FEATURE AND PROTOTYPE EVOLUTION FOR NEAREST NEIGHBOR CLASSIFICATION OF WEB DOCUMENTS

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

MICHELLE ANDREEN CHEATHAM
B.S., University of Kentucky, 2001
M.B.A., University of Kentucky, 2001

2005
Wright State University

WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

Aug 17, 2005

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Michelle Andreen Cheatham ENTITLED Feature and Prototype Evolution for Nearest Neighbor Classification of Web Documents BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Mateen M. Rizki, Ph.D.
Thesis Director

Forouzan Golshani, Ph.D.
Department Chair

Committee on
Final Examination

Mateen M. Rizki, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Thomas A. Sudkamp, Ph.D.

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

ABSTRACT

Cheatham, Michelle Andreen. M.S., Department of Computer Science and Engineering, Wright State University, 2005. Feature and Prototype Evolution for Nearest Neighbor Classification of Web Documents.

The problem of information overload could be greatly reduced if reliable automatic text categorization were possible. This thesis considers the application of a nearest neighbor classifier (NNC) approach to classification of web pages. NNCs are based on the idea that unknown inputs are most likely to belong to the same class as the closest known prototype. The utility of this method is dependent on choosing appropriate features to represent the documents and finding a set of prototypes that can accurately represent the documents within that feature space. The computation and memory requirements of the NNC increase with the number of features and prototypes. We will show that it is possible to use a standard genetic algorithm to optimize both the feature and prototype sets for an NNC applied to the text classification domain. In addition, we will demonstrate that the features and prototypes may be evolved sequentially rather than simultaneously and still produce comparable results.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Mateen Rizki, for his patience and valuable insight, my husband Jason for endless proofreading and LaTeX advice, and Robert Ehret of the Collaborative Technologies Branch of the Air Force Research Laboratory for his support throughout my graduate work. I would also like to thank Drs. Sudkamp and Krishnaprasad for serving on my thesis committee.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 MOTIVATION

A study done at the University of California at Berkely in 2002 found that 5 exabytes of information was created that year (1 exabyte is 10^{18} bytes), 92% of which was stored on magentic media such as hard disks. Google claims to have indexed over 8 billion documents, including PowerPoint, Word, text, pdf, and Flash as well as HTML files. With this staggering amount of data available and more on the way, information overload is a serious concern.

Researchers have been trying almost since computers were invented to automatically classify similar documents into groups. Reliable text classification is the foundation for a variety of applications with the potential to mitigate the effects of information overload. For example, email could be categorized as spam or legitimate, as well as automatically routed to an appropriate folder. Essays submitted by students could be assigned a grade. Document management systems could file new content into the correct folders. This thesis will examine one potential method for text classification.

1.2 PROBLEM DESCRIPTION

Many different algorithms for text classification exist. As is true in most interesting problem domains, none of these methods completely dominates the others. Some are very sensitive to the underlying distribution of the data, some require that the

number of categories be known in advance, and others require the user to fine-tune a set of parameters in order to achieve good results. An overview of the most common approaches to the text classification problem is given in Section 2.1.

We have chosen a nearest neighbor classifier (NNC) for use on this project. This is a simple, fast, and well-researched text categorization method. The principle behind the NNC is the notion that if two items are close together in the feature space, they probably belong to the same class. The theory and application of an NNC, including the time complexity, memory requirements and error bounds, are discussed in detail in Section 2.2.

Text classification methods do not generally operate on the raw text of the documents to be classified. Instead, the text documents are converted into numeric feature vectors. These features are most commonly some measure of the frequency with which a dictionary of words appears in the document, although other characteristics of the document could also be used. The selection of which features to use strongly impacts the overall performance of the classifier. Section 2.3 explains what makes a good feature set and describes several different approaches to the feature selection problem.

Many supervised text categorization methods, including the NNC, base the classification of unknown documents on a set of documents whose categories are known. These documents are termed prototypes. The goal in prototype selection is to choose prototypes that are representative of the general distribution of the documents in the feature space. There are several potential pitfalls here: not choosing enough prototypes to characterize the distribution, choosing an atypical example that results in misclassifications, and overfitting the training data are some examples. This issue is covered in more detail in Section 2.4.

We have chosen to explore the utility of using a genetic algorithm (GA) to opti-

mize both the features and prototype set used by a nearest neighbor classifier for text categorization. A GA is a heuristic search method that explores a large search space by mimicking the natural evolutionary processes of reproduction, mutation, and competition. Several other researchers have used a GA for either prototype or feature selection, but only a couple have attempted to evolve both sets. Section 2.5 gives an overview of this work. The majority of this research has been done in domains that contain 20 or 30 potential features, while text classification is a more complex problem that typically involves hundreds or thousands of features.

1.3 APPROACH

Chapter 3 describes our implementation in detail, including the chosen document representation, the generation of an initial feature and prototype set from which to begin the optimization process, the representation used by the GA, and the specifics of the genetic operators.

This thesis attempts to answer two main questions: Is a genetic algorithm an effective method for selecting the features and prototypes used by an NNC? and Does evolving the feature and prototype sets simultaneously produce better results than sequential optimization? Several tests were run to determine the answers to these questions. We chose to use the BankSearch dataset, a collection of 10,000 web pages classified into 10 different groups, as our document corpus. Each test was run on two groups of documents on very different subjects, two similar groups of documents, and all ten groups. To get a baseline accuracy for the NNC, we first classified the documents using all available features and prototypes. The prototype set was then held constant, and the dictionary was evolved using the GA. The process was then reversed, fixing the dictionary and evolving the prototype set. The results of these tests were compared to the unoptimized NNC and to the results of choosing

a dictionary and prototype set randomly instead of using the GA to evolve them. Finally, we compared the system's performance when first the dictionary and then the prototypes were evolved versus optimizing the prototypes first and then the dictionary or evolving both simultaneously using a nested GA.

1.4 RESULTS

The nearest neighbor classifier was found to be a reasonably accurate approach to the text classification problem, although its accuracy is not high enough for use in applications where a misclassification could cause serious repercussions, such as spam filtering. Contrary to the results of some other researchers, we found that the optimized NNC could achieve slightly higher accuracy than the baseline version while still cutting the feature and prototype sets by 50 percent or more. The GA produced smaller sets than random selection, indicating the value of the crossover operator's ability to exploit useful subsets of features and prototypes. Evolving either the features or the prototype set improved the NNC's accuracy by about the same amount. Evolving both sets did not further increase accuracy, but did improve speed and reduce memory requirements. Optimizing the feature and prototype sets simultaneously did not produce better results than sequential evolution, implying that it is unnecessary to use a computationally expensive nested GA to achieve good results.

2 BACKGROUND

2.1 TEXT CLASSIFICATION

Automated text classification has been researched for decades. It was recognized early on that with so much information available in the computer age, it would be increasingly difficult to find the particular documents needed. This prompted researchers to attempt to automatically classify documents. Searches could then be focused on only those items in categories that might be relevant to a particular query. Early work in this area focused on analyzing documents at the semantic level, but these efforts were largely unsuccessful. By the time van Rijsbergen published his seminal work in the field of information retrieval in 1975, text classification techniques almost exclusively used a statistical approach [1]. Baeza-Yates and Ribeiro-Neto provide an updated survey of this field in [2].

Both clustering and categorization fall under the broad heading of text classification. Clustering is an unsupervised task, meaning the correct classification is not known for any documents. These algorithms attempt to uncover relationships among the data that can serve as criteria to group similar documents into clusters. The k-means algorithm, introduced by Hartigan in 1975, is the most widely used and researched clustering algorithm [3]. In this very intuitive method, each of the k clusters is represented by the mean of the points within that cluster. New documents are assigned to the cluster of the nearest centroid, and the centroid is recomputed.

This type of clustering is straightforward to apply to numeric attributes, and there are many different optimizations available. Hierarchical methods are another type of clustering technique. These approaches create a tree of clusters, with siblings further partitioning the space covered by their parent. This is done by either starting with every document in its own cluster and recursively merging similar clusters or by starting with a single cluster and recursively splitting it into smaller pieces. These algorithms can handle non-numeric data and provide information about the documents at varying levels of granularity [4]. However, the user must generally specify the number of clusters to form in advance. There are many hierarchical clustering methods, which differ primarily in the function used to determine when to merge or split a cluster and the shapes of clusters that can be created. A third group of clustering methods includes neural networks and other machine learning approaches. Kohonen's Self Organizing Map (SOM) is the most common example of this type of method [5]. The system begins with a regular pattern of neurons. In its simplest form, a winner-take-all competitive learning system is used, in which the closest neuron fires to indicate the cluster of the given input, and that neuron is moved closer to the input vector. This type of approach results in a neural network that "maps" the topology of the data. A more complete survey of text clustering techniques can be found in [4].

Text categorization is a supervised learning task in which the classifier attempts to learn a relationship between a training set of documents and their categories, which are known apriori. The classifier then applies that relationship to a test set of documents whose categories are not known. We will give only a brief overview of several text categorization methods here and then focus specifically on the nearest neighbor classifier, which is the basis for our current work. Support Vector Machines (SVM) are a popular new method that attempts to solve the two-class problem by learning

the location of a hyperplane that creates a boundary with the largest possible margins between itself and items of either class [6]. Quadratic programming is used to find this hyperplane based on a training set. The original SVM approach is limited to the two-class case and those classes must be linearly separable. However, some work has been done to relax those restrictions including chaining SVMs to deal with more classes and combining features in order to make the classes linearly seperable [7]. Another method, Linear Least Squares Fit (LLSF), uses as input the feature vectors of the training documents and a binary vector for each document that indicates the category (or categories) to which that document belongs. The algorithm then finds the linear least square fit between these two matrices. The result is a mapping between a feature vector and the proper categories [8]. The most popular method for use in practical applications is currently the Bayes classifier, which uses the joint probabilities of words and categories to estimate the probability that a given document belongs to each of the possible categories [9]. There are also many categorization approaches based on decision trees and neural networks [10], [11]. Yang and Liu compared many of these classifiers in [7], and report that the methods studied all achieve comparable accuracies when provided with a good sized training set.

2.2 NEAREST NEIGHBOR CLASSIFIER

The nearest neighbor classifier is one of the most commonly used categorization methods. It is based on the appealingly simple notion that items which are close together in the feature space belong to the same class. In essence, a distance measure is computed between an item whose class is unknown and a set of prototypes of known classification. The unknown item is assigned the classification of the prototype nearest it. A more formal description can be found in [12].

An example involving loan applications is shown in Figure 2.1. In this simplified

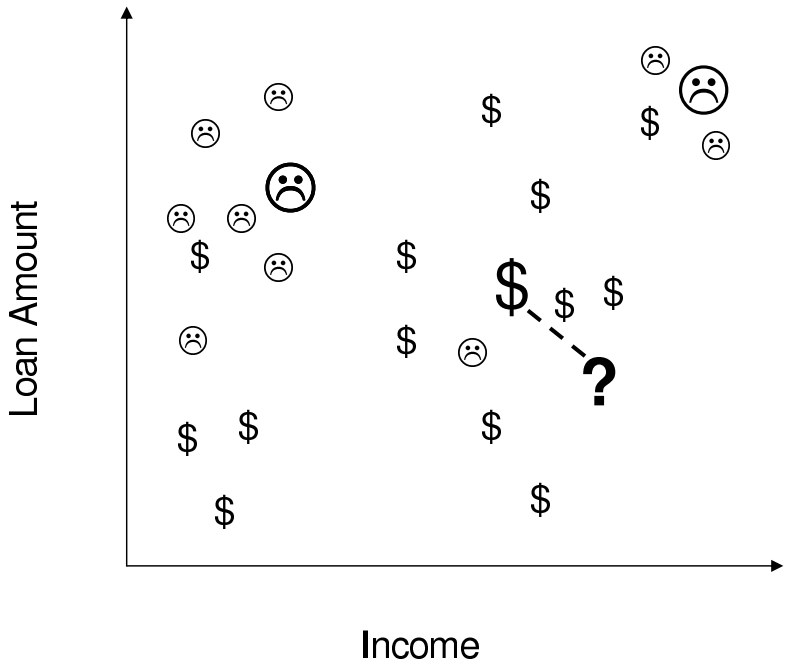

Figure 2.1: Use of a nearest neighbor classifier to determine whether or not a loan should be approved based on the loan amount and the income of the loan applicant.

case, the two features used to determine whether or not to approve the loan are the amount requested and the income of the applicant. A dollar sign indicates the application was approved and a sad face that it was denied. The three larger icons have been selected as prototypes. The question mark is a new loan application. Because it is closest to an approved prototype, this application will be classifed as approved.

The error rate of an NNC has been shown by Cover and Hart to be bounded below by the Bayesian probability of error and above by twice this value [12]. That is to say, if one has a complete statistical understanding of the distribution of the samples and their categories, a Bayesian classifier is optimal and the NNC, which does not use any knowledge of the probability densities, can therefore not perform better than this. The upper limit on NNC error is twice the Bayesian probablity of error. This is

only the case when the size of the training set approaches infinity, however. Langley and Iba have done some theoretical work to compute the average-case accuracy as a function of the number of training cases and the number of relevant and irrelevant attributes [13]. More convincing, perhaps, is that the NNC has been shown by many studies to perform as well or better than alternative classification schemes on both synthetic and real data sets [7], [14].

Both the computation time and memory required for the NNC increase with the number of prototypes and the size of the feature vector. For an NNC using manhattan distance, there are $p(2d-1)$ operations required to compute the distance between the document to be classified and all of the prototypes, where p is the number of prototypes and d is the number of words in the dictionary. Another p comparison operations must be done to find the closest prototype. The p prototypes, each of length d, must also be stored in memory. Much of the literature surrounding NNCs focuses on this issue ([15] and [16] for example). However, with the speed and memory of today's computers and the rate at which these resources are increasing, we believe this should not be the main focus. Features and prototypes need to be chosen carefully, but primarily because redundant or irrelevant choices may be very detrimental to classification accuracy rather than merely to reduce computation time.

It is difficult to determine where the NNC concept originated, but one of the earliest mentions is from Fix and Hodges in 1951 [17]. Because an NNC does not depend on knowledge of the probability density of the input feature vectors, it is commonly used in practical categorization applications [16]. These applications include domains such as MPEG video traffic classification [18], speech recognition [19], and classification of newspaper articles [7].

Due to its long history and widespread use, a large body of research exists concerning every aspect of the NNC. Our primary focus in this project is on the selection

Document	*Features*			*Class*
Document 1	credit	mortgage	point	Banking
Document 2	game	point		Soccer

Table 2.1: Features may be irrelevant, weakly relevant, or strongly relevant.

of prototypes and features for use by an NNC. We therefore go into more detail below on work related specifically to these areas.

2.3 FEATURE SELECTION

Feature selection has been an area of interest for several decades, both because it can allow a better insight into the nature of the data and because it can enhance the performance of classifiers. Blum looks at features as either strongly relevant, weakly relevant, or irrelevant [20]. A strongly relevant feature is one where there exist two samples that differ only with respect to this feature and whose classes are different. If a feature is weakly relevant it is possible to remove a subset of other features and make the given feature strongly relevant. In other words, a weakly relevant feature is important but redundant. In the simple example shown in Table 2.1, *game* is strongly relevant, *credit* and *mortgage* are weakly relevant, and *point* is irrelevant.

It seems intuitive to state that an optimal feature set should contain all strongly relevant features and potentially some that are weakly relevant. This would be the case if the goal is to understand the data, however, if the objective is to build an accurate classifier other factors come into play. In general, irrelevant features should be removed [21], but if a large number of relevant features are highly correlated to one another, it may be unnecessary to make use of all of them in a classifier [22]. However, sometimes better class separation can be achieved by adding redundant variables [22].

Nearest neighbor classifiers in particular are very sensitive to the feature vector

used. According to [13], the number of prototypes needed to achieve a given accuracy increases exponentially with the number of irrelevant features. The situation is complicated because features which may seem to have little value on their own are highly relevant when considered as a group. Thus, feature selection algorithms that attempt to generate near-optimal solutions must consider subsets rather than individual features. Selecting an optimal feature set is known to be an NP complete problem [23]. The size of the search space is 2^N, where N is the number of potential features. This can easily be in the thousands for text classification domains. This may be somewhat offset by the fact that there are likely to be many good solutions, especially if there is a high degree of correlation among the potential features [21]. However, the more classes that are involved, the less likely this is [22].

There are many feature selection methods. Some of these are not truly "selection" methods, but instead derive new features from existing ones. There are also unsupervised techniques such as K-means clustering, which replaces a group of similar features by their centroid. These unsupervised methods use properties such as saliency, entropy, smoothness, density, and reliability to determine the viability of a feature set [22]. Our focus in this work will be on supervised feature selection methods, however. These are generally divided into two groups: filter and wrapper methods.

2.3.1 Filter Methods

Filter methods choose features based on their relation to each other and/or to the training data. They are independent of the classifier used, which speeds up the feature selection process but may limit the efficacy of the chosen features. Instead of relying on the classifier to evaluate potential features, a generic evaluation function is used.

The simplest filter methods rank the features in isolation, and choose the top-

rated features for inclusion in the final set. The evaluation functions used to rate the features are typically based on an individual feature's frequency within a document or a class. These are often normalized to deal with documents and classes of varying sizes. These methods are fast, generally $O(n)$, but they cannot take into account interactions among features and therefore often generate poorly performing feature sets. This is particularly problematic for high-dimensional data that may contain a large number of redundant features (as in text classification problems), because redundant features will generally be ranked similarly by any evaluation method [24]. However, these methods are often used to thin the number of potential features before invoking a more complex algorithm.

Other filter methods consider the correlations between potential features when ranking them. The evaluation functions for these methods are more complex, usually involving the correlation of a feature both to a particular class and to the other potential features [25]. Common methods of this type include Relief-F and FOCUS. Relief-F ranks features according to their relevance to the class labels. Items are taken randomly from the training set, and the correlation between each feature and each class is determined. Features are then ordered based on their positive correlation with the correct class and negative correlation to the other classes [26]. FOCUS considers every possible subset of features and selects the smallest set that can accurately classify the training set [27]. These methods generate reasonable feature sets in a limited amount of time, but the performance can vary widely depending on the type of classifier chosen.

2.3.2 Wrapper Methods

Wrapper methods select features by using the classifier to judge a subset of features. This approach is considerably more time consuming than a filter, but the resulting

feature set is optimized for the classifier and inter-feature relationships are considered. Wrapper methods vary based on their approaches to four decisions: starting state, search algorithm, evaluation function, and stopping criteria. The algorithm can either start with an empty feature set, a set containing all potential features, or a random subset of potential features. Starting from an empty feature set and working up is generally faster than beginning with the entire feature set, but it can generate less than optimal results. Search algorithms are either complete, heuristic, or random. A complete search is guaranteed to find the optimal solution, usually by searching all possible alternatives (although the ability to backtrack may allow for some improvements to increase efficiency). Heuristic search functions make decisions about which areas of the search space to explore next based on a search strategy. Random searches, as the name implies, randomly wander through the search space. The evaluation function of a wrapper method is essentially the performance of the chosen classifier. Finally, the stopping condition can be when a certain performance level is achieved, when a given amount of time has passed, or when the algorithm ceases to improve performance by a specified amount.

One of the simplest wrapper methods is sequential forward selection (SFS), which starts from an empty feature set and tries each remaining potential feature. The feature that leads to the largest performance increase is added to the set, and the process is repeated. Sequential backward selection (SBS) is similar, except that it begins from the entire set of potential features. Random generation plus sequential selection is a variation on these techniques that begins the SFS or SBS algorithms from a randomly chosen feature subset [25]. It should be noted that while Dash [25] classifies these as wrapper methods, this is because they are often evaluated based on the accuracy of the classifier. SFS and SBS could also be used as filter methods by choosing a different evaluation method. Both SFS and SBS may generate sub-optimal

feature sets because once a feature is added or removed from the set, it cannot be reconsidered at a later time. Beam search [28] keeps a queue of candidate feature sets, sorted according to their performance. The feature set at the front of the queue is expanded to all possible subsets by adding a feature to it. These new subsets are then placed in their proper locations within the queue, and the process is repeated. The size of the queue controls how completely the algorithm explores the search space. Other methods include plus l-take away r, simulated annealing, branch-and-bound, and node pruning of neural networks [29].

2.4 PROTOTYPE SELECTION

With a good feature set, the input data will hopefully fall into relatively distinct clusters. The goal of the prototype set is then to have at least one representative per cluster. If the intra-cluster distances are all smaller than the inter-cluster distances then only one prototype per class is required 2.2(a). If the distance across the clusters is greater than the distance between them, some items may be incorrectly classified 2.2(b). In that case, it is necessary to add prototypes until the distance between all of the inputs are closer to a prototype of their own class than to that of another class 2.2(c). However, care must be taken not to overfit the training data by choosing many outliers as prototypes 2.2(d). In that case, the NNC will be considerably less accurate on the test data than the training data.

There are literally dozens of methods for optimizing the prototype set of a nearest neighbor classifier. We provide a brief overview here that is intended to illustrate the primary strategies used and refer the interested reader to [30] for a more detailed survey of the field.

Prototype optimization techniques can be divided into two main groups: those that select certain representatives from the training set to serve as prototypes as-is

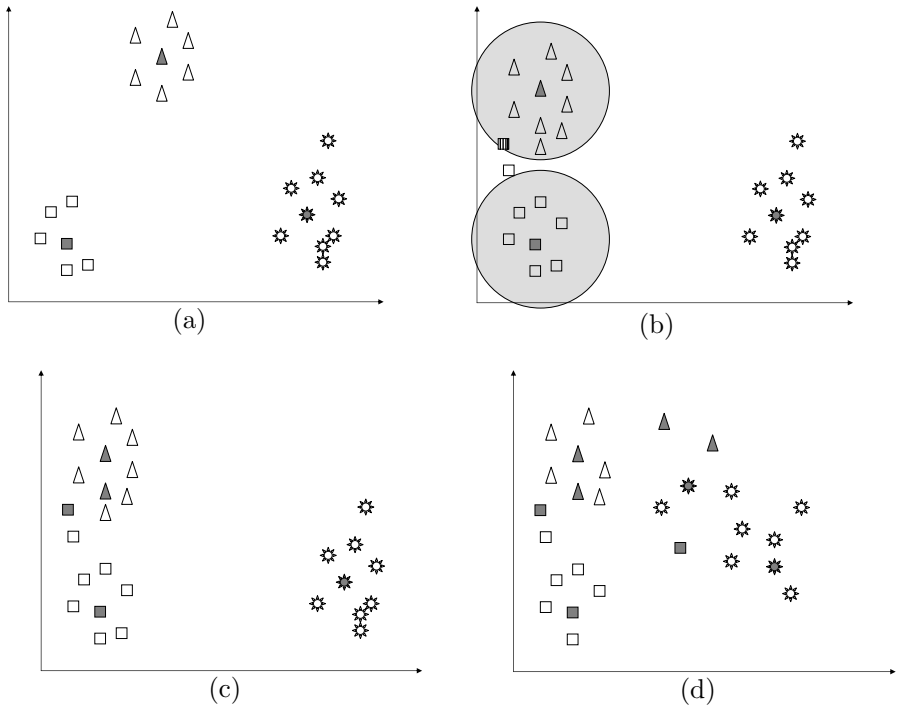

Figure 2.2: Problematic Prototype Selection

and those that generate prototypes based on the training set but that do not actually occur within that set. There is not a standard terminology for these groups, but we will refer to them as prototype *selection* and prototype *creation*, respectively.

2.4.1 Prototype Selection Methods

The most-cited selection method is Hart's Condensed Nearest Neighbor (CNN) rule [31]. A training input is chosen randomly to be the initial prototype. Then each training sample is classified using that prototype. Any samples that are misclassified are added to the prototype set. The process is repeated until a consistant prototype set, i.e. one that results in no classification errors on the training data, is generated. Interestingly Wilson's method, another very common technique, takes the opposite approach. A knn classification (where the k nearest neighbors "vote" on the correct category) is done on the entire training set, and all misclassified inputs are *removed*

from the prototype set [32]. A plethora of other algorithms are based on these two methods.

CNN and Wilson's method iteratively build up the prototype set from a seed or pare it down from the entire training set, respectively. Another group of selection methods evaluates entire subsets of prototypes at once. This group includes genetic algorithms, tabu search, and random search [30].

2.4.2 Prototype Creation Methods

Learning Vector Quantization (LVQ), originally proposed by Kohonen [33], is probably the most well-known creative method. An initial prototype set is chosen randomly with the restriction that it contain at least one sample from each class. For each element in the training set, the closest prototype is computed. If the prototype belongs to the same class as the training input, it is updated using $p' = p + \alpha(x - p)$, where p is the winning prototype, p' is the updated prototype, x is the training sample, and α is a varying learning rate. This moves the prototype towards the input vector. If the classification was incorrect, the update equation is $p' = p - \alpha(x - p)$, which pushes the prototype away from the input vector. Many other creation methods are similar to LVQ, differing primarily in the selection of prototypes to be updated and the update equation itself. These include Decision Surface Mapping (DSM) [34], Prototypes for Nearest Neighbor (PNN) [35], simulated annealing [36], and various approaches involving converting the NNC to a neural network, training it, and then converting it back [37], [36].

Other creation methods differ significantly from LVQ. For example, bootstrap approaches iteratively replace each training input with the mean of its k nearest neighbors of the same class [38]. Support Vector Machines (SVM) attempt to find the hyperplane that best separates two classes by solving a quadratic programming

problem [6], [39]. There are also a variety of other techniques that first cluster the training set and then use the centroids of these clusters as prototypes [30].

2.4.3 Comparison

Comparing the many methods for generating a prototype set is extremely difficult. As Bezdek's results in [30] illustrate, performance is intimately dependent upon the nature of the data involved, particularly the dimensionality of the feature space and the underlying probability density of the training input.

Both [30] and [40] present comparisons of several prototype set generation methods on a variety of data. It is interesting to note that in each case there is a fairly large group with similar performance. This implies that factors other than accuracy should also be of concern when choosing an algorithm for a particular application. These may include simplicity of the algorithm, the number of parameters that must be specified by the user, the sensitivity of the algorithm to the size of the training set, and whether or not the algorithm is parallelizable. Overall, iterative selection methods tend to be the most straightforward and do not generally involve user-specified parameters, but they are sensitive to both the size and the order of the training input. In addition, there is no way for these methods to trade off accuracy in order to generate smaller prototype sets [30]. Non-iterative selection methods such as GA and random search are capable of producing passable solutions more quickly (and can be executed in parallel), but the resulting prototype set is not guaranteed to be consistent with the training data. For GA and tabu search, the user must also fine-tune several parameters. Creation methods are capable producing prototype sets that achieve slightly higher accuracies [30], but these methods are on the whole more complex and slower than selection algorithms.

2.5 GENETIC ALGORITHM

This section provides a brief overview of genetic algorithms and then focuses specifically on how this technique has been applied to feature and prototype selection.

The genetic algorithm (GA) belongs to a class of global optimization methods that takes its inspiration from nature. This group also includes simulated annealing, which attempts to find a global minimum in the same way that a heated solid will crystalize into a state with a minimum of free energy, and tabu search, which occasionally moves in non-optimal directions in an effort to avoid areas of the search space that have already been considered (as humans have been observed to do).

The genetic algorithm uses reproduction, random variation, competition, and selection of a population of individuals to explore the search space. GA is not guaranteed to find the optimal solution, but it is a generic process that can be tailored to many different problem domains in order to produce good solutions. One of the main advantages of GA is that through the choice of various parameters and the selection method employed, various constraints on the form of solutions can be easily put in place [41]. It is also a robust method in the sense that even poor parameter choices generally yield relatively good solutions [42].

The genetic algorithm was first proposed by Holland in the early 1960s. Much of the early GA work done by Holland and his students is summarized in his 1975 book [43]. The general procedure is given in Figure 2.3. Once a representation is chosen for the problem, an initial population is generated. Then a set of individuals is selected for reproduction and offspring are created via the crossover operator. Following this, some individuals are chosen for possible mutation, and a small percentage of individual genes within these individuals are mutated. It is common practice to shield the most fit individuals from mutation to avoid losing the best solution found thus far.

$t \leftarrow 0$;

$P(t)[1:\mu] \leftarrow$ **initialize**(μ, ν);

while $(t < \theta_t$ or $max(F(t)[1:\mu]) < \theta_f)$

$P'(t)[1:\mu+\gamma] \leftarrow$ **recombine**$(P(t)[1:\mu], \gamma)$;
$P''(t)[1:\mu+\gamma] \leftarrow$ **mutate**$(P'(t)[m], \lambda)$;
$F(t)[1:\mu+\gamma] \leftarrow$ **evaluate**$(P''(t)[1:\mu+\gamma])$;
$P(t+1)[1:\mu] \leftarrow$ **select**$(P''(t)[1:\mu+\gamma], F(t)[1:\mu+\gamma], \mu)$;
$t \leftarrow t+1$;

end while

t – epoch
μ – population size
ν – population bias
γ – number of offspring to generate
λ – mutation rate
m – individuals to mutate
θ_t – generation limit
θ_f – fitness goal

Figure 2.3: Genetic Algorithm

The population is then judged by evaluating each individual with respect to a fitness function. Finally, a subset of these individuals is chosen to proceed to the following generation. This process is repeated until a certain fitness is achieved or for a chosen number of generations. The algorithm can be tuned to explore the search space in different ways based on the values of various parameters and the methods chosen for reproduction and selection. One thing that should be noted concerning Figure 2.3 is the large number of parameters that must be specified to instantiate the GA for a particular problem. In his original work, Holland used a bit string representation, one-point crossover, a very low mutation rate, proportional selection and only generated one offspring per generation [43]. The choices for our system will be discussed in detail in Chapter 3.

A variety of work has been done regarding using a GA to select features for classifiers. Brill performed feature selection for an NNC using a modified genetic algorithm and then applied the resulting feature set to a counterpropagation neural network, which is conceptually similar to the NNC but takes a long time to train [44]. The GA is "modified" in that it uses punctuated equilibria, meaning that several distinct populations are evolved and periodically the best individuals from each population are exchanged. Brill also uses training set sampling to limit the time spent evaluating different feature subsets – only a subset of the training set is used to evaluate each generation, but the subset is different each time to avoid overfitting to a small training set. Brill's results indicate that this method does not improve on the classifier's performance using all potential features, but it did reduce the number of features by an average of 40%. Yoshida et al compared a GA to Fisher's ratios (a filter method) to choose features for linear discriminant analysis and partial least squares discriminant classifiers [45]. Neither method improved the performance over the entire feature set, however both significantly reduced the number of features without sacrificing accuracy. The GA performed slightly better than the Fisher ratio, especially in cases where the Fisher ratios were low with a wide distribution. The majority of features selected by both methods were the same, however the GA generally selected several more than the Fisher approach. Oh and his colleagues propose a hybrid method in which a local optimization step is added to the GA before the survivors of the current generation are chosen [46]. The local optimization is done by removing the r least significant features and substituting the l that are most significant, as determined by a filter algorithm. They claim better accuracy than sequential floating forward search, the dominant sequential method, and indicate that conflicting results among other researchers in the field may be due to poorly chosen parameters or inefficient genetic operators.

Kangas [15] and Kuncheva [41] have both researched prototype selection for an NNC using genetic algorithms. Kangas's method applies the GA to the actual feature vectors of the prototypes and can therefore create new prototypes instead of selecting a subset of the training data. The GA attempts to minimize the intraclass distances while maximizing the distances between classes. It was shown that this method compared favorably to the Condensed Nearest Neighbor rule when used to categorize characters in the sense that it did choose prototypes that resulted in larger distances between classes, but the author did not actually give accuracy information for the two methods. Kuncheva compared a GA prototype selection method to random search and several clustering prototype creation techniques. This GA represented the prototype set as a binary vector the length of the training set. A 1 indicated that the given document was to be used as a prototype and a 0 that it was not. Experimental results on the IRIS data set indicated that GA is recommended as the best choice, although the author notes that random search performed surprisingly well.

A small amount of research has also been done on using a GA to select both the prototype set and the features for a classifier. Kuncheva and Jain compare using a GA for both purposes to using CNN and Wilson's method for prototype selection and SFS for feature selection [14]. An individual in the population consisted of a binary string of length $N + n$ where N is the number of potential prototypes and n is the number of potential features. Their results indicate that GA is the best choice among the methods studied when both accuracy and the number of features and prototypes are considered, however, SFS and Wilson's method used in conjunction performed very similarly. Other work has built upon this approach by attempting to add "intelligence" to the GA [47]. This approach uses orthogonal experimental design (OED) within the crossover operator to determine how much each gene is

contributing to the overall fitness and chooses the most useful genes for propagation to the offspring. In this way, the algorithm attempts to explore the (very large) search space more efficiently than a traditional GA. The authors indicate that this method does indeed produce better results when accuracy and size are considered.

3 METHODS

3.1 PROBLEM DOMAIN

The overall goal of this effort is to be able to simulaneously categorize as many different document types as possible: plain text, Word, PowerPoint, HTML, email, etc. For this initial phase of the project, however, we have limited ourselves to the categorization of web pages only. As mentioned previously, web page categorization is a major research area due to the massive number of existing web pages and the astonishing rate at which new pages are being created. With such a large amount of data available, finding a specific piece of information is becoming difficult. As pointed out in [48], if web pages can be reliably categorized without human intervention, keyword searches and other techniques can then be applied to more limited categories instead of the web as a whole.

The BankSearch web page dataset was chosen as the document corpus. It was created by Mark Sinka and David Corne in 2002 in an effort to allow various researchers in document clustering and categorization to compare their results. The data is freely available online at http://www.pedal.reading.ac.uk/banksearchdataset. There are ten classes of documents, with 1000 documents per class. These documents are web pages that have been human-categorized as part of the Open Directory Project and Yahoo! Categories. A sample document is shown in Figure 3.1. Very small web pages were not included because the lack of content makes them inherently difficult

to classify. The dataset's creators point out that in this way the corpus is not a completely accurate representation of the current state of the world wide web [49]. The four major categories within the dataset – Banking and Finance, Programming Languages, Science, and Sport – each have two or three subcategories (Table 3.1). The dataset will therefore support experiments of varying levels of complexity. For example, separating two of the major categories such as Science and Sport is a relatively easy task, while categorizing closely related subclasses such as Java and C/C++ is more difficult. More information about the BankSearch dataset can be found in [49].

3.2 SYSTEM DESIGN

Our approach to the text classification problem is to use a standard nearest neighbor classifier (NNC) to categorize the documents. A preprocessing phase generates a potential feature set and prototypes for the NNC. These are then optimized by a genetic algorithm. A high level system architecture diagram is shown in Figure 3.2. While this is similar to other work mentioned in Chapter 2 [44], [45], [15], [41], little work has been done on generating a refined feature set and prototypes simultaneously. It is likely that the computational complexity of such an approach has limited its consideration. Of the work that has been done in this area ([14], [47]), the problem domains that were considered were very different from the text categorization attempted here. In particular, previous work attempted to optimize feature sets involving 30 or so potential features, while those under consideration here are an order of magnitude larger. The primary contribution of this thesis will therefore be to determine whether simultaneous evolution of the feature set and prototypes is a viable method to optimize a nearest neighbor classifier used for document classification.

Figure 3.1: Sample Web Document

3.2.1 Document Representation

Documents must be converted from free form text to feature vectors in order to be classified. There are many features that can be considered. Some of these are specific to web documents, such as the number and target of links on the page, the number and source filenames of images, or the number of user input fields. Because the eventual goal for this system is to be able to classify multiple documents types, it

Class	Specific Topic	General Topic
1	Commercial Bank	Banking and Finance
2	Building Societies	Banking and Finance
3	Insurance Agencies	Banking and Finance
4	Java	Programming Languages
5	C / C++	Programming Languages
6	Visual Basic	Programming Languages
7	Astronomy	Science
8	Biology	Science
9	Soccer	Sport
10	Motor Sport	Sport

Table 3.1: Coarse and fine-grained classes within the BankSearch web page dataset .

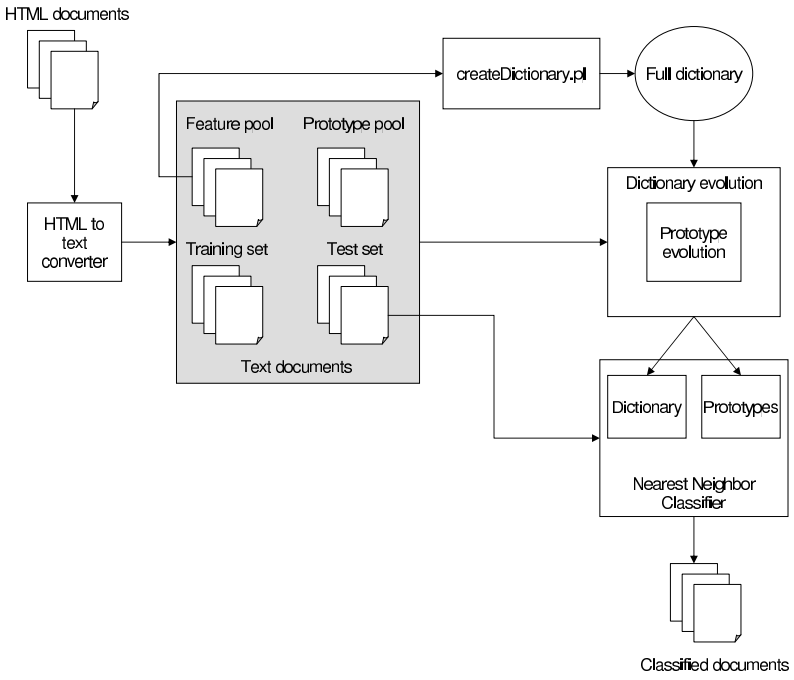

Figure 3.2: System Architecture

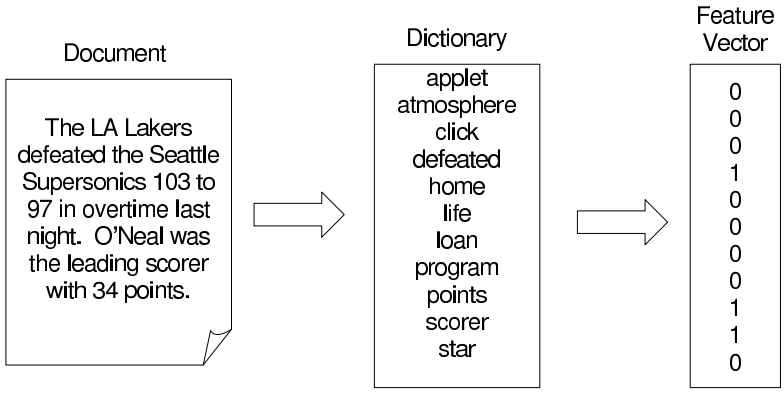

Figure 3.3: Documents are parsed with respect to a dictionary to create binary feature vectors. A 1 indicates the word appears in the document and a 0 that it does not.

does not make use of HTML-specific attributes. Other potential features are statistical properties of the entire text document. These may include characteristics such as the number of words in a document, the punctuation to character ratio, or the average sentence length. The most common feature set for document classification systems is various forms of word frequency. The primary methods are term frequency (TF), which is a simple count of the number of times each word occurs; document frequency (DF), which is term frequency normalized with respect to the total number of words in each document and summed or averaged over all of the documents; and term frequency x inverse document frequency (TFIDF), where the inverse document frequency term serves to emphasize words that occur in few documents and de-emphasize those that appear in many documents. IDF equals the logarithm of the total number of documents in the corpus divided by the number of documents that contain a given word. The simplest word-based feature set is a binary vector where a 1 indicates that document contains a word and a 0 indicates that it does not (see Figure 3.3).

The current system uses binary feature vectors. The speed of a nearest neighbor

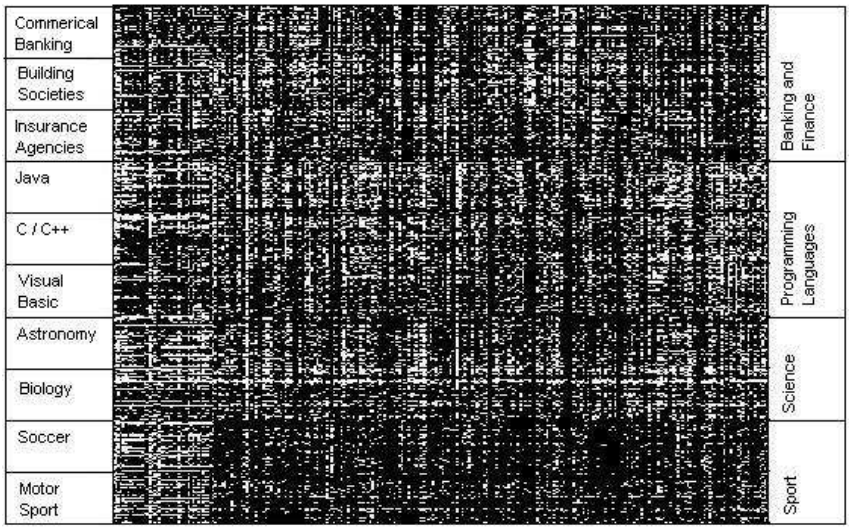

Figure 3.4: Binary feature vectors for 100 documents from each class.

classifier is chiefly dependent on the distance function used [15]. With a binary feature vector, manhattan distance can be used rather than euclidean distance and is therefore faster to compute. Other research done using this dataset has found boolean feature vectors to be highly effective at classification [48]. Figure 3.4 shows a subset of each class in the document corpus converted to a boolean feature vector. The x-axis represents a feature set (creation of this feature set is discussed in the next section). So each column shows which documents contain the word represented by that value of x, and each row represents the feature vector of an individual document. The NNC will need to be able to correctly differentiate among the horizontal bands in this plot. From inspection, it seems that categorizing the major classes will be a relatively easy task. The subclasses are harder to distinguish. Figure 3.5 shows the same information for the subclasses of the sports category.

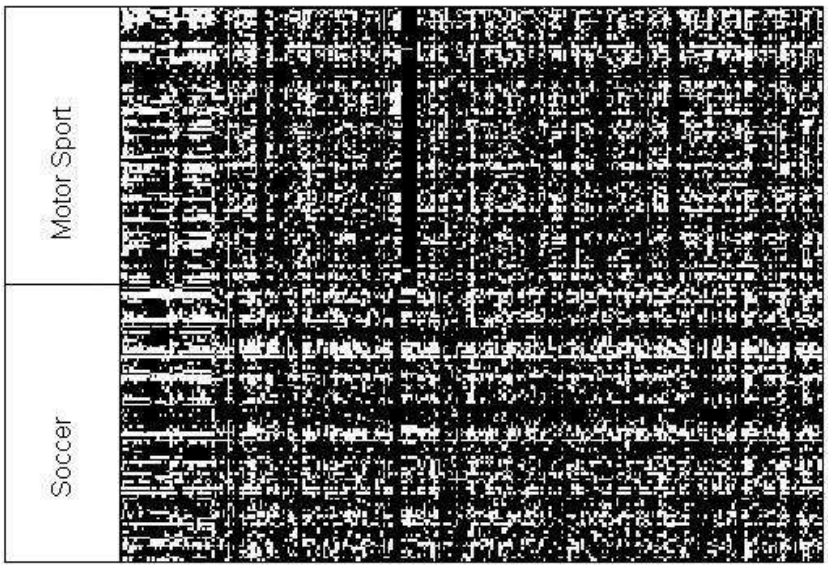

Figure 3.5: Binary feature vectors for 100 documents from each Science subclass.

3.2.2 Preprocessing

As mentioned previously, this system is meant to process multiple document types. The first step is therefore to convert all documents to the lowest common denominator: text. In the case of the web pages used for this project, this is done using a freeware tool called HTMLess, which can be downloaded at http://www.oz.net/sorth/media/htmls13d32.exe. This tool attempts to produce the text actually seen by someone viewing the page in a browser, and so it ignores things like comments and JavaScript.

The second step in the preprocessing phase is to generate the feature set that will be optimized by the genetic algorithm. In the document classification domain, the feature set is also known as a dictionary. The documents will be parsed with respect to this dictionary, as shown in Figure 3.3. While it is theoretically possible for the GA to start from a feature set containing every word that occurs in any of the

input documents, this would be prohibitively time-consuming as the system would be wasting time considering words such as "the" that obviously have little classification value. There are two primary approaches to extracting a feature set for a given group of documents: using either the term frequency or the distribution of a term over the documents [1]. The approach used in this system is to consider an averaged document frequency. Specifically, a Perl script takes as input a training set and generates the term frequency for each word in each document, normalized with respect to the length of the document. The script then computes the average of the normalized term frequency for each word over all documents in the training set. This has the effect of considering how important a word is in a particular document and how many documents contain that word, which are both indicative of the term's classification value. When the script was run for classes 1, 3, 7 and 9 of the BankSearch dataset, using fifteen documents randomly chosen from each class, 6648 distinct words were found. A graph of the average DF for these words is shown in Figure 3.6. As the graph indicates, the average document frequency is minimal for most words, but is extremely high for a small subset. Luhn, in some of the earliest work on automated text processing, suggested the need for upper and lower cutoff levels when choosing features. The upper cutoff eliminates extremely common words such as "a" and "the". The lower cutoff excludes words that appear very seldomly in a document (and therefore are less likely to be indicative of its subject) and words that appear in few documents in a set (and so are not representative of most members of the set). The exact values for these cutoffs need to be determined experimentally [50]. However, when we looked at the word list associated with the DF values, it became evident that some of the words with the highest average DF values would make very useful classifiers. So we opted to use the words with the top average DF values to create the initial dictionary instead of establishing explicit cutoff points. In order to

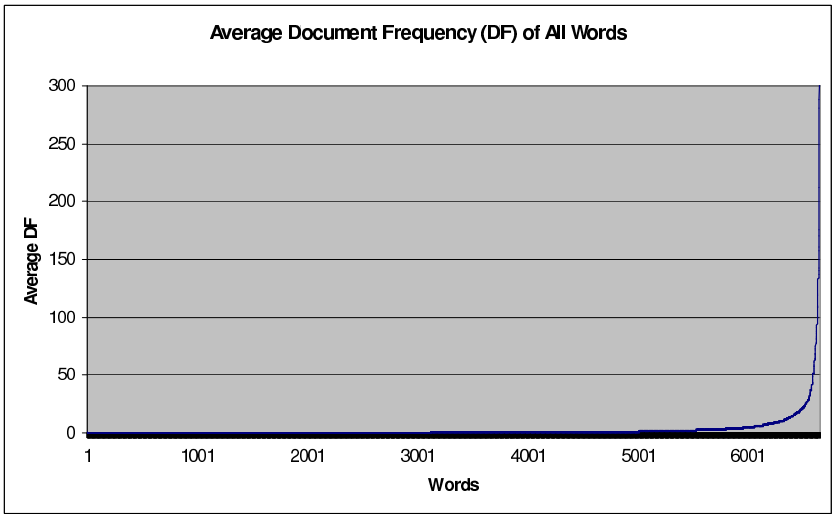

Figure 3.6: Average document frequencies (DF) over the training set.

keep very common words from being included, a stopwords file was used as a filter. There has been some work in creating stopwords lists specifically for web document classification [51], but we chose a more generalized approach. The stopwords file used by the current system consists of the 1000 most commonly used words in the English language. Figure 3.2 shows an initial dictionary created to attempt to classify groups 1, 4, 7, and 9 of the BankSearch dataset. This example contains 100 words. The Perl script can be set to generate an initial dictionary of any size. After experimentation, 100 words times the number of classes to be separated was chosen as a good initial size. van Rijsbergen suggests applying word stemming techniques (for instance, treating "differ", "differentiate", and "different" as a single word) to further conflate the dictionary at this point, but this approach was not implemented in the current system [1].

Word	Avg DF	Word	Avg DF	Word	Avg DF	Word	Avg DF
2002	0.07	supplier	0.07	internet	0.1	atltico	0.14
3131998	0.07	united	0.07	les	0.1	score	0.14
core	0.07	web	0.07	news	0.1	wireless	0.14
credit	0.07	supporters	0.08	null	0.1	conditions	0.15
cups	0.07	1999	0.08	physics	0.1	email	0.15
details	0.07	casimir	0.08	stars	0.1	financial	0.15
football	0.07	cole	0.08	v20	0.1	string	0.15
frente	0.07	current	0.08	were	0.1	lloyds	0.16
geovisit	0.07	plc	0.08	1st	0.11	products	0.16
his	0.07	scholes	0.08	2nd	0.11	stadium	0.16
insurance	0.07	stephen	0.08	borrowing	0.11	tsb	0.16
ireland	0.07	text	0.08	browser	0.11	euro	0.17
isa	0.07	topics	0.08	code	0.11	terms	0.17
lazio	0.07	tunisia	0.08	link	0.11	2000	0.19
limited	0.07	v30	0.08	saving	0.11	2001	0.19
madridss	0.07	wembley	0.08	shearer	0.11	documentwrite	0.2
mass	0.07	100	0.09	star	0.11	site	0.21
museum	0.07	access	0.09	applets	0.12	june	0.22
nav	0.07	astronomy	0.09	galaxy	0.12	applet	0.23
owen	0.07	everton	0.09	investing	0.12	league	0.32
photos	0.07	useful	0.09	cup	0.13	banking	0.33
pts	0.07	years	0.09	galaxies	0.13	services	0.33
registered	0.07	attendance	0.1	legal	0.13	java	0.35
senegal	0.07	click	0.1	online	0.13	var	0.36
sun	0.07	contact	0.1	universe	0.13	team	0.38

Table 3.2: Initial dictionary generated to classify groups 1, 4, 7, and 9

3.2.3 Genetic Algorithm

The options available when implementing the GA for a given problem must be viewed in light of their effects on the search process. In general, some choices will cause the GA to converge to a solution more quickly while others will result in it exploring the search space more thoroughly. The computational complexity of the nested GA approach necessitates a relatively small population size and number of generations, which limit the breadth and depth, respectively, to some extent.

The first step in the GA is to initialize the population. This is typically done randomly, although it is possible to bias the initial population towards a specific type of solution. For instance, because we are interested in finding small dictionaries and prototype sets that perform well we could bias the initial populations to contain more zeros than ones. However, our population size is limited and biasing the initial population towards one area of the search space may cause the algorithm to get stuck

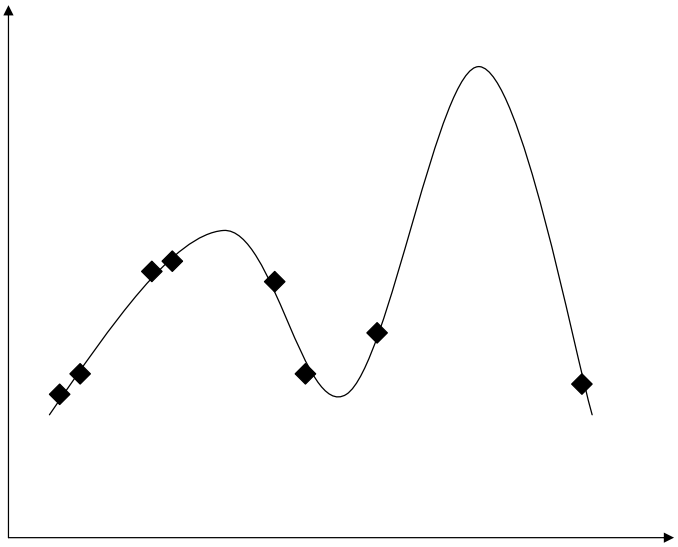

Figure 3.7: Biasing the initial population may cause the GA to converge to a local maximum.

in a local maximum instead of converging on a more optimal solution (see Figure 3.7). To avoid this situation, both populations were randomly initialized. This is trading off depth in the interest of breadth.

The next step is recombination. This is done via a crossover operator. The goal of this operator is to take the best groups of genes available in the parent population and combine them to create better-performing offspring. Consequently, crossover examines a given area of the search space in greater detail rather than exploring widely different solutions. For instance, say that we are trying to evolve an NNC to separate sports and science documents. One parent that performs well on this task uses the words "ball", "point" and "biology". Another good parent contains the words "win" and "cell". The crossover operation would hopefully be able to take advantage of this by creating a new individual that uses all five of these words. Because the GA only operates on the genotypes (bit strings) and has no knowledge of

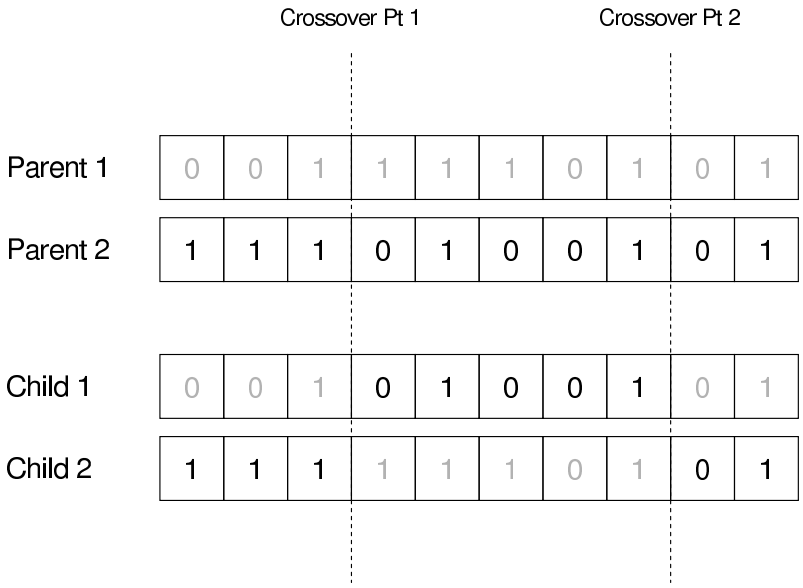

Figure 3.8: Two Point Crossover

the phenotypes (words/prototypes), the crossover operator randomly selects subsets of genes from the parents to pass on to the children. In two-point crossover, which is used for this system, this is done by randomly choosing two points along the length of the chromosome. The children are then created by alternating subsections from each parent (Figure 3.8).

Another issue in recombination is how to choose which individuals to use as parents. The goal in selection is to choose individuals with useful components to mate. One way to do this is to give individuals with a higher fitness a greater chance at reproducing. Various methods including proportional, ranked, and tournament selection are based on this principle. However, if the selective pressure is too great, the population will become saturated with a particular individual and its descendants and may converge prematurely. Selection is also an issue at the end of each generation, when the algorithm must determine which individuals will survive to the next

generation. The current system takes a balanced approach. All individuals have an equal chance of reproducing. This is the strategy with the least selective pressure and will insure a diverse population. Conversely, we use an elitist selection strategy to decide survival, exerting as much selective pressure as possible at this stage.

While crossover explores a confined area of the search space in greater depth, the mutation operator serves to prevent the algorithm from prematurely converging on a local maximum. This is done by randomly mutating individual genes within randomly selected individuals. The probability of any given gene being mutated is kept relatively low to avoid requiring a large number of generations for convergence. We used Bremerman's suggested value of $\frac{1}{n}$, where n is the number of bits used to encode an individual [52]. To avoid losing the best-performing solution, the individual with the highest fitness in the population is not mutated.

Our system is a wrapper method as described in Section 2.3.2 – each dictionary/prototype set combination is evaluated by using it in an NNC to categorize a training set of documents. This training set contains an equal number of representatives chosen randomly from each class. It is disjoint from the set of documents chosen as potential prototypes. The fitness function used is given by Equation 3.1. The parameter α controls the emphasis placed on accuracy versus size, and β determines which is more important to optimize – the number of words in the dictionary or the number of prototypes. Evaluation is by far the most expensive step in the GA.

$$\alpha * accuracy + (1 - \alpha) * (\beta * -\frac{numFeatures}{maxFeatures} + (1 - \beta) * -\frac{numPrototypes}{maxPrototypes}) \quad (3.1)$$

3.3 SYSTEM COMPLEXITY

One of the primary goals of this thesis is to test the idea that the features and prototype set for an NNC need to be evolved simultaneously. Specifically, should the system evolve the dictionary in an outer evolutionary process and evolve the prototype set with respect to each individual in the dictionary population in an inner evolutionary loop? We wish to determine if this approach produces better results than first using a GA to evolve the dictionary and then applying the GA again to optimize a prototype set with respect to that dictionary (or evolving the prototype set first and then the dictionary).

In the case of the dictionary, the algorithm is attempting to find a subset of the words generated by the Perl process during the preprocessing phase. The prototype set being optimized is a group of randomly chosen documents from each class. Both the dictionaries and prototype sets are represented as bit strings, where a 1 indicates that the corresponding item in the full set is included in this individual and a 0 that it is not. This means that the search space for the dictionary is 2^w, where w is the size of the full dictionary. Similarly, the search space for the prototype set contains 2^p items where p is the total number of potential prototypes pulled from all of the classes being categorized. The size of the combined search space is therefore 2^{w+p}. This is considerably larger than the $2^w + 2^p$ that would be the size if the features and prototypes were evolved sequentially.

The algorithm to simultaneously evolve the dictionary and prototype sets using a GA is shown in Figure 3.9. This figure summarizes the implementation decisions discussed in this chapter. Assuming the size of the dictionary and prototype populations are equal, and that both are evolved for the same number of generations, the complexity of this approach is of $O(s^2g^2)$ where s is the population size and g

is the number of generations. Clearly, evolving the dictionary and prototype set sequentially (shown in Figure 3.10 with the dictionary being evolved first) would be preferable. The complexity is then $O(2sg)$. It is possible, however, that the dictionary and prototype set are so intricately interdependent that they must be evolved simultaneously. We will attempt to determine this in Chapter 4.

It should be noted that the method proposed here for simultaneously evolving the features and prototypes differs from that described in [14]. In that work, a single GA was used instead of two nested evolutionary processes. A single chromosome contained genes specifying both features and prototypes. The size of the search space is the same in that case, but the computational complexity is reduced at the expense of custom-tailoring the prototype set to a particular dictionary.

```
initialize the dictionary population with a random distribution
  of ones and zeros

for (1:dictionary_epochs)
   double the dictionary population by randomly selecting
     parents and recombining using two-point crossover

   subject each gene in all but the best-performing dictionary
     to mutation with probability 1/size(dictionary)

   assign a fitness score based on size and accuracy to each
     dictionary in conjunction with its best prototype set by
     executing the following {

      for (1:dictionary_population_size)
         initialize the prototype population with a random
           distribution of ones and zeros

         for (1:prototype_epochs)
            double the prototype population by randomly selecting
              parents and recombining using two-point crossover

            subject each gene in all but the best-performing
              prototype set to mutation with probability
              1/number(prototypes)

            use the NNC with the current dictionary and prototype set
              to classify the training set and compute fitness based
              on accuracy

            reduce the prototype population by half by selecting the
              best-performing prototype sets
         end

         return the best-performing prototype set and its fitness
           score up to the dictionary->evaluate method
      end
   }

   reduce the dictionary population by half by selecting the
     best-performing dictionaries
end
```

Figure 3.9: Algorithm for Simultaneous Evolution of Features and Prototypes

```
initialize the dictionary population with a random distribution
  of ones and zeros

for (1:dictionary_epochs)
   double the dictionary population by randomly selecting
     parents and recombining using two-point crossover

   subject each gene in all but the best-performing dictionary
     to mutation with probability 1/size(dictionary)

   use the NNC with the initial prototype set and current dictionary
     set to classify the training set and compute fitness based
     on accuracy and size

   reduce the dictionary population by half by selecting the
     best-performing dictionaries
end

initialize the prototype population with a random distribution
  of ones and zeros

for (1:prototype_epochs)
   double the prototype population by randomly selecting
     parents and recombining using two-point crossover

   subject each gene in all but the best-performing
     prototype set to mutation with probability
     1/number(prototypes)

   use the NNC with the optimized dictionary and current prototype
     set to classify the training set and compute fitness based
     on accuracy and size

   reduce the prototype population by half by selecting the
     best-performing prototype sets
end
```

Figure 3.10: Algorithm for Sequential Evolution of Features and Prototypes

4 ANALYSIS

This analysis is meant to answer two questions: Does using a GA to optimize the dictionary and prototype set improve the accuracy of an NNC on text classification problems? and How does the performance differ when the dictionary and prototypes are evolved sequentially versus simultaneously?

4.1 NNC ACCURACY

In order to effectively judge the contribution made by the GA to the system, the accuracy of the NNC without any refinement of the dictionary and prototype set was first determined as a baseline for comparison. The system randomly chooses documents from each class to use for testing purposes, so this experiment was run several times to determine the variation produced by this random sampling.

The first run of this test attempts to classify documents from classes 1 and 9 of the dataset, which correspond to commercial bank and soccer. The second run looks at documents from classes 2 and 3, which are about building societies and insurance agencies. The prototypes are drawn from 50 documents of each class. Another 50 documents from each group are used to generate the dictionary, which contains 200 words. The test set is comprised of a disjoint set of 100 documents from each class. The NNC uses all 200 words and 100 prototypes when classifying the test set. Finally, all ten classes are categorized. In this last case, the dictionary contains 500 words.

Classes 1 and 9		*Classes 2 and 3*		*All Classes*	
Correct	Accuracy	Correct	Accuracy	Correct	Accuracy
177	.885	155	.775	535	.535
176	.880	165	.825	554	.554
168	.840	163	.815	574	.574
Mean	.868	Mean	.805	Mean	.554
StdDev	.025	StdDev	.026	StdDev	.020
RelError	.028	RelError	.033	RelError	.035

Table 4.1: Accuracy of the NNC with no evolution of dictionary or prototype set.

The results are shown in Table 4.1.

On average, the NNC classified the diverse documents with 87% accuracy and the similar documents with 81% accuracy. The accuracy on the full category set averaged 55%. It is somewhat surprising that the NNC performed as well as it did in these tests. This is an indication that the strategy used to create the initial dictionary is at least marginally effective. These values will serve as a baseline when various optimizations are made later.

The standard deviation was relatively small, and 3 trials were sufficient to bring the relative error to around 3%. The other experiments in this work will therefore be averaged over 3 trials to smooth the variations in results caused by randomly choosing the test and training sets.

4.2 DICTIONARY EVOLUTION

In order to determine the efficacy of using a GA to evolve the dictionary used by the NNC, the prototype set was fixed. In this case, it consisted of a randomly chosen set of 50 documents from each class. The dictionary was then evolved using the system setup described in Chapter 3. The specific parameters used are given in Table 4.2.

The results of this experiment on seperating classes 1 and 9 are shown in Figure 4.1. Over three runs, the system achieved an average accuracy of 93% on the test set. This is a 7% improvement over applying the NNC using the full dictionary. In

Training Size	50/class
Test Size	100/class
Generations	30
Population Size	10
α	.5
β	.5

Table 4.2: GA Parameters

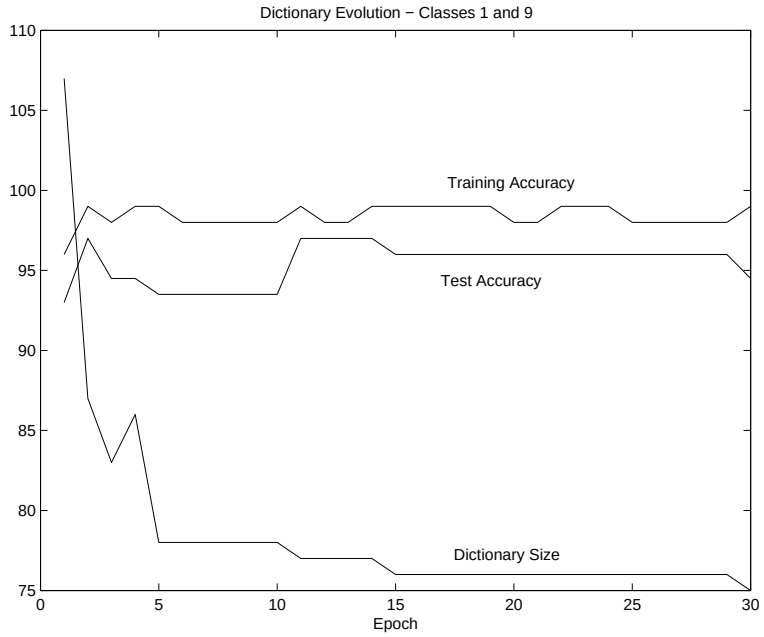

Figure 4.1: Dictionary Evolution - Classes 1 and 9

addition, the GA reduced the size of the dictionary by an average of 60%, from 200 to 81 words. Furthermore, the GA neared its best results after just 15 generations. The ability to quickly find good solutions is important considering the computational cost of this approach.

The same experiment was then done to classify groups 2 and 3 of the dataset. In this case, the evolved dictionary did not perform better than the initial one. The average accuracy was 81%, the same as the unoptimized NNC. The dictionary was again reduced by 60% to 81 words. An example of the evolution is shown in Figure 4.2. Comparing figures 4.1 and 4.2, it is evident that the system had a more difficult time dealing with the more similar documents. The dictionary size nearly monotonically

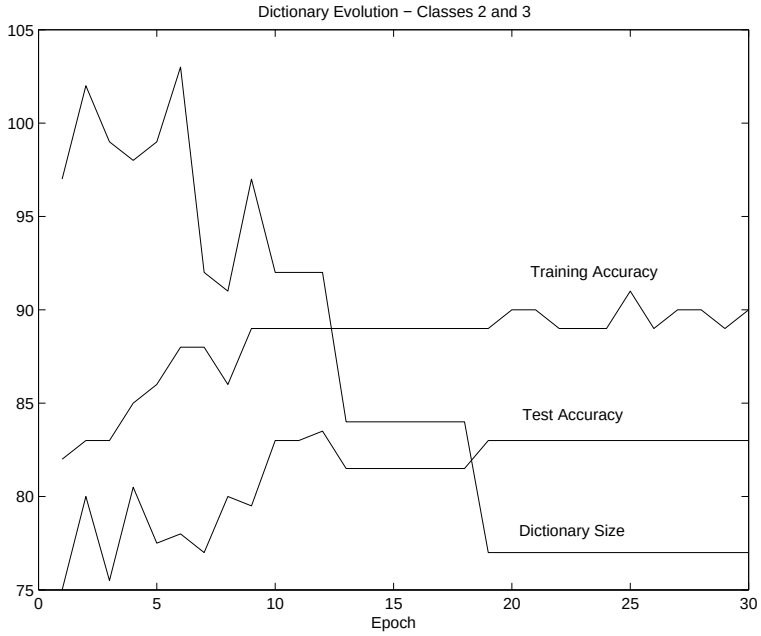

Figure 4.2: Dictionary Evolution - Classes 2 and 3

decreases in the first case, while the system makes more accuracy/size tradeoffs when categorizing the similar documents. Still, the GA converges after 19 generations. This test was run again with the parameter α set to 1 to allow the system to achieve the best possible accuracy on the dataset, regardless of the dictionary size required. The system then chose a dictionary containing 96 words, but this surprisingly did not impact the accuracy. This may be a limitation of the method used to create the initial dicitonary, or it may simply be a peformance limitation of the NNC on this problem domain.

Finally, this experiment was run again to attempt to categorize all ten classes of the dataset. The initial dictionary consisted of 500 words. The resulting system's accuracy was 59% (a 7% improvement) with a dictionary of 236 words. The system reached this state in about 20 generations.

It has been suggested by [41] that random search can sometimes perform comparably to a GA in feature selection. To determine if the genetic operators were helpful

in generating good solutions, the crossover, mutation and selection operators were commented out and the population was randomly reinitialized every generation. The only memory between generations was the retention of the best-performing individual. Using this setup, classes 1 and 9 were categorized with 90% accuracy and classes 2 and 3 with 82%. The dictionary sizes were 87 and 88, respectively. While more experimentation may need to be done, we preliminarily conclude that random search does perform surprisingly well on this feature selection problem. While the feature set produced were less optimal (larger) on average, the accuracy was comparable to the GA for the dissimilar documents. This seems to be because the potential features fall into groups, where each word in the group provides roughly the same utility when classifying documents. Choosing more than one word from each group is redundant. The fitness function of the GA contains a size penalty that allows it to focus the search on solutions that contain fewer redundant features. When considering the similar documents, the difference between random search and the GA was greater. This is possibly because the groups of equivalent words are smaller in this case, and there may be fewer good feature combinations to find. The GA's ability to capitalize on useful subsets of features is therefore more of an advantage in this situation.

Table 4.3 is a subset of the initial dictionary considered by the GA when attempting to categorize documents from classes 1 and 9 (Commercial Banking and Soccer). The numbers under the class headings indicate how many documents from the test set contained each word. For instance, 19 of the 100 documents on commercial banking from the test set contained the word *advice*, while only one page on soccer contained that word. These are meant to give a coarse indication of the value of each word as a classifier. The words in bold are those included in the dictionary evolved by the GA. The majority of the chosen words are obviously useful classifiers and would likely have been selected by a human if they were hand-generating a dictionary. These

Word	Class 1	Class 9	Word	Class 1	Class 9	Word	Class 1	Class 9
100	25	12	**fans**	0	19	**payment**	31	1
1998	7	14	**fcm**	0	4	**pensions**	17	1
2000	12	17	fees	14	0	players	0	34
2002	19	45	financial	44	3	**ploiesti**	0	5
access	32	5	fixture	0	7	**poland**	2	12
acorns	0	0	following	23	15	**property**	16	0
advice	19	1	**france**	10	25	provided	22	10
aston	0	26	**further**	25	5	**rates**	46	1
attendance	1	6	**georgia**	0	4	**repayment**	19	0
average	3	1	**germany**	8	28	request	27	1
balance	18	2	goals	1	19	rights	8	20
banks	24	1	greece	8	11	russia	1	8
belgium	5	8	hsbc	7	0	**savings**	47	1
birmingham	4	16	insurance	38	4	**scorers**	0	19
borrowing	18	0	investing	19	0	scottish	5	10
brazil	2	14	**investments**	26	0	security	43	0
bucuresti	0	11	**italy**	7	27	sheffield	0	10
card	31	5	jul	0	6	soccer	0	23
cash	28	9	**latest**	7	23	**stadium**	0	16
charges	29	1	leeds	2	32	**statistics**	0	14
clubs	2	28	limited	23	6	subs	0	3
cole	0	9	links	15	15	teamgoals	0	4
conditions	36	9	loan	29	6	**telephone**	37	1
contract	44	22	**location**	4	7	travel	11	4
corporate	18	1	made	26	18	tsb	12	0
credit	48	1	**match**	1	33	uefa	0	26
cup	0	63	**moldova**	0	3	valuation	9	1
current	38	10	mortgage	28	0	**variable**	21	0
cwc	0	4	**nav**	4	0	visa	9	0
derby	0	21	**netscape**	14	9	**wednesday**	0	11
dinamo	0	11	**news**	16	40	**were**	12	18
email	17	15	**northern**	15	10	**written**	21	2
euro	11	17	**online**	31	19			
explorer	12	2	owen	0	8			

Table 4.3: Evolved Dictionary - Classes 1 and 9

include soccer terms and team locations such as *brazil*, *germany*, *stadium* and *match* and banking vocabulary including *credit*, *savings*, and *repayment*. A few words, such as *wednesday* appear to be useful classifiers but are unlikely to have been considered by a human because they are not strongly associated with the subject in our minds, even though they may appear frequently in documents on the subject. Some words that survived the evolutionary process are poor classifiers. Examples include *acorns* and *were*. It is possible that running the GA for more generations or with a larger population size could reduce the occurence of this. Alternatively, the fitness function parameters could be adjusted to introduce a larger penalty for dictionary size.

4.3 PROTOTYPE EVOLUTION

In order to determine the utility of evolving the prototype set, the dictionary was fixed at its initial state and the prototype set was evolved using the parameters in Table 4.2. Each test was again run three times.

When classifying groups 1 and 9, the system achieved an average accuracy of 95% on the test data using 45 prototypes out of the possible 100. This is a 9% improvement over the 87% accuracy of the unoptimized NNC, with a 55% reduction in the number of prototypes. A graph of the evolutionary process is shown in Figure 4.3. On groups 2 and 3, the average accuracy was 82% using 49 prototypes (Figure 4.4). This is roughly the same performance as the plain NNC, but uses half the prototypes. To determine the best possible accuracy in this case, the fitness function parameter α was set to 1 to avoid penalizing the system for the number of prototypes used. This improved the accuracy of groups 2 and 3 to 88%, without significantly increasing the number of prototypes used (Figure 4.5). Interestingly, this fitness function slightly lowered the accuracy on classes 1 and 9, to 92%. We speculate that the initial dictionary is not sufficient to well separate classes 2 and 3 in the feature space, and therefore more prototypes are required than in the case of classes 1 and 9. Removing the size penalty allows the GA to consider solutions involving more prototypes, which can drive accuracy higher even when the final solution does not need to contain an excessive number of prototypes. When this system configuration was used to categorize all of the groups in the dataset, the average accuracy was 57% using 243 of 500 potential prototypes.

When the crossover, mutation and selection operators were commented out and the system performed a random search, the accuracy was 90% on classes 1 and 9 using 51 prototypes and 80% with 52 prototypes on groups 2 and 3. These results

are slightly more accurate than the unoptimized NNC and use considerably fewer prototypes. Compared to the GA, the accuracy of the best randomly generated solution was noticeably lower (5% lower for classes 1 and 9 and 9% for classes 2 and 3). The number of prototypes chosen was about the same in both cases. Unlike the feature selection case, the document distribution in the feature space seems to limit the combinations of prototypes that are capable of providing good results. The crossover operator is particulary important in this case because it allows useful subsets of prototypes to be combined in offspring.

Looking more closely at Figures 4.3 and 4.5 reveals that the system converged to a prototype set within 15 generations. The increases in accuracy achieved by evolving the prototype set were comparable to those attained by evolving the dictionary. Prototype evolution was complicated more by the similarity of the documents being considered than was dictionary evolution, but this can be accomodated by adjusting the fitness function parameters α and β higher when the classification task involves more similar documents. This will cause the system to work to achieve more accurate solutions at the expense of size and will penalize larger dictionaries more than larger prototype sets.

Some characteristics of the evolved prototype sets are interesting. Tables 4.4 through 4.6 show which documents were chosen as prototypes for each trial. To keep the number of prototypes small enough to analyze easily, the initial prototype set contained 15 documents from each class for this experiment (instead of the 50 used previously), and the test set contained 30 documents of each type. The documents are numbered according to their class, so 1 through 1000 are class 1, 1001 through 2000 are class 2, and so on. In each case, the prototype set contains a roughly equal number of examples from each class being considered. The utilized column shows how many times a given prototype was used to make a classification, and the accuracy

column indicates how often that classification was correct. Ideally, the prototype set would contain a small number of frequently-used prototypes that are reasonably accurate. If the feature space is such that there are several pockets of outliers among the documents to be classified, several prototypes may need to be included in the set in order to properly categorize these outliers. These prototypes would be utilized less often, but that would be acceptable if they were highly accurate. In Table 4.4, which illustrates classification of two groups of very dissimilar documents, the prototypes chosen display these traits. Documents 758, 550, 8668, 8452, and 8391 are used to make the majority of the classifications. Documents 550 and 8668 lead to some errors, but on the whole these prototypes are doing more good than harm. Documents 86, 578, and 8665 are only used a handful of times, but are perfectly accurate. In the other two trials, which involve more similar documents, the evolved prototype sets are not as coherent. For instance, in Table 4.5 document 2347 is used to make one classification, and it was incorrect. Document 2309 is included in the prototype set despite not being used. These types of cases occur even more frequently in Table 4.6. It is clear that these prototypes cannot be helping the NNC. The GA could be "encouraged" to remove them by increasing the fitness penalty for the number of prototypes used. Alternatively, it could be forced to remove them by adding a specific check on the utilization vs accuracy of the prototypes in the fitness function. Because the overall classification accuracy was reasonable, neither of these alternatives was implemented in the current system.

4.4 SIMULTANEOUS VS SEQUENTIAL EVOLUTION

We now turn our attention to comparing the results of evolving the prototype set and dictionary sequentially versus simultaneously. As discussed in section 3.3, sequential evolution is clearly more computationally efficient. However, it is possible that the

Document	Utilized	Accuracy
86	1	1.0000
578	1	1.0000
758	19	1.0000
550	8	.8750
8668	7	.7143
8452	6	1.0000
8665	2	1.0000
8391	16	1.0000

Table 4.4: Evolved Prototype Set - Classes 1 and 9

Document	Utilized	Accuracy
1901	5	1.0000
1487	5	1.0000
1455	2	.5000
1369	3	1.0000
1038	2	1.0000
1540	3	.6667
2435	2	1.0000
2897	1	1.0000
2310	28	.6071
2141	6	1.0000
2309	0	0
2267	2	1.0000
2347	1	0

Table 4.5: Evolved Prototype Set - Classes 2 and 3

features and prototypes are so interrelated that performance will suffer if they are not evolved simultaneously. To determine the answer to this question, the system was configured in three different ways and used to categorize classes 1 and 9 and classes 2 and 3. In the interest of time, we did not attempt to categorize all of the documents for these tests. The parameters for the GA are shown in Table 4.7. In particular, the number of generations was reduced from 30 to 15 due to the additional computation time required to evolve both the features and prototypes, particularly for the nested approach. The results given in sections 4.2 and 4.3 indicate that the GA is able to produce good results within this time period, however, so this should not be a problem.

In the first configuration, the dictionary was evolved first using the entire prototype set. Then the best dictionary was chosen and held constant while the prototype set was evolved. The general algorithm is given in Figure 3.10. The results are shown

Document	Utilized	Accuracy	Document	Utilized	Accuracy	Document	Utilized	Accuracy
187	31	.3871	4950	0	0	7525	6	0
278	3	.3333	4954	0	0	7767	3	1.0000
944	1	1.0000	4692	1	0	7854	1	1.0000
712	1	1.0000	4332	1	1.0000	7375	2	.5000
866	1	0	4881	0	0	7878	2	1.0000
290	0	0	4940	0	0	7268	0	0
249	0	0	4579	2	1.0000	7313	3	1.0000
1888	4	.5000	4658	0	0	8266	5	1.0000
1165	2	1.0000	5738	1	0	8002	1	0
1510	0	0	5765	6	.5000	8389	15	.1333
1378	16	.1875	5724	4	.7500	9000	1	1.0000
1924	2	1.0000	5429	0	0	8742	1	1.0000
1680	4	.7500	5753	3	1.0000	8869	1	1.0000
1593	0	0	5065	0	0	8796	2	1.0000
1445	1	0	5565	3	.3333	8849	3	1.0000
1823	0	0	5458	12	.4167	8571	4	1.0000
2857	4	0	5064	0	0	8327	4	.2500
2394	3	.6667	6840	0	0	8557	7	0
2982	5	.4000	6430	0	0	9057	1	1.0000
2581	1	1.0000	6076	14	.5714	9691	1	1.0000
2614	1	1.0000	6553	1	1.0000	9603	66	.2727
2160	2	1.0000	6960	2	1.0000	9267	0	0
3117	0	0	6767	1	1.0000	9813	0	0
3564	4	1.0000	6368	1	0	9616	10	.3000
3767	3	.6667	6841	0	0	9165	13	.2308
3461	1	1.0000	6948	1	0	9391	0	0
3616	0	0	6648	4	.5000			
3505	1	1.0000	7818	0	0			

Table 4.6: Evolved Prototype Set - All Classes

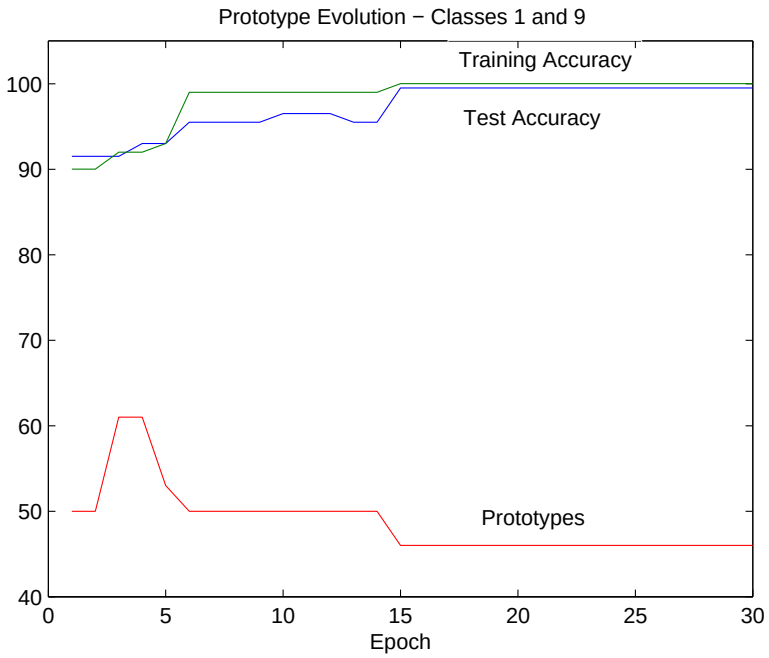

Figure 4.3: Prototype Evolution - Classes 1 and 9

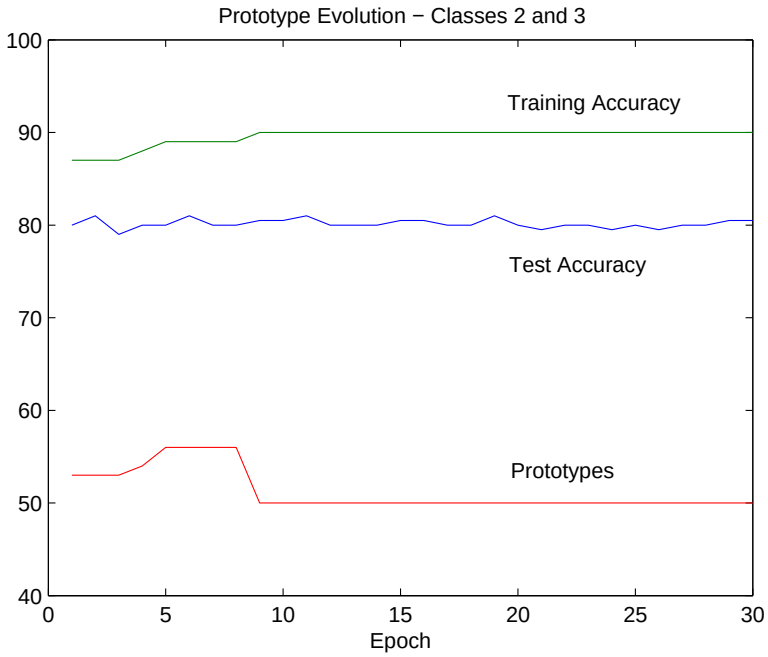

Figure 4.4: Prototype Evolution - Classes 2 and 3

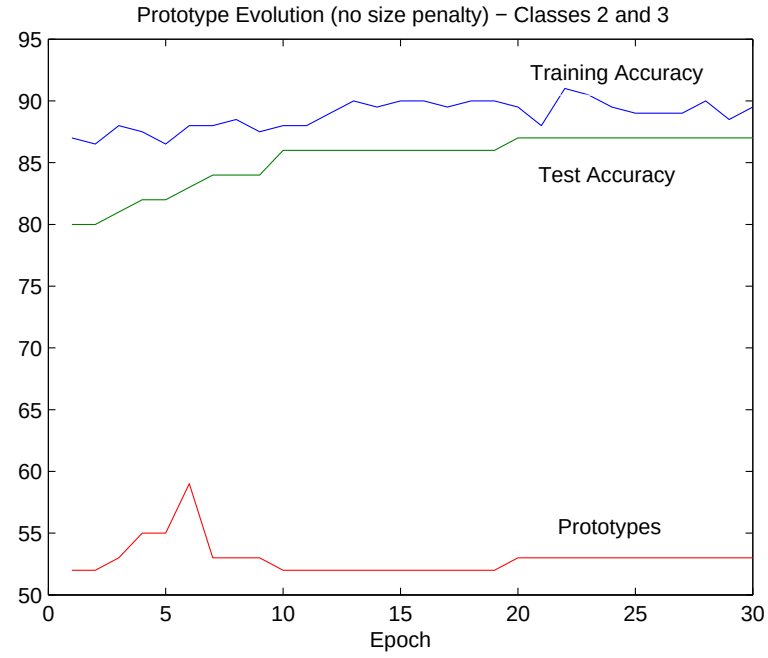

Figure 4.5: Prototype Evolution (no size penalty) - Classes 2 and 3

Training Size	50/class
Test Size	100/class
Dictionary Generations	15
Prototype Generations	15
Dictionary Population Size	10
Prototype Population Size	10
α	.5
β	.5

Table 4.7: GA Parameters

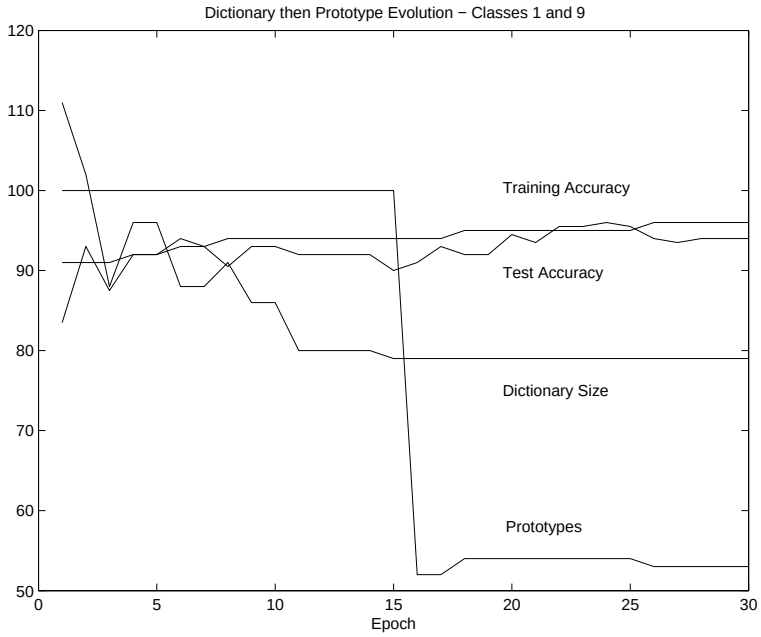

Figure 4.6: Dictionary then Prototype Evolution - Classes 1 and 9

in Figures 4.6 and 4.7. On classes 1 and 9, the system achieved 94% accuracy using 52 out of 100 possible prototypes and 82 out of 200 potential features. For classes 2 and 3, these figures were 87%, 53 prototypes, and 88 features.

The system was then modified to use the initial dictionary to evolve the prototype set first, and then use the chosen prototypes to optimize the dictionary. Figures 4.8 and 4.9 show the evolutionary process. Classes 1 and 9 were categorized correctly 93% of the time using 48 prototypes and 84 features. The accuracy was 85% on classes 2 and 3 with 50 prototypes and 81 words in the dictionary.

Finally, the system was set up to evolve the dictionary in an outer evolutionary

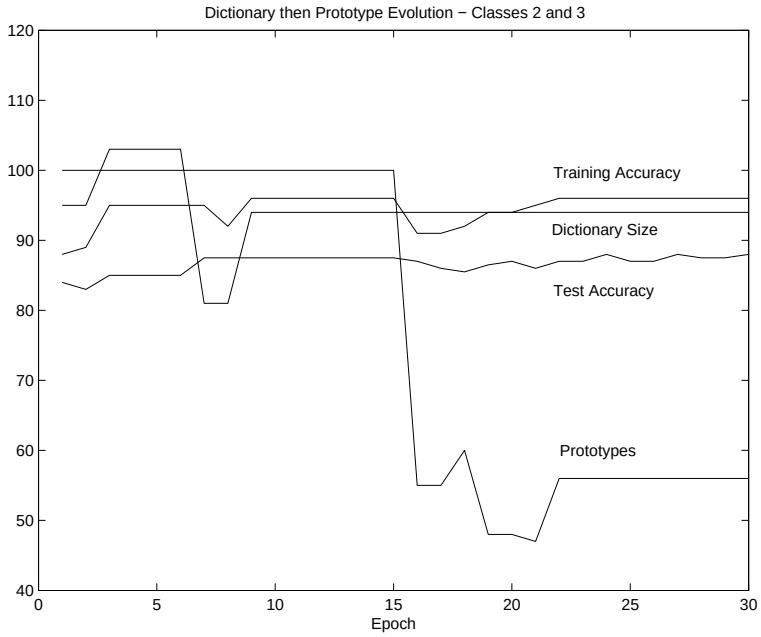

Figure 4.7: Dictionary then Prototype Evolution - Classes 2 and 3

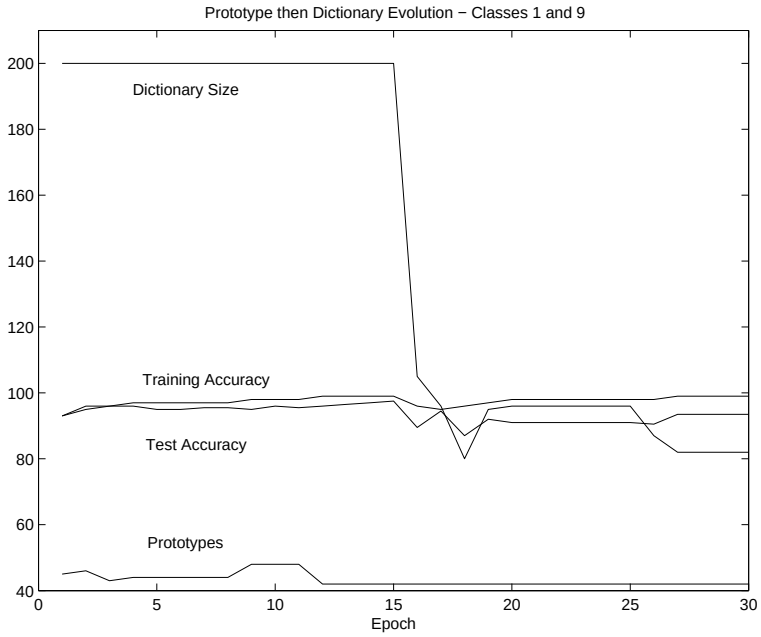

Figure 4.8: Prototype then Dictionary Evolution - Classes 1 and 9

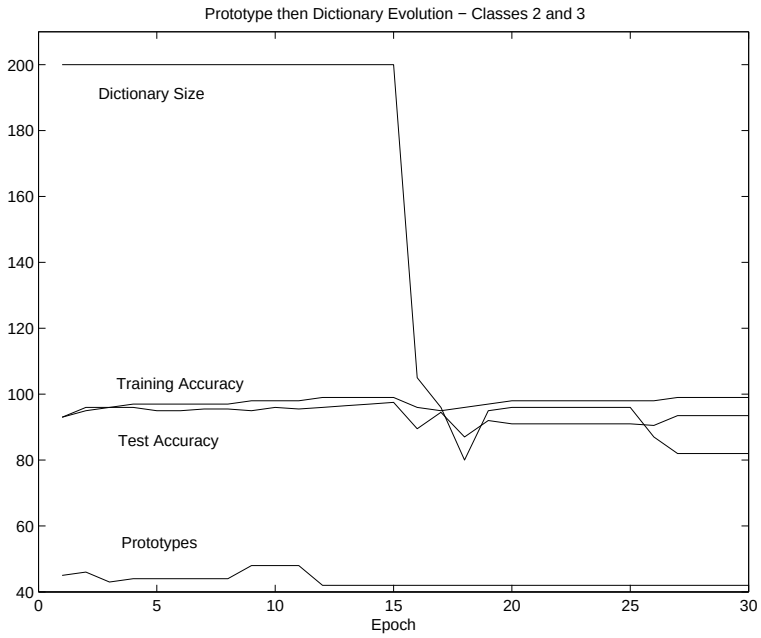

Figure 4.9: Prototype then Dictionary Evolution - Classes 2 and 3

process. For each generation in the dictionary evolution, a prototype set is evolved for each member of the dictionary population using another GA. The high-level algorithm is shown in Figure 3.9. The results on classes 1 and 9 were 94% accuracy, 46 prototypes, and 86 features. For classes 2 and 3, the system used 49 prototypes and 82 features to achieve 86% accuracy. Figures 4.10 and 4.11 show the evolutionary process.

In each configuration, the accuracy of the evolved system was roughly the same as evolving the features or prototypes alone. This represents a significant improvement over the unoptimized NNC. Furthermore, evolving both the dictionary and prototype sets resulted in about the same number of words and prototypes as when either was optimized individually. Most importantly, the nested genetic algorithm did not perform better than sequential evolution. This is encouraging as it allows us to both improve performance and decrease the computation time and memory requirements of the NNC using the much faster sequential algorithm. It is likely that the extremely

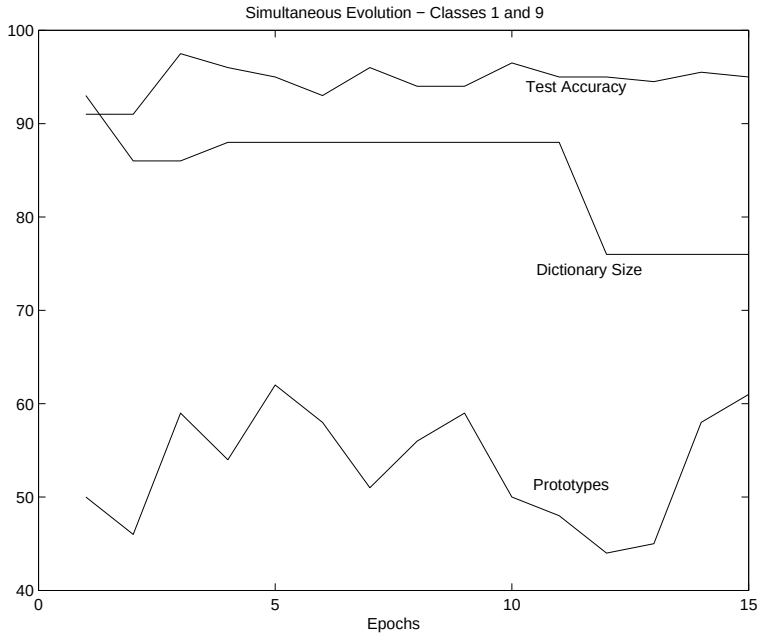

Figure 4.10: Simultaneous Evolution - Classes 1 and 9

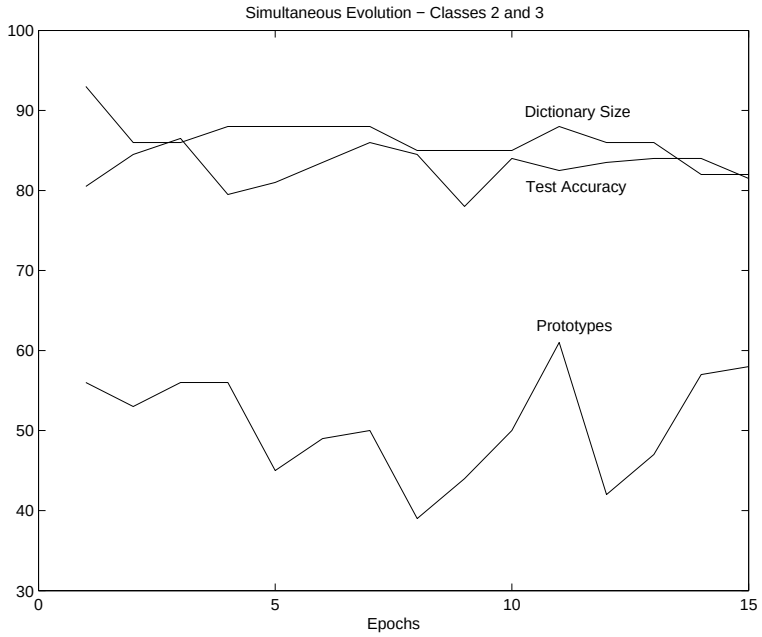

Figure 4.11: Simultaneous Evolution - Classes 2 and 3

high dimensionality of the feature space in text classification domains is the reason the feature and prototype sets are not so interrelated that they must be evolved simultaneously. The large number of features greatly increases the likelihood that documents will be more nearly linearly separable, which simplifies the process of choosing prototypes. Our results indicate that it does not matter which is optimized first - the features or the prototypes. More work needs to be done in order to generalize these results beyond the particular document corpus studied here, however. This is discussed in more detail in the following chapter.

5 CONCLUSION

The results of the experiments described in the previous chapter are summarized in Table 5.1.

More research needs to be done to analyze the performance of other types of classifiers on this particular document corpus and on other text classification problems so that relative comparisons can be made. However, the results presented in Chapter 4 do indicate that the nearest neighbor classifier performs reasonably well in absolute terms in this domain. While the accuracy falls off considerably when more categories are considered, the system's classification may be useful in optimizing searches for some domains, particularly web portals where the number of topics involved is nat-

Test	Documents	Accuracy	Prototypes	Features
Unoptimized NNC	Classes 1 and 9	87	100	200
	Classes 2 and 3	81	100	200
	All Classes	55	500	500
Dictionary Evolution	Classes 1 and 9	93	100	81
	Classes 2 and 3	81	100	81
	All Classes	59	500	236
Randomized Dictionary	Classes 1 and 9	90	100	87
	Classes 2 and 3	82	100	88
Prototype Evolution	Classes 1 and 9	95	45	200
	Classes 2 and 3	88	49	200
	All Classes	57	243	500
Randomized Prototypes	Classes 1 and 9	90	51	200
	Classes 2 and 3	80	52	200
Dictionary then Prototypes	Classes 1 and 9	94	52	82
	Classes 2 and 3	87	53	88
Prototypes then Dictionary	Classes 1 and 9	93	48	84
	Classes 2 and 3	85	50	81
Simultaneous Evolution	Classes 1 and 9	94	46	86
	Classes 2 and 3	86	49	82

Table 5.1: Summary of Results

urally limited. If a user is looking for documents similar to one they already have, this system can assist by directing searches to a particular category first. The search of this smaller space will be faster, and the documents are likely to be more relevant. If the desired information is not found by searching that group of documents, the search could then be widened to consider all documents.

The level of accuracy achieved by the NNC in this study was not high enough for some applications, however. For instance, if we are attempting to sort spam from legitimate email or automatically assign a grade to an SAT essay, the consequences of a misclassification are serious. Using larger training sets may help to improve accuracy. It is also possible that choosing a different method to generate the initial dictionary or using another classifier in combination with, or instead of, the NNC would enhance performance.

We have found genetic algorithms to be a useful method for optimizing the feature and prototype sets. Several studies in this area (using GA and other techniques) have reported that it was possible to reduce the feature set and number of prototypes while maintaining performance, but that these optimizations did not increase accuracy [44], [45], [14]. Our results contradict this. They indicate that optimizing either the dictionary or the prototype set can increase accuracy, particularly in the easiest case of two very distinct subject areas. This may be because the stopwords list was not completely effective in removing words with no correlation to the correct class. These words could cause misclassifications in the unoptimized NNC, but are very likely to be removed by the GA. More experimentation is required to confirm this hypothesis. While random selection of the dictionary and prototypes did produce reasonable results, the ability of the GA to exploit useful subsets of features via the crossover operator proved useful in both cases. It can not be said that optimizing the dictionary is more important than reducing the prototype set or vice versa – the GA

reduced both by about the same percentage, and the accuracy went up to about the same level in both cases.

As mentioned in Chapter 2, the computation time and memory requirements of an NNC scale with the product of the number of features and prototypes. In the studies cited, the number of features and prototypes was in the dozens. The savings from optimizations are therefore limited in these cases. However, in the area of text classification the number of features can range from hundreds to thousands. In this case, any reduction of the feature or prototype sets translates into a large increase in computational performance. This was the case when both the dictionary and prototype sets were optimized. Reducing both sets did not translate into a further increase in performance. However, when the features and prototypes were both evolved, each one could be reduced by roughly the same amount as when only it was evolved. Furthermore, it did not prove necessary to optimize the features and prototypes simultaneously to achieve the best results.

BIBLIOGRAPHY

[1] C. van Rijsbergen, *Information Retrieval: Second Edition.* Butterworth and Company, 1979.

[2] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval.* ACM Press / Addison-Wesley, 1999.

[3] J. Hartigan, *Clustering Algorithms.* John Wiley & Sons, 1975.

[4] P. Berkhin, "Survey of clustering data mining techniques," tech. rep., Accrue Software, San Jose, CA, 2002.

[5] T. Kohonen, S. Kaski, K. Lagus, J. Salojarvi, J. Honkela, V. Paatero, and A. Saarela, "Self organization of a massive document collection," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 574–585, 2000.

[6] V. Vapnik, *The nature of statistical learning theory.* Springer-Verlag, 1995.

[7] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *Proceedings of the Special Interest Group on Information Retrieval*, pp. 42–49, 1999.

[8] Y. Yang and C. Chute, "An example-based mapping method for text categorization and retrieval," *ACM Transactions on Information Systems*, vol. 12, no. 3, pp. 252–277, 1994.

[9] J. Graham-Cumming, "Naive bayesian text classification," *Dr. Dobb's Journal*, vol. 372, pp. 16–20, May 2005.

[10] S. Chakrabarti, S. Roy, and M. Soundalgekar, "Fast and accurate text classification via multiple linear discriminant projections," in *Proceedings of the 28th Very Large DataBase Conference*, 2002.

[11] M. Ruiz and P. Srinivasan, "Automatic text categorization using neural networks," in *Proceedings of the 8th ASIS/SIGCR Workshop on Classification Research*, pp. 59–72, 1997.

[12] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[13] P. Langley and W. Iba, "Average-case analysis of a nearest neighbor algorithm," in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 889–894, 1993.

[14] L. Kuncheva and L. Jain, "Nearest neighbor classifier: Simultaneous editing and feature selection," *Pattern Recognition Letters*, no. 20, pp. 1149–1156, 1999.

[15] J. Kangas, "Prototype search for a nearest neighbor classifier by a genetic algorithm," *Proceedings of the Third International Conference on Computational Intelligence and Multimedia Applications*, pp. 117–121, 1999.

[16] H. Yau and M. Manry, "Iterative improvement of a nearest neighbor classifier," *Neural Networks*, vol. 4, no. 4, pp. 517–524, 1991.

[17] E. Fix and J. Hodges, "Discriminatory analysis, non-parametric discrimination," tech. rep., USAF School of Aviation Medicine, 1951.

[18] Q. Liang, "MPEG VBR video traffic classification using bayesian and nearest neighbor classifiers," in *Proceedings of the International Symposium on Circuits and Systems*, pp. 77–80, 2002.

[19] D. Lubensky, "Word recognition using neural nets, multi-state gaussian and k-nearest neighbor classifiers," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 141–144, 1991.

[20] A. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245–271, 1997.

[21] R. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 1, no. 2, pp. 273–324, 1997.

[22] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[23] M. Brown and N. Costen, "Non-linear feature selection for classification," in *Proceedings of the British Machine Vision Conference*, 2004.

[24] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of Machine Learning Research*, vol. 5, pp. 1205–1224, 2004.

[25] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, pp. 131–156, 1997.

[26] I. Kononenko, "Estimating attributes: Analysis of extensions of relief," in *Proceedings of the European Conference on Machine Learning*, pp. 171–182, 1994.

[27] H. Almuallim and T. Dietterich, "Learning with many irrelevant features," in *Proceedings of the 9th National Conference on Artificial Intelligence*, pp. 547–552, 1991.

[28] W. Siedlecki and J. Sklansky, "On automatic feature selection," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 2, pp. 197–220, 1988.

[29] D. Zongker and A. Jain, "Algorithms for feature selection: an evaluation," in *Proceedings of the International Conference on Pattern Recognition*, pp. 18–22, August 1996.

[30] J. Bezdek and L. Kuncheva, "Nearest prototype classifier designs: An experimental study," *International Journal of Intelligent Systems*, vol. 16, pp. 1445–1473, 2001.

[31] P. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Infomation Theory*, vol. 14, pp. 515–516, 1968.

[32] D. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-2, pp. 408–421, 1972.

[33] T. Kohonen, "Improved versions of learning vector quantization," in *Proceedings of the International Joint Conference on Neural Networks*, pp. 545–550, 1990.

[34] S. Geva and J. Sitte, "Adaptive nearest neighbor pattern classification," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 318–322, 1991.

[35] C. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Transactions on Computers*, vol. C-23, no. 11, pp. 1179–1184, 1974.

[36] Y. Huang, K. Liu, C. Suen, A. Shie, I. Shyu, M. Liang, R. Tsay, and P. Huang, "A simulated annealing approach to construct optimized prototypes for nearest-neighbor classification," in *Proceedings of the International Conference on Pattern Recognition*, pp. 483–487, 1996.

[37] H. Yan, "A two-layer perceptron for nearest neighbor classifier optimization," in *Proceedings of the International Joint Conference on Neural Networks*, pp. III–585–III–590, June 1992.

[38] Y. Hamamoto, S. Uchimura, and S. Tomita, "A bootstrap technique for nearest neighbor classifier design," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 1, pp. 73–79, 1997.

[39] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[40] C. Liu and M. Nakagawa, "Prototype learning algorithms for nearest neighbor classifier with application to handwritten character recognition," in *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 378–381, 1999.

[41] L. Kuncheva and J. Bezdek, "Nearest prototype classifier: Clustering, genetic algorithms, or random search?," *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 28, no. 1, pp. 160–164, 1998.

[42] H.-P. Schwefel, "Advantages (and disadvantages) of evolutionary computation over other approaches," in *Evolutionary Computation 1: Basic Algorithms and Operators*, 2000.

[43] J. Holland, *Adaptation in Natural and Artifical Systems*. University of Michigan Press, 1975.

[44] F. Brill, D. Brown, and W. Martin, "Fast genetic selection of features for neural network classifiers," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 324–328, 1992.

[45] H. Yoshida, R. Leardi, K. Funatsu, and K. Varmuza, "Feature selection by genetic algorithms for mass spectral classifiers," *Analytica Chimica Acta*, vol. 446, pp. 485–494, 2001.

[46] I. Oh, J. Lee, and B. Moon, "Hybrid genetic algorithms for feature selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1424–1437, 2004.

[47] S. Ho, C. Liu, and S. Liu, "Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm," *Pattern Recognition Letters*, no. 23, pp. 1495–1503, 2002.

[48] M. Sinka and D. Corne, "Evolving document features for web document clustering: a feasibility study," in *Proceedings of the IEEE Congress of Evolutionary Computation*, June 2004.

[49] M. Sinka and D. Corne, "A large benchmark dataset for web document clustering," *Soft Computing Systems: Design, Management and Applications*, vol. 87, pp. 881–890, 2002.

[50] H. Luhn, "The automatic creation of literature abstracts," *IBM Journal of research and development*, vol. 2, pp. 159–165, 1958.

[51] M. Sinka and D. Corne, "Towards modernised and web-specific stoplists for web document analysis," in *Proceedings of the IEEE/WIC International Conference on Web Intelligence*, pp. 396–402, Oct 2003.

[52] H. Bremermann, "Optimization through evolution and recombination," *Self-Organizing Systems*, 1962.