Representative Selection Strategies for Dissimilarity Representations

by

Zane G. Reynolds

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Abraham Kandel, Ph.D.
Member: Dewey Rundus, Ph.D.
Member: Miguel Labrador, Ph.D.

Date of Approval:
May 11, 2004

Keywords: Dissimilarity, representative selection, neural networks, document classification, dimensionality reduction.

© Copyright 2004, Zane G. Reynolds
Acknowledgements

Foremost, I would like to thank Professor Abraham Kandel for seeing potential in me and actively helping me to cultivate my ideas. Professors Horst Bunke and Mark Last have provided invaluable advice and guidance in my research, and I thank them sincerely for taking an interest in my work. I would like to acknowledge my faculty committee members, Professors Dewey Rundus and Miguel Labrador, for their useful comments and time spent on my behalf. Special thanks to Judy Hyde, who has always worked hard to ensure my best interests are kept in mind, to Tom Barr, for his much appreciated practical guidance and instruction, and to the NISTP team for their constant support and interest in my work.

This work was supported in part by the National Institute for Systems Test and Productivity at the University of South Florida under the US Space and Naval Warfare Systems Command Contract No. N00039-02-C-3244 and by the Fulbright Foundation that has granted Prof. Kandel the Fulbright Research Award at Tel-Aviv University, College of Engineering during the academic year 2003-2004.
# Table of Contents

List of Tables .................................................................................................................. ii

List of Figures ................................................................................................................ iii

Representative Selection Strategies for Dissimilarity Representations ....................... iv

Chapter 1 ........................................................................................................................ 1
  1.1 Motivation ....................................................................................................... 1
  1.2 Current Dimensionality Reduction Techniques ................................................ 3
  1.3 Previous Work Applying Neural Networks to Document Classification ........... 4
      1.3.1 Applications of Feed-Forward Neural Networks .................................. 5
      1.3.2 Applications of Non-Linear Networks.................................................... 6
  1.4 Organization .................................................................................................... 6

Chapter 2 ........................................................................................................................ 7
  2.1 Basic Terminology .......................................................................................... 7
  2.2 Document Classification .................................................................................. 7
  2.3 The k-Nearest Neighbor Algorithm ................................................................ 10
  2.4 Artificial Neural Networks ............................................................................. 10
      2.4.1 Feed-Forward Networks ......................................................................... 10
      2.4.2 Non-Linear Networks ............................................................................. 12
      2.4.3 Overfitting ............................................................................................. 13
  2.5 Embedded Spaces and Dimensionality Reduction Techniques ......................... 13
      2.5.1 Quality Measures ................................................................................... 15

Chapter 3 ...................................................................................................................... 17
  3.1 Data and Pre-Processing .................................................................................. 17
  3.2 Representative Selection Strategies ................................................................ 19
  3.3 Evaluation of Representative Selection Strategies ......................................... 20
  3.4 Application of Representative Dissimilarity Data to Neural Networks .......... 35
  3.5 Time and Storage Space Costs ....................................................................... 35

Chapter 4 ...................................................................................................................... 38
  4.1 Summary of Contributions ............................................................................. 38
  4.2 Future Work .................................................................................................. 38

References .................................................................................................................... 40

Appendices ................................................................................................................... 47
  Appendix A: Stop Words ......................................................................................... 48
List of Tables

Table 2.1. Various term-weighting schemes and their weight definitions. 9

Table 3.1. The classes in the Yahoo K-series dataset and their abbreviations. 17

Table 3.2. Accuracy of neural networks trained on the embeddings created by each representative selection strategy. 35

Table 3.3. Storage requirements for the described classification systems. 36

Table 3.4. Shows the time requirements of the described classification systems. 36
List of Figures

Figure 2.1. A sigmoid threshold unit. 11
Figure 2.2. Typical overfitting behavior of neural networks. 13
Figure 3.1. The histogram of the Yahoo K-series dataset and the selected subset. 18
Figure 3.2. Accuracy and classification time of the document vectors. 20
Figure 3.3. The dependence of accuracy on the number of representatives for each selection strategy. 22
Figure 3.4. The dependence of accuracy on the number of neighbors for each selection strategy. 23
Figure 3.5. The dependence of total time on the number of representatives for each selection strategy. 24
Figure 3.6. The dependence of representative selection time on the number of representatives for each selection strategy. Note the difference in scale. 25
Figure 3.7. The dependence of $c_1$ on the number of representatives for each selection strategy. 26
Figure 3.8. The dependence of $c_2$ on the number of representatives for each selection strategy. 27
Figure 3.9. The distortion (both $c_2$ and $c_2$) of each selection strategy as the representatives are increased. 28
Figure 3.10. The dependence of stress on the number of representatives for each selection strategy. 29
Figure 3.11. The dependence of precision on the number of representatives for each selection strategy. 31
Figure 3.12. The dependence of precision on the number of neighbors for each selection strategy. 32
Figure 3.13. The dependence of recall on the number of representatives for each selection strategy. 33
Figure 3.14. The dependence of recall on the number of neighbors for each selection strategy. 34
Representative Selection Strategies for Dissimilarity Representations

Zane G. Reynolds

ABSTRACT

Many of the computational intelligence techniques currently used do not scale well in data type or computational performance, so selecting the right dimensionality reduction technique for the data is essential. By employing a dimensionality reduction technique called representative dissimilarity to create an embedded space, large spaces of complex patterns can be simplified to a fixed-dimensional Euclidean space of points. Unfortunately, the only current suggestion as to how the representatives should be selected to create the representative dissimilarity representation is by performing principal component analysis, which is computationally expensive. Here, we propose several alternative representative strategies and perform an empirical evaluation of those selection strategies as a means of dimensionality reduction on a set of term vectors constructed from HTML documents. The resulting embedded spaces are modeled with artificial neural networks to demonstrate the general applicability of this technique to any type of data with a distance metric. The results indicate that using a representative dissimilarity representation with at least 50 representatives can achieve a significant increase in classification speed, with a minimal sacrifice in accuracy, and when the representatives are selected randomly, the time required to create the embedded space is significantly reduced with a small penalty in accuracy.
Chapter 1
Introduction

1.1 Motivation

With the explosive growth of content on the Internet, automatically discovering, monitoring, and classifying that content is impractical at best, but remains an absolute necessity. Currently the most common type of content classification is categorization; that is, organizing content into topical categories. Large electronic document repositories, news and press release archives, and legal case databases must be able to quickly categorize new documents as they arrive without requiring human intervention. Although categorization is the most typical application of document classification, classification can be extended to user preference or the level of risk posed by the content; that is, organizing content into non-topical classes such as “like” and “dislike” or “safe” and “risky.” As an example of this type of classification, Internet companies are offering more and more personalized web services for their users and customers. Users can listen to streaming radio stations that play songs that they enjoy or are likely to enjoy according to their feedback, or companies can display advertisements or suggest products according to each user’s browsing or buying habits. With the content changing continuously, it is becoming increasingly important for these companies to keep track of user trends.

Computational intelligence techniques have been active areas of research and have shown tremendous promise in solving these problems. Like many concepts, “computational intelligence” has no clearly accepted definition; however, it is generally considered to encompass the methods from artificial intelligence, machine learning, data mining, fuzzy systems, pattern recognition, and soft computing. Applying computational intelligence techniques to Internet content is a research area called web mining [78], which has it’s foundations in the field of information retrieval [3]. Web mining includes examining document usage, structure, or inter-relationships, and content [48].

Most of the computational intelligence techniques in use have strong mathematical roots, making assumptions that the data is highly structured or organized, and that their relationships are either known or can be measured. Web mining data such as documents tend to be only loosely structured, making the application of computational intelligence techniques more difficult. Content such as audio or video clips, images, and complex commodities, which are highly-variable and more sophisticated than documents, can make the application of computational intelligence techniques awkward or clumsy at best. In order to handle these data types, their dimensionality needs to be reduced to a manageable level.

The traditional method employed to reduce the dimensionality of these complex objects is a set of domain-specific descriptive or characteristic features, called feature vectors. Usually, the features that comprise the feature vector are selected by an expert in the particular object domain, and can be quantitative (continuous, discrete, or intervals), qualitative (nominal, ordinal, or cardinal), or a mixture of both [17]. This representation
of objects is simple, and allows the use of computational intelligence methods that deal with a Euclidean feature space. Automated feature selection techniques are more completely discussed in [47]. However, there are several disadvantages of using the feature vector representation of objects. First, the features must be defined by a domain expert, and even if the expert is able to define relevant and measurable features, hidden dependencies between the features can exist. Sometimes the features are too inefficient to compute, features that seem relevant may have poor discriminatory power, or relevant features are left out all together. Selecting too many features can lead to the ‘dimensionality curse’ problem, in which the performance of the applied computational intelligence technique degrades exponentially as a function of the dimensionality of the data [40].

In order to overcome these problems, a dissimilarity representation could be employed to create an embedded Euclidean space [18]. The dissimilarity representation describes objects relatively; that is, objects are defined by their distance or dissimilarity to the other objects in the set rather than by the absolute features in the feature vector representation. The objects are mapped into the embedded space by measuring their distances to every other object in the set. The computational intelligence techniques could then be easily applied to the embedded space. The main advantages of this method are that there are minimal domain-specific knowledge requirements (a distance metric between the objects is required) and the application of a computational intelligence technique to the resulting embedded space is more straight-forward. The main disadvantages of this method are that a distance metric for the objects is required (preferably symmetric and obeying the triangle inequality, otherwise it is possible that the distances are not perfectly preserved in the embedded space), it has poor scalability (the distance between all pairs of objects in the set must be measured), and the performance is decreased with a small set of objects.

However, by selecting a set of representatives (or prototypes) rather than using the entire set of objects to create the dissimilarity representation, the number of required distance computations is significantly reduced [57]. Using a set of representatives, classifying a new object would consist of measuring the dissimilarity between the object and the selected representatives, thus mapping the object into the embedded space, then applying the trained classifier. Unfortunately, other than creating a full dissimilarity representation of the data and performing a principal component analysis, no suggestions have been made as to how these representatives could be selected.

Several strategies for selecting the representative set for a dissimilarity representation are presented and evaluated for the task of classifying HTML documents, examining the dependencies on the number of representatives and the strategy used to select those representatives. A feed-forward neural network is applied to the resulting representation. The purpose of this work is: 1) to evaluate the feasibility of applying complex objects to the more mathematically-based computational intelligence techniques using a generally applicable dimensionality reduction technique, 2) to determine if extensive processing of an entire data set is required to select effective representatives for a dissimilarity representation, and 3) to identify which factors when selecting representatives have the most impact on the resulting representation. The storage requirements as well as time and accuracy performance for the $k$-nearest neighbor
algorithm in the original space and in the embedded space created by each set of selected representatives are empirically assessed along with embedding quality measures.

1.2 Current Dimensionality Reduction Techniques

Many dimensionality reduction techniques are in use today, for a variety of purposes. The goal of dimensionality reduction is usually to reduce the computational cost of handling the data, improve the accuracy of models built on that data, or permit a visual inspection of the data. The ‘dimensionality curse’ [40] is a well-known problem, where the performance of models built on the data degrades as data dimensionality increases. This problem is captured by the “ugly duckling theorem”, which states that any two arbitrary patterns can be made similar when described with a sufficiently large set of redundant features [71]. On the other hand, reducing the dimensionality too much can lead to a loss in discriminatory power.

The most straight-forward and certainly the most popular way to represent data is using characteristic feature vectors, as described in Section 2.5, thus it stands to reason that most of the work in dimensionality reduction is directed towards feature vector representations. Note that representing any data object with a feature vector is itself a reduction in dimensionality. It is also not uncommon to combine dimensionality reduction techniques to attain a good data representation. In the case of features, feature extraction generally refers to the measurement of characteristics, to which feature selection is applied in order to obtain a concise set of features.

Many methods exist for reducing the dimensionality of the data, and these techniques can be broken down into two basic categories: linear and non-linear. The most popular of the linear methods is principal component analysis (PCA), also referred to as Karhunen-Loève transforms [26] [32]. Let the starting dimensionality of the data be \( d \), the desired dimensionality be \( m \), and the size of the dataset containing the \( d \)-dimensional objects be \( n \). PCA computes the \( m \) eigenvectors with the largest eigenvalues of the \( dxd \) covariance matrix of the \( n \) objects. That is, using a matrix-based linear transformation to identify, by means of mean squared error, which \( m \) dimensions have the greatest discriminatory power. PCA is considered a fast technique, which is good for Gaussian-distributed data. Other linear methods such as projection pursuit and independent component analysis are better suited for non-Gaussian distributed data [34].

Linear discriminant analysis takes advantage of cases in which the data is labeled to linearly extract the most discriminatory dimensions. This method replaces the covariance matrix in PCA with some general seperability measure, which finds the between-class scatter matrix (i.e. the dimensions that have the most impact on the class labels). In general, linear discriminant analysis is considered better than PCA for classification tasks [26].

One non-linear method, called kernel PCA, consists of mapping the data into some intermediate space typically with some non-linear mapping function, and applying PCA to the mapped space. During the process, a kernel matrix is defined, given some kernel function, which can be used to map new objects into the space [34]. Another more popular non-linear method is multi-dimensional scaling (MDS). MDS is used to represent a multi-dimensional dataset in either two or three dimensions for visualization purposes. The goal is to preserve the distances in the distance matrix as much as possible in the embedded space. Unfortunately, there is no explicit mapping function, so it is not
possible to map new objects to the embedded space without repeating the entire process or altering the procedure to produce a mapping function [9].

Neural networks are also often used to reduce the dimensionality of input data. The hidden neurons in feed-forward neural networks can be viewed as new, salient input dimensions. When unsupervised, as in the self-organizing map (SOM), or Kahonen network, where neurons are arranged in an $m \times m$ grid with $d$ input units, neural networks can be used to create an embedded space [34].

If the data being operated on is time-sensitive, then discrete Fourier transforms [54] and wavelet transforms [10] are often used to reduce the dimensionality of the data. These methods focus on preserving the relevant information local neighborhoods, rather than trying to retain the global topology of the dataset.

Unfortunately, the dimensionality reduction techniques mentioned thus far require object distances or similarities to be repeatedly computed in the original space. A new dimensionality reduction technique attempting to address this problem is the random mapping method. Given a set of data represented as $N$-dimensional vectors, and a matrix of random values, each vector is multiplied by the random matrix, thus yielding a set of reduced-dimensional vectors. This method is computationally inexpensive, allows new data objects to be quickly mapped into the reduced space, and allows the number of dimensions to be selected. Interestingly, the random-mapping method has performed competitively in terms of accuracy of classifiers built on the reduced-dimensional space [37]. Other applications of random mapping can be found in [1] and [4]. An explanation is suggested in [30] as to why random mapping works well is that there exist a much larger number of almost orthogonal directions than orthogonal directions in a high-dimensional space, thus selecting directions at random will likely produce directions that are close to orthogonal.

1.3 Previous Work Applying Neural Networks to Document Classification

The seminal works in the application of automated techniques for document classification were first published in the 1960s, and more specifically, the application of neural networks to the task began to interest researchers in the early 1990s [69]. There are many types of neural networks, as discussed in Chapter 2, but the most widely-studied neural network is a feed-forward multilayer network and its single-layer counterpart, the perceptron. Other types of networks include recurrent networks, logistic regression networks, Hopfield networks, as well as other specialized architectures. Attempts have been made at applying these networks to document classification with limited success, since document classification is an inherently difficult learning task due to the multi-class nature of documents (that is, natural language documents in general tend to fit multiple classifications or categories). There are also many possible representations of documents and types of document classification problems, which are also discussed in Chapter 2 and surveyed in [69]. The primary difference in each application is the representation of the documents and the type of network used. All of the applications struggle with how to keep the dimensionality of the input low enough for the networks to generalize well.

The application of neural networks has not been thoroughly researched, likely due to the seemingly awkward fit of the variable document representations and the fixed neural network architecture. Each application investigates a different approach to the problem, whether it is varying the type of network, varying the document representation,
or varying the preprocessing strategy. The main thrust of the research has been directed toward using recurrent networks, non-linear networks, or some other customized architecture, although there have been a number of works in applying feed-forward networks.

1.3.1 Applications of Feed-Forward Neural Networks

Most of the variations in the applications of feed-forward neural networks to document classification are focused on altering the preprocessing and representations of the documents. In [22] and [23], equivalence classes of keywords are constructed for each document category, and each document was represented with a feature vector that consists of the degree of membership the document had to each category. The degree of membership the document had to each category was determined by the frequency of the keywords in each equivalence class that occurred in the document. In this way, the number of features for each document was equal to the fixed number of document categories that exist in the data. Thus, the data was suitable for learning with a feed-forward network.

In [38], documents were classified into authorship categories rather than topical categories, relying on the theory that authors tend to use the same sequences of characters repeatedly. Thus, each document was examined for character sequences of length $n$, where $n$ is incrementally increased, by using a sliding window that shifts by one character at a time through the document. The 50 most frequent $n$-tuples of characters were selected and a Karhunen-Loève transform [32] was applied to reduce the samples to 16 features. Finally the features were normalized, and a three-layer feed-forward neural network was trained.

A few years later in [53] and in [73], documents were represented with feature vectors constructed from a dictionary of terms occurring over all documents in the same category and a perceptron was constructed for each category. Other types of non-linear networks were also considered.

In [16], rigorous preprocessing was employed to try and increase the accuracy of the neural network. In this work, a reference library, which is a collection of documents that captures all concepts of interest, is constructed. The reference library is represented with a matrix where each column is the vector of terms occurring in a single document. The matrix is reduced using latent semantic indexing [27] to a user-specified dimension. The authors conduct experiments with the documents that contribute to the reference library, words that are included in the term vectors of each document, and a category based feature extractor that does not employ a reference library. In each case, the dimensionalities of the vectors that represent the documents are all a fixed, user-defined size. The experiments were conducted on three-layer feed-forward networks. A similar approach compared the performance of a three-layer feed-forward network on documents represented by vectors weighted with each DF, CF-DF, $tfidf$, and also by terms selected by principal component analysis [43].

The following two applications approach the problem of document classification from an information retrieval perspective. In [77], a system of autonomous agents, each outfitted with a feed-forward network that is trained at query time, are sent to scour the web and retrieve documents that contain information relevant to the query. In [49], a feed-forward network is applied to the task of document filtering. Unlike the other applications of feed-forward networks, the networks in these experiments are trained to
discriminate an identity function that defines if a document is interesting. Only positive examples are provided to the network for to construct the identity function, and documents presented must be accepted by that identity function in order to activate the network’s output.

1.3.2 Applications of Non-Linear Networks

A greater share of the work in this area was done on non-linear or unsupervised networks such as recurrent, Hopfield, logistic regression networks, specialized architecture networks, self-organizing maps, or support-vector machines. This may be due to the inadequacy of standard feed-forward networks to classify the high-dimensional document data.

One approach used a recurrent network whose inputs were the words in the order they occurred in the document [21] [24]. This approach avoids the problem of discarding possibly useful information when reducing the dimensionality of the documents. In [58], a fuzzy lattice neural network with mapping and voting is applied to documents represented by each DF, Boolean vector, relative DF, and normalized frequency. More information on fuzzy lattice networks is available in [36]. A Hopfield network is employed to assign subjects to documents in [12], where each subject is represented by a stable state, and documents that are applied to the network settle into the nearest subject. The natural hierarchy of document topics is exploited in [63] and [72], where a network with a specialized architecture was constructed to identify higher-levels of interactions between terms in the documents. A network model for document retrieval is proposed in [74], where the purpose of the network is to accept a query represented as a set of terms, and return a ranking of relevant documents.

Many applications of self-organizing maps, which are competitive learning (a form of unsupervised learning) methods based on neural networks, have been researched for document classification, retrieval, or filtering [25] [39] [41] [61] [62]. A support vector machine with a radial-basis kernel trained for relationships between semantic terms is applied in [70]. A comparison of text categorization methods using support vector machines, k-nearest neighbor, neural networks, linear least-squares fit mapping, and naïve Bayes classifiers is performed in [76].

1.4 Organization

Chapter 2 includes the necessary background information on document classification including the vector-space model and the distance metrics, the k-nearest neighbor classification algorithm, artificial neural networks and related technologies, and the dissimilarity representation. Chapter 3 describes the data sets used, the experiments performed, and the values measured in order to evaluate the quality of the embedded spaces created by the selected representatives. Chapter 4 summarizes the work, presents the conclusions that can be drawn from the experiments including advantages and disadvantages of the described technique, and indicates the follow-up work to be done in this area.
Chapter 2
Background

This chapter covers the basic concepts that are required to understand the work presented in the following chapters. In Section 2.1, the basic terminology used throughout the chapter is introduced. Section 2.2 provides an overview of the practices in document classification. Section 2.3 discusses the k-nearest neighbor algorithm and its applications. Section 2.4 introduces neural networks and some of the designs that are referred to during the discussion of neural networks in document classification in Chapter 1. And finally, the fundamentals of embedded spaces and dissimilarity representations are covered in Section 2.5.

2.1 Basic Terminology

Although many varieties of computational intelligence techniques exist, there are two basic categories: supervised and unsupervised. The goal of the supervised techniques is to take a set of examples, called the training data or training set, each of which contain a set of inputs labeled with the desired output, and to “learn” or model the underlying function, called the target function, that govern the given examples. The ability for the learned model, sometimes referred to as a hypothesis, to generalize refers to its accuracy on unseen inputs, or instances. Supervised learning techniques run the risk of overfitting the training data, which occurs when the learned model fits the given training examples too closely and lacks the ability to generalize new instances. One way of limiting the effects of overfitting is to keep some of the data out of the learning process in a set called the validation set, which is used to assess the generalization capabilities of the model. Sometimes a set of data called a test set is kept aside from learning as in a validation set, but the test set is used to tune the model after it has been built to improve is generalization capabilities.

On the other hand, unsupervised learning typically consists of grouping or clustering similar objects in the provided data set. The goal is to place the objects into naturally occurring groups for the purposes of organization or data analysis. The primary difference between supervised and unsupervised learning is that in supervised learning, a training set of labeled data is provided, whereas in unsupervised learning a set of unlabeled data is provided. A review of clustering is available in [33], and a review of statistical pattern recognition is available in [34].

2.2 Document Classification

The area of document classification is closely related to the field of information retrieval, which provides a formal specification of retrieval models. An information retrieval model is a quadruple \([D, Q, F, R(q_i, d_j)]\), where \(D\) represents a set of logical views (or representations) of documents in a collection, \(Q\) represents a set of logical views (or representations) of user queries, \(F\) represents a framework for modeling document representations, queries, and their relationships, and \(R(q_i, d_j)\) is a ranking
function that associates a real number with the query \( q \) and a document \( d_i \). Typically, the system is either *ad hoc* (where the collection remains relatively static and new queries are arriving), *filtering* (where the queries remain relatively static, and the collection is changing constantly), or *routting* (where user preferences and trends are followed to determine what documents are interesting to the user) [3].

In information retrieval, most of the methods devised to deal with text and documents are based on the occurrence of *index terms*, which are words that appear in the text of a document from a collection of documents. Of course, this approach assumes that the semantics of the documents can be expressed through sets of indexed terms, and many approaches based on this model assume that the index terms are mutually independent. In the classical representations of documents, each of the index terms is assigned a numerical weight. There are three classical models of documents: Boolean, vector, and probabilistic. In the *Boolean model* (a set-theoretic model), a document is represented by a vector of index terms whose weights are binary values that indicate whether or not that index term was contained in the document. A more powerful representation is the *vector model* or *vector-space model* (an algebraic model), in which documents are represented as a vector of index terms, but the weights are some non-Boolean numeric weight. Using this model, the weights can be used to compute the degree of *similarity* between documents. The *probabilistic model*, given a user query, attempts to retrieve the “ideal” answer set by making an initial guess, and having the user refine the results with a ranking system. The vector model is the most popular model since it enables the similarity between documents to be measured, allowing partial matches of documents, unlike the Boolean model which can only predict whether a document is relevant or not. Here some formal notation of the vector model is presented from [3].

Let \( t \) be the number of index terms in the system and \( k_i \) be a generic index term, where \( K = \{k_1, \ldots, k_t\} \) is the set of all index terms. A document \( d_i \) is represented by an index term vector \( \vec{d}_j \), where \( \vec{d}_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j}) \). Given a query \( q \), let \( w_{k_i} \) be the weight associated with the pair \( [k_i, q] \), where \( w_{k_i} \geq 0 \). Thus, the query \( q \) is represented as a vector \( \vec{q} \), where \( \vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q}) \). Therefore, a document \( d_j \) and a user query \( q \) are represented as \( t \)-dimensional vectors, and the correlation or similarity between the two vectors can be quantified by the cosine of the angle between them. The equation for the cosine similarity measure is:

\[
\text{sim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{||\vec{d}_j|| \times ||\vec{q}||} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}
\]

With a distance metric in hand, many mathematical approaches can be applied to the representation. The vector model also allows a variety of index term weighting options. Here, only the weighting techniques that are referred to or used in this work are covered; however, a more thorough survey can be found in [66]. Now, the above formal notation is extended to help describe the various weighting schemes. Let \( \text{freq}_{k_i} \) be the
raw frequency (i.e. number of occurrences) of term $k_i$ in document $d_j$, and let $N$ be the total number of documents in the collection, and let $n_i$ be the number of documents in which term $k_i$ appears at least once. Further, let the inverse document frequency of term $k_i$ be given by:

$$ \text{idf}_{k_i} = \log \frac{N}{n_i} $$

The inverse document frequency factor is useful for identifying which words are more useful for distinguishing between relevant and irrelevant documents, since words that appear in many documents contain less information about the subject of the document in which they occurred [3]. Table 2.1 shows some of the weighting schemes used in and referenced by this paper and how the term weights are calculated under each scheme.

<table>
<thead>
<tr>
<th>Weighting Scheme</th>
<th>Weight Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw frequency</td>
<td>$w_{i,j} = \text{freq}_{i,j}$</td>
</tr>
<tr>
<td>Relative frequency</td>
<td>$w_{i,j} = \frac{\text{freq}<em>{i,j}}{\sum</em>{x \in d_j} \text{freq}_{x,j}}$</td>
</tr>
<tr>
<td>Normalized frequency</td>
<td>$w_{i,j} = \frac{\text{freq}<em>{i,j}}{\max</em>{x \in d_j} (\text{freq}_{x,j})}$</td>
</tr>
<tr>
<td>Normalized document frequency</td>
<td>$w_{i,j} = \frac{\text{freq}<em>{i,j}}{\max</em>{x \in \text{documents}} (\text{freq}_{i,x})}$</td>
</tr>
<tr>
<td>Term frequency x inverse document frequency (tf·idf)</td>
<td>$w_{i,j} = \text{freq}_{i,j} \times \log \frac{N}{n_i}$</td>
</tr>
<tr>
<td>Normalized term frequency x inverse document frequency (normalized tf·idf)</td>
<td>$w_{i,j} = \frac{\text{freq}<em>{i,j}}{\max</em>{x \in d_j} (\text{freq}_{x,j})} \times \log \frac{N}{n_i}$</td>
</tr>
</tbody>
</table>

| Table 2.1. Various term-weighting schemes and their weight definitions. |

Other variations of these definitions are commonly used. For example, in a method sometimes called the document frequency (DF) method, a vocabulary is built for each document category, consisting of all the words that occurred in all the documents of the category, ranked by the number of documents in which they appeared, and only the $d$ highest-ranked terms are kept. The category frequency – document frequency (CF-DF)
method is closely related, where each term has a category frequency which corresponds to the number of categories in which the term appeared along with the document frequency. Terms with a category frequency of greater than \( t \) are removed, and then the \( d \) highest-ranked terms are kept based on their document frequencies [43].

2.3 The \( k \)-Nearest Neighbor Algorithm

Unlike many of the supervised learning methods that build explicit models of the target function from a set of training examples, instance-based learning methods simply store the given training examples, and generalization beyond those examples occurs only when a new instance is presented. The relationship between the stored examples and the new instance is examined, and a prediction or classification is assigned based on that relationship.

The \( k \)-nearest neighbor algorithm (\( k \)NN) is the simplest and most widely-used of the instance-based learning methods. It assumes that all instances correspond to points in an \( n \)-dimensional space \( \mathbb{R}^n \), and that the distances between the instances are measured by the standard Euclidean distance. When a new instance \( x \) is presented, its distance to all stored examples is measured, and the \( k \) examples with the smallest distance to \( x \) are selected. If the target function were discrete-valued, the most common value among the \( k \) selected examples is assigned to \( x \). If the target function were real-valued, the mean value among the \( k \) selected examples is assigned to \( x \) [51].

This algorithm never forms an explicit general hypothesis of the target function, but still manages to generalize well when enough examples are stored. Instead, it makes local approximations of the target function around the stored examples. This is a very robust and generally-applicable technique, but the cost of classifying new instances can be very high since almost all computation is done at classification time. The performance of the \( k \)NN algorithm, in terms of accuracy, also depends on the quality of the mapping of the objects into the points in \( \mathbb{R}^n \). If \( n \) is very large, and many of the dimensions are not relevant to capturing the relationship between objects, similar examples may be a large distance apart [51].

2.4 Artificial Neural Networks

An artificial neural network (ANN) is a supervised modeling technique based on an analogy with the human brain that can approximate real-valued, discrete-valued, and vector-valued target functions. Neural networks are one of the most practical and effective modeling techniques, since they can handle real-world complex sensor data. They have been successfully applied to recognizing handwritten characters [45], spoken words [44], and faces [14], automatic driving [59], and a host of other applications [64]. In general, neural network modeling is best suited for problems with noisy, complex sensor data or problems where symbolic representations are used.

2.4.1 Feed-Forward Networks

In order to understand how a neural network models functions, one must have an understanding of the fundamental computational units, or neurons. Figure 2.1 illustrates a basic artificial neuron where \( x_j \) is the output of unit \( j \), \( w_{ij} \) is the weight of the link coming from unit \( j \) to unit \( i \), and \( x_i \) is the output of unit \( i \). Each neuron computes its input function, \( \text{net}_i \), which is usually a sum of its weighted inputs. The output of the unit, \( x_i \), is determined by the unit’s activation function. The activation function sets the threshold at
which the unit will be activated (produce a positive output). There are many possible activation functions, but the most common is the sigmoid activation function (illustrated in Figure 2.1) because it allows the easy computation of derivative, which is necessary for modeling [51].

\[
\text{net} = \sum_{j=0}^{n} w_{ij} x_j
\]

\[
x_i = \sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}}
\]

**Figure 2.1.** A sigmoid threshold unit.

Neural networks are collections of connected neurons or units. There are three types of neurons: input, hidden, and output. *Input neurons* act as receptors for signals from outside the network, but do not perform any computation. *Hidden neurons* perform most of the computation, and are not interacted with by entities outside the network. *Output neurons* perform computation and send their output signals out of the network. If the neurons in the network are connected in a cascading fashion such that a directed acyclic graph is formed, the network is called *feed-forward*. Usually, the neurons are organized into *layers*, or groups of parallel neurons where the input layer is generally not counted as a layer, since it performs no computation. In general, there are two types of feed-forward networks: a *perceptron*, which has no hidden units, and a *multi-layer* network, which has at least one layer of hidden units [51].

Neural networks can be adjusted or “trained” to model or “learn” a particular target function. In order to build a neural network, the network’s layout or architecture must be established, and remain fixed throughout the training process. There are automated techniques that can be incorporated in the training process for adjusting the hidden structure of the network. For example, using genetic algorithms to optimize the structure of the network [50], using optimal brain damage to prune the network [46], or using cascade correlation to add hidden units only when necessary [20]. Examples of the inputs and their desired outputs are provided to the network repeatedly, and the weights of the connections between the units are adjusted such that the error of the network is reduced. Usually the error of a network is defined as the sum of the squared differences between the desired output and the output actually produced by the network averaged over the number of output units. More formally, let \( d \) be a training example selected from the training set \( D \), let \( t_{kd} \) be the target output and \( o_{kd} \) be the actual or observed value of the \( k \)th output unit of the network. \( E(\vec{w}) \) is the error produced by a particular set of weights. The error is most commonly defined as the sum of errors over all output units:

\[
E(\vec{w}) = \frac{1}{2} \sum_{d} (t_{kd} - o_{kd})^2
\]
The most commonly used learning algorithm for neural networks is the backpropagation algorithm, which determines the weights for a multi-layer network given a fixed set of neurons and interconnections [11]. The algorithm employs gradient descent to attempt to minimize the error, and thus can be applied to any function that is differentiable with respect to the parameterized hypothesis space (i.e. the space of possible weights for the given network). The behavior of the gradient descent can be modified by altering the definition of the error. This could be used to implement a weight decay strategy or to minimize the cross entropy of the network [51]. Other error minimization techniques include the conjugate gradient and the line search methods [5]. Neural networks trained with the backpropagation algorithm are appropriate for learning tasks in which instances are represented by many numeric features, the target function output is discrete-valued, real-valued, or a vector of real- or discrete-valued functions, the training examples are noisy, long training times are acceptable, fast evaluation of the learned target function is required, or the ability of humans to learn the target function is unimportant [51].

Feed-forward networks have a very wide variety of functions that they can learn. Every Boolean function can be represented exactly with a 2-layer network, every bounded continuous function can be approximated with an arbitrarily small error by a 2-layer network, and any function can be approximated with to an arbitrary accuracy by a 3-layer network [15]. Unfortunately, there is still the problem of being able to find the right architecture and set of weights for those functions.

2.4.2 Non-Linear Networks

If a network contains cycles in the connections between the neurons, it is called a recurrent network. Recurrent networks are generally used for time-sensitive data where the output at time $t+1$ depends on the output at time $t$ as well as the inputs to the network (e.g. stock market prices). There are several variations of recurrent networks along with several proposed training methods [19] [35] [75]. One such training method is an extension of the backpropagation algorithm that “unfolds” a recurrent network by linking together copies of the network in sequence and trains it as a feed forward network [52]. In general, recurrent networks have more complicated training algorithms and they do not generalize as well in practice as feed-forward networks, however, they do have more representational power [51].

One type of recurrent network is a Hopfield network, sometimes referred to as an associative network because of its similarity to an associative (content addressable) memory in that a portion of the desired data is provided and the complete data is returned. Hopfield networks usually operate on binary inputs and outputs, where the inputs and outputs are of the same length. A set of desired outputs, called stable states, are chosen to be “remembered” by the network, and the weights are calculated from the chosen stable states. When inputs are applied to the network, it is transformed to the closest stable state [29].
2.4.3 Overfitting

The training methods of neural networks attempt to minimize the error on the provided training set. Since neural networks are able to represent any function, it is possible to achieve a zero-error neural network for the training data. However, the performance of the network is better judged by its performance on unseen data (i.e. its ability to generalize), rather than on the training data. Figure 2.2 illustrates the typical behavior exhibited by a neural network during training [51].

![Figure 2.2. Typical overfitting behavior of neural networks.](image)

After a certain amount of training, the network begins to fit the training data too closely, modeling any noise or non-representative distributions and causing the error on the test set to rise. Neural networks are prone to this behavior when there are not enough training examples to adequately generalize. The number of required training examples depends on the size of the network. Generally, as the number of links between neurons in the network increases, so does the required number of training examples, although the exact relationship is not known. Thus, building neural networks for high-dimensional data has a greater likelihood of overfitting than one built on lower-dimensional data.

2.5 Embedded Spaces and Dimensionality Reduction Techniques

The performance of all learning techniques depends on the quality of the data being analyzed. The raw data contains the most information, but most raw data is unsuitable for learning techniques due to its highly-dimensional nature. Selecting the right representation of the data is extremely important, since the goal is to reduce the dimensionality of the data such that operations on that data are less expensive while preserving as much information as possible. Many learning techniques depend on the distances between objects, making the most valuable information is the inter-relationships between the data objects. That is, when the relative similarities between the data objects are preserved, more accurate models can be built from that data. In this section, we introduce some of the dimensionality reduction techniques used or discussed. Note that the terms ‘vector’ and ‘point’ are often used interchangeably when referring to a particular space, since a point $p$ gives rise to a vector $\mathbf{v} = 0\hat{p}$ with the same coordinate values, where 0 is the origin [31].

The most common dimensionality reduction technique is feature extraction and selection. A feature can be defined as a piece of information extracted from a data object [3], and feature extraction involves obtaining a set of features which can capture relevant
information from the data object. Feature selection involves choosing the subset of features that contains the most relevant information. The task of selecting which features are appropriate is generally reserved for experts in the problem domain, however, there are automated techniques for performing feature selection [47]. The result of performing feature extraction on an object is a feature vector, which maps to a point in an embedded Euclidean feature space. The feature space created by feature vectors of length \( n \) can be thought of as an embedded \( n \)-dimensional space, where each dimension or axis corresponds to a feature. Euclidean spaces are built on orthogonal axes, thus, if the selected features are not independent (i.e. if they overlap in someway), then the axes will not be orthogonal, and the actual relationships between the objects will not be as well preserved, reducing the quality of the models that can be built in that space. Other problems with feature vectors are that the features may be difficult or inefficient to obtain, experts may not be able to define the features in a straight-forward way, or features may consist of both continuous and nominal variables.

In [18], the authors propose the dissimilarity representation that attempts to preserve the most valuable information from a set of data objects: their inter-relationships. The dissimilarity representation is created solely from a distance function \( d \), by measuring distance from every object in the set to every other object in the set. Given \( n \) objects, the objects are mapped to an embedded \( n \)-dimensional space, where each dimension or axis corresponds to one of the objects in the set. Although \( d \) is not required to be a distance metric (i.e. satisfies the triangle inequality), the distances between the objects will not be well preserved if it is not. In general, a higher-quality distance metric yields a higher-quality mapping of objects to the embedded space.

The main advantage of this method is that the objects are represented in a relative way, since an object is projected into the embedded space based solely on the other objects in the set. The feature vector, on the other hand, is an absolute mapping. For an intuitive example, consider a room with 100 people. An absolute mapping of people would be to measure the height of each person from the floor to the top of their head, attempting to answer how tall each person is. A relative measurement would be to measure the differences from the top of one person’s head to the top of another, attempting to answer how much taller one person is than another. The authors show in [56] that it is impossible to construct a perfect (zero-error on a validation set) classifier using a feature-based representation, and that such a limitation is not present when using a dissimilarity representation. Furthermore, each object is represented by a vector of normalized distances, making the representation compact for storage and convenient for mathematically-based modeling techniques.

The main disadvantage of this technique is that creating the dissimilarity representation requires \( n^2 \) distance measurements to be made, and mapping a single object into the embedded space requires \( n \) distance measurements. This is often computationally infeasible or at least inefficient for most applications. However, the same authors propose that the dimensionality of the dissimilarity representation can be further reduced by selecting a set of prototypes or representatives from among the data objects, thus creating a representative dissimilarity representation [57]. By selecting \( m \) representatives, the objects are mapped into an \( m \)-dimensional embedded space such that each dimension in the space corresponds to a representative. However, even the representative dissimilarity representation requires that a distance function (and
preferably a distance metric) be available for the objects. Other commonly used dimensionality reduction techniques are discussed in Section 1.2.

2.5.1 Quality Measures

The distances between the objects mapped into an embedded space will not correspond exactly to the distances between the objects in the original space, so classifiers built on the embedded space will most likely be less accurate than those built on the original space. In order to determine how well an embedded space resembles (or preserves) the object relationships, some quality measures are required.

When building a kNN classifier or performing a query on an embedded space, two useful quality measures are precision and recall. In order to define these two measures, some formal notation is required. Let \( R_o \) be the set of objects that are returned by a similarity query in the original space, and \( R_e \) be the set of objects that are returned by the corresponding query in the embedded space. Since the elements in \( R_o \) may differ from those in \( R_e \), and \( R_o \) is assumed to be the correct result, we can establish two types of error in \( R_e \): some elements in \( R_o \) may not be found in \( R_e \), and some elements are found in \( R_e \) that are not found in \( R_o \). Precision measures the first type of error, by determining the proportion of objects in \( R_o \) that are also in the correct result \( R_o \). Precision is defined as:

\[
\text{precision} \equiv \frac{|R_e \cap R_o|}{|R_e|}
\]

Recall measures the second type of error, by determining the proportion of objects that are in \( R_o \) that are also in \( R_e \). Recall is defined as:

\[
\text{recall} \equiv \frac{|R_e \cap R_o|}{|R_o|}
\]

When the precision is 100\%, \( R_e \) contains only correct objects, but may not contain all of the correct objects. When the recall is 100\%, \( R_e \) contains all of the correct objects, but may also contain incorrect objects [31].

Another way to measure the quality of an embedded space is to measure directly how well the distances between objects in the embedded space correspond to the distances between objects in the original space. Let \( d(o_1, o_2) \) be the distance between object \( o_1 \) and object \( o_2 \) in the original space. These objects are mapped into an embedded space with a mapping procedure \( F \), where their distances in the embedded space is given by \( \delta(F(o_1), F(o_2)) \). The distortion of the distances refers to how much larger or smaller the distances between objects in the embedded space when compared to the distance between the same objects in the original space. Formally, the distortion is defined as the smallest values for \( c_1 \) and \( c_2 \) such that it is guaranteed that [31]:

\[
\frac{1}{c_1} \cdot d(o_1, o_2) \leq \delta(F(o_1), F(o_2)) \leq c_2 \cdot d(o_1, o_2)
\]
Stress measures the overall deviation of the original distances and the embedded distances with respect to a data set $S$. That is, the extent to which the functions $d$ and $\delta$ differ for a set of objects. Stress is typically defined in terms of variance [31]:

$$
stress \equiv \frac{\sum_{o_1, o_2 \in S} (\delta(F(o_1), F(o_2)) - d(o_1, o_2))^2}{\sum_{o_1, o_2 \in S} d(o_1, o_2)^2}\$$
Chapter 3
Experiments and Results

3.1 Data and Pre-Processing
The Yahoo K-series dataset is a publicly available dataset consisting of news articles organized into topical categories [6] [7] [8]. More specifically, the Yahoo dataset consists of 2,340 HTML documents organized into 20 classes, which are shown with their abbreviations in Table 3.1. There is some degree of skew within this dataset, as there are about 55 times more documents in the most-frequent class than in the least-frequent class (494:9). Since the experiments described in this work require full dissimilarity representations of the data to be built, a subset of 585 documents (25% of the entire dataset) was drawn uniformly from the original data, thus preserving the original distribution, while reducing the required amount of computation. Figure 3.1 illustrates the distribution of the Yahoo dataset and the distribution of the selected subset. This subset represents only 19 of the original 20 classes, and other classes are quite under-represented.

<table>
<thead>
<tr>
<th>Class</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business</td>
<td>B</td>
</tr>
<tr>
<td>Entertainment (general)</td>
<td>E</td>
</tr>
<tr>
<td>Entertainment (art)</td>
<td>Ea</td>
</tr>
<tr>
<td>Entertainment (cable)</td>
<td>Ec</td>
</tr>
<tr>
<td>Entertainment (culture)</td>
<td>Ecu</td>
</tr>
<tr>
<td>Entertainment (film)</td>
<td>Ef</td>
</tr>
<tr>
<td>Entertainment (industry)</td>
<td>Ei</td>
</tr>
<tr>
<td>Entertainment (media)</td>
<td>Em</td>
</tr>
<tr>
<td>Entertainment (multimedia)</td>
<td>Emm</td>
</tr>
<tr>
<td>Entertainment (music)</td>
<td>Emu</td>
</tr>
<tr>
<td>Entertainment (online)</td>
<td>Eo</td>
</tr>
<tr>
<td>Entertainment (people)</td>
<td>Ep</td>
</tr>
<tr>
<td>Entertainment (review)</td>
<td>Er</td>
</tr>
<tr>
<td>Entertainment (stage)</td>
<td>Es</td>
</tr>
<tr>
<td>Entertainment (television)</td>
<td>Et</td>
</tr>
<tr>
<td>Entertainment (variety)</td>
<td>Ev</td>
</tr>
<tr>
<td>Health</td>
<td>H</td>
</tr>
<tr>
<td>Politics</td>
<td>P</td>
</tr>
<tr>
<td>Sports</td>
<td>S</td>
</tr>
<tr>
<td>Technology</td>
<td>T</td>
</tr>
</tbody>
</table>

Table 3.1. The classes in the Yahoo K-series dataset and their abbreviations.
The documents in the dataset must then be converted into a form in which a distance metric can be applied. For documents, the most effective distance metric operates on a vector of index terms for each document, which, unfortunately, requires a lot of information to be discarded. For example, word orderings, word negation, phrases, titles, and keywords are all valuable information that is discarded while constructing document vectors. Reducing documents to a weighted vector of index terms is itself a reduction in dimensionality, to which, further reduction is applied in these experiments. However, the goal here is to assess the tradeoffs of this representation, and not to outperform existing methods. Thus, the documents in the datasets were converted to the vector-space model, using the inverse document frequency (tf-idf) weighting scheme, enabling use of the cosine distance metric between the documents.

Since the documents being processed are semi-structured, they contain formatting tags and other characters that do not contribute to the actual content of the document. These tags and characters are removed from the document using a standard HTML language parser, leaving behind only the ‘meaningful’ text. Next, all punctuation, special characters, and stop words are removed. Stop words are common words that occur frequently in the English language, such as “the”, “and”, “of”, and so on. Since they are not considered in context, these words do not help determine which class the document in which they occurred belongs. For these experiments, a list of about 600 stop words was used as a basis for word removal, which are listed in Appendix A.

Next, the words are placed in a vector, where simple stemming is performed. Stemming is used to reduce the number of index terms in a document by conflating words with the same root word or ‘stem’. Stemming is used to determine which words should be considered identical. For example, “dog” and “dogs” both refer to the same fundamental subject, and can be conflated into a single word. The most common stemming algorithm involves applying some simple word transformation rules in

![Histogram of Yahoo K-series dataset and the selected subset.](image-url)
sequence such that the words are reduced to their stems [60]. However, the stemming algorithm used in these experiments is even simpler [67]. For each word in the vector, common suffixes are added to the term and if a matching word is found, the two words are conflated. Conflating equivalent words consists of adding the frequencies of each word, adopting the more frequent word as the new word, and then removing the original words from the vector. For example, in an article about “data mining”, words that are variations of “mine” will most likely be conflated into “mining” and not “mine”. The common suffixes added are the plural endings “s” and “es” as well as the verb endings “ing”. If the word ends in an “e”, the “e” is removed before adding the “ing”.

Finally, the weight of each word in every vector is calculated, using the inverse document frequency (tf·idf) weighting scheme, thus allowing the use of the cosine distance metric. Combining the original filename, term vector, and desired classification of each document yields a set of training examples suitable for learning. The examples are then randomly shuffled to avoid any learning biases created by the order in which the documents were read. For example, if we constructed a training set with the first 100 examples read, there is a high likelihood that these examples do not follow the original distribution of the entire dataset, since datasets are usually constructed in some organized fashion. On the other hand, if those were 100 randomly chosen examples, they will follow the original distribution of the data with greater likelihood.

3.2 **Representative Selection Strategies**

As discussed in Chapter 2, in order to make dissimilarity representations more scalable, representatives are selected from among the training set to reduce the number of required distance computations and the number of dimensions in the embedded space. Each representative becomes a dimension of the embedded space, such that $m$ representatives create an $m$-dimensional Euclidean space $\mathbb{R}^m$. Currently, the literature suggests only one option for selecting the representatives: performing principal component analysis to determine which objects are the most orthogonal. In other words, principal component analysis selects the objects which have the greatest impact on the resulting embedded space. However, using this method moves away from the original goal of scalability, since the entire dissimilarity representation must still be constructed along with performing principal component analysis.

Here several other strategies for selecting the representatives from among a training set are examined in order to assess the impact on the resulting representation, and reveal the qualities of a good set of representatives. The strategies are broken into two parts: selecting the first representative and selecting the remaining representatives. To clarify, the *median* object of a set is defined as the object with the smallest average distance to all other objects in the set. The *outlier* object of a set is the object with the largest minimum distance to all other objects in the set. That is, the outlier is the object whose closest neighbor is farther away than any other object’s closest neighbor. The strategies for selecting the first representative are:

1. Selecting an object at random
2. Selecting the *median* - the object with the smallest average distance to all other objects in the training set
3. Selecting the *outlier* - the object with the largest minimum distance to the other objects in the training set
The strategies for selecting the remaining representatives are:
1. Selecting objects at random (where each object may be selected at most one time)
2. Selecting the outlier to the selected representatives - the object with the largest minimum distance to the existing set of representatives

Using all combinations of the above selection strategies yields six unique strategies for selecting representatives from among the training set. These are referred to from now on by the selection strategy for the first representative followed by the representative selection strategy for the remaining representatives, separated by a dash (e.g. “Random-Random” and ‘Median-Outlier”).

3.3 Evaluation of Representative Selection Strategies

In order to establish a base-line for comparison, a \(k\)NN classifier was applied to the term vectors directly. That is, the space created by the standard vector model of the documents with \(tfidf\) weighting is considered to be the ‘original space’, to which each embedding method that is created is compared. Although the vector model is itself a dimensionality reduction technique, the emphasis here is on measuring the effects of using an embedded space and demonstrating how this method can be used to apply objects to highly-structured classifiers, rather than attempting to devise a method that yields superior document classification accuracy. Figure 3.2 shows the results of the classification on the Yahoo dataset. Note that in all figures, all data points are the mean values obtained from 10 runs, where each run consists of a 10-fold cross validation, and the error bars illustrate the 95% confidence intervals.

![Figure 3.2. Accuracy and classification time of the document vectors.](image)

At this point, representatives were selected from among the document vectors, and an embedded space was created. Each document vector was mapped into the embedded space by measuring their cosine distances to each of the representative document vectors, such that each document vector is represented only by a set of coordinates. These document coordinates were then classified with a \(k\)NN classifier. The controlled variables for these experiments were \(k\) (i.e. the number of neighbors used in the \(k\)NN classifier), the number of representatives chosen, and the representative selection strategy.

Figure 3.3 shows the dependence of accuracy on the number of representatives for each selection strategy. It can be seen that the outlier-based selection strategies (i.e. the
ones that choose representatives that are outliers to the currently selected representatives) have a slightly higher accuracy than the random-based strategies (i.e. the ones that choose representatives randomly). It is also interesting to note that the accuracy seems to saturate at about 60 representatives. Figure 3.4 illustrates the dependence of accuracy on k. It can be seen that the outlier-based selection strategies again perform slightly better than the random-based selection strategies by about 3-4%. In fact, the outlier-based selection strategies rival the original document vector representation, staying within 1-4% for all values of k.

Figure 3.5 shows the effect of the number of representatives on the total time taken per run. As previously mentioned, each run performs a 10-fold cross validation using a kNN classifier, where, in each fold, a new set of representatives is selected and every document vector is mapped into the new embedded space. Notice that the outlier-based selection strategies take about 200 seconds longer than the random-based strategies, due to their need to construct a full dissimilarity representation in order to select their representatives. Figure 3.6 shows the time taken to select the representatives from among the population. The random-based selection strategies grow at a very small, linear rate with respect to the number of representatives, whereas, the outlier-based strategies grow at a super-linear rate.

Looking more closely at each selection strategy, the distortion and stress of the embeddings created by each strategy is measured. As described in Section 2.5.1, distortion is defined by the two bounding constants, c1 and c2, and stress is measured directly. The ideal values for c1 and c2 are 1, which would mean that all of the distances in the embedded space are exactly the same as the original distances. If c1 grows and c2 remains constant, the distances in the embedded space are becoming smaller than the distances in the original space. Conversely, if c2 grows and c1 remains constant, the distances in the embedded space are becoming larger than the distances in the original space. The ideal value for stress is 0, which means that there is not deviation in the distances in the embedded space and the distances in the original space.

Figure 3.7 illustrates the effect of the number of representatives on c1, Figure 3.8 illustrates the effect of the number of representatives on c2, Figure 3.9 shows the effects of both c1 and c2 at the same time, and Figure 3.10 illustrates the effect of the number of representatives on stress. From these figures we can see some interesting behavior in the distortion. Notice in Figure 3.9 that the values for c2 begin to grow as the values for c1 shrink. This indicates that the distances between the objects in the embedded space are getting larger as the space itself grows larger. This behavior also corroborates the saturation effect that is observed in accuracy when varying the number of representatives (Figure 3.3). There is no clear separation among the selection strategies in terms of c1, but the values for c2 are slightly smaller among the outlier-based strategies. It is also apparent that the stress is reduced as the number of representatives grows, and that the outlier-based strategies have a marginally higher stress than the random-based strategies.
Figure 3.3. The dependence of accuracy on the number of representatives for each selection strategy.
Figure 3.4. The dependence of accuracy on the number of neighbors for each selection strategy.
Figure 3.5. The dependence of total time on the number of representatives for each selection strategy.
Figure 3.6. The dependence of representative selection time on the number of representatives for each selection strategy. Note the difference in scale.
Figure 3.7. The dependence of $c_1$ on the number of representatives for each selection strategy.
Figure 3.8. The dependence of $c_2$ on the number of representatives for each selection strategy.
Figure 3.9. The distortion (both $c_2$ and $c_3$) of each selection strategy as the representatives are increased.
Figure 3.10. The dependence of stress on the number of representatives for each selection strategy.
Since a $k$NN classifier can be viewed as a query from a data store, we can measure the precision and recall rates for each selection strategy, where each query involves returning the $k$ nearest objects to the object provided in the query. This is a very good test for each embedded space, especially if the average point density in the space is high, which is more likely in lower-dimensional spaces.

In Figure 3.11, the precision for each selection strategy are rising with the number of representatives, and, although there is not much separation, the outlier-based selection strategies have slightly higher precision than the random-based strategies. Similar behavior is observed in Figure 3.13 for the recall rates; however, there is a more consistent occurrence of a drop-off in recall rate at 100 representatives. Again, the outlier-based selection strategies slightly outperform the random-based selection strategies. Interestingly, in Figure 3.12 and Figure 3.14, a very pronounced drop-off in precision and recall rates are observed as the number of neighbors asked to be returned by the query is increased. This is likely attributed to the increased possibility of error in queries that require more results to be returned, and the behavior corroborates the higher accuracy of the $k$NN classifier when $k = 1$.

After examining the representative selection strategies, two clear groups emerged: the random-based selection strategies (where all representatives except the first are chosen at random) and the outlier-based selection strategies (where each representative is outlier to the representatives already in the representative set). On the other hand, the selection of the first representative had no significant observable effect on either the quality of the embedding or the accuracy of the classifier. From the results we can see that the outlier-based selection strategies were marginally better in terms of $k$NN accuracy and upper bound of distortion, while the random-based selection strategies were significantly better in terms of representative selection time and marginally better in terms of stress. In practice, the argument for using a strategic representative selection strategy is very weak considering the amount of computation that is required to determine the median, outliers, or principal components. The finding that randomly selecting components for axes in a Euclidean space is nearly as good as strategic selection is corroborated in [30]. A more detailed look at the time and storage requirements for this method is given in Section 3.5.
Figure 3.11. The dependence of precision on the number of representatives for each selection strategy.
Figure 3.12. The dependence of precision on the number of neighbors for each selection strategy.
Figure 3.13. The dependence of recall on the number of representatives for each selection strategy.
Figure 3.14. The dependence of recall on the number of neighbors for each selection strategy.
3.4 Application of Representative Dissimilarity Data to Neural Networks

Using the settings that create the best embeddings for each of the representative selection strategies as determined by experimentation, a dataset is constructed that is suitable for neural network learning. Ten neural networks were built for each selection strategy, selecting 60 representatives and a random subset of 90% of the data. Each neural network had 60 inputs, one for each representative chosen, and 19 outputs, which represented each class (refer to Section 3.1 for why only 19 of the original 20 classes are represented). To ensure a fair estimation of the performance of each selection strategy on its own, no input selection or data transformations were used for these experiments.

The hidden-unit structures of these networks were composed only of sigmoid units, where the number and arrangement of these units were determined by using a highest average classification rate strategy. That is, the network architecture is determined by training the network on many different configurations of hidden units, and taking the structure with the highest average classification rate. Each configuration was trained using the backpropagation algorithm. Table 3.2 shows the results of the neural network experiments. It is interesting to note that the random-based selection strategies had both a higher mean accuracy and a higher variance.

<table>
<thead>
<tr>
<th>Representative Selection Strategy</th>
<th>Neural Network Accuracy (95% Confidence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random-Random</td>
<td>71.25 ± 5.4%</td>
</tr>
<tr>
<td>Random-Outlier</td>
<td>63.51 ± 4.2%</td>
</tr>
<tr>
<td>Median-Random</td>
<td>71.61 ± 6.5%</td>
</tr>
<tr>
<td>Median-Outlier</td>
<td>68.31 ± 3.6%</td>
</tr>
<tr>
<td>Outlier-Random</td>
<td>70.98 ± 5.9%</td>
</tr>
<tr>
<td>Outlier-Outlier</td>
<td>66.30 ± 3.8%</td>
</tr>
</tbody>
</table>

Table 3.2. Accuracy of neural networks trained on the embeddings created by each representative selection strategy.

3.5 Time and Storage Space Costs

In this section, the time and storage space required by a kNN using the standard document representation, a kNN using the representative dissimilarity representation, and an ANN trained on the representative dissimilarity representation is compared. This comparison helps reveal the trade-offs involved with each system. In order to make this comparison, some formal notation is required. Let \( n \) be the number of documents in a particular collection or training set, and let \( m \) be the number of representatives. From the experiments in Section 3.3, we can see that \( m << n \).

Let the space required to store the average document term vector be given by \( v\text{Size} \). To avoid confusion, a document represented by only its coordinates in the embedded space is referred to as a document point. Let the average space required to store a document as a point be given by \( p\text{Size} \). Since storing a large vector of index terms requires much more space than storing a small vector of real values, \( p\text{Size} << v\text{Size} \). Finally, let the space required to store a trained ANN be given by \( \text{annSize} \). Storing a
$kNN$ classifier requires that all training examples be stored for comparison, therefore, storing a $kNN$ classifier using the standard document term vectors requires the space given by $n \cdot vSize$. Since using the representative dissimilarity representation requires a new document to be converted into a document point using the set of representatives, we must store the set of representatives as document vectors. So storing a $kNN$ classifier using the representatives is $(m \cdot vSize) + (n \cdot pSize)$, and storing an ANN trained on the representative dissimilarity representation is $(m \cdot vSize) + annSize$. These requirements are summarized in Table 3.3.

<table>
<thead>
<tr>
<th>Representation/Classifier</th>
<th>Required Storage Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term vector $kNN$</td>
<td>$n \cdot vSize$</td>
</tr>
<tr>
<td>Representative dissimilarity $kNN$</td>
<td>$(m \cdot vSize) + (n \cdot pSize)$</td>
</tr>
<tr>
<td>Representative dissimilarity ANN</td>
<td>$(m \cdot vSize) + annSize$</td>
</tr>
</tbody>
</table>

Table 3.3. Storage requirements for the described classification systems.

Let $p$ be the mean pre-processing time of a particular document. That is, $p$ represents the mean amount of time to convert a raw document to a weighted vector of index terms. Let $cosDist$ be the mean time to measure the cosine distance between two term vectors, and let $eDist$ be the mean time to measure the Euclidean distance between two coordinate vectors of length $m$. In our experiments, using a Sun 450 with four 333 MHz processors and 2 GB of RAM, $cosDist = 1.15 \times 10^{-3}$, while $eDist$ ranged from $2.50 \times 10^{-5}$ for 10 representatives to $2.10 \times 10^{-4}$ for 100 representatives. That is, the $eDist$ time varies for the number of dimensions in the embedded space. Thus, $eDist << cosDist$ when the number of dimensions is less than about $10^{3}$. Let $trainANN$ represent the training time and $runANN$ be the running time of a neural network on a set of $n \cdot m$-dimensional coordinate vectors. Table 3.4 shows the overall time required to build the classifier as well as the classification time required for a single document presented to the model after it is built for each of the classifier systems.

<table>
<thead>
<tr>
<th>Representation/Classifier</th>
<th>Time to Build Classifier</th>
<th>Single Document Classification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term vector $kNN$</td>
<td>$n \cdot p$</td>
<td>$p + (n \cdot cosDist)$</td>
</tr>
<tr>
<td>Non-random-based dissimilarity $kNN$</td>
<td>$(n \cdot p) + (n^2 \cdot cosDist)$</td>
<td>$p + (m \cdot cosDist) + (n \cdot eDist)$</td>
</tr>
<tr>
<td>Random-based dissimilarity $kNN$</td>
<td>$(n \cdot p) + (n \cdot m \cdot cosDist)$</td>
<td>$p + (m \cdot cosDist) + (n \cdot eDist)$</td>
</tr>
<tr>
<td>Rep. dissimilarity ANN</td>
<td>$(n \cdot p) + (n^2 \cdot cosDist) + trainANN$</td>
<td>$p + (m \cdot cosDist) + runANN$</td>
</tr>
</tbody>
</table>

Table 3.4. Shows the time requirements of the described classification systems.
Following Table 3.4 downward, it is apparent that the computational weight shifts away from the single document classification time and toward the building of the classifier. Depending on the application, one may prefer to take longer building the classifier so that the time required to classify a single document is reduced.
Chapter 4
Conclusions and Future Work

4.1 Summary of Contributions
In Chapter 3, several strategies for selecting representatives for use in creating representative dissimilarity representations are presented and empirically evaluated to quantify the effects of the information lost when mapping objects into the embedded space. Previously, the only currently suggested method of selecting representatives for this dimensionality reduction technique was to perform principal component analysis on the full dissimilarity representation of the data. From the evaluation performed in this work, the only benefits of building a full dissimilarity representation of the available data and strategically selecting every representative over simply randomly picking representatives are a slightly higher accuracy and marginally lower distortion of the distances in the embedded space. In addition, the documents in the dataset were mapped to the embedded spaces created by each of the representative selection strategies, modeled with a neural network, and compared. The results indicated that the random-based selection strategies yielded a slightly higher accuracy on average than the outlier-based selection strategies, but also with higher variance.

Using a set of representatives from among an object population and creating a representative dissimilarity representation allows any type of object with a distance metric to be applied to neural networks and other classifiers that rely on highly-structured data. Note that the distance metric and its meaning should be determined by a domain expert, however, once a distance metric is defined, no additional domain expertise is required. As in any dimensionality reduction technique, some information is lost in the transformation, but using the representative dissimilarity representation the most vital information is retained as well as possible: the object relationships. If using a non-random selection strategy that requires knowledge about all objects in the set, the construction of the embedded space is as computationally expensive as a full dissimilarity representation.

4.2 Future Work
There are many exciting avenues of exploration related to this dimensionality reduction technique. Most immediately pertaining to this work is experimenting with other representative selection strategies; for example, centers of clusters that occur naturally within the data, objects identified by competitive learning techniques, or the set of objects returned from a heuristic search. Examining how many representatives should be selected as a rule-of-thumb for a given dataset would be important contribution. Another important direction is determining which characteristics of the dataset (e.g. number of elements, distribution as compared to the actual distribution, etc.) from which the representatives are chosen affect the performance of the embedded spaces.

Other data structures with distance metrics could be used. Many recent advanced have been made in graph-theoretic representations of data, which have been shown to
contain more information than feature vector representations, and applying this dimensionality reduction technique to data represented with graphs could potentially increase the accuracy of many learning tasks while keeping the required computation at a manageable level. Proteins and protein synthesis is another area of significant research interest, where the main problem is processing the huge data structures that represent the proteins.

The results in Section 3.4 indicated that using a neural network on the embedded spaces created by each representative selection strategy yielded higher accuracy on the random-based strategies than on the outlier-based strategies. This may indicate that other learning methods, such as clustering, may also demonstrate the benefits of this dimensionality reduction technique.
References


Appendices
# Appendix A: Stop Words

- a
- about
- above
- according
- accordingly
- across
- actually
- after
- afterwards
- again
- aint
- all
- allow
- allows
- almost
- alone
- along
- already
- also
- although
- always
- among
- amongst
- an
- and
- another
- any
- anybody
- anyhow
- anyone
- anything
- anyway
- anyways
- anywhere
- apart
- appear
- appreciate
- appropriate
- approximately
- are
- arent
- around
- as
- aside
- ask
- asking
- associated
- at
- available
- ave
- away
- awfully
- back
- be
- became
- because
- become
- becomes
- becoming
- been
- before
- beforehand
- behind
- being
- believe
- below
- beside
- besides
- best
- better
- between
- beyond
- blvd
- both
- brief
- browser
- but
- by
- came
- can
- cannot
- cant
- cause
- causes
- certain
- certainly
- changes
- clearly
- click
- cmom
- co
- com
- come
- comes
- comment
- concerning
- contact
- contain
- containing
- contains
- copyright
- corresponding
- could
- couldnt
- course
- currently
- date
- definitely
- described
- despite
- did
- didnt
- different
- do
- does
- doesnt
- doing
- done
- dont
- down
- download
- downwards
- dr
- due
- during
- each
- edu
- eg
- eight
- either
- else
- elsewhere
- email
- enough
- entirely
- especially
- et
- etc
- even
- ever
- every
- everybody
- everyone
- everything
- everywhere
- ex
- exactly
- example
- except
- far
- fax
- few
- fifth
- find
- first
- five
- fl
- followed
- following
- follows
- for
- former
- formerly
- found
- four
- fourth
- frame
- free
- from
- ftp
- further
- furthermore
- get
- gets
- getting
- give
<table>
<thead>
<tr>
<th>was</th>
<th>when</th>
<th>whom</th>
<th>www</th>
</tr>
</thead>
<tbody>
<tr>
<td>wasnt</td>
<td>whence</td>
<td>whos</td>
<td>yes</td>
</tr>
<tr>
<td>way</td>
<td>whenever</td>
<td>whose</td>
<td>yet</td>
</tr>
<tr>
<td>we</td>
<td>where</td>
<td>why</td>
<td>you</td>
</tr>
<tr>
<td>web</td>
<td>whereafter</td>
<td>wide</td>
<td>youd</td>
</tr>
<tr>
<td>wed</td>
<td>whereas</td>
<td>will</td>
<td>youll</td>
</tr>
<tr>
<td>welcome</td>
<td>whereby</td>
<td>willing</td>
<td>your</td>
</tr>
<tr>
<td>well</td>
<td>wherein</td>
<td>wish</td>
<td>your</td>
</tr>
<tr>
<td>went</td>
<td>whereupon</td>
<td>with</td>
<td>youre</td>
</tr>
<tr>
<td>were</td>
<td>wherever</td>
<td>within</td>
<td>yours</td>
</tr>
<tr>
<td>were</td>
<td>which</td>
<td>without</td>
<td>yourself</td>
</tr>
<tr>
<td>werent</td>
<td>while</td>
<td>wonder</td>
<td>yoursves</td>
</tr>
<tr>
<td>weve</td>
<td>whither</td>
<td>wont</td>
<td>youve</td>
</tr>
<tr>
<td>what</td>
<td>who</td>
<td>world</td>
<td>zero</td>
</tr>
<tr>
<td>whatever</td>
<td>whoever</td>
<td>would</td>
<td></td>
</tr>
<tr>
<td>whats</td>
<td>whole</td>
<td>wouldn't</td>
<td></td>
</tr>
</tbody>
</table>