.789	0.793	0.770	0.781
+ earn	0.783	0.786	0.770	0.782

0.5, the genes (weights) were initialized uniform randomly between 0 and 1, the chance of mutating a particular gene was 0.5, and gene mutation was performed by adding Gaussian noise with zero mean and 0.1 standard deviation. If mutation resulted in a weight obtaining a value smaller than 0 or larger than 1, the weight was set to 0 or 1 respectively. No crossover was employed. Each run of the experiment consisted of 2000 training generations. The results were averaged over 20 runs and are shown in the third column (weighted letter trigrams) of Table 4.2. The unweighted results are shown in the second column (unweighted letter trigrams) of Table 4.2. Experiment 2 differs from Experiment 1 only in that it employs phoneme trigrams instead of letter trigrams. The results from Experiment 2 are shown in the fifth column (weighted phoneme trigrams) of Table 4.2. The unweighted results are shown in the fourth column (unweighted phoneme trigrams) of Table 4.2.

Experiment 3 and Experiment 4 mirror Experiment 1 and Experiment 2 respectively, except for the training set size which was increased from 30 to 50 (except in the case of *money-supply* which could only be increased to 37 due to its smaller size). The results for Experiment 3 are shown in the third column (weighted letter trigrams) of Table 4.3. The unweighted results are shown in the second column (unweighted letter trigrams) of Table 4.3. The results for Experiment 4 are shown in the fifth column (weighted phoneme trigrams) of Table 4.3. The unweighted results are shown in the fourth column (unweighted phoneme trigrams) of Table 4.3.

Table 4.3: Experiments 3 & 4: results

	letter trigrams		phoneme trigrams	
Topics	unweighted	weighted	unweighted	weighted
Coffee, trade	0.983	0.980	0.967	0.973
+ crude	0.933	0.966	0.922	0.958
+ money-fx	0.925	0.938	0.908	0.942
+ sugar	0.920	0.933	0.920	0.938
+ money-supply	0.867	0.895	0.900	0.909
+ ship	0.833	0.846	0.852	0.872
+ interest	0.833	0.843	0.850	0.853
+ acq	0.815	0.826	0.844	0.839
+ earn	0.823	0.831	0.857	0.850

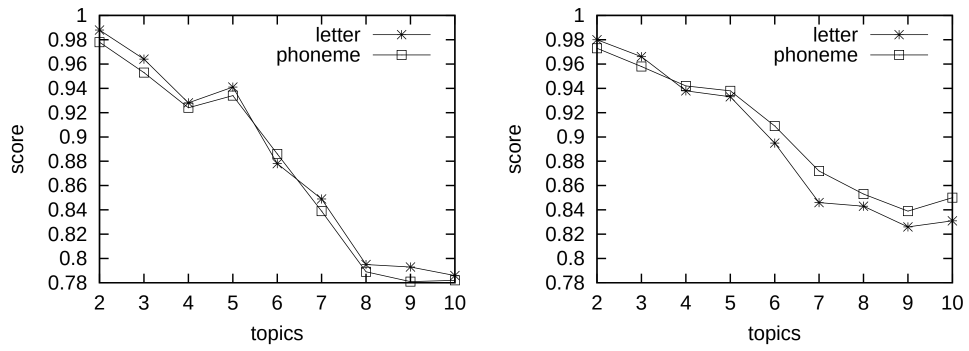
To be able to better compare the results of the four experiments, plots are provided showing letter versus phoneme results (Figure 4.2) and training set size = 30 versus 50 results (Figure 4.3).

The weighted letter trigram versus weighted phoneme trigram plots for a training set size of 30 in Figure 4.3(a) show that those results are very similar, with, overall, a slight lead for the weighted letter trigrams. However, when the training set size is increased to 50, Figure 4.3(b) reveals that for more than three topics the weighted phoneme trigrams are in the lead. In both cases, however, the differences are quite small. The training set size = 30 versus 50 plots for letter trigrams in Figure 4.4(a)) show that for more than eight topics the training set size = 50 results are superior to the training set size = 30 results. This superiority manifests itself more strongly for phoneme trigrams, as one can see in Figure 4.4(b)), with the training set size = 50 results beating the training set size = 30 results by an increasingly large margin as the number of topics increases beyond five.

4.5 Comparison

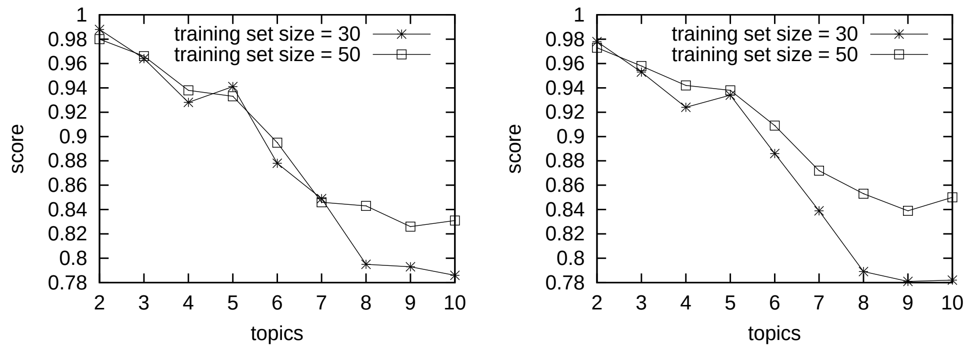
Langdon has conducted related research based on an earlier version of our C++ n-gram class library [Lan00a, Lan00b]. In his experiments Langdon employed the same data set as we employed in ours, thus making a direct comparison possible. The difference lies in the classifier and the EA: he

Figure 4.2: Weighted letter trigram versus weighted phoneme trigram results



(a) training set size = 30 (b) training set size = 50

Figure 4.3: Training set size = 30 versus training set size = 50 results



(a) weighted letter trigrams (b) weighted phoneme trigrams

Table 4.4: Langdon's KNN letter trigram results

Topics	unweighted	weighted
Coffee, trade	0.91	0.99
+ crude	0.79	0.93
+ money-fx	0.77	0.91
+ sugar	0.76	0.91
+ money-supply	0.76	0.89
+ ship	0.72	0.87
+ interest	0.69	0.82

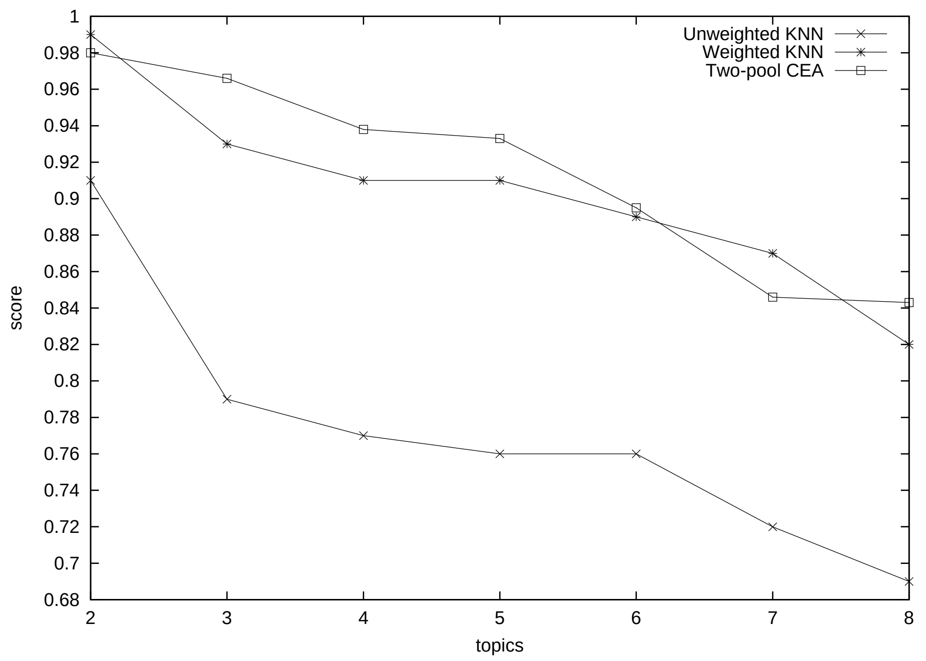
employed the *k nearest neighbours (KNN)* classifier (with k=1) and a GA implemented using a customized version of the QGAME C++ library. A summary of his results is presented in Table 4.4. A comparison of his results with ours is provided in Figure 4.4. The unweighted KNN results are taken from the second column (unweighted) of Table 4.4, the weighted KNN results are taken from the third column (weighted) of Table 4.4, and the two-pool CEA results are taken from the third column (weighted letter trigrams) of Table 4.3. Overall our two-pool EA seems to outperform Langdon's QGAME KNN algorithm.

4.6 Chapter conclusions

This chapter provides an extensive historic overview of the application of evolutionary computation in the information sciences, covering various subfields such as information retrieval, document classification, and (adaptive) information filtering. Our AIF simulation system is described at varying levels of detail, first by presenting a schematic overview of the system, then a more detailed algorithmic overview, and finally an in-depth look at the many parameters. The experimental section of the chapter poses the following questions:

1. How well does the two-pool CEA perform for our classification task?
2. Is there a difference in performance depending on whether letter trigrams or phoneme trigrams are employed?
3. What are the optimal parameter values?

Figure 4.4: Letter trigram comparison



4. How does the two-pool CEA compare to other EAs when applied to the same problem?

The answer to the first question is that for small numbers of topics the two-pool CEA performs quite well, but as the number of topics increases, the performance drops considerably. And, although for a training set size of 50 the drop in performance slows down as the number of topics increases, it is not sufficient to make the system scalable. Ways in which the scalability issue might be addressed will be explored in the future research section of the concluding chapter (Section 6.3). The answer to the second question is that for a training set size of 30, letter trigrams perform overall just a hair better than phoneme trigrams, but for a training set size of 50 phoneme trigrams perform overall better than letter trigrams. The third question is more difficult to answer, because the lack of positive examples is not conclusive. The only parameter that has positively been shown to have an influence on the results is the training set size: the results for larger numbers of topics improve markedly when the training set size is increased from 30 to 50. The answer to the final question is that the two-pool CEA seems to outperform a

QGAME based GA and KNN hybrid algorithm in the only known comparative research. In the next chapter neural networks will be investigated as an alternative classification and optimization method. A comparison with the results obtained in this chapter will also be provided.

Chapter 5

Neural classification

Synopsis

Chapter 3 introduced evolutionary computation as an efficient optimization method in the absence of an a priori solution. One of the main problems which employing evolutionary computation to optimize the weights associated with particular n-grams brings with it is the huge and complex search space, involving a very large number of calculations and numerous parameters. Another approach is to replace the clustering algorithm and evolutionary optimization technique with a neural network capable of classification as well as optimization.

This chapter will introduce neural networks as a promising document classification method for n-gram document representations, well suited to adaptive information filtering. Section 5.1 explains what neural networks are, how they work, and presents a short overview of the different types of neural networks. Section 5.2 focuses on the application of neural networks in the information sciences. Section 5.3 describes the employment of the simple perceptron neural network architecture in our research, followed by Section 5.4 which presents the accompanying experimental results. Section 5.5 describes the employment of a multi-layer feed-forward architecture using the backpropagation learning algorithm and presents some experimental results. The perceptron and backpropagation results are compared with the evolutionary computation results of Chapter 4 in Section 5.6. Section 5.7 contains the chapter conclusions.

5.1 Introduction to neural networks

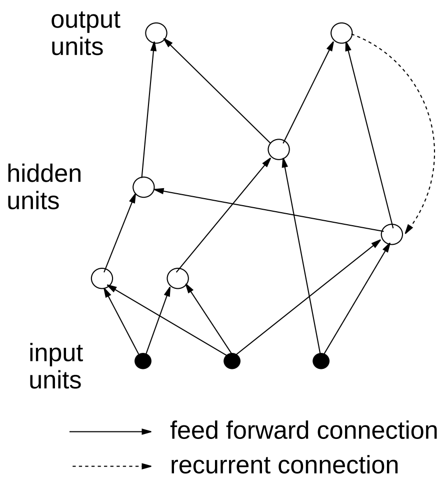
Artificial Neural Networks, further referred to as neural networks, are parallel-distributed computational systems inspired by biological neural networks. They are also known as parallel-distributed networks or connectionist networks. For an excellent in-depth introduction to neural networks, see [Bis95]. More comprehensive mathematical introductions to neural networks can be found in [HKP91, HN91], while more applied introductions can be found in [FS92, Fu94]. This section will provide a basic introduction, with emphasis on the type of neural networks employed in the dissertation research.

Neural networks are systems composed of units, connections, and weights associated with the connections. Units correspond to neurons in biological neural networks, connections correspond to axons/synapses and weights correspond to synaptic strength. The topology of a neural network is defined by its units and their interconnections. When the units can be separated into non-trivial groups having solely inter-group connections, that is to say, no intra-group connections, the neural network topology is called layered. Two neural network topologies, one layered and one not, are shown in Figure 5.1. Intuitively, a connection is called recurrent when it “loops back” to itself or an “earlier” unit. For a precise definition see [HKP91]. Some examples of recurrent connections are shown in Figure 5.1. In the absence of recurrent connections the network is called a feed-forward network, otherwise it is called a recurrent network.

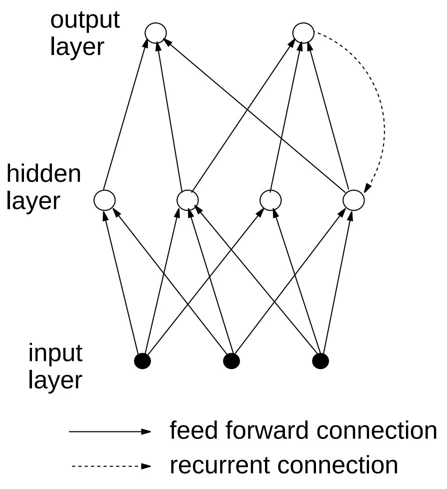
Units can be classified into three types: input units, hidden units, and output units. Data is presented to the neural network by setting the activation levels of the input units, which can be expressed in either discrete or continuous values. Input units do not perform computations. The results are expressed as the activation levels of the output units. If there are other units, they are called hidden units, “hidden” because they have no direct connection to the system input nor the system output. Layered neural networks contain one input layer, one output layer and zero or more hidden layers. When counting the number of layers in a layered neural network it is customary not to count the input layer because input units do not perform computations. Thus, for example, the neural network depicted in Figure 5.2(b) is referred to as a two-layer neural network.

The activation levels of all the other units are computed by applying an activation function (also called a transfer function, gain function, or squashing function) to their net input. The net input for units in the first layer is

Figure 5.1: Neural network models



(a) non-layered



(b) layered

computed as follows:

$$net_j = \sum W_{ji} X_i - \theta_j \tag{5.1}$$

where net_j is the net input of the j^{th} unit, W_{ji} is the weight associated with the connection from input unit X_i to the j^{th} unit, and θ_j is the unit threshold of the j^{th} unit. The type of activation function employed is problem dependent. For instance, if a Boolean activation level (e.g., on or off) is desired, the hard-limiting function F_h (also called the unit step function or Heaviside function) may be employed:

$$F_h(net_j) = \begin{cases} 1 & \text{if } net_j > 0 \\ 0 & \text{else} \end{cases} \tag{5.2}$$

If a continuous activation level in the range [0,1] is desired, the sigmoid function F_β may, for instance, be employed:

$$F_\beta(net_j) = \frac{1}{1 + e^{-2\beta net_j}} \tag{5.3}$$

The closer the value of β is to zero, the more "smoothly" the function F_β goes from 0 to 1 as net_j goes from $-\infty$ to ∞. For a more detailed discussion of the parameter β see, for example, [HKP91].

The dynamic behaviour of a neural network is controlled by its learning rule. A learning rule specifies how to adapt weights in order to optimize performance. In many types of neural networks learning only occurs during training. When a certain performance level has been achieved, the weights in such neural networks are fixed and the networks are ready for use. It is important to note that while training can be very time consuming, once the weights are fixed and learning has ceased neural networks tend to be very fast in operation. In some neural networks (i.e., Kohonen networks) the rate of adaptation — also called the learning rate — is decreased after each adaptation. In addition to the benefit of faster operation when no learning has to be performed, decreasing the learning rate ensures a certain amount of stability because new knowledge then has increasingly less ability to supplant knowledge previously encoded in the weights of the neural network. The consequence is, however, that over time the neural network loses more and more of its plasticity. This tension between stability and plasticity is called the *stability-plasticity dilemma* [CG87b] and can be posed as follows [CG88]: "How does the system know how to switch between its stable and its plastic modes to achieve stability without rigidity and plasticity without chaos?"[1] Stephen Grossberg's Adaptive Resonance Theory (ART) overcomes this dilemma [Gro76a, Gro76b]. The term "resonance" refers here to the so called resonant state of the network in which a category prototype vector matches the current input vector closely enough for it to be selected and modified to resemble the input vector. If the input vector does not match any of the category prototype vectors within a certain tolerance, as specified by the *vigilance parameter*, then a new category is created by storing a category prototype vector similar to the input vector. Consequently, no category is ever modified unless the input vector matches the category prototype vector within a certain tolerance. This means that an ART network has both plasticity and stability: new categories can be formed when the environment does not match any of the stored patterns, but the environment cannot change stored patterns unless they are sufficiently similar. The original ART-1 neural network is capable of classifying binary input vectors. The ART-2 neural network extended this capability to real-valued input vectors [CG87a]. A large family of ART neural networks has been developed based on the original ART-1 and ART-2 neural networks, further extending the ART capabilities to include hierarchical clustering, fuzzy clustering, agglom-

[1] *When* to switch would seem the more pertinent question.

erative clustering, and more. For an overview of the many ART models, an extensive ART bibliography, and anything else ART related, see The Adaptive Resonance Theory (ART) clearinghouse[2]. For an easy introduction to ART see [HT95] which describes ART-1 in detail.

Learning rules can be divided into two distinct classes, supervised and unsupervised. We can distinguish two types of supervised learning, namely *learning with a teacher* and *learning with a critic.* In *learning with a teacher* the solution (output) of the neural network is compared with the known solution, and the neural network receives instructive feedback about any errors it has made. In *learning with a critic*, also called *reinforcement learning*, the only feedback the neural network receives is whether the solution was correct or not. Examples of supervised learning are the Hebb rule, the perceptron learning rule, the delta rule, the backpropagation algorithm and the Boltzmann rule [HKP91]. The Hebb rule, the earliest and weakest of the learning rules, has been very influential in inspiring many of the later rules. It can be written as follows:

$$W_{ji}(t+1) = W_{ji}(t) + X_i \cdot T_j \tag{5.4}$$

where $W_{ji}(t)$ is the weight associated with the connection from input unit X_i to output unit O_j at time t (the t^{th} iteration) and where T_j is the desired (target) output activation. The Hebb rule can, for example, be employed in single-layer feed-forward neural networks.

The delta rule is one of the learning rules employed in the dissertation research and will be discussed in Section 5.3. The backpropagation algorithm is the other learning rule employed in the dissertation research and will be discussed in Section 5.5. We can also distinguish two types of unsupervised learning, namely *unsupervised Hebbian learning* and *competitive learning.* In unsupervised learning there is neither “teacher” nor “critic” and no notion of correct or incorrect solutions; the network must discover for itself categories or features in the input data. In *unsupervised Hebbian learning* a modified Hebb rule is employed and the output units do not have a winner-take-all character, while in *competitive learning* the output units compete for being the one to fire and only one, or one per group, is on at a time. Examples of *unsupervised Hebbian learning* are Oja’s rule and Sanger’s rule [HKP91]. Examples of *competitive learning* are Kohonen’s learning rule and Adaptive Resonance Theory (ART), except for the branch of the ART family of neural networks based on ARTMAP, which employ supervised learning

[2] `http://www.liacs.nl/art/`

[CGR91, CGM$^+$92]. There are also some hybrid learning schemes which combine supervised and unsupervised learning in the same neural network. An example of such a hybrid learning scheme is counterpropagation [HN91].

5.2 Neural networks in the information sciences

According to J.C. Scholtes [Sch93b] Michael Mozer is credited for the first application of neural networks in information retrieval [Moz84]. However, as pointed out by Richard Belew in [Bel89], Mozer's model lacked the ability to learn. One of the earliest works on the application of "learning" neural networks in the field of information retrieval was Belew's dissertation in 1986 [Bel86]. In it he describes an approach he calls "Adaptive Information Retrieval" (AIR). His AIR system employs relevance feedback for the purpose of representation adaptation, leading to incremental improvement in the retrieval process. His dissertation was followed by a succession of papers on AIR, including a paper presented at the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in 1989 [Bel89]. Belew was Daniel Rose's dissertation advisor. Rose's 1991 dissertation [Ros91] extended the AIR system to a hybrid connectionist and symbolic artificial intelligence system called SCALIR (Symbolic and Connectionist Approach to Legal Information Retrieval) which employs analogical reasoning to find relevant documents for legal research (see also [RB91]).

While the AIR and SCALIR systems employ neural networks with architectures and learning rules specifically designed for their application, many other approaches have concentrated on the application of existing neural networks. One neural network that has received considerable attention in the domain of the information sciences is the Kohonen SOM [Koh01], a member of the class of self-organizing feature maps, also called topology preserving maps, topographic maps, or self-organizing maps. The abbreviation for the latter, SOM, is the most widely used. Unlike most other neural networks, with SOMs the geometrical arrangement of the competitive output units is of importance. The location of the winning output unit conveys information, with nearby outputs corresponding to nearby input patterns. Currently, by far the most popular learning rule for SOMs is the Kohonen rule [Koh01]. Because of this, Kohonen SOMs are often referred to as simply SOMs and

SOMs are sometimes referred to as Kohonen networks. In the context of the information sciences, a SOM organizes documents onto a two-dimensional grid so that related documents appear close to each other. In other words, a SOM provides a topographic map of the document space.

One of the first applications of SOMs for information retrieval was reported in 1990 [GR90]. This was followed in 1991 by [LSM91] and [Sch91c]. One of the first applications of SOMs for information filtering was also reported in 1991 [Sch91b]. J.C. Scholtes' dissertation in 1993 [Sch93b] reviews the state of the art in connectionist information retrieval and filtering as of 1993. It presents his extensive research into the employment of the Kohonen SOM for Natural Language Processing, Information Retrieval, and Information Filtering. Jakub Zavrel's masters thesis in 1995 [Zav95] discusses two variants of the Kohonen SOM for information retrieval which overcome some of the Kohonen SOM's limitations. The two variants are Blackmore and Miikkulainen's incremental growing grid [BM93] and Fritzke's growing cell structures [Fri94].

These early publications were followed by a flood of SOM applications in the information sciences, such as [Mer95, Roz95, Mer97, Nik97, Mer98, Mer99]. Surprisingly, it was not until 1996 that Kohonen started applying SOMs to document classification with his WEBSOM[3]. WEBSOM is a method of organizing miscellaneous text documents onto meaningful maps for exploration and search. For a comprehensive overview of the WEBSOM, see [KKL$^+$00].

Many other neural network architectures have been employed in the information sciences, in particular backpropagation and counterpropagation networks. A comparison of these two neural networks for automatic text categorization was made in [RS97]. That paper presents the results obtained from a series of experiments in automatic text categorization of MEDLINE articles. MEDLINE is the USA National Library of Medicine's (NLM) premier bibliographic database, covering the fields of medicine, nursing, dentistry, veterinary medicine, the health care system, and the preclinical sciences. The main goal of this research was to build neural networks and to train them in assigning MeSH phrases based on term frequency of single words from title and abstract. MeSH stands for Medical Subject Headings, NLM's controlled subject vocabulary used for indexing and cataloging. The experiments compared the performance of a counterpropagation network ver-

[3]http://websom.hut.fi/websom/

sus a backpropagation neural network. Results obtained by using a set of 2,344 MEDLINE documents are presented and discussed. Some other papers dealing with the application of miscellaneous types of neural networks in the information sciences include: [GBW93, SHP95, WPW95, LL99, RS99, WPA99, WAP99, YL99a, YL99b, Wer00, YL00, Tan01].

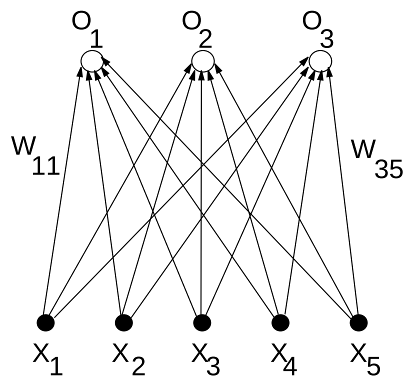
While most research on neural networks employed for document classification, retrieval and filtering employ term based representations, some have employed n-gram based representations. J.C. Scholtes experimented with n-grams and Kohonen networks [Sch93b] and also cited research done in 1990 by David Mitzman and Rita Giovannini employing n-grams and backpropagation networks [MG91].

5.3 Neural network simulation system

As it is our aim to optimize the weights associated with the full set of all occurring n-grams, a neural network with as many input units as there are occurring n-grams in the Reuters dataset is required. For letter trigrams that number is 8,013 (see Table 2.6 on page 30) and for phoneme trigrams that number is 10,303 (see Table 2.7 on page 30). While Mitzman and Giovannini employed a backpropagation network [MG91], they only used bigrams. And although Scholtes experimented with higher values of n employing Kohonen networks, in his experiments the number of n-grams was limited to several hundreds. In order to be able to handle the large number of inputs required in our research we employ a one-layer perceptron, often referred to as a simple perceptron. An example of a simple perceptron with five input units and three output units is shown in Figure 5.2. The activation level of an input unit is determined by the instance presented to the neural network, in this case an n-gram distribution vector. The net input of an output unit is computed according to Equation 5.1. Output units fire to indicate a category, otherwise remaining inactive. Therefore, the activation of an output unit is calculated according to the hard-limiting function F_h as given in Equation 5.2. As the training and test sets consist of pairs of n-gram distribution vectors and single categories $(\vec{v}, c)$, the neural network classification of $\vec{v}$ can be defined to be correct if and only if the output unit associated with category c is active and all other output units are inactive. The weights are trained by adjusting them as follows:

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji} \tag{5.5}$$

Figure 5.2: Simple perceptron



where $W_{ji}(t)$ is the weight associated with the connection from input unit X_i to output unit O_j at time t (the t^{th} iteration) and ΔW_{ji} is the weight adjustment. The adjustment can be made in different ways. In our research the delta rule was employed:

$$\Delta W_{ji} = \eta(T_j - O_j)X_i \tag{5.6}$$

where η is the learning rate with $0 < \eta < 1$, T_j is the desired (target) output activation and O_j is the actual output activation.

An algorithmic overview of the neural simulation system is shown in Algorithm 10. First the random number generators are initialized and the parameter file processed. Then a data structure is created containing the following information for each topic employed in a particular experiment:

- topic label (e.g. *coffee*)
- directory containing AIL files[4]
- file listing the names of the individual AIL files
- training set size
- test set size

[4]AIL stands for Alphabet Index List and is discussed in depth in Appendix B

Next a pass is made through all the AIL files employed in the current experiment, indexing every n-gram occurrence. The result is a list of all the n-grams occurring in the experiment data set. The need for time consuming incremental updating of the master index is avoided by employing this preconstructed list as the master index in the experiment, thus considerably reducing the time required to run the experiment. In a dynamic non-simulated environment in which the data set is not known beforehand and documents arrive in a temporal fashion, this would obviously not be possible; instead, however, the master index could be initialized with a list of frequently occurring n-grams, reducing the frequency of a non-listed n-gram occurrence and thus minimizing the overhead for incremental updating of the master index without losing any of the benefits of employing the full set of possible n-grams.

This is followed by the outer loop of the system which averages the experimental results by running the experiment multiple times and averaging the results of the individual runs. The connection weights are initialized and an initial test is carried out for each run. Then the inner loop is executed, each execution being one cycle of the system. A cycle consists of presenting a pattern (document vector) to the neural network and comparing the activity pattern (the output units that fire) with the class the presented document vector belongs to. If the only output unit which is fired is the one associated with that class, the classification is correct. Otherwise, the neural network weights are adjusted to better represent the correct activity pattern. Every *plot step size* cycles, the document vectors belonging to the test set are presented and the score is stored for later averaging. The exception is the case in which the delta rule has not been applied since the last time the test set was presented. This is the case if no misclassifications of training set documents have occurred since that time. In this case the test set result will be identical to the previous test set result and can therefore be stored without further computation. When all the cycles of all the runs have been executed, the stored results of all the runs are averaged and the averages written away to a log file for later inspection and plotting.

5.4 Experimental results

A number of experiments were run in order to answer the following questions:

- How well does a simple perceptron perform for our classification task?

Algorithm 10 AIF Neural simulation system

```
initialize random number generators
read parameter file
load dataset file indexes
build $n$-gram index
**for** $run = 1, 2, \ldots, number\ of\ runs$ **do**
  initialize connection weights
  test phase
  **for** $cycle = 1, 2, \ldots, number\ of\ cycles$ **do**
    training phase
    **if** $cycle$ is an integer multiple of $plot\ step\ size$ **then**
      **if** delta rule applied **then**
        test phase
      **end if**
      record result of last executed test phase
    **end if**
  **end for**
**end for**
write averaged test results to plot file
```

Table 5.1: Experiment 1: parameter values

parameter	value
n	3
alphabet size	27
number of runs	5
number of topics	2,. . .,10
training set size	80
test set size	30
weight initialization	uniform random in [-0.1,0.1]
learning rate η	0.3
number of training cycles	10,000

- Is there a difference in performance depending on whether letter n-grams or phoneme n-grams are employed?
- What values of n provide the best results?
- How does the learning rate η affect the performance?
- How do the results for the simple perceptron compare to the results obtained in Chapter 4 with the 2-pool CEA?

To this end experiments were conducted using varying parameter sets. The parameter set for Experiment 1 is shown in Table 5.1; the parameter sets for Experiments 2, 3 and 4 are very similar, the differences being noted in the text.

Experiment 1 employed letter trigrams ($n = 3$, alphabet size $= 27$) and was consecutively executed for the number of topics ranging from 2 through 10. To facilitate comparisons between the perceptron results reported in this chapter and the evolutionary results reported in the previous chapter, identical test sets were employed. The training set size was chosen to be equal to the sum of the initialization set size and training set size employed in the second evolutionary experiment (Section 4.4 on page 65). Thus, the training set consists of the first 80 documents of each topic, while the test set consists of the following 30 documents of each topic. Each run of the experiment consisted of 10,000 training cycles. Initially the weights were uniform randomly distributed over the interval [-0.1,0.1] and the learning rate η was set to 0.3. The results were averaged over five runs and are shown

Table 5.2: Simple perceptron results experiments 1-4

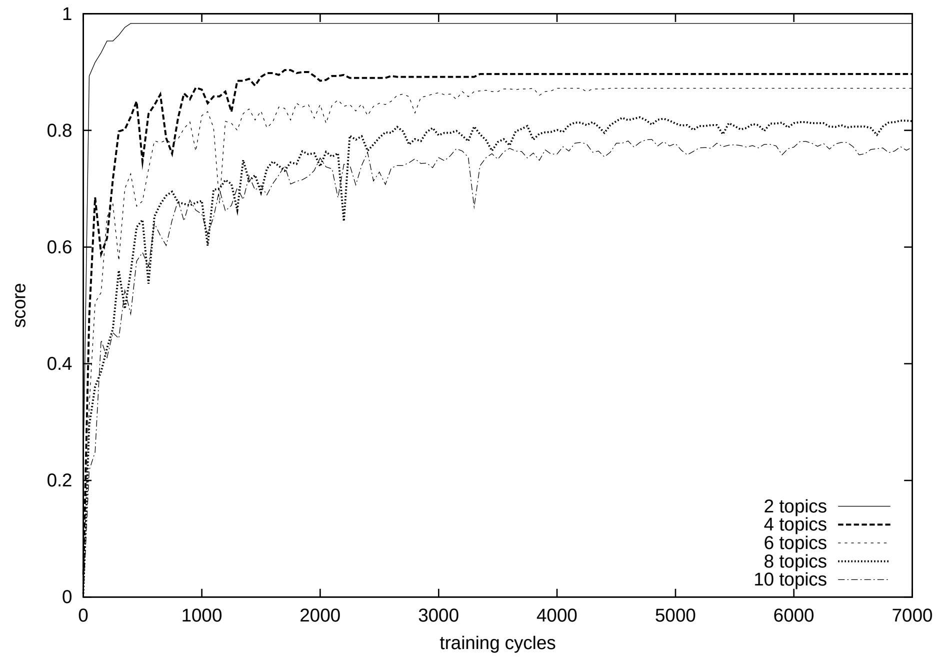
	trigrams		bigrams	
Topics	letter	phoneme	letter	phoneme
Coffee, trade	0.983	0.987	0.947	0.973
+ crude	0.911	0.920	0.876	0.876
+ money-fx	0.897	0.908	0.845	0.842
+ sugar	0.893	0.888	0.831	0.839
+ money-supply	0.872	0.889	0.793	0.818
+ ship	0.820	0.856	0.749	0.788
+ interest	0.816	0.827	0.698	0.719
+ acq	0.790	0.791	0.652	0.711
+ earn	0.786	0.796	0.599	0.690

in the second column (letter trigrams) of Table 5.2. To illustrate the results a plot is provided in Figure 5.3 for even values of the number of topics parameter. Adding more results to the plot would make it cluttered and therefore less legible. The results reveal a decreasing score as the number of topics increases. Also, the number of training cycles required for obtaining peak performance increases along with the number of topics. One can glean from the plot that for more than six topics the results might be further improved by increasing the number of training cycles. Note that the problem of overlearning does not seem to be present; the performance levels off at around its peak instead of dropping off as would happen if overlearning had occurred.

Experiment 2 differs from Experiment 1 only in that it employs phoneme trigrams instead of letter trigrams. The results for Experiment 2 are shown in the third column (phoneme trigrams) of Table 5.2 and a plot is provided in Figure 5.4. While for all but one of the values of the number of topics parameter the results are slightly better than those of Experiment 1, the plots are very similar and further improvement of the results by increasing the number of training cycles for more than six topics is again possible.

Experiment 3 is identical to Experiment 1, except that letter bigrams are employed instead of letter trigrams. The results for Experiment 3 are shown in the fourth column (letter bigrams) of Table 5.2 and a plot is provided in Figure 5.5. While letter trigrams clearly outperform letter bigrams, the possibility of improving the results through further training seems to be

Figure 5.3: Experiment 1: letter trigrams

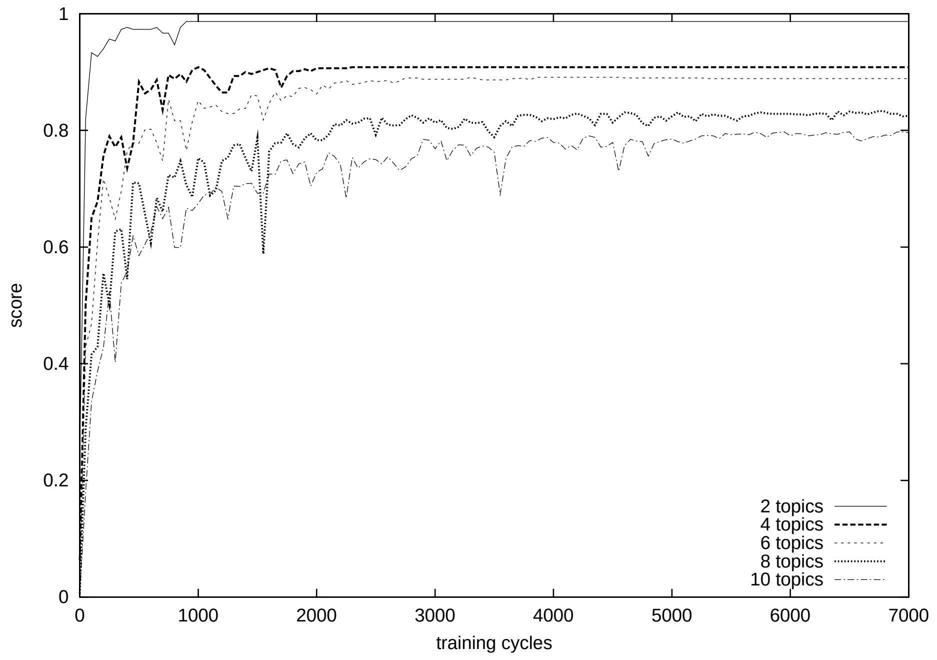


present for values of the number of topics parameter larger than four versus larger than six for letter trigrams. This is surprising because the letter bigram search space is far smaller than the letter trigram search space, leading to the expectation that there would be less opportunity for further learning.

Experiment 4 in its turn mirrors Experiment 2, except that phoneme bigrams are employed instead of phoneme trigrams. The results for Experiment 4 are shown in the fifth column (phoneme bigrams) of Table 5.2 and a plot is provided in Figure 5.6. While phoneme trigrams clearly outperform phoneme bigrams, the possibility of improving the results through further training seems to be present only for eight or more topics. This is again unexpected, considering that the *letter bigrams experiment* (Experiment 3) showed the possibility of improved results through further training for six or more topics, while the phoneme bigram search space is larger than the letter bigram search space.

To be able to better compare the results of the four experiments, plots are provided showing letter versus phoneme results (Figure 5.7) and bigram versus trigram results (Figure 5.8).

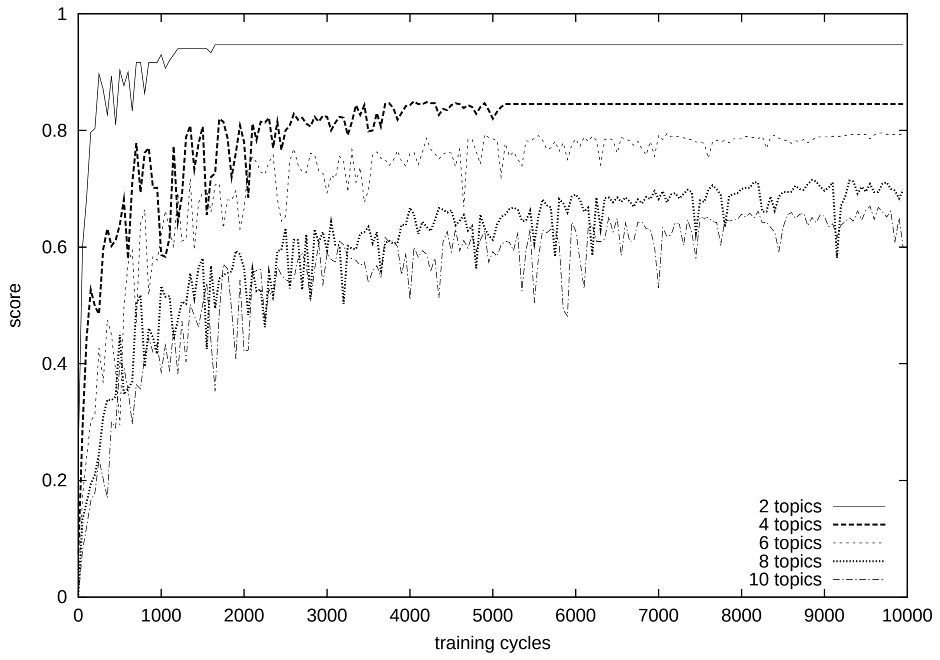
Figure 5.4: Experiment 2: phoneme trigrams



The letter trigram versus phoneme trigram plot in Figure 5.8(a) shows again how similar those results are, with a very slight lead for the phoneme trigrams. However, there also seems to be the suggestion of a decreasing performance penalty for an increasing number of topics. It is interesting to compare this to the letter bigram versus phoneme bigram plot in Figure 5.8(b). While the plots look the same at first, as the number of topics increases they show a marked degree of variation. The phoneme bigram results show the same suggestion of a decreasing performance penalty for an increasing number of topics, but the letter bigram results reveal a linear drop in performance as the number of topics increases. The superiority of trigrams over bigrams is especially clear in Figure 5.8. As the number of topics increases, the gap between letter trigrams and letter bigrams grows quickly (Figure 5.9(a)), while the gap between phoneme trigrams and letter trigrams grows at a decreasing rate (Figure 5.9(b)).

The preceding experiments demonstrate conclusively the advantage of employing trigrams over bigrams. In Experiment 5 the same parameters are employed as in Experiment 1, except that the number of topics is fixed at

Figure 5.5: Experiment 3: letter bigrams



10 and n ranges from 1 through 5. This 10-topic letter n-gram experiment provides a good comparison of how performance is affected by the value of n. The results are plotted in Figure 5.9. The plot shows that unigrams perform very badly, trigrams outperform bigrams as earlier noted, and that quadgrams outperform trigrams by a small margin. However, increasing n to 5 shows no further improvement.

The effect of the learning rate is measured in Experiment 6. The parameters are the same as those employed in Experiment 1, except that the number of topics is fixed at 10 as in Experiment 5 and the learning rate η ranges from 0.1 to 1.0. This 10-topic letter trigram experiment will permit a useful comparison to be made of how performance is affected by the value of η. The results are plotted in Figure 5.10. The plot shows that, surprisingly, the value of η does not have a significant effect on the performance.

Figure 5.6: Experiment 4: phoneme bigrams

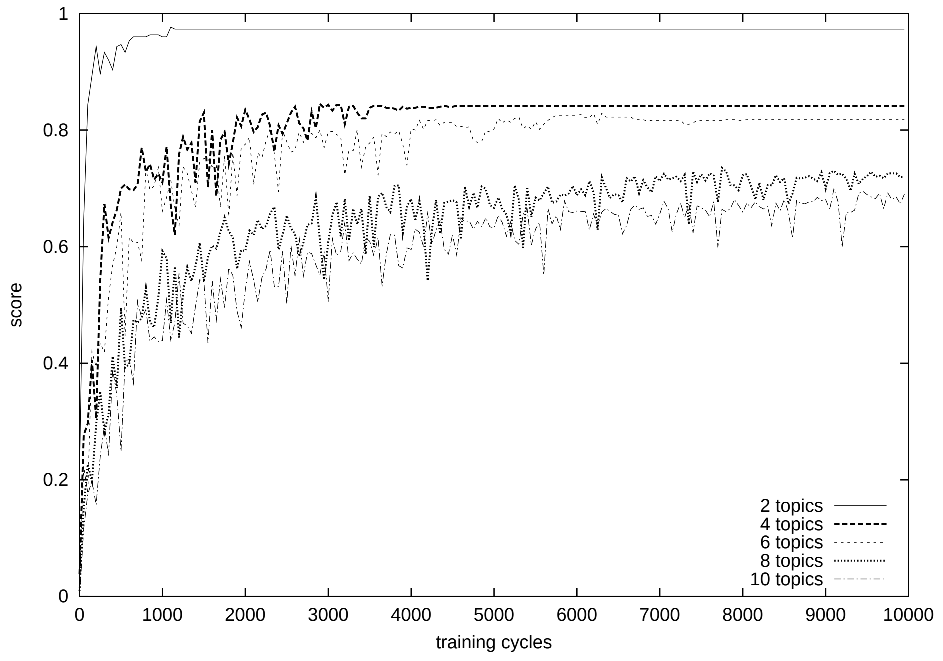
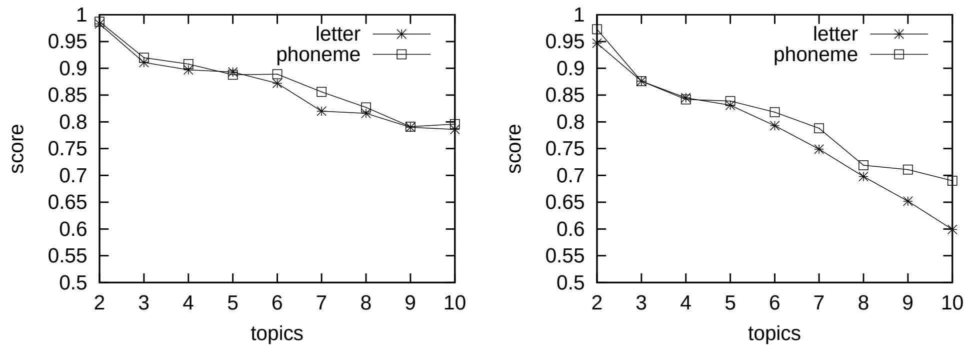


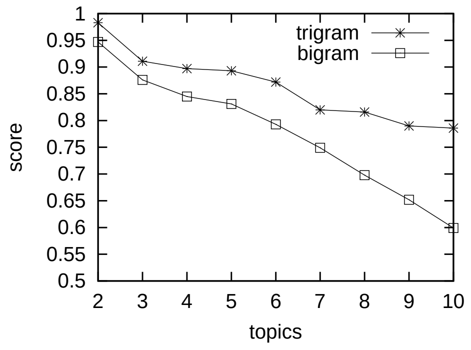
Figure 5.7: Letter versus phoneme results



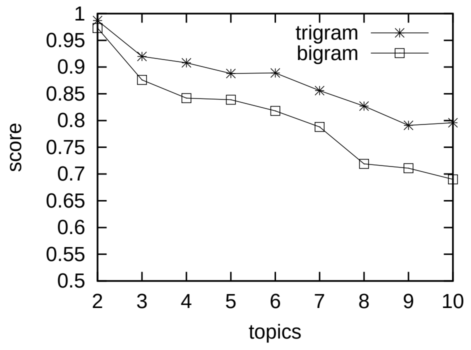
(a) trigrams

(b) bigrams

Figure 5.8: Bigram versus trigram results



(a) letter n-grams



(b) phoneme n-grams

Figure 5.9: Experiment 5: n parameter (10-topic, letter n-grams)

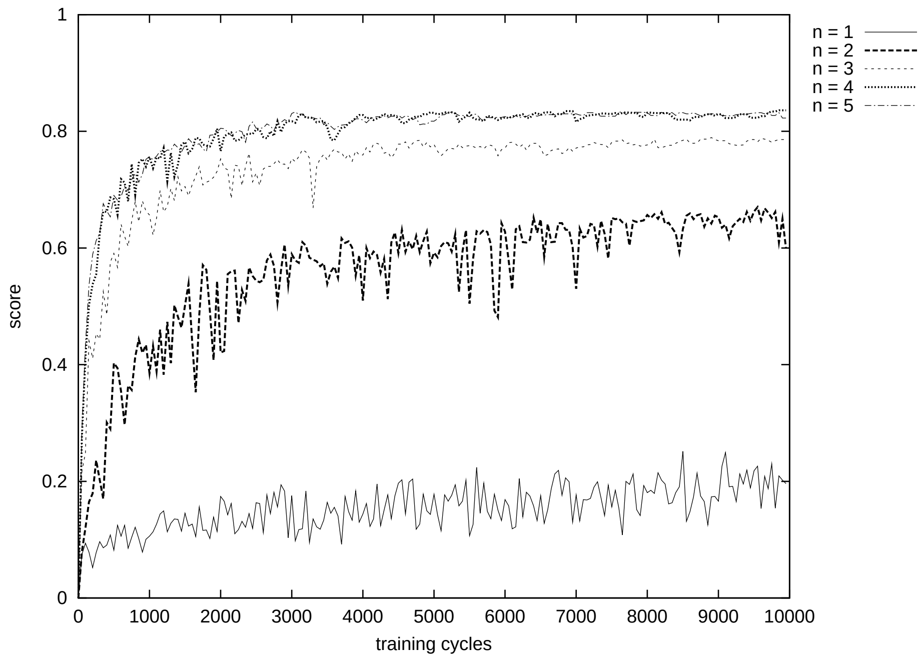
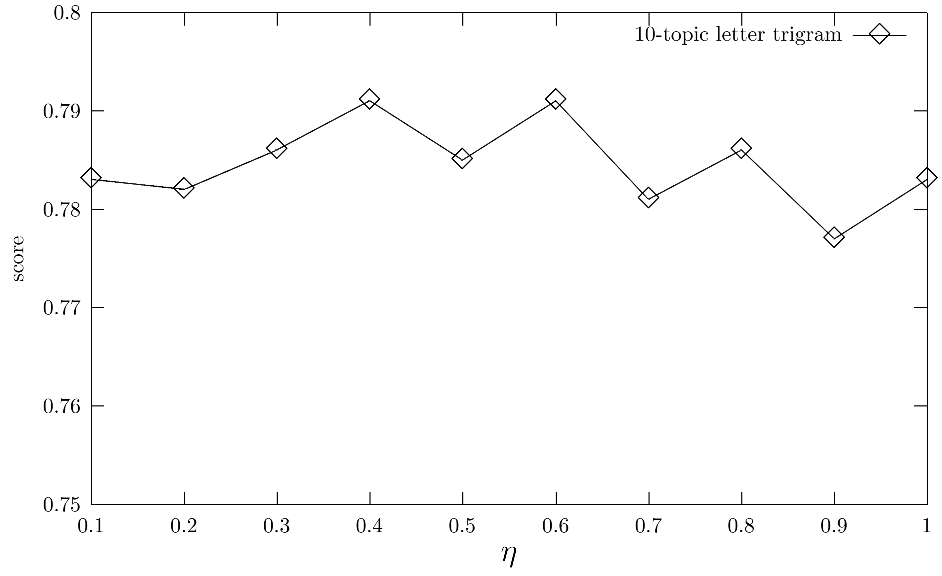


Figure 5.10: Experiment 6: learning rate η



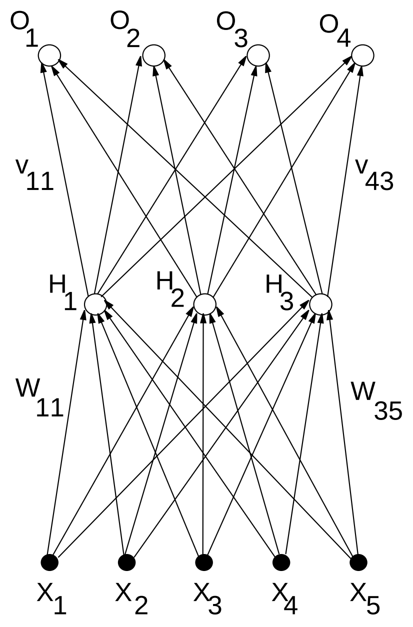
5.5 Backpropagation

This section reports on the employment of multi-layer feed-forward neural networks in the AIF simulator. Specifically, we will investigate the potential of a two-layer feed-forward network using the backpropagation algorithm as its learning rule. The first part of the section will provide a detailed overview of the backpropagation algorithm. The second part will describe its use in the AIF simulator and report some results. For more information on multi-layer feed-forward neural networks in general, and the backpropagation algorithm in particular, see, for instance, [HN91, HKP91, FS92, Fu94].

5.5.1 Overview

An example of a two-layer feed-forward network with five inputs, three hidden units, and four output units is shown in Figure 5.11. Initially, the weights w_{ji} are uniform randomly distributed over the interval $[-\gamma/\sqrt{num\_of\_inputs}, \gamma/\sqrt{num\_of\_inputs}]$ and the weights v_{ji} are initialized uniform randomly in the interval $[-\Gamma/\sqrt{num\_of\_hidden\_units}, \Gamma/\sqrt{num\_of\_hidden\_units}]$ [BK92]. The value of γ is dependent on the distribution of input activations and the value of Γ is dependent on the activation function.

Figure 5.11: Two-layer feed-forward network



One iteration of the backpropagation algorithm as applied to a two-layer feed-forward network can be formulated as follows:

Present instance The activation level of an input unit is determined by the instance presented to the neural network, in our case an n-gram distribution vector.

Calculate activation The net input for units in the hidden layer is computed in the same manner as the net input for units in the output layer of a simple perceptron (Equation 5.1 on page 75). The net input for units in the output layer is computed using the same equation, with the exception that the X_i's represent the outputs of the hidden units instead of the input units. The activation level of the hidden units and the output units is determined by the sigmoid function F_β (Equation 5.3 on page 75). Note that in our backpropagation experiments β was set at 0.5.

Perform backpropagation Begin at the output units, working backward to the hidden layer. The weights are adjusted in the same manner as with the simple perceptron (Equation 5.5 on page 80). The weight change of the output units is determined as follows:

$$\Delta W_{ji} = \eta \delta_j H_i \tag{5.7}$$

and for the hidden units as follows:

$$\Delta W_{ji} = \eta \delta_j X_i \tag{5.8}$$

where η is the learning rate with $0 < \eta < 1$ and δ_j is the error gradient at unit j. The error gradient of the output units is calculated as follows:

$$\delta_j = O_j(1 - O_j)(T_j - O_j) \tag{5.9}$$

where T_j is the desired (target) output activation and O_j is the actual output activation at output unit j. The error gradient of the hidden units is calculated as follows:

$$\delta_j = H_j(1 - H_j) \sum_k \delta_k v_{kj} \tag{5.10}$$

where δ_k is the error gradient at output unit k which is connected to the hidden unit j with v_{kj} being the connection weight.

The number of iterations of the backpropagation algorithm can be fixed, or dependent on, for example, an error criterion.

5.5.2 Results

The parameter set for the backpropagation experiments is shown in Table 5.3. For comparison purposes the parameters were chosen as follows. The experiments employ letter and phoneme trigrams respectively, with the number of topics ranging from 2 to 10. The training set size and the test set size are respectively 80 and 30, which is identical to the values of those parameters in the simple perceptron experiments. The other parameters were chosen after evaluating some trial runs. The number of runs was set at 10, the number of hidden units was set at 10, the learning rate η was set at 0.1, γ was set to 30, and Γ was set to 1 since the activation function employed was the sigmoid function. Finally, the number of training cycles was set at 10,000.

The results of the backpropagation experiments are shown in Table 5.4. To illustrate the backpropagation results plots are provided in Figure 5.12

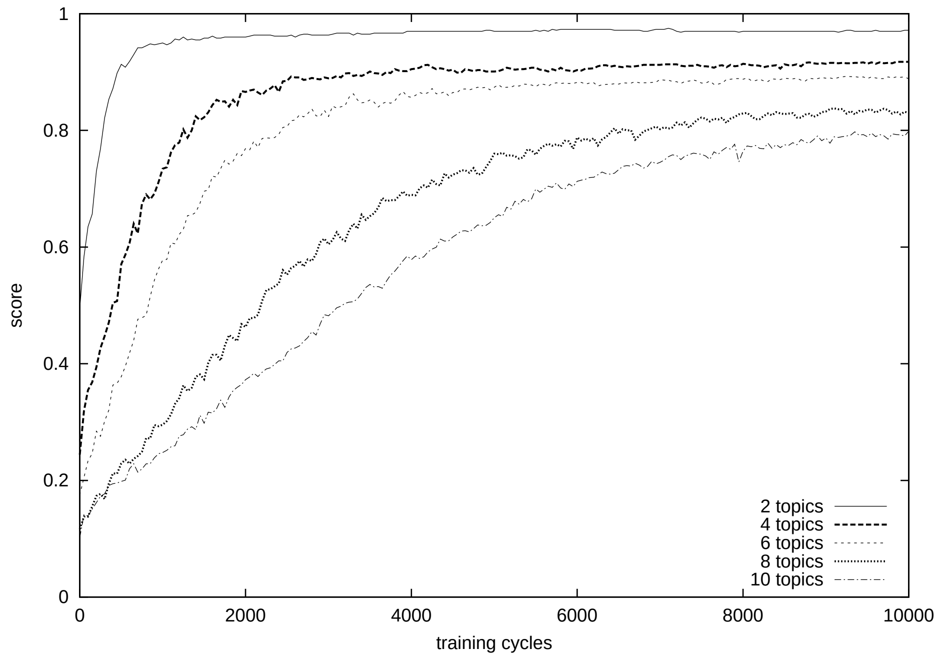
Table 5.3: Backprop parameter values

parameter	value
n	3
alphabet size	27,40
number of runs	10
number of topics	2,...,10
training set size	80
test set size	30
number of hidden units	10
learning rate η	0.1
γ	30
Γ	1
number of training cycles	10,000

Table 5.4: Backprop results

Topics	letter	phoneme
Coffee, trade	0.972	0.960
+ crude	0.920	0.940
+ money-fx	0.918	0.928
+ sugar	0.911	0.932
+ money-supply	0.888	0.911
+ ship	0.847	0.871
+ interest	0.830	0.864
+ acq	0.797	0.827
+ earn	0.799	0.829

Figure 5.12: Backprop experiment: letter trigrams

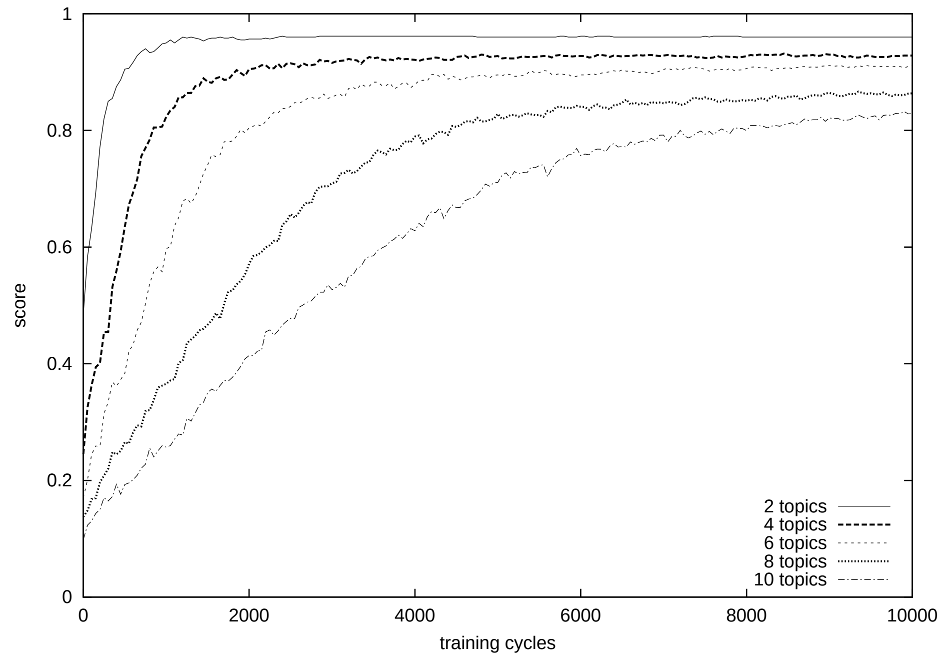


and Figure 5.13 for even values of the number of topics parameter. Adding more results to the plot would clutter it, decreasing legibility. The results show a decreasing score as the number of topics increases. Also, the number of training cycles required for obtaining peak performance increases along with the number of topics. One can glean from the letter trigram plot that the results might be further improved by increasing the number of training cycles, though probably only marginally. This is true also for the phoneme trigram plot, when the value of the number of topics parameter is greater than four. Note that the problem of overlearning does not seem to be present; the performance levels off at around its peak instead of dropping off as would happen if overlearning had occurred.

5.6 Comparison

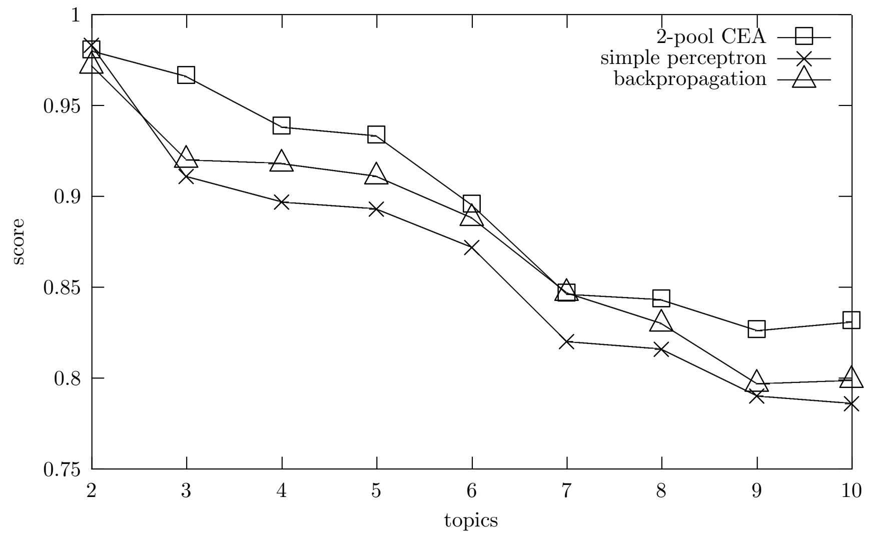
In Figure 5.15(a) and Figure 5.15(b) results obtained running the two-pool EA, the simple perceptron, and the backpropagation experiments are compared for letter and phoneme trigrams respectively. The perceptron results

Figure 5.13: Backprop experiment: phoneme trigrams

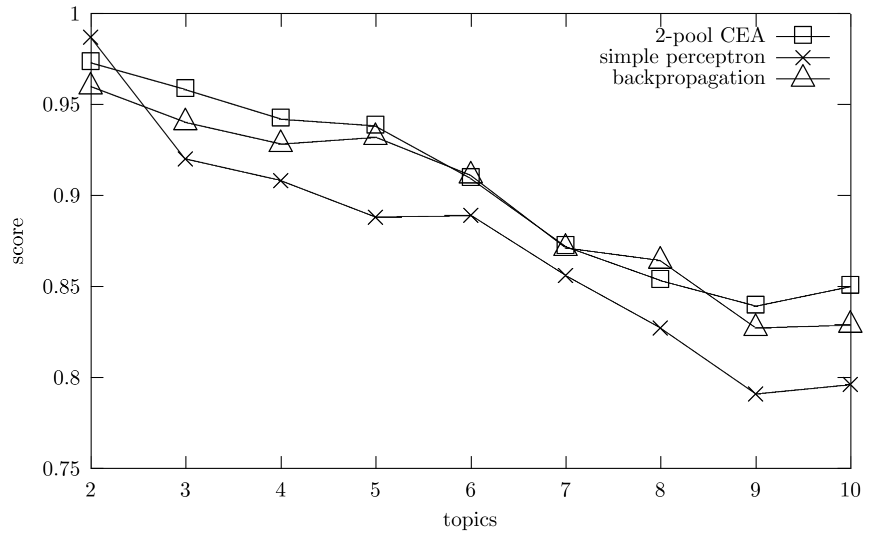


are taken from Table 5.2 (page 85), the backpropagation results are taken from Table 5.4 (page 94), and the 2-pool CEA results are taken from Table 4.3 (page 68). Both for letter trigrams and for phoneme trigrams, the 2-pool CEA delivers the best overall performance and the perceptron the worst. For the number of topics equal to two, the results for all three methods are almost identical for letter trigrams, and pretty close for phoneme trigrams. For letter trigrams the difference between the 2-pool CEA and the perceptron are considerable between three and five topics, from six to eight topics the margin narrows, and after that the gap starts to grow again. The backprop results remain in between; they are generally closer to the perceptron results except between six and eight topics when they are closer to the CEA results. The same pattern emerges for phoneme trigrams, except that the backprop results are always closer to the CEA results, sometimes even surpassing them. We know from the perceptron experiments as well as from the backpropagation experiments that further training has the potential to improve the results, especially for higher values of the number of topics parameter. Therefore, it is probable that a narrowing of the gap between

Figure 5.14: Comparison of 2-pool EA, simple perceptron, and backpropagation



(a) letter trigrams



(b) phoneme trigrams

the CEA results and the neural network results could be obtained for higher values of the number of topics parameter by increasing the training time.

5.7 Chapter conclusions

This chapter provides an introduction to neural networks. Various topologies, activation functions, and learning rules are discussed. The general introduction to neural networks is followed by an overview of the application of neural networks in the information sciences. Some notable early systems such as AIR and SCALIR are mentioned, followed by a more extensive look at self-organizing maps (SOMs) and a number of other neural network models. In Section 5.3 the implementation of the AIF system employing a neural network, in particular a simple perceptron, is described in detail and an algorithmic overview of the system is shown in Algorithm 10. Section 5.4 poses the following questions:

1. How well does a simple perceptron perform for our classification task?

2. Is there a difference in performance depending on whether letter n-grams or phoneme n-grams are employed?

3. What values of n provide the best results?

4. How does the learning rate η affect the performance?

5. How do the results using the simple perceptron compare to the results obtained in Chapter 4 with the 2-pool CEA?

The answer to the first question is that for small numbers of topics a simple perceptron performs quite well, but as the number of topics increases the performance drops considerably. And although the drop in performance slows down as the number of topics continues to increase, it is not sufficient to make the system scalable. Ways in which the scalability issue might be addressed will be explored in the future research section of the conclusions chapter (Section 6.3). The answer to the second question is that phoneme trigrams perform just a hair better than letter trigrams, but phoneme bigrams outperform letter bigrams by an increasing margin as the number of topics increases. The third question is more difficult to answer, because of the need to take into account the exponential increase in computational resources required as n increases. Unigrams perform terribly, bigrams far better, but not sufficiently better for real-world use, trigrams perform quite well, and quadgrams slightly better. Considering the computational cost advantage of trigrams over quadgrams it is doubtful, however, whether the performance advantage of quadgrams over trigrams is sufficient to be of interest. Higher values of n are definitely not worthwhile as they provide little or no performance improvement. Note that this might not be the case for more advanced neural networks. Surprisingly, the value of the learning parameter η does not seem to have a noticeable effect on the learning process. The answer to the final question is that while the difference between the performance of the 2-pool CEA and the perceptron is not very large, and further training of the perceptron has the potential for decreasing the performance difference for a larger number of topics, overall, the 2-pool CEA outperforms the perceptron. Section 5.5 introduces a multi-layer feed-forward network using the backpropagation algorithm and presents some experimental results. The comparison between the perceptron, backpropagation, and 2-pool CEA results in Section 5.6 suggests that the backpropagation network outperforms the simple perceptron, and is very close in performance to the 2-pool CEA, sometimes even outperforming it. Further tuning of the backpropagation parameters, together with increased training, therefore has the potential to improve its performance beyond that of the 2-pool CEA.

Chapter 6

Conclusion

Synopsis

This chapter consists of three sections. A summary of the most important results is presented in Section 6.1. The research questions posed in Section 1.2 are answered in Section 6.2. Suggestions are made for future research in Section 6.3.

6.1 Summary of results

Representing textual as well as phonetic documents with their n-gram frequency distributions is shown in Chapter 2 to be an effective statistical method for document classification. That method does not, however, have sufficient discriminatory power for real-world use in an adaptive information filtering system; weighted n-gram frequency distributions are a solution, but require a mechanism to incrementally optimize the weights associated with the n-grams. Such a mechanism will also facilitate adaptation to any changes in the information stream and in the user's interests. Chapter 3 introduces a possible mechanism based on evolutionary computation, namely the two-pool classification evolutionary algorithm (CEA). Chapter 4 demonstrates how the two-pool CEA can be employed in an adaptive information filtering system. Table 6.1 and Table 6.2 give an overview of the most important results found in this dissertation for letter trigrams and phoneme trigrams respectively. Letter trigram results for the two-pool CEA are shown in the second column of Table 6.1, phoneme trigram results in the second column

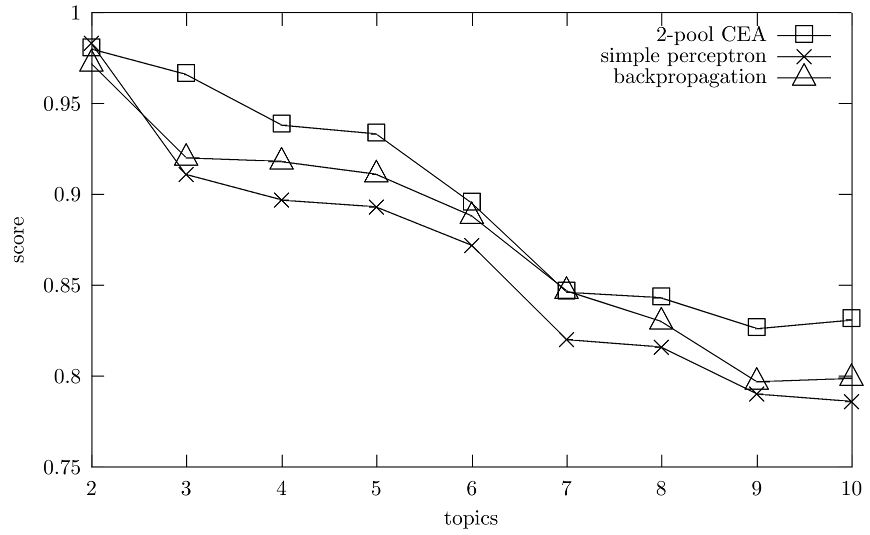
Table 6.1: Letter trigram results

Topics	2-pool CEA	simple perceptron	back-prop	unweighted KNN	weighted KNN
Coffee, trade	0.980	0.983	0.972	0.91	0.99
+ crude	0.966	0.911	0.920	0.79	0.93
+ money-fx	0.938	0.897	0.918	0.77	0.91
+ sugar	0.933	0.893	0.911	0.76	0.91
+ money-supply	0.895	0.872	0.888	0.76	0.89
+ ship	0.846	0.820	0.847	0.72	0.87
+ interest	0.843	0.816	0.830	0.69	0.82
+ acq	0.826	0.790	0.797	-	-
+ earn	0.831	0.786	0.799	-	-

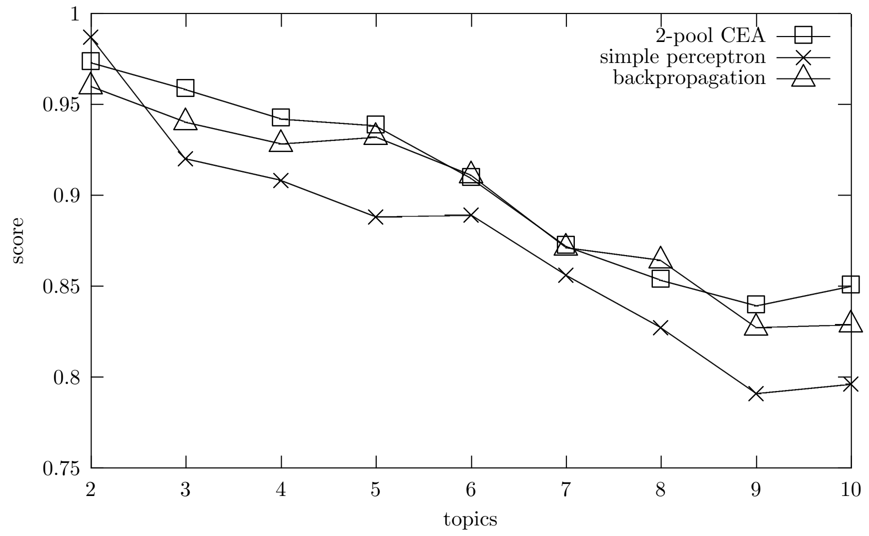
of Table 6.2. The results of related research by Langdon employing the k *nearest neighbours (KNN)* classifier and a genetic algorithm based on the QGAME library is shown in the fifth and sixth columns of Table 6.1. A graph comparing the two-pool CEA, unweighted KNN, and weighted KNN results is displayed in Figure 6.2. Chapter 5 introduces two mechanisms for incremental optimization based on neural computation, namely simple perceptrons and feed-forward networks employing the backpropagation learning algorithm. Letter trigram results for each are shown in the third and fourth columns of Table 6.1 respectively, phoneme trigram results in the third and fourth columns of Table 6.2 respectively. Graphs comparing the two-pool CEA, simple perceptron, and backpropagation network for letter trigrams and phoneme trigrams are displayed in Figure 6.2(a) and in Figure 6.2(b) respectively.

Overall, the two-pool CEA seems to outperform the other methods by a narrow margin. The feed-forward network with backpropagation learning algorithm ties with the weighted KNN approach, closely followed by the simple perceptron. The unweighted KNN approach renders the worst result by far, demonstrating the superiority of the weighted methods.

Figure 6.1: Comparison of 2-pool CEA, simple perceptron, and backpropagation



(a) letter trigrams

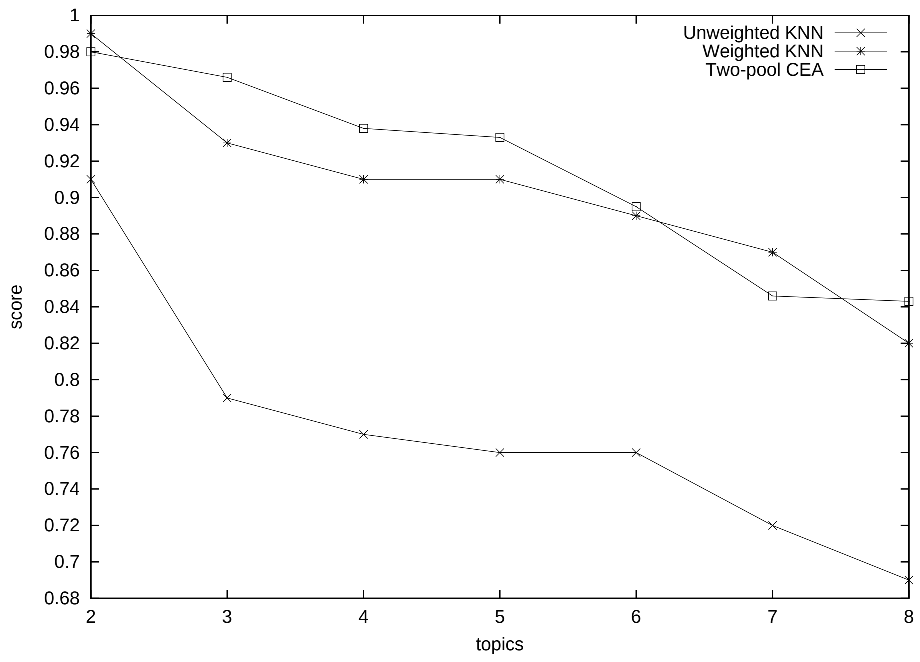


(b) phoneme trigrams

Table 6.2: Phoneme trigram results

Topics	2-pool CEA	simple perceptron	backprop
Coffee, trade	0.973	0.987	0.960
+ crude	0.958	0.920	0.940
+ money-fx	0.942	0.908	0.928
+ sugar	0.938	0.888	0.932
+ money-supply	0.909	0.889	0.911
+ ship	0.872	0.856	0.871
+ interest	0.853	0.827	0.864
+ acq	0.839	0.791	0.827
+ earn	0.850	0.796	0.829

Figure 6.2: Letter trigram comparison



6.2 Research questions

Each of the research questions posed in Section 1.2 are answered in this section. Our first question was:

> Do n-grams have the potential to provide document representations more suited to adaptive information filtering than traditional methods of representation?

Employing n-grams for adaptive information filtering has several advantages over traditional methods of representation which are typically based on terms. First, n-grams provide a robust representation in the face of spelling variations/errors. They are also language/topic independent. Both of these advantages are due to the fact that employing n-grams relies on statistics whereas employing terms requires linguistic knowledge. A more extensive examination of the advantages associated with employing n-grams can be found in Section 2.3 and in Section 2.4. The method of representing documents by computing their *n-gram frequency distribution*, as introduced in Section 2.2, was successfully employed in the dissertation research as reported in Chapter 4 and Chapter 5. Discussion of the other research questions in this chapter will further expand on this point. The answer to our first question is thus affirmative: the dissertation research has demonstrated the potential of n-grams to provide document representations more suited to adaptive information than traditional methods of representation.

Our second question was:

> Do machine learning techniques have the potential to satisfy the specific requirements of adaptive information filtering systems?

Adaptive information filtering entails adapting to changes in both the user's needs and the nature of the information streams. Machine learning allows this adaptation to be performed in an intelligent manner. The specific machine learning techniques employed in the dissertation research are reviewed in the answer to the next research question. Their successful application as reported in this dissertation (the results of which are summarized in the previous section) indicates that this research question too can be answered in the affirmative.

The third question concerns the specific machine learning techniques:

> How can machine learning techniques be employed to optimize the weights associated with n-grams in n-gram document representations?

In the dissertation research the adaptation is encoded in the weights associated with the n-grams. This weighted representation is introduced in Section 2.5. Adaptation is thus achieved through continuous optimization of the weights. In Chapter 3 the two-pool CEA is proposed as an optimization method well suited to this temporal task and experimental results for this method are presented in Chapter 4. An alternative method is to employ neural computation; this is explored in Chapter 5.

Our fourth question was:

> What type of evolutionary algorithm is capable of addressing the specific requirements of adaptive information filtering?

Adaptive information filtering imposes special requirements on data processing systems. The first requirement holds true for any information filtering system: it must be able to process data in a temporal fashion (i.e., incrementally). The second requirement is specific to adaptive information filtering: the system must be able to adapt to changes in users' information needs as well as changes in the data streams being filtered. Traditional evolutionary algorithms optimize in a batch fashion as opposed to in an incremental fashion; a major part of the dissertation research involved the development of an incremental classification evolutionary algorithm: the two-pool CEA. The details of its development are discussed in Chapter 3 and experimental results produced by its implementation in a simulated adaptive information filtering system are presented in Chapter 4. The only comparable research we are aware of are Langdon's KNN QGAME experiments, which we reported in Section 4.5. Typically our results were better with a margin of 2% to 4%, but in a few cases our results were only equal to or slightly worse than Langdon's. A graph illustrating this comparison is displayed in Figure 6.2. It should be kept in mind, however, that this comparison is between our specific combination of an EA and a classification algorithm and his, preventing a direct comparison of the EAs alone.

Our fifth and final question was:

> What degree of improvement in classification results can be achieved through the use of weighted n-gram document representations?

This is a difficult question to answer for a number of reasons. The upper-bound depends on the particular data set employed, the specific classification algorithm used, and the exact values of numerous parameters. However, one can glean a rough estimate from the experimental results (summarized in the previous section). The degree of improvement reported in Chapter 4 for the AIF simulation system ranged typically from 1% to 3%, with in a few cases a negative result and in one case an improvement exceeding 4%. A comparison of the three methods explored in the dissertation research, namely the two-pool CEA, the simple perceptron, and the two-layer feed-forward network using the backpropagation learning algorithm, are shown in Figure 6.1. This comparison is discussed in detail in Section 5.6.

The research reported in this dissertation has demonstrated the potential of combining statistical methods of representation (e.g., n-grams) with machine learning techniques (e.g., evolutionary algorithms and neural networks) for enhancing adaptive information filtering systems to address the important problem of *Information Overload*. The next section will outline the most promising directions for future research.

6.3 Future research

Document similarity was estimated by employing the Minkowski ℓ_p-metric for p equal to one, which is sometimes known as the Manhattan metric. Future research should investigate other values of p and other metrics as such.

The classification algorithm we employed, the *nearest cluster algorithm* (Algorithm 1 on page 18), was purposely chosen to be of a simple nature with as few parameters as possible, thus facilitating the optimization of the other system parameters. Better results can, however, probably be attained employing a more advanced classification algorithm.

As token alphabet for textual representation we employed the Latin alphabet with space delimiter, for a total of 27 tokens. Experimenting with more elaborate alphabets might yield better results. For instance, one could distinguish between lower and upper case letters, and expand the token alphabet by placing significance on numerals and grammatical symbols.

The data set employed in the dissertation research is the Reuters-21578 text categorization collection, as described in Section 2.3. The use of additional data sets would allow less data set dependent conclusions to be drawn and would facilitate comparisons with related research. A particularly rewarding collection of data sets are those employed in the TREC Conference series[1].

While one of the advantages of employing weighted n-gram representations is that one need not use preprocessing methods (see Section 2.3), minor improvements in the results were obtained using stop word filtering. The employment of stemming and conflation routines might result in further improvements.

In Table 2.5 it was shown that the use of quadgrams sometimes leads to poorer results than when employing trigrams. This may be due to the "curse of dimensionality"; investigating this phenomenon and finding an explanation could provide important insights.

The two-pool CEA derived in Section 3.3 and employed in the dissertation research as specified in Section 3.4 is inspired by Evolutionary Strategies (ES). However, our two-pool CEA lacks some of the more advanced features of ES, such as the self-adaptation component that provides local fine-tuning of the mutation rate. Incorporating this feature into the two-pool CEA has the potential to significantly speed up the convergence rate.

The *simplified two-pool CEA* (Algorithm 7 on page 46) is employed in the dissertation research. It would be interesting to compare the results with those obtained by employing the *two-pool CEA* (Algorithm 6 on page 45) from which the *simplified two-pool CEA* is derived.

The *evolve algorithm* (Algorithm 8 on page 47) currently supports evolution with or without crossover. This could be improved by introducing a *crossover chance* parameter, ranging from 0 (no crossover) to 1 (always crossover).

The AIF simulation system employs static non-hierarchic clusters. A more advanced model would use dynamic hierarchic clusters.

One of the main problems with the current two-pool CEA based AIF simulation system is the tuning of the EA parameters. In [EHM99], *parameter control* was suggested as a promising approach for solving this problem.

The neural network architectures employed in the dissertation research, as reported in Chapter 5, were a *simple perceptron* and a *multi-layer feed-forward network employing the backpropagation learning algorithm.* More advanced neural networks would most likely yield better results. Their use would entail, however, far greater computational demands. To address this requires either the allocation of greater computational resources or a reduc-

[1]`http://trec.nist.gov/`

tion in the number of inputs (one each per n-gram). One possiblity worth exploring is the use of principal component analysis [HKP91]. If this issue can be resolved, some neural network architectures worth investigating include SOMs and ART networks [HKP91].

Both neural networks and evolutionary algorithms are inspired by nature. While in the dissertation research they are employed independently, in nature they have symbiotic roles. Our DNA provides a blueprint for our "construction", including the architecture of the neural networks that form our brain. Learning occurs thus at two levels: at the individual level via training of neural networks, and at the species level via evolving of the gene population. This suggests two ways of improving on the current design, namely:

- The use of hybrid EA/NN systems.
- Evolving NN architectures using an EA.

For the first see, for instance, [MS94]. For the second see [BBSK95, BSK01].

To conclude, the future research outlined in this section holds the promise of further advancing the ideas, methods and algorithms presented in this dissertation. Much work is still to be done in order to develop the powerful and practical tool that we envision.

Appendix A

C++ n-gram class library

This appendix details the C++ n-gram class library. The newest version of this library is always available from the *n-gram clearinghouse*[1]. A general description and user manual are presented in Section A.1; in Section A.2, a set of tables provides the library details.

A.1 Overview

The C++ n-gram class library defines the class *ngramdistribution* and some supporting functions, types and constants. This library supports the use of n-gram distributions for various applications and a range of n values. As in many applications of n-grams, especially for n equal to or larger than three, sparse distributions are employed, this library fully supports sparse n-gram distributions. This is implemented by employing separate data structures to hold the indices of the occurring n-grams. It is up to the user of the library to keep track of which indices go with which n-gram distributions.

A short user manual for the library, with examples, is presented below; for a detailed overview of the library see Section A.2. Note that for the sake of brevity the inclusion of the library header file *ngram.h* is not shown, though it is assumed.

[1]`http://www.liacs.nl/home/dtauritz/ngram/`

A.1.1 Creation and destruction

The library offers a default constructor, a copy constructor, and a destructor for the class *ngramdistribution*. In the following example, the default constructor is called first, then the copy constructor; upon exiting the main function, the destructor is invoked automatically.

```
int main(void) {
  ngramdistribution d1;
  try {ngramdistribution d2(d1);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  return 0;
}
```

Two initialization functions are offered, one for constant initialization and one for uniform random. In the following example, $d1$ is initialized uniformly employing the constant c and $d2$ is initialized uniform randomly in the interval [0,1]. The alphabet size a is set to 27 and the vector size is set to 100.

```
int main(void) {
  unsigned n=3;
  unsigned a=27;
  unsigned size=100;
  double c=0.5;
  RUniform frand(1);
  ngramdistribution d1,d2;
  try {d1.init(n,a,size,c);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  try {d2.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  return 0;
}
```

The create function constructs an *ngramdistribution* from an AIL file. In the following example a trigram distribution ($n = 3$) is created from the AIL file *file.ail*. If the create function executed correctly, *unlisted* will contain the indices of the trigrams occurring in the AIL file.

```
int main(void) {
  unsigned n=3;
  ngram_set index,unlisted;
  ngramdistribution d;
  try {unlisted=d.create("file.ail",n,index);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  return 0;
}
```

A.1.2 Updating and storage

A critical component of the *master index method* is to keep both the master index and all the associated lists updated at all times. This is accomplished via the uniform constant update function and the uniform random update function. In the following example, first a trigram distribution $d1$ is constructed from the file *file1.ail*, then a weight vector w is initialized uniform randomly, next a trigram distribution $d2$ is constructed from the file *file2.ail*. This is followed by updating $d1$ through the addition of zeros for the trigram indices added to the master index during the creation of $d2$ (using the uniform constant update function). Finally, w is updated through the addition of random values in the interval [0,1] for the trigram indices added to the master index during the creation of $d2$ (using the uniform random update function).

```
int main(void) {
  unsigned n=3;
  unsigned a;
  RUniform frand(1);
  ngram_set index,unlisted;
  ngramdistribution d1,d2,w;
  try {index=d1.create("file1.ail",n,index);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  a = d1.return_a();
  try {w.init(n,a,index.size(),frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  try {unlisted=d2.create("file2.ail",n,index);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  try {d1.index_update(index,unlisted,0);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  try {w.index_update(index,unlisted,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  index.insert(unlisted.begin(),unlisted.end());
  return 0;
}
```

Functions are provided for the storing and loading of an *ngramdistribution*. In the following example, a trigram distribution is created from the AIL file *file.ail*, stored in the file *file.dist*, and then loaded again from the last mentioned file.

```
int main(void) {
  unsigned n=3;
  unsigned a;
  ngram_set index,unlisted;
  ngramdistribution d1,d2;
  try {index=d1.create("file.ail",n,index);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  a = d1.return_a();
  try {d1.store("file.dist",index);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  try {unlisted=d2.load("file.dist",index);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    };
  return 0;
}
```

A.1.3 Metrics and other functions

A function is provided to normalize an *ngramdistribution* such that the sum of its elements is precisely one. This is accomplished by dividing each element by the sum of the elements.

```
int main(void) {
  unsigned n=3;
  unsigned a=27;
  unsigned size=100;
  double c=0.5;
  ngramdistribution d;
  try {d.init(n,a,size,c);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d.normalize();}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  return 0;
}
```

While in the next subsection covering operators, a multiply operator is provided, a special purpose function is offered which assigns the multiple of two distributions to a third distribution. This function is substantially faster than using separate multiply and assignment operators. In the dissertation research this is of critical importance because the assignment of the multiple of two distributions to a third distribution is one of the most frequently occurring operations.

```
int main(void) {
  unsigned n=3;
  unsigned a=27;
  size=100;
  RUniform frand(1);
  ngramdistribution d1,d2,d3;
  try {d1.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d2.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d3.assign_multiple(d1,d2);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  return 0;
}
```

Two metric functions are provided, namely for the Manhattan metric and for the Euclidean metric. In the following example two uniform random distributions are created after which the Manhattan distance and the Euclidean distance between the two are displayed.

```
int main(void) {
  unsigned n=3;
  unsigned a=27;
  unsigned size=100;
  double dist;
  RUniform frand(1);
  ngramdistribution d1,d2;
  try {d1.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d2.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {dist=d1.ManhattanDist(d2);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  cout << dist << endl;
  try {dist=d1.EuclideanDist(d2);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  cout << dist << endl;
  return 0;
}
```

A.1.4 Operators

Operators are provided for addition, multiplication, assignment, and the testing of equality. In the following example distributions $d1$ and $d2$ are created uniform randomly, then $d2$ is added to $d1$, next $d1$ is multiplied by $d2$. This is followed by assigning $d1$ to $d2$, and finally $d1$ and $d2$ are tested for equality.

```
int main(void) {
  unsigned n=3;
  unsigned a=27;
  unsigned size=100;
  RUniform frand(1);
  ngramdistribution d1,d2;
  try {d1.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d2.init(n,a,size,frand);}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d1+=d2;}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d1*=d2;}
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  try {d2=d1;}
```

```
    catch (ngram_exception exception) {
      cerr << return_ngram_exception_message(exception) << endl;
      exit(1);
    }
  if (d1==d2) cout << "correct" << endl;
    else cout << "incorrect" << endl;
  return 0;
}
```

A.2 Details

Table A.1 shows the public members of the class *ngramdistribution*; Table A.2 shows the protected members of the class *ngramdistribution*; Table A.3 shows the types, constants and supporting functions; and Table A.4 shows for each class function which exceptions can occur.

Table A.1: Public members of *ngramdistribution*

Member declaration	Description
ngramdistribution()	default constructor; guarantees that value is initialized as a NULL pointer
ngramdistribution(const ngramdistribution &original)	copy constructor
∼ngramdistribution()	destructor
void init(unsigned n, unsigned a, unsigned size, RUniform &frand)	initializes value uniform randomly in the range [0,1] using random number generator frand
void init(unsigned n, unsigned a, unsigned size, double uniform_value)	initializes value homogeneously to uniform_value
ngram_set create(string s, unsigned n, ngram_set index)	create *n*-gram distribution from AIL file s and return indices of any occurring *n*-grams not listed in index
void index_update(ngram_set index, ngram_set unlisted, RUniform &frand)	expand value for *n*-grams indexed in unlisted and initialize the new entries in value uniform randomly in the range [0,1] using random number generator frand; index specifies the *n*-grams already indexed
void index_update(ngram_set index, ngram_set unlisted, double uniform_value)	expand value for *n*-grams indexed in unlisted and initialize the new entries in value homogeneously to uniform_value; index specifies the *n*-grams already indexed
ngram_set load(string s, ngram_set index)	load *n*-gram distribution from *n*-gram file s and return indices of any occurring *n*-grams not listed in index
void store(string s, ngram_set index)	store *n*-gram distribution in *n*-gram file s; index lists the occurring *n*-grams
void normalize()	divide all the values in value by sum and then assign the value 1 to sum
void assign_multiple(const ngramdistribution &dist1, const ngramdistribution &dist2)	assign the multiple of *n*-gram distributions dist1 and dist2; note that this is considerably faster than using separate assignment and multiplication operators
double ManhattanDist(const ngramdistribution &dist)	return the Manhattan Distance with *n*-gram distribution dist
double EuclideanDist(const ngramdistribution &dist)	return the Euclidean Distance with *n*-gram distribution dist
ngramdistribution & operator+=(const ngramdistribution &dist)	add to self *n*-gram distribution dist
ngramdistribution & operator*=(const ngramdistribution &dist)	multiply self with *n*-gram distribution dist
ngramdistribution & operator=(const ngramdistribution &dist)	assign *n*-gram distribution dist
bool operator==(const ngramdistribution &dist)	return true if equal to *n*-gram distribution dist, otherwise false; note that this only works as long as value contains solely integer values; this is a test function and will be removed from future releases
double* return_value()	return a pointer to value
double return_sum()	return the value of sum

Table A.2: Protected members of *ngramdistribution*

Member declaration	Description
unsigned n	the value of n being used
unsigned a	the size of the alphabet
unsigned size	the number of n-grams indexed by value
double *value	values for each n-gram occurring in this distribution
double sum	the sum of the values indexed by value

Table A.3: Types, constants, and supporting functions

Definition	Description
const string _ngram_version ("0.41")	_ngram_version contains revision information for the n-gram C++ class library
typedef set<unsigned> ngram_set	ngram_set defines a data type to hold the indices of occurring n-grams
enum ngram_exception {N_MISMATCH, A_MISMATCH, SIZE_MISMATCH, FILE_OPEN_FAILURE, INCORRECT_FILE_FORMAT, MEMORY_ALLOCATION_FAILURE, N_OUT_OF_BOUNDS, VALUE_ARRAY_UNDEFINED, DIVIDE_BY_ZERO_SUM}	in table A.4 for each class function a list of the possible exception values is given
unsigned compute_num_of_ngrams (unsigned n, unsigned a)	returns the total number of n-grams for given values of n and a
void ngram_token (unsigned index_value, unsigned n,char s[])	given index_value and n return the corresponding n-gram in s under the assumption that the alphabet consists of the letters a through z and the space delimiter
unsigned return_ngram_position (char ngram[], unsigned n)	given an n-gram and n, return the position value of the n-gram (with 'a'=0, 'z'=25, and *space*=26)

Table A.4: Exceptions

Function	N_MISMATCH	A_MISMATCH	SIZE_MISMATCH	FILE_OPEN_FAILURE	INCORRECT_FILE_FORMAT	MEMORY_ALLOCATION_FAILURE	N_OUT_OF_BOUNDS	VALUE_ARRAY_UNDEFINED	DIVIDE_BY_ZERO_SUM
ngramdistribution()									
ngramdistribution(const ngramdistribution &original)						√			
∼ngramdistribution()									
void init(unsigned n, unsigned a, unsigned size, RUniform &frand)						√	√		
void init(unsigned n, unsigned a, unsigned size, double uniform_value)						√	√		
ngram_set create(string s,unsigned n,ngram_set index)				√	√	√	√		
void index_update(ngram_set index, ngram_set unlisted, RUniform &frand)						√			
void index_update(ngram_set index, ngram_set unlisted, double uniform_value)						√			
ngram_set load(string s, ngram_set index)				√		√			
void store(string s, ngram_set index)			√	√					
void normalize()								√	√
void assign_multiple(const ngramdistribution &dist1, const ngramdistribution &dist2)	√	√	√			√		√	
double ManhattanDist(const ngramdistribution &dist)	√	√	√					√	
double EuclideanDist(const ngramdistribution &dist)	√	√	√					√	
ngramdistribution & operator+=(const ngramdistribution &dist)	√	√	√					√	
ngramdistribution & operator*=(const ngramdistribution &dist)	√	√	√					√	
ngramdistribution & operator=(const ngramdistribution &dist)						√			
bool operator==(const ngramdistribution &dist)									
double* return_value()									
double return_sum()									

Appendix B

Alphabet Index Lists

An Alphabet Index List (AIL) is a vector of the form:

$$(< token\_index >, < token\_index >, \ldots, < token\_index >)$$

with $< token\_index >$ being an integer in the interval $[0, |\mathcal{A}| - 1]$ and each token index corresponding to the token in the token stream represented by the AIL. For example, if the token alphabet $\mathcal{A}$ consisted of the colors red, white, and blue, and the token stream contained first two times red, then once blue, once more red, and finally three times white, the AIL would be as follows:

$$(0, 0, 2, 0, 1, 1, 1)$$

To facilitate the storage of AILs, the following standard form of storing AILs is adopted:

$$\begin{array}{llll}
ail & & & \\
< alphabet\_size > & & & \\
< token\_index > & < token\_index > & \ldots & < token\_index > \\
< token\_index > & < token\_index > & \ldots & < token\_index > \\
\vdots & \vdots & \vdots & \vdots \\
< token\_index > & < token\_index > & \ldots & < token\_index >
\end{array}$$

The first line contains the letters "ail", for the purpose of identifying the file type. The second line contains the size of the alphabet, and the following lines contain the token indices. While the line breaks are not significant, it enhances the readability of the file to break the lines after delimiter token indices, if those are present. To illustrate the AIL file structure, the above color example is now shown stored in an AIL file:

```
ail
3
0 0 2 0 1 1 1
```

The following two sections contain the source code employed in the dissertation research for converting text files to AIL files: Section B.1 for $\mathcal{T}$ and Section B.2 for $\mathcal{P}$.

B.1 Text to Latin Alphabet Index List

This section is concerned with converting text files to AIL files for $\mathcal{T}$, the Latin alphabet specified in Section 2.3. Such "Latin" AIL files will further be referred to as LAIL files. Consider the following example text:

international coffee organization

The corresponding LAIL file is:

```
ail
27
9 14 20 5 18 14 1 20 9 15 14 1 12 0
3 15 6 6 5 5 0
15 18 7 1 14 9 26 1 20 9 15 14 0
```

The first line contains the file type identifying text "ail". The second line contains the number 27, which is equal to $|\mathcal{T}|$. And each following line contains the token indices of the corresponding word in the text (the 'i's in *international* are the 9th letter of the Latin alphabet), followed by a zero, the index of the delimiter token. The source code of the conversion program is as follows:

```
// t2lail - text to Latin alphabet index list
//
// Author      : Daniel R. Tauritz
// Created     : June 21st 2001
// Last revised: July 11th 2001
// Compiler    : GNU C++
//
// Input       : either a text file to be processed or a text file
//               containing the file names of the text files to be
//               processed (note that in the latter case the
//               filenames need to contain one single dot)
// Output      : Latin alphabet index list file(s)

#include <fstream>
#include <string>
#include <map>
#include <vector>
#include <set>

typedef vector<char> phoneme_vector_type;
map<string,phoneme_vector_type> tp_map;
set<string> unknown_set;

void loop(string);
void process(string,string);
void write_unknown_terms(void);

int main(int argc, char *argv[])
{
  cout << "t2lail : Text-to-Latin Alphabet Index List\n" << endl;
  switch (argc) {
  case 2:  loop(argv[1]);
           break;
  case 3:  process(argv[1],argv[2]);
           break;
  default: cout << "Incorrect number of arguments." << endl;
           cout << "Format: t2lail [text file] [pail file] or" << endl;
           cout << "        t2lail [list file]" << endl;
  }
  return 0;
}

void loop(string listfile_name)
{
  ifstream listfile (listfile_name.c_str());
  if (!listfile) {
    cerr << "Error: Unable to open " << listfile_name << endl;
    exit(1);
  }

  // open lail list file
  string laillistfilename = "lail.ind";
  ofstream laillistfile (laillistfilename.c_str());
  if (!laillistfile) {
    cerr << "Error: Unable to open " << laillistfilename << endl;
    exit(1);
  }

  cout << "Processing files" << flush;
  char inputfilename[255],outputfilename[255];
  while (listfile >> inputfilename) {
    cout << '.' << flush;
    strcpy(outputfilename,inputfilename);

    // Find dot in outputfilename (dot in filename required!)
    unsigned i=0;
    while (outputfilename[i]!='.') i++;

    // Replace suffix with "lail"
    outputfilename[i+1]='l';
    outputfilename[i+2]='a';
    outputfilename[i+3]='i';
    outputfilename[i+4]='l';
    outputfilename[i+5]='\0';

    process(inputfilename,outputfilename);
    laillistfile << outputfilename << '\n';
  }
  cout << endl;

  laillistfile.close();
  listfile.close();
}

void process(string tf_name, string lail_name)
{
  // open text file
  ifstream tf (tf_name.c_str());
  if (!tf) {
    cerr << "Error! Unable to open " << tf_name << endl;
    exit(1);
  }

  // open alphabet index vector file
  ofstream ailf (lail_name.c_str());
  if (!ailf) {
    cerr << "Error! Unable to open " << lail_name << endl;
    exit(1);
  }
  ailf << "ail" << endl; // label
  ailf << "27" << endl;  // alphabet size : 26 letters + delimiter

  // process text file
  char ch;
  bool previous_char_is_delimeter = true;
  while(tf.get(ch)) {
    if (!isalpha(ch)) {
      if (!previous_char_is_delimeter) {
        ailf << (int)0 << '\n';
        previous_char_is_delimeter = true;
      }
    } else {
      ailf << (int)(1+tolower(ch)-'a') << ' ';
      previous_char_is_delimeter = false;
    }
  }

  // close files
  tf.close();
  ailf.close();
}
```

B.2 Text to Phoneme Alphabet Index List

This section is concerned with converting text files to AIL files for $\mathcal{P}$, the Latin alphabet specified in Section 2.4. Such "phonetic" AIL files will further be referred to as PAIL files. Consider the following example text:

international coffee organization

The corresponding PAIL file is:

```
ail
40
17 23 31 12 23 2 30 3 23 3 21 0
20 1 14 18 0
4 28 15 3 23 3 38 13 30 3 23 0
```

The first line contains the file type identifying text "ail". The second line contains the number 40, which is equal to $|\mathcal{P}|$. And each following line contains the token indices of the corresponding word in the text, followed by a zero, the index of the delimiter token. The source code of the conversion program is as follows:

```
// t2pail - text to phoneme alphabet index list
//
// Author      : Daniel R. Tauritz
// Created     : May 31st 2001
// Last revised: July 11th 2001
// Compiler    : GNU C++
//
// Input          : either a text file to be processed or a text
//                  file containing the file names of the text files
//                  to be processed (note that in the latter case
//                  the filenames need to contain one single dot)
// Auxiliary input: cmudict 0.6d - stripped
// Output         : phoneme alphabet index list file(s)

#include <fstream>
#include <string>
#include <map>
#include <vector>
#include <set>
```

```
typedef vector<char> phoneme_vector_type;
map<string,phoneme_vector_type> tp_map;
set<string> unknown_set;

void loop(string);
void process(string,string);
void write_unknown_terms(void);

int main(int argc, char *argv[])
{
  cout << "t2pail : Text-to-Phoneme Alphabet Index List\n" << endl;

  // open stripped cmudict file
  ifstream cmudict_stripped ("cmudict.0.6d-stripped");
  if (!cmudict_stripped) {
    cerr << "Error! Unable to open cmudict.0.6d-stripped" << endl;
    exit(1);
  }

  // build term-phoneme index vector lookup table
  cout << "Building term-phoneme index vector lookup table" << endl;
  string term;
  cmudict_stripped >> term;
  for (unsigned i=1;i<=112151;i++) {
    char phoneme[256];
    phoneme_vector_type v;
    while(1) {
      cmudict_stripped >> phoneme;
      if (atoi(phoneme) == 0) {
        tp_map.insert(pair<string,phoneme_vector_type>(term,v));
        term=phoneme;
        break;
      }
      v.push_back(atoi(phoneme));
    };
  }
  cmudict_stripped.close();

  // process command line arguments
  switch (argc) {
  case 2:  loop(argv[1]);
           break;
  case 3:  process(argv[1],argv[2]);
           break;
  default: cout << "Incorrect number of arguments." << endl;
           cout << "Format: t2pail [text file] [pail file] or" << endl;
           cout << "        t2pail [list file]" << endl;
  }

  write_unknown_terms();
  return 0;
}

void loop(string listfile_name)
{
  ifstream listfile (listfile_name.c_str());
  if (!listfile) {
    cerr << "Error: Unable to open " << listfile_name << endl;
    exit(1);
  }

  // open pail list file
  string paillistfilename = "pail.ind";
  ofstream paillistfile (paillistfilename.c_str());
  if (!paillistfile) {
    cerr << "Error: Unable to open " << paillistfilename << endl;
    exit(1);
  }

  cout << "Processing files" << flush;
  char inputfilename[255],outputfilename[255];
  while (listfile >> inputfilename) {
    cout << '.' << flush;
    strcpy(outputfilename,inputfilename);

    // Find dot in outputfilename (dot in filename required!)
    unsigned i=0;
    while (outputfilename[i]!='.') i++;
    // Replace suffix with "pail"
    outputfilename[i+1]='p';
    outputfilename[i+2]='a';
    outputfilename[i+3]='i';
    outputfilename[i+4]='l';
    outputfilename[i+5]='\0';

    process(inputfilename,outputfilename);
    paillistfile << outputfilename << '\n';
  }
  cout << endl;

  paillistfile.close();
  listfile.close();
}

void process(string tf_name, string pail_name)
{
  // open text file
  ifstream tf (tf_name.c_str());
  if (!tf) {
    cerr << "Error! Unable to open " << tf_name << endl;
    exit(1);
  }

  // open alphabet index vector file
  ofstream ailf (pail_name.c_str());
  if (!ailf) {
    cerr << "Error! Unable to open " << pail_name << endl;
    exit(1);
  }
  ailf << "ail" << endl; // label
  ailf << "40" << endl;   // alphabet size : 39 phonemes + delimiter

  // process text file
  map<string,phoneme_vector_type>::iterator p;
  string term;
  while(1) {
    tf >> term;
    if (tf.eof()) break;
    p = tp_map.find(term);
    if (p != tp_map.end()) {
      for (unsigned i=0;i<p->second.size()-1;i++) {
        ailf << (int)p->second[i] << ' ';
      }
      ailf << (int)p->second[p->second.size()-1] << ' ' << 0 << '\n';
    } else unknown_set.insert(term);
  }
  tf.close();
  ailf.close();
}

void write_unknown_terms(void)
{
  ofstream unknown ("unknown.txt");
  if (!unknown) {
    cerr << "Error! Unable to open unknown.txt" << endl;
    exit(1);
  }
  set<string>::iterator q;
  q = unknown_set.begin();
  while (q != unknown_set.end())
    unknown << *q++ << endl;
  unknown.close();
}
```

Appendix C

A mathematical proof concerning normalized vectors

An operation occurring frequently in the dissertation research is the computation of prototype cluster vectors by averaging a certain number of document vectors in the form of normalized weighted frequency distribution vectors. If the resulting vectors were already normalized, a separate normalization operation would be unnecessary, thus saving computational time. The following mathematical proof shows that this is the case.

The number of document vectors to average is indicated with l and the dimension of the document vectors is indicated with k. The indexing variables i and j will range as follows: $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, l$. The unnormalized document vectors are indicated with $\overrightarrow{d^j}$. Let $\overrightarrow{d^j} = (d^j_1, d^j_2, \ldots, d^j_k)$ with $d^j_i \geq 0$ and $\forall j \; \exists i : d^j_i > 0$ (in plain English: empty documents are not allowed). The normalized document vectors are indicated with $\overrightarrow{\overrightarrow{d^j}}$. Let $\overrightarrow{\overrightarrow{d^j}} = (\overline{d^j_1}, \overline{d^j_2}, \ldots, \overline{d^j_k})$ with $\overline{d^j_i} = d^j_i / |\overrightarrow{d^j}|$. The cluster prototype vector $\vec{c}$ is defined as follows as an average of document vectors: $\vec{c} = (c_1, c_2, \ldots, c_k)$ with $c_i = 1/l \cdot \sum_j \overline{d^j_i}$. The theorem we want to prove is shown in Equation C.1:

$$\vec{c} = \vec{\vec{c}} \tag{C.1}$$

In order to do that we will first prove the lemma shown in Equation C.2:

$$|\overrightarrow{\overrightarrow{d^j}}| = 1 \tag{C.2}$$

The proof of the lemma is as follows:

$$|\overrightarrow{\overline{d^j}}| = \sum_i |\overline{d_i^j}| = \sum_i |\frac{d_i^j}{|\overrightarrow{d^j}|}| = \sum_i \frac{d_i^j}{|\overrightarrow{d^j}|} = \frac{\sum_i d_i^j}{|\overrightarrow{d^j}|} = \frac{|\overrightarrow{d^j}|}{|\overrightarrow{d^j}|} = 1$$

Note that the fact that $d_i^j \geq 0$ is essential. The following equalities hold:

$$\vec{c} = \overrightarrow{\overline{c}} \Leftrightarrow \forall i : c_i = \overline{c_i} \Leftrightarrow \forall i : c_i = \frac{c_i}{|\vec{c}|}$$

So to prove the theorem, all we have to prove is that $|\vec{c}| = 1$:

$$|\vec{c}| = \sum_i |c_i| = \sum_i \left|1/l \cdot \sum_j \overline{d_i^j}\right| = 1/l \cdot \sum_i \left|\sum_j \overline{d_i^j}\right| = 1/l \cdot \sum_i \sum_j \overline{d_i^j} = 1/l \cdot \sum_j \sum_i \overline{d_i^j}$$

$$= 1/l \cdot \sum_j \sum_i |\overline{d_i^j}| = 1/l \cdot \sum_j |\overrightarrow{\overline{d^j}}| = 1/l \cdot \sum_j 1 = 1/l \cdot l = 1 \; \square$$

Bibliography

[Ada91] Elizabeth Shaw Adams. *A Study of Trigrams and Their Feasibility as Index Terms in a Full Text Information Retrieval System.* PhD thesis, George Washington University, 1991.

[AFW83] Richard C. Angell, George E. Freund, and Peter Willett. Automatic spelling correction using a trigram similarity measure. *Information Processing & Management*, 19(4):255–261, 1983.

[AP92] Elizabeth Shaw Adams and Gregory Popovici. On using a relational database to store full text for information retrieval with a trigram based index. In *Proceedings of the First International Conference on Information and Knowledge Management (CIKM-92)*, pages 112–119, Baltimore, Maryland, U.S.A., November 1992. International Society for Computers and Their Applications (ISCA).

[Ass99] International Phonetic Association. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet.* Cambridge University Press, Cambridge, England, UK, July 1999.

[Bac91] Paul E. Baclace. Personal information intake filtering. In *Proceedings of Bellcore Workshop on High-Performance Information Filtering: Foundations, Architectures and Applications*, November 1991. `http://www.baclace.net/Resources/ifilter1.html`.

[Bäc96] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, USA, 1996.

[BBSK95] Egbert J.W. Boers, M.V. Borst, and Ida G. Sprinkhuizen-Kuyper. Evolving artificial neural networks using the "baldwin effect". In D.W. Pearson, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms, Proceedings of the International Conference*, pages 333–336, New York, NY, USA, 1995. Springer-Verlag.

[BC92] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, 35(12):29–38, December 1992. `http://www.acm.org/pubs/citations/journals/cacm/1992-35-12/p29-belkin/`.

[Bel86] Richard K. Belew. *Adaptive Information Retrieval: Machine Learning in Associative Networks.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1986.

[Bel89] Richard K. Belew. Adaptive information retrieval: Using a connectionist representation to retrieve and learn about documents. In *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–20, June 1989. `http://www.acm.org/pubs/citations/proceedings/ir/75334/p11-belew/`.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[BK92] Egbert J.W. Boers and H. Kuiper. Biological metaphors and the design of artificial neural networks. Master's thesis, Leiden University, 1992.

[Bla90] David C. Blair. *Language and Representation in Information Retrieval.* Elsevier, Amsterdam, The Netherlands, 1990.

[BM93] Justine Blackmore and Risto Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proceedings ICNN'93, International Conference on Neural Networks*, volume I, pages 450–455, Piscataway, NJ, USA, 1993. IEEE Service Center.

[BMK94] Bert R. Boyce, Charles T. Meadow, and Donald H. Kraft. *Measurement in Information Science*. Academic Press, 1994.

[BS01] Glenn B. Bell and Anil Sethi. Matching records in a national medical patient index. *Communications of the ACM*, 44(9):83–88, September 2001. `http://www.acm.org/pubs/citations/journals/cacm/2001-44-9/p83-bell/`.

[BSK01] Egbert J.W. Boers and Ida. G. Sprinkhuizen-Kuyper. Chapter 6: Combined biological metaphors. In M. Patel, V. Honavar, and K. Balakrishnan, editors, *Advances in the Evolutionary Synthesis of Intelligent Agents*, pages 153–183. MIT Press, Cambridge, MA, USA, 2001.

[BYT92] Managing infoglut. *Byte Magazine*, 17(6), June 1992.

[Cav93] William B. Cavnar. n-gram-based text filtering for trec-2. In D.K. Harman, editor, *Overview of the Second Text REtrieval Conference (TREC-2)*, volume 500-215 of *Special Publications*, pages 171–179, Gaithersburg, Maryland, U.S.A., 1993. National Institute of Standards and Technology (NIST). `http://www.nonlineardynamics.com/trenkle/papers/trec93ps.gz`.

[Cav94] William B. Cavnar. Using an n-gram-based document representation with a vector processing retrieval model. In D.K. Harman, editor, *Overview of the Third Text REtrieval Conference (TREC-3)*, volume 500-225 of *Special Publications*. National Institute of Standards and Technology (NIST), April 1994. `http://trec.nist.gov/pubs/trec3/papers/cavnar\_ngram\_94.ps`.

[CG87a] Gail A. Carpenter and Stephen Grossberg. Art 2: Selforganisation of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.

[CG87b] Gail A. Carpenter and Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing*, 37(1):54–115, January 1987.

[CG88] Gail A. Carpenter and Stephen Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *IEEE Computer*, 21(3):77–88, March 1988.

[CGM$^+$92] Gail A. Carpenter, Stephen Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, September 1992.

[CGR91] Gail A. Carpenter, Stephen Grossberg, and J.H. Reynolds. Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4:565–588, 1991.

[Che95] Hsinchun Chen. Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, 46(3):194–216, April 1995.

[CMS01] Maria Fernanda Caropreso, Stan Matwin, and Fabrizio Sebastiani. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. In Amita G. Chin, editor, *Text Databases and Document Management: Theory and Practice*, pages 78–102. Idea Group Publishing, Hershey, US, 2001. `http://faure.iei.pi.cnr.it/~fabrizio/Publications/TD01a/TD01a.pdf`.

[Coh97] Jonathan D. Cohen. Recursive hashing functions for *n*-grams. *ACM Transactions on Information Systems*, 15(3):291–320, 1997. `http://www.acm.org/pubs/articles/journals/tois/1997-15-3/p291-cohen/p29%1-cohen.pdf`.

[Com90] Fatih Mehmet Comlekoglu. *Optimizing a Text Retrieval System Utilizing N-gram Indexing*. PhD thesis, The George Washington University, May 1990. University Microfilms Order Number ADG90-20669.

[Coo94] William S. Cooper. The formalism of probability theory in ir: a foundation or an encumbrance? In *Proceedings of the seventeenth* *annual international ACM-SIGIR conference on Research and development in information retrieval*, pages 242–247, New York, NY, USA, 1994. Springer-Verlag.

[CT94] William B. Cavnar and John M. Trenkle. n-gram-based text categorization. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval (SDAIR-94)*, pages 161–175, Las Vegas, Nevada, USA, April 1994. UNLV Publications/Reprographics. `http://www.nonlineardynamics.com/trenkle/papers/sdrbcps.gz`.

[CVBW92] Vladimir Cherkassky, Nikolaos Vassilas, Gregory L. Brodt, and Harry Wechsler. Conventional and associative memory approaches to automatic spelling correction. *International Journal of Engineering Applications of Artificial Intelligence*, 5(3):223–237, 1992.

[Dam95] Marc Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(10):843–848, February 1995.

[DeH82] T. DeHeer. The application of the concept of homeosemy to natural language information retrieval. *Information Processing & Management*, 5(18):229–236, 1982.

[Dow99] J. Stephen Downie. *Evaluating a simple approach to Music Information Retrieval: conceiving melodic n-grams as text.* PhD thesis, University of Western Ontario, London, Ontario, Canada, July 1999. `http://alexia.lis.uiuc.edu/~jdownie/mir_papers/thesis_missing_some_music_figs.pdf`.

[EHM99] Ágoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2), July 1999.

[ELW95] F. Çuna Ekmekçioglu, Michael F. Lynch, and Peter Willett. Language processing techniques for the implementation of a document retrieval system for turkish text databases. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Posters: Abstracts, pages 369–370, 1995. `http://www.acm.org/pubs/articles/proceedings/ir/215206/p369-liddy/p369-%liddy.pdf`.

[FD92] Peter W. Foltz and Susan T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35(12):51–60, December 1992.

[Fog95] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, 1995.

[FOW66] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, New York, U.S.A., 1966.

[Fox01] Robert Fox. News track. *Communications of the ACM*, 44(11):9–10, November 2001.

[Fri94] Bernd Fritzke. Growing cell structures—a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[FS92] James A. Freeman and David M. Skapura. *Neural Networks: algorithms, applications, and programming techniques.* Addison-Wesley, July 1992.

[Fu94] LiMin Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill Series in Computer Science. McGraw-Hill, 1994.

[GBW93] Petra Geutner, Uli Bodenhausen, and Alex Waibel. Flexibility through incremental learning: Neural networks for text categorization. In *Proceedings of WCNN-93, World Congress on Neural Networks*, pages 24–27, 1993. `http://www.is.cs.cmu.edu/papers/speech/1993/WCNN\_93\_petra\_geutner.ps.gz`.

[GNOT92] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, December 1992. Special Issue on Information Filtering, `http://www.acm.org/pubs/citations/journals/cacm/1992-35-12/p61-goldberg%/`.

[Gor85] Michael David Gordon. A genetic algorithm for document retrieval. In *Proceedings of Workshop on Foundations of Adaptive Information Processing*, June 1985.

[Gor88] Michael David Gordon. Probabilistic and genetic algorithms in document retrieval. *Communications of the ACM*, 31(10):1208–1218, October 1988.

[Gor91] Michael David Gordon. User-based document clustering by redescribing subject descriptions with a genetic algorithm. *Journal of the American Society for Information Science*, 42(5):311–322, June 1991.

[GR90] M. Gersho and R. Reiter. Information retrieval using self-organizing and heteroassociative supervised neural network. In *Proc. INNC'90, Int. Neural Network Conference*, volume 1, pages 361–364, Dordrecht, Netherlands, 1990. Kluwer.

[Gro76a] Stephen Grossberg. Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121–134, 1976. Reprinted in Anderson and Rosenfeld, 1988.

[Gro76b] Stephen Grossberg. Adaptive pattern recognition and universal recoding: II. Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, 23:187–202, 1976.

[HD94] Stephen Huffman and Marc Damashek. Acquaintance: A novel vector-space N-gram technique for document categorization. In Donna K. Harman, editor, *Proceedings of TREC-3, 3rd Text Retrieval Conference*, pages 305–310, Gaithersburg, US, 1994. National Institute of Standards and Technology, Gaithersburg, US.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*, volume 1 of *Santa Fe Institute Studies In The Sciences of Complexity Lecture Notes*. Addison-Wesley, 1991.

[HKW94] Max Höfferer, Bernd Knaus, and Werner Winiwarter. Using genetics in information filtering. In *Proceedings of the 22nd Annual Conference of the Canadian Association for Information Science*, May 1994.

[HN91] Robert Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, 1991.

[Hol75] John H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, Ann Arbor, MI, U.S.A., 1975.

[HRF76] A.R. Hanson, E.M. Riseman, and E. Fisher. Context in word recognition. *Pattern Recognition*, 8:35–45, 1976.

[HS82] Jonathan J. Hull and Sargur N. Srihari. Experiments in text recognition with binary n-gram and viterbi algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(5):520–530, September 1982.

[HSV87] J. Henseler, J. C. Scholtes, and C. R. J. Verhoest. The design of a parallel knowledge-based optical-character recognition system. Master's thesis, Delft University, Delft, Netherlands, 1987.

[HT95] Lucien G. Heins and Daniel R. Tauritz. Adaptive resonance theory (art): An introduction. Technical Report 95-35, Leiden University, 1995. `http://www.liacs.nl/home/dtauritz/papers/art.ps.gz`.

[Huf95] Stephen Huffman. Acquaintance: Language-independent document categorization by N-grams. In Donna K. Harman and Ellen M. Voorhees, editors, *Proceedings of TREC-4, 4th Text Retrieval Conference*, pages 359–371, Gaithersburg, US, 1995. National Institute of Standards and Technology, Gaithersburg, US. `http://trec.nist.gov/pubs/trec4/papers/nsa.ps.gz`.

[JRSW95] Gareth Jones, Alexander M. Robertson, Chawchat Santimetvirul, and Peter Willett. Non-hierarchic document clustering using a genetic algorithm. *Information Research News*, 5(3):2–6, 1995.

[JRW94] Gareth Jones, Alexander M. Robertson, and Peter Willett. An introduction to genetic algorithms and to their use in information retrieval. *Online & CDROM Review*, 18(1):3–13, February 1994.

[KBPP93] Donald H. Kraft, Bill P. Buckles, Fred E. Petry, and Devaraya Prabhu. Generating fuzzy information retrieval queries via genetic programming. In *Proceedings of the 5th IFSA World Congress*, Seoul, Korea, July 1993.

[Kei01] Daniel A. Keim. Visual exploration of large data sets. *Communications of the ACM*, 44(8):38–44, August 2001.

[KKL+00] Teuvo Kohonen, Samuel Kaski, Krista Lagus, Jarkko Salojärvi, Vesa Paatero, and Antti Saarela. Organization of a massive document collection. *IEEE Transactions on Neural Networks, Special Issue on Neural Networks for Data Mining and Knowledge Discovery*, 11(3):574–585, May 2000.

[Koh01] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Information Sciences*. Springer, Berlin, Heidelberg, New York, 3 edition, 2001. `http://www.cis.hut.fi/research/som-research/book/`.

[Koz90] John R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University Computer Science Department, June 1990.

[KPB+94] Donald H. Kraft, Fred E. Petry, Bill P. Buckles, T. Sadasivan, and Devaraya Prabhu. Construction of Boolean queries for document retrieval via genetic algorithms. In *Proceedings of AISB Workshop on Evolutionary Computation*, pages 307–315, Leeds, England, April 1994.

[KPBP93] Donald H. Kraft, Fred E. Petry, Bill P. Buckles, and Devaraya Prabhu. Fuzzy information retrieval using genetic algorithms and relevance feedback. In *1993 Proceedings of the 56th Annual Meeting of the American Society for Information Science*, volume 30, Columbus, OH, USA, October 1993.

[KPBS95] Donald H. Kraft, Fred E. Petry, Bill P. Buckles, and T. Sadasivan. Applying genetic algorithms to information retrieval systems via relevance feedback. In Patrick Bosc and Janusz Kacprysk, editors, *Fuzziness in Database Management Systems*, pages 330–344. Physica-Verlag, Heidelberg, Germany, 1995.

[KPBS97] Donald H. Kraft, Fred E. Petry, Bill P. Buckles, and T. Sadasivan. Genetic programming for query optimization in information retrieval: Relevance feedback. In Elie Sanchez, Takanori Shibata, and Lotfi A. Zadeh, editors, *Genetic Algorithms and Fuzzy Logic Systems, Soft Computing Perspectives*, pages 155–173. World Scientific Publishing Co., Singapore, 1997.

[Krz95] Roman M. Krzanowski. Compression of spatial data. In *Twelfth International Symposium on Computer- Assisted Cartography*, volume 4, pages 331–340, Charlotte, North Carolina, 1995.

[Kuk92a] Karen Kukich. Spelling correction for the telecommunications network for the deaf. *Communications of the ACM*, 35(5):80–90, May 1992. `http://www.acm.org/pubs/citations/journals/cacm/1992-35-5/p80-kukich/`.

[Kuk92b] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992. `http://www.acm.org/pubs/citations/journals/surveys/1992-24-4/p377-kukic%h/`.

[LA96] Joon Ho Lee and Jeong Soo Ahn. Using n-grams for korean text retrieval. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Asian Languages, pages 216–224, 1996. `http://www.acm.org/pubs/articles/proceedings/ir/243199/p216-lee/p216-le%e.pdf`.

[Lan00a] William B. Langdon. Natural language text classification and filtering with trigrams and evolutionary nearest neighbour classifiers. Technical Report SEN-R0022, CWI, July 2000. `http://www.cwi.nl/ftp/CWIreports/SEN/SEN-R0022.ps.Z`.

[Lan00b] William B. Langdon. Natural language text classification and filtering with trigrams and evolutionary NN classifiers. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 210–217, Las Vegas, Nevada, USA, 8 July 2000. `ftp://cs.ucl.ac.uk/genetic/ga_papers/WBL.kneighbours.ps.gz`.

[LCP99] Joon Ho Lee, Hyun Yang Cho, and Hyouk Ro Park. *n*-gram-based indexing for korean text retrieval. *Information Processing and Management*, 35(4):427–441, 1999.

[LL99] Savio L. Lam and Dik L. Lee. Feature reduction for neural network based text categorization. In Arbee L. Chen and Frederick H. Lochovsky, editors, *Proceedings of DASFAA-99, 6th IEEE International Conference on Database Advanced Systems for Advanced Application*, pages 195–202, Hsinchu, TW, 1999. IEEE Computer Society Press, Los Alamitos, US. `http://dlib.computer.org/conferen/dasfaa/0084/pdf/00840195.pdf`.

[LSM91] X. Lin, D. Soergel, and G. Marchionini. A self-organizing semantic map for information retrieval. In *Proceedings 14th. Annual International ACM/SIGIR Conf. on R & D In Information Retrieval*, pages 262–269, 1991.

[Mae94] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, July 1994. `http://pattie.www.media.mit.edu/people/pattie/CACM-94/CACM-94.p1.html`.

[Mar00] Carla Marceau. Characterizing the behavior of a program using multiple-length N-grams. In *Proceedings of the 2000 new security paradigm workshop*, pages 101–110, September 2000. `ftp://ftp.oracorp.com/documents/MultiLengthStrings.pdf`.

[MC75] Robert Morris and Lorinda L. Cherry. Computer detection of typographical errors. *IEEE Transactions on Professional Communication*, PC-18(1):54–56, March 1975.

[ME62] Constance K. McElwain and Martha B. Evens. The degarbler—A program for correcting machine-read Morse code. *Information and Control*, 5(4):368–384, December 1962.

[Mer95] Dieter Merkl. Content-based document classification with highly compressed input data. In F. Fogelman-Soulié and P. Gallinari, editors, *Proceedings ICANN'95, International Conference on Artificial Neural Networks*, volume II, pages 239–244, Nanterre, France, 1995. EC2.

[Mer97] Dieter Merkl. Lessons learned in text document classification. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pages 316–321. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[Mer98] Dieter Merkl. Text classification with self-organizing maps: SOM lessons learned. *Neurocomputing*, 21(1):61–77, 1998.

[Mer99] Dieter Merkl. Document classification with self-organizing maps. In E. Oja and S. Kaski, editors, *Kohonen Maps*, pages 183–197. Elsevier, Amsterdam, The Netherlands, 1999.

[MG91] David Mitzman and Rita Giovannini. Activitynets: A neural classifier of natural language descriptions of economic activities. In *Proceedings of the International Workshop on Neural Nets for Statistics and Economic Data*, December 1991.

[Mic96] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, third edition, June 1996.

[MM97] James Mayfield and Paul McNamee. Indexing using both N-grams words. In Ellen M. Voorhees and Donna K. Harman, editors, *The Seventh Text REtrieval Conference (TREC 7)*, pages 419–424. USA Department of Commerce and National Institute of Standards and Technology, November 1997. `http://trec.nist.gov/pubs/trec7/papers/JHUAPL.pdf`.

[Moz84] Michael C. Mozer. Inductive information retrieval using parallel distributed computation. Technical Report TR8406, Institute for Cognitive Science, UCSD, La Jolla, CA, USA, May 1984.

[MS94] Michael McElligott and Humphrey Sorensen. An evolutionary connectionist approach to personal information filtering.

In *INNC 94 (Fourth Irish Neural Network Conference)*, pages 141–146. University College Dublin, September 1994. `ftp://odyssey.ucc.ie/pub/filtering/INNC94.ps`.

[MSL$^+$99] Ethan L. Miller, Dan Shen, Junli Liu, Charles Nicholas, and Ting Chen. Techniques for gigabyte-scale n-gram based information retrieval on personal computers. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1410–1416, June 1999.

[MSLN00] Ethan L. Miller, Dan Shen, Junli Liu, and Charles Nicholas. Performance and scalability of a large-scale n-gram based information retrieval system. *Journal of Digital Information (online refereed journal)*, January 2000. `http://journals.ecs.soton.ac.uk/jodi/`.

[Neu75] David L. Neuhoff. The viterbi algorithm as an aid in text recognition. *IEEE Transactions on Information Theory*, IT-21:222–226, March 1975.

[Ng00] Kenney Ng. *Subword-based Approaches for Spoken Document Retrieval*. PhD thesis, MIT, Cambridge, Massachusetts, USA, February 2000. `http://www.sls.lcs.mit.edu/kng/papers/phd_thesis.pdf`.

[Nik97] Kazuhisa Niki. Self-organizing information retrieval system on the web: SirWeb. In Nikola Kasabov, Robert Kozma, Kitty Ko, Robert O'Shea, George Coghill, and Tom Gedeon, editors, *Progress in Connectionsist-Based Information Systems. Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems*, volume 2, pages 881–884. Springer, Singapore, 1997.

[NWZ00] Corinna Ng, Ross Wilkinson, and Justin Zobel. Experiments in spoken document retrieval using phonetic n-grams. *Speech Communication, special issue on Accessing Information in Spoken Audio*, 32(1–2):61–77, September 2000.

[Pai90] Chris D. Paice. Another stemmer. *ACM-SIGIR Forum*, 24(3):56–61, 1990.

[PBK$^+$97] Fred E. Petry, Bill P. Buckles, Donald H. Kraft, Devaraya Prabhu, and Thyagarajan Sadasivan. The use of genetic programming to build queries for information retrieval. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages G2.1:1–6. Institute of Physics Publishing and Oxford University Press, Bristol, New York, USA, 1997.

[PGF00] Praveen Pathak, Michael Gordon, and Weiguo Fan. Effective information retrieval using genetic algorithms based matching functions adaptation. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, January 2000.

[PPF96] Ulrich Pfeifer, Thomas Poersch, and Norbert Fuhr. Retrieval effectiveness of proper name search methods. *Information Processing and Management*, 32(6):667–679, 1996. `http://ls6-www.informatik.uni-dortmund.de/ir/publications/1996/Pfeifer\%_etal:96.html`.

[RA87] V. V. Raghavan and B. Agarwal. Optimal determination of user-oriented clusters: An application for the reproductive plan. In *Proceedings of the Second International Conference on Genetic Algorithms and their applications*, pages 241–246, Hillsdale, NJ, USA, July 1987. Lawrence Erlbaum Associates.

[Rav67] Josef Raviv. Decision making in markov chains applied to the problem of pattern recognition. *IEEE Transactions on Information Theory*, 3(4):536–551, October 1967.

[RB79] Vijay V. Raghavan and Kim Birchard. A clustering strategy based on a formalism of the reproductive process in natural systems. In *Information Implications into the Eighties, Proceedings of the Second International Conference on Information Storage and Retrieval*, pages 10–22. ACM, 1979.

[RB91] Daniel Eric Rose and Richard K. Belew. A connectionist and symbolic hybrid for improving legal research. *International Journal of Man-Machine Studies*, 35(1):1–33, July 1991.

[RE71] Edward M. Riseman and Roger W. Elrich. Contextual word recognition using binary digrams. *IEEE Transactions on Computers*, 20(4):397–403, April 1971.

[Rec73] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, volume 15 of *Problemata*. Friedrich Frommann Verlag · Günter Holzboog, Stuttgart, Germany, 1973.

[RH74] Edward M. Riseman and Allen R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *IEEE Transactions on Computers*, 23(5):480–493, May 1974.

[Rij79] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, second edition, 1979. `http://www.dcs.gla.ac.uk/Keith/Preface.html`.

[Ros91] Daniel Eric Rose. *A symbolic and connectionist approach to legal information retrieval*. PhD thesis, University of California, San Diego, 1991.

[Roz95] J.M. Rozmus. Information retrieval by self-organizing maps. In M.E. Williams, editor, *16th National Online Meeting Proceedings—1995*, pages 349–54, Medford, NJ, USA, 1995. Smart Syst., USA, Learned Inf.

[RS97] Miguel E. Ruiz and Padmini Srinivasan. Automatic text categorization using neural networks. In Efthimis Efthimiadis, editor, *Proceedings of the 8th ASIS/SIGCR Workshop on Classification Research*, pages 59–72, Washington, US, 1997. American Society for Information Science, Washington, US. `http://www.cs.uiowa.edu/~mruiz/papers/sigcr97/sigcrfinal2.html`.

[RS99] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical neural networks for text categorization. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 281–282, Berkeley, US, 1999. ACM Press, New York, US. `http://www.acm.org/pubs/articles/proceedings/ir/312624/p281-ruiz/p281-r%uiz.pdf`.

[Rub76] Frank Rubin. Experiments in text file compression. *Communications of the ACM*, 19(11):617–623, November 1976.

[RW95] Alexander M. Robertson and Peter Willett. The use of genetic algorithms in information retrieval. Technical Report BLRDD Report no. 6201, Research and Development Department, British Library, London, UK, 1995.

[RW96] Alexander M Robertson and Peter Willett. An upperbound to the performance of ranked-output searching: Optimal weighting of query terms using a genetic algorithm. *Journal of Documentation*, 52(4):405–420, December 1996.

[RW98] Alexander M. Robertson and Peter Willett. Applications of n-grams in textual information systems. *Journal of Documentation*, 54(1):48–69, January 1998.

[Sal89] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley, 1989.

[Sch81] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, UK, 1981.

[Sch91a] J. C. Schmitt. Trigram-based method of language identification, October 1991. U.S. Patent No. 5,062,143.

[Sch91b] Johannes Cornelis Scholtes. Filtering the Pravda with a self-organizing neural net. In *Worknotes of the Bellcore Workshop on High Performance Information Filtering*, Chester, NJ, USA, 1991. Bellcore.

[Sch91c] Johannes Cornelis Scholtes. Unsupervised learning and the information retrieval problem. In *Proceedings IJCNN'91, International Joint Conference on Neural Networks*, volume I, pages 95–100, Piscataway, NJ, USA, 1991. IEEE; International Neural Networks Society, IEEE Service Center.

[Sch93a] Johannes Cornelis Scholtes. *Neural Networks in Natural Language Processing and Information Retrieval.* PhD thesis, University of Amsterdam, 1993.

[Sch93b] Johannes Cornelis Scholtes. *Neural Networks in Natural Language Processing and Information Retrieval.* PhD thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, 1993.

[Sen94] Rene Sennhauser. Integration of contextual knowledge sources into a blackboard- based text recognition system. In *Proceedings of DAS94 (Workshop on Document Analysis Systems)*, pages 211–228, 1994.

[SH73] E.J. Schuegraf and H.S. Heaps. Selection of equifrequent word fragments for information retrieval. *Information Storage and Retrieval*, 9:697–711, 1973.

[She94] Beered Dilip Sheth. A learning approach to personalized information filtering. Master's thesis, Massachusetts Institute of Technology, 1994.

[SHP95] Hinrich Schütze, David A. Hull, and Jan O. Pedersen. A comparison of classifiers and document representations for the routing problem. In Edward A. Fox, Peter Ingwersen, and Raya Fidel, editors, *Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval*, pages 229–237, Seattle, US, 1995. ACM Press, New York, US. `ftp://parcftp.xerox.com/pub/qca/papers/sigir95.ps.gz`.

[SJW97] Karen Sparck-Jones and Peter Willett, editors. *Readings in Information Retrieval.* Morgan Kaufmann, San Francisco, CA, USA, 1997.

[Smi97] Martin Smith. The use of genetic programming to build boolean queries for text retrieval through relevance feedback. *Journal of Information Science*, 23(6):423–431, 1997.

[SR96] Penelope Sibun and Jeffrey C. Reynar. Language identification: Examining the issues. In *5th Symposium on Document Analysis and Information Retrieval*, pages 125–135, Las Vegas, Nevada, U.S.A., April 1996. `http://www.cis.upenn.edu/~jcreynar/sdair96.ps.gz`.

[Tan01] Ah-Hwee Tan. Predictive self-organizing networks for text categorization. In David Cheung, Qing Li, and Graham Williams, editors, *Proceedings of PAKDD-01, 5th Pacific-Asia Conferenece on Knowledge Discovery and Data Mining*, pages 66–77, Hong Kong, CN, 2001. Springer Verlag, Heidelberg, DE. Published in the "Lecture Notes in Computer Science" series, number 2035, `http://link.springer.de/link/service/series/0558/papers/2035/20350066.p%df`.

[Tau96a] Daniel R. Tauritz. Adaptive information filtering as a means to overcome information overload. Master's thesis, Leiden University, September 1996. `http://www.liacs.nl/home/dtauritz/papers/thesis.ps.gz`.

[Tau96b] Daniel R. Tauritz. Concepts of adaptive information filtering. Technical Report 96-19, Leiden University, July 1996. `http://www.liacs.nl/home/dtauritz/papers/concepts.ps.gz`.

[Tau96c] Daniel R. Tauritz. Optimization of the discriminatory power of a trigram based document clustering algorithm using evolutionary computation. Technical Report 96-5, Leiden University, April 1996. `http://www.liacs.nl/home/dtauritz/papers/trigram.ps.gz`.

[TKSK97] Daniel R. Tauritz, Joost N. Kok, and Ida G. Sprinkhuizen-Kuyper. Adaptive information filtering using evolutionary computation. In *Proceedings of JCIS'97*, volume 1, pages 77–80, March 1997. `http://www.liacs.nl/home/dtauritz/papers/jcis97.ps.gz`.

[TKSK00] Daniel R. Tauritz, Joost N. Kok, and Ida G. Sprinkhuizen-Kuyper. Adaptive information filtering using evolutionary computation. *Information Sciences*, 122(2–4):121–140, February 2000. `http://www.elsevier.nl/gej-ng/10/23/143/56/27/27/show/`.

[TS88] Bernd Teufel and Stephanie Schmidt. Full text retrieval based on syntactic similarities. *Information Systems*, 13(1):65–70, 1988.

[TSK99a] Daniel R. Tauritz and Ida G. Sprinkhuizen-Kuyper. Adaptive information filtering: improvement of the matching technique and derivation of the evolutionary algorithm. Technical Report

99-04, Leiden University, April 1999. `http://www.liacs.nl/TechRep/1999/tr99-04.html`.

[TSK99b] Daniel R. Tauritz and Ida G. Sprinkhuizen-Kuyper. Adaptive information filtering algorithms. In David J. Hand, Joost N. Kok, and Michael R. Berthold, editors, *Advances in Intelligent Data Analysis, Third International Symposium, IDA-99*, volume 1642 of *Lecture Notes in Computer Science*, pages 513–524. Springer-Verlag, 1999. `http://link.springer.de/link/service/series/0558/bibs/1642/16420513.htm%`.

[TSK00] Daniel R. Tauritz and Ida G. Sprinkhuizen-Kuyper. Adaptive information filtering: evolutionary computation and n-gram representation. In Antal van den Bosch and Hans Weigand, editors, *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference*, pages 157–164, November 2000. `http://www.liacs.nl/home/dtauritz/papers/bnaic00.pdf.gz`.

[TSKK97] Daniel R. Tauritz, Ida G. Sprinkhuizen-Kuyper, and Joost N. Kok. Evolutionary computation applied to adaptive information filtering. In K. van Marcke and W. Daelemans, editors, *Proceedings NAIC'97*, pages 17–26, 1997. `http://www.liacs.nl/home/dtauritz/papers/naic97.ps.gz`.

[Ull77] J.R. Ullmann. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *The Computer Journal*, 20(2):141–147, May 1977.

[UZ98] Alexandra L. Uitdenbogerd and Justin Zobel. Manipulation of music for melody matching. In *Proceedings of the sixth ACM international conference on Multimedia*, pages 235–240, New York, NY, USA, 1998. ACM Press. `http://www.mds.rmit.edu.au/~sandra/mm98/`.

[UZ99] Alexandra L. Uitdenbogerd and Justin Zobel. Melodic matching techniques for large music databases. In *Proceedings of the seventh ACM international conference on Multimedia*, pages 57–66, New York, NY, USA, 1999. ACM Press. `http://www.kom.e-technik.tu-darmstadt.de/acmmm99/ep/uitdenbogerd/index.html`.

[vdP00] Ruud W. van der Pol. *Knowledge-based Query Formulation in Information Retrieval.* PhD thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2000.

[Vra98] Dana Vrajitoru. Crossover improvement for the genetic algorithm in information retrieval. *Information Processing and Management*, 34(4):405–415, 1998.

[WAP99] Stefan Wermter, Garen Arevian, and Christo Panchev. Recurrent neural network learning for text routing. In *Proceedings of ICANN-99, 9th International Conference on Artificial Neural Networks*, pages 898–903, Edinburgh, UK, 1999. Institution of Electrical Engineers, London, UK. `http://www.his.sunderland.ac.uk/ps/icann99.pdf`.

[Wer00] Stefan Wermter. Neural network agents for learning semantic text classification. *Information Retrieval*, 3(2):87–103, 2000. `http://www.his.sunderland.ac.uk/ps/ir4.pdf`.

[Wie98] Floris J. Wiesman. *Information Retrieval by Graphically Browsing Meta-Information.* PhD thesis, Universiteit Maastricht, Maastricht, The Netherlands, 1998.

[Wil79] Peter Willett. Document retrieval experiments using indexing vocabularies of varying size. ii. hashing, truncation, digram and trigram encoding of index terms. *Journal of Documentation*, 35(4):296–305, December 1979.

[Win99] Werner Winiwarter. PEA - a Personal Email Assistant with evolutionary adaptation. *International Journal of Information Technology*, 5(1), 1999.

[Wis87] Janusz L. Wisniewski. Effective text compression with simultaneous digram and trigram encoding. *Journal of Information Science: Principles & Practice*, 13(3):159–164, 1987.

[WPA99] Stefan Wermter, Christo Panchev, and Garen Arevian. Hybrid neural plausibility networks for news agents. In *Proceedings of AAAI-99, 16th Conference of the American Association for Artificial Intelligence*, pages 93–98, Orlando, US, 1999. AAAI

Press, Menlo Park, US. `http://www.his.sunderland.ac.uk/ps/aaai99.pdf`.

[WPW95] Erik D. Wiener, Jan O. Pedersen, and Andreas S. Weigend. A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 317–332, 1995. `http://www.stern.nyu.edu/~aweigend/Research/Papers/TextCategorization/Wiener.Pedersen.Weigend\_SDAIR95.ps`.

[YH92] E. J. Yannakoudakis and P. J. Hutton. An assessment of n-phoneme statistics in phoneme guessing algorithms which aim to incorporate phonotactic constraints. *Speech Communications*, 11(6):581–602, December 1992.

[YKR93] Jing-Jye Yang, Robert R. Korfhage, and Edie Rasmussen. Query improvement in information retrieval using genetic algorithm:a report on the experiments of the trec project. In *Text Retrieval Conference (TREC-1)*, pages 31–58, 1993.

[YL99a] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, pages 42–49, Berkeley, US, 1999. ACM Press, New York, US. `http://www.cs.cmu.edu/~yiming/papers.yy/sigir99.ps`.

[YL99b] Edmund S. Yu and Elizabeth D. Liddy. Feature selection in text categorization using the Baldwin effect networks. In *Proceedings of IJCNN-99, 10th International Joint Conference on Neural Networks*, pages 2924–2927, Washington, DC, 1999. IEEE Computer Society Press, Los Alamitos, US.

[YL00] Hsin-Chang Yang and Chung-Hong Lee. Automatic category generation for text documents by self-organizing maps. In Shun-Ichi Amari, C. Lee Giles, Marco Gori, and Vincenzo Piuri, editors, *Proceedings of IJCNN-00, International Joint Conference on Neural Networks*, volume 3, pages 581–586, Como, IT, 2000.

IEEE Computer Society Press, Los Alamitos, US. `http://dlib.computer.org/conferen/ijcnn/0619/pdf/06193581.pdf`.

[Zav95] Jakub Zavrel. Neural information retrieval: An experimental study of clustering and browsing of document collections with neural networks. Master's thesis, University of Amsterdam, 1995.

[ZD95] Justin Zobel and Philip W. Dart. Finding approximate matches in large lexicons. *Software: Practice and Experience*, 25(3):331–345, March 1995.

[ZKL96] Byoung-Tak Zhang, Ju-Hyun Kwak, and Chang-Hoon Lee. Building software agents for information filtering on the internet: A genetic programming approach. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1996 Conference (GP-96)*, page 196. Stanford University Bookstore, 1996.

[ZPZ81] E. M. Zamora, J. J. Pollock, and Antonio Zamora. The use of trigram analysis for spelling error detection. *Information Processing and Management*, 17(6):305–316, June 1981.

Nederlandstalige samenvatting

Adaptieve informatiefiltering houdt zich bezig met het filteren van informatiestromen in dynamische (veranderende) omgevingen. De veranderingen kunnen plaatsvinden zowel aan de transmissiezijde — de aard van de stromen kan veranderen — als aan de receptiezijde — de interesses van de gebruiker (of gebruikersgroep) kunnen veranderen. Hoewel informatiefiltering en informatievergaring veel gemeen hebben, richt deze dissertatie zich juist op de verschillen. Het temporele aspect vereist flexibelere methoden van documentrepresentatie voor informatiefiltering dan voor informatievergaring, waar alle voorkomende termen van tevoren bekend zijn. Ook houdt informatiefiltering typisch gebruikersprofielen bij in plaats van de statische queries (vragen) van informatievergaring. Dit maakt een lerend systeem met het vermogen om met dynamische omgevingen om te gaan noodzakelijk.

Het onderzoek beschreven in deze dissertatie exploreert de toepassing van twee karakteristieke *machine learning* technieken, namelijk evolutionaire computatie en neurale computatie (neurale netwerken), voor de intelligente optimalisatie van de incrementele classificatie van informatiestromen. De documentrepresentatie die in dit onderzoek gehanteerd werd, is die van gewogen frequentiedistributies van n-grammen. De gewichten geassocieerd met de n-grammen zijn de attributen die geoptimaliseerd worden.

De resultaten in deze dissertatie geven een positieve indicatie voor het gebruik van de *machine learning* technieken zoals beschreven in de vorige alinea. Zowel geschreven documenten als gesproken documenten worden succesvol geclassificeerd, met inachtneming van de beperkingen gesteld door adaptieve informatiefiltering. Een belangrijk aspect dat nog verder onderzoek vraagt is schaalbaarheid: de classificatieresulten zakken van meer dan 95% correct voor twee onderwerpen tot iets onder de 85% correct voor tien onderwerpen, hoewel het dalen van de classificatieresultaten lijkt te stoppen wanneer het aantal onderwerpen meer dan acht is.

Curriculum Vitae

Daniel Remy Tauritz was born in Leiden, The Netherlands, on July 22nd, 1973. He finished his pre-academic secondary education in 1991 at Rijnlands Lyceum in Wassenaar. After studying computer science and mathematics for about two years at Leiden University, he concentrated on computer science and specialized in the fields of computational intelligence (evolutionary computation in particular), information filtering, and automatic document classification. He received his Masters degree in 1996 after completing a research internship at the NATO C3 Agency (formerly known as SHAPE Technical Center) in The Hague, where he investigated the application of evolutionary computation for the optimization of adaptive information filtering systems.

In 1997 he taught for two quarters as an instructor in the School of Computer and Information Sciences at the University of South Alabama in Mobile, Alabama, USA. From December 1st 1997 through November 30th 2001 he performed his Ph.D. research under a grant from NWO (the Dutch National Science Foundation). His dissertation advisor was Prof. dr. Joost N. Kok and his daily supervisor was Dr. Ida G. Sprinkhuizen-Kuyper.

Titles in the IPA Dissertation Series

A.W. Heerink. *Ins and Outs in Refusal Testing.* Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering.* Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation.* Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller.* Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model.* Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing.* Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes.* Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search.* Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection.* Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Schedulere Optimization in Real-Time Distributed Databases.* Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics.* Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems.* Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods.* Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems.* Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules.* Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System.* Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars.* Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Progam Construction.* Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic.* Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of* *distributed algorithms.* Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes.* Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols.* Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor.* Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant.* Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs.* Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language.* Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure.* Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication.* Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle.* Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection.* Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes.* Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes.* Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking.* Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ.* Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking.*