Text Categorization and Prototypes

Alexander Bergo

August 17, 2001

"There is nothing more basic to thought and language than our sense of similarity; our sorting of things into kinds."

Kant

Acknowledgements:

First and foremost to Maarten de Rijke and his brilliant guidance, effort to keep me going and consistent support far beyond the reaches of what is to be expected. To Christof Monz for his encouraging words, and for sharing his knowledge. To Ingrid van Loon and Dick de Jongh for accepting me! To all my new-won friends for their warmth. To Benjamin and Hanno (subset of friends) for voluntarily including me in their discussion groups. To Viil for moving down to Amsterdam ++. To all my lecturers for showing me a glimpse of the beautiful world(s) of logic. Last but not least, to the whole community of ILLC for their incredible attitude.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Text Categorization

This thesis is about the automated categorization of texts into categories of certain topics. The subject goes at least back to the 1960's [22]. Since the 1960s we have seen an immense growth in the production and availability of digital libraries, news sources and online documents. As a result, automated text categorization has witnessed an increased and renewed interest.

A generally accepted definition of text categorization is

> "Text Categorization (TC) is the task of deciding whether a piece of text belongs to any of a set of prescribed categories. It is a generic text processing task useful in indexing for later retrieval, as a stage in natural language processing systems, for content analysis, and in many other roles."
>
> *Lewis*

The core problem in automated text categorization is this: how can documents be assigned to a category with a highest possible chance of being correct without assigning too many incorrect categories, and at acceptable computational costs?

Machine learning paradigm [17, 22] has emerged as one of the main approaches in the area. The machine learning approach to text categorization consists of a general inductive process which automatically builds a classifier based on the characteristics of each of the categories. These characteristics are learned from documents already classified and then utilized on the documents to be classified. Advantages of this approach include domain independence and resource saving by focusing on construction of systems instead of the classifier.

Although there have been many and rapid developments in the field, we believe that there is still room to improve various aspects of the task of assigning a category to a document. In this thesis we will try to adapt theories from other disciplines for TC purposes. More specifically, we will try to improve two aspects which may seem to be at odds with each other: how to achieve a good text categorization system and at a low processing time.

To evaluate the effectiveness of our new methods for document categorization, we will compare them to two well-known existing ones: *k Nearest Neighbor* and *Rocchio*. Before proceeding, let us briefly recall some details about these methods.

1.2 Two Well-Known Approaches

1.2.1 k Nearest Neighbour

The first approach we will test and use as a basis for comparison is the classic k Nearest Neighbours algorithm (kNN) [17, 15]. This approach has become a standard within the field of text categorization and is included in numerous experiments as a basis for comparison, the main reason being that it is by far the better performing algorithm. kNN takes an arbitrary input document and ranks the k nearest neighbour among the training documents through the use of a similarity score. It then adapts the category of the most similar document or documents; k denotes the number of neigbours included in the evaluation. As our documents can have more than one category assigned to them, we will try a number of ways of selecting categories from kNN.

Although not perfect, research has shown that kNN is the best overall performing system on diverse sets [9, 32, 31, 11]. In practice kNN compares and ranks each of the already categorized documents with the document to be categorized. This leads, as Mitchell and others [17, 15] points out, to an undesirable amount of computational costs.

1.2.2 Rocchio

To reduce the demand for processing cost we will also implement a centroid[1] classifier, called the *Rocchio classifier*. The one we focus on is a well-known information retrieval algorithm developed by Rocchio [14, 24, 26] in the early 1970s, originally to optimize queries from relevance feedback.

This approach makes a centroid document for each category and then ranks the categories, through the use of similarity scores according to which one compares each feature of the category's centroid document to the features of the test document. The processing costs are not only reduced, but potentially much lower than for kNN. The higher number of documents that has been categorized already, the bigger the difference in processing costs.

[1] Defined as the point in a system of masses each of whose coordinates is a weighted mean of coordinates of the same dimension of point within the system, the weights being determined by the density function of the system.

But there are problems because, however good the intentions of using a centroid or prototype for each category, the performance is quite poor as we can see in [31].

1.3 Our Proposal

As the discussion so far seems to indicate, there is a basic dichotomy in text categorization between having a good categorizer (kNN) and low processing costs (Rocchio). We will try to reconcile the two forces by developing centroid- or, as we prefer to call them, prototype-based algorithms that look for similarities and *dissimilarities*. The basic idea is to measure the *distance* between two objects, not the closeness.

In general, our algorithms will consist of two phases:

1. generating prototypes, and

2. comparing documents to be classified to prototypes.

In some settings, however, the two steps will be combined into a single one as seen in [27]. This is done by first measuring the distance between the document to be categorized with the previously categorized documents. The dissimilarity score for each category are then averaged and ranked. The category which has the least average dissimilarity score is adopted to the document we want to categorize.

One of the core issues, then, is to come up with a good notion of prototype. We will try to implement notions of prototype that are inspired by fields like psychology, cognitive linguistics and philosophy, mainly by measuring dissimilarities between objects. We find the pairwise distance between two objects and then, for each object sub-space, we find the mean distance from the object we want to place in n-dimensional term space and thereby assign a category. The aim is to choose the correct categories for the test documents, by adopting the least dissimilar category. Our approach to distance measuring is based on variations of the so-called Minkowski [12, 8, 33] and Canberra [12, 8] metrics. Among others, we will use these notions of dissimilarity to implement Rocchio prototypes.

Our main aim is to utilize the new approach in such a manner that it possibly can outperform the better approaches in the field today. We will try to utilize methods that perform well, both according to correctly categorizing documents, and to save computation time.

1.4 Outline

The rest of the thesis is organized as follows. In Chapter 2 we recall further technical facts about kNN and the Rocchio classifiers; these will serve as our starting points. Then, in Chapter 3 we present the basic ideas about the dissimilarity measures that we will use, based on the the Canberra and Minkowski metrics. The next step is to *evaluate* our newly developed methods; in Chapter 4 we provide the basis for a reliable test of the systems, by taking a brief look at the Reuters collection, at ways of representing documents, and at an example of how the relevant computations are done. In Chapter 5 we present our experimental results, first for kNN, then the Rocchio classifier, and then for a variety of dissimilarity systems. The final chapter of the thesis is devoted to a discussion, conclusion and some thoughts on how to develop the systems from here onwards.

Chapter 2

Two Well-Known Methods in Text Categorization

In this chapter we briefly describe two well-known methods in text categorization: k Nearest Neighbour Classification and Rocchio. These are the two methods against which we will compare our own proposals.

2.1 k Nearest Neighbour Classification

The k *Nearest Neighbour* (kNN) algorithm is a well-known approach in text categorization. It has been in use since the early stages of TC research, and is one of the best performing methods within the field [31].

The basic idea of the kNN approach in TC can easily be explained: calculate a similarity score between the new document and each of the already categorized documents via a data representation model. In this thesis, we adopt a so-called vector space model, where, for each document, each of the words occurring in it is represented by a point in an n-dimensional term space. That is, each document is represented as a vector of weighted word counts, where the weights are relative to the occurrences of the same words in other documents. The documents that have already been categorized are ranked, descending from the document that has the highest similarity score with the document that is to be classified. If $k > 1$ we use the k top ranking documents.

Now, what methodology should we use when we choose categories for our unclassified document? In similarity measures we compare a document to other documents that are already categorized. The documents will then be ranked according to their similarity score with the original document. There are basically two ways of assigning a category for our document to be classified;

- Adopt, from the ranked list, the category from the nearest document in the ranked list

- or for $k > 1$, that is, if $k = 10$, then we will include all documents in the ranked list smaller and equal to the document ranked as number 10. We can then use different means of finding a category for our document, including
  - sum for the individual categories the score for all the documents in which the category occurs.
  - sum for all the categories, the $\frac{1}{rank}$ in which the category occurs

We will go into greater detail about this at a later stage in this thesis. We note that information about each of the categories score are being extracted from a ranked list of similar documents. These documents are already assigned a category, and we can adapt categories from the ranked documents in a number of ways.

When there are documents which have the same similarity level there is a tie. To decide if a category is correct the elements from a category are chosen and divided by the number of all elements that have the same similarity score. When using the k-Nearest Neighbours, we need to adjust the impact of the elements relative to their similarity score, so that documents that have a better similarity score will also count more when comparing the different documents in different classes.

Let's consider an abstract example of categorizing within the kNN approach. A weighted document $\vec{x}$ is to be categorized based on a training set T; we assume $k = 1$. Determine the similarity score between $\vec{x}$ and the elements in the training set $\vec{t} \in T$, and choose all $\vec{t}$ where $\vec{t} = sim_{max}$:

$$sim_{max}(\vec{x}) = max_{\vec{t} \in T} sim(\vec{x}, \vec{t}).$$

Look for the subset of T that has the highest similarity with $\vec{x}$.

$$A = \{\vec{t} \in T \mid sim(\vec{t}, \vec{x}) = sim_{max}(\vec{x})\},$$

let $n_1, \ldots, n_i$ be the number of elements in A that belong to the categories $c_1, \ldots, c_i$ for all $c_i \in C$, respectively. That is, $n_j = |\{d \mid cat(d) = c_j\}|$. Next, we estimate conditional probabilities of membership:

$$P(c_i \mid \vec{x}) = \frac{n_i}{\sum_{i=1}^{n} n_i},$$

and we adopt c_j if $P(c_j \mid \vec{x}) > P(c_k \mid \vec{x})$ for all k where $k \neq j$.

Note that the example only takes care of one nearest neighbour. For $k > 1$, with suitable ties, the above-mentioned approach decides the category based on the majority class of these k neighbours. Here it is feasible to weigh neigbours in a hierarchial manner according to their similarity score. Furthermore, if several of the similar documents share a category then the per-neighbour weights of that category are added together. The resulting sum is used as the basis for comparison with the document to be classified.

Research has shown that the kNN approach performs remarkably well [31, 32]; see Chapter 6 on page 56 for further discussions on evaluation.

2.1.1 Computational Concerns

Let us shortly outline the computational pitfall the above mentioned approach poses; The kNN approach classifies on the intersection between two documents: it checks the weights occurring in both the unclassified document and all classified documents. That is, it compares each of the test documents with each of the documents in the training set. The bigger training set, the higher computational cost. We will later argue more in detail why this is the case. Now, as earlier research [3, 13, 14, 25, 28] has shown the prototype per category remedy this cost.

2.2 Rocchio

Instead of considering similarity of an unclassified document to all docs in a category, a natural alternative is to somehow take a single representative document per category, called a *prototype*, and to compare the unclassified document to each of the different category prototypes. At least intuitively, this is bound to save computation if compared to the kNN approach in the previous section.

We will have more to say about prototypes in the following chapter; for now, we will discuss one of the best known prototype based approaches to text categorization: the *Rocchio algorithm*. Originally the Rocchio algorithm was designed for relevance feedback in information retrieval [14, 13]. Retrieving useful documents for a user-query has always been a challenging problem in the field of information retrieval. For optimized retrieval of correct documents there has to be a good categorization. The Rocchio algorithm has been adapted from information retrieval into the field of text categorization.

The basic idea behind applying the Rocchio approach to text categorization is to construct a prototype vector for each category. For a given category the prototype vector is the category representative vector of the documents assigned to this particular category. There are a number of different approaches to what is considered the optimal prototype vector, and the Rocchio prototype is suggested as one of them. Each weight in the prototype vector is computed by averaging the co-occurring weights from all documents in a given category and subtracting the average of the same co-occurring weights in all documents that do not occur in the category. The result is an efficient method that is relatively easy to implement. The original Rocchio algorithm, as vector-space-model method and relevance feedback algorithm for document routing and filtering in information retrieval, consists of heuristic components that offer a wide variety of possible implementations [14].

The prototype for each category is calculated as the weighted difference between the sum of both the positive and negative training examples. Let us make things more formal. To be able to express this more formally, we need the following notation:

Definition 1 Learning the Rocchio prototype vector represented as $\vec{c_k}$ is achieved by combining each document vector $\vec{d_j}$ in a given category C_k, where C_k is the set of documents in category k and then by calculating the score for each of the co-occurring weights dividing by the number of documents $|C_k|$ in the category. Then subtracting the average score for each of the weights in the document vectors not in this category, $\overline{C}_k$. This gives the prototype vector $\vec{c}_k$ [14, 25, 28].

$$\vec{c_k} = (\alpha\cdot)\frac{1}{|C_k|} \cdot \sum_{i \in C_k} w_{ik} - (\beta\cdot)\frac{1}{|\overline{C}_k|} \cdot \sum_{i \in \overline{C}_k} \overline{w}_{ik}. \tag{2.1}$$

where $|C_k|$ is the number of documents in the category. If an instance in the prototype vector has a value below 0 it is set to 0.

The α and β in (2.1) are parameters that adjust the relative impact of positive and negative training examples. For our experiments we adopt the settings recommended in [3, 14]: $\alpha = 16$ and $\beta = 4$

Once the prototype for each category has been calculated, the unclassified document is compared with the prototype for each category and thus assigned to the category which has the best similarity score. To avoid preference for larger prototypes we will implement the so-called *cosine similarity* between the documents.

Definition 2 For the cosine similarity we use a similarity score *sim* between two vectors $\vec{d_{ij}}$ and $\vec{c_{ik}}$. So $sim(\vec{d_{ij}}, \vec{c_{ik}})$ where each of the co-occurring weights in the vectors are compared. This is denoted by $\bullet$, which again is defined as $\sum_{i=1}^{n}(\vec{d_{ij}} \cdot \vec{c}_{ik})$.

$$sim(\vec{d}, \vec{c}) = (\vec{d} \bullet \vec{c}) = \sum_{i=1}^{n}(\vec{d_{ij}} \times \vec{c}_{ik}),$$

The unclassified document vector is denoted by $\vec{d_{ij}}$ and the prototype vector is denoted by $\vec{c}_{ik}$. We then turn our attention to how we can look at similarities where we normalize the vector length. The cosine similarity cos between two vectors $\vec{d_{ij}}$ and $\vec{c}_{ik}$ is defined as

$$\cos(\vec{d_{ij}}, \vec{c}_{ik}) = \frac{\vec{d_{ij}} \bullet \vec{c}_{ik}}{\|\vec{d_{ij}}\| \times \|\vec{c}_{ik}\|} = \frac{\sum_{i=1}^{n}(\vec{d_{ij}} \times \vec{c}_{ik})}{\sqrt{\sum_{i=1}^{n}(\vec{d_{ij}})^2 \times \sum_{i=1}^{n}(\vec{c}_{ik})^2}}$$

This similarity score is called the *cosine similarity score*; it reduces the relative impact on the similarity score of the larger of the prototypes [2, 6].

All in all, the combination of the two steps just explained (generating prototypes and computing similarity to prototypes) makes the Rocchio approach a very flexible and workable algorithm. Whenever a document is categorized, a calculation for each document is not necessary, but only a comparison to the prototype vectors of the available categories. After assigning the new document to a particular category, the prototype for this category is recalculated. One of the advantages with the Rocchio approach is that there is little need for preprocessing of the data. It is also very computationally attractive compared to many other approaches [24].

2.3 kNN vs. Rocchio

One of the reasons to why text categorization approaches based on prototypes are computationally more attractive than the kNN algorithm can easily be imagined when working with large set of documents. Rather than comparing each new document to every document that is already categorized, one only has to compare the new documents to the median document of each category. A database of documents would normally contain more than 10000 documents, and when the amount of documents increase the demand for processing, when using the kNN, would increase proportionally. This leads to an extremely expensive computation. On reasonable small sets of data there would already be a noticeable difference between kNN and Rocchio, with respect to the average computation time used [26].

However, the more modest computational demands of the Rocchio algorithm do come at a price: the Rocchio approach does not perform as well as the kNN approach [31]; see also Chapter 5. The relatively poor categorization performance and attractiveness in computation is what we will explore in the remainder of this thesis. Our aim in the remainder of this thesis is to try to combine the good computational behavior and the intuitive character of prototype-based approaches to text categorization (such as the Rocchio algorithm) with the good categorization results of methods such as the kNN algorithm. Our first step on this road is to take a closer look at prototypes, and after that we will look at alternative methods of comparing unclassified documents to prototypes.

Chapter 3

Prototypes and Dissimilarities

In this chapter we describe our own proposal for text categorization, based on an effort to combine the prototype-based ideas that we have already encountered in the Rocchio approach with novel ways to use prototypes in the classification process based on dissimilarity instead of similarity.

3.1 Prototypes

Why do we utilize prototypes? Prototype theory, in linguistics and psychology, provides an explanation for the way word meanings are organized in the mind [7, 27, 21, 5]. It is argued that words are categorized on the basis of a whole range of typical features. For example, a prototypical bird has feathers, wings, a beak, the ability to fly and so on. Decisions about category membership are then made by matching the features of a given object against a prototype object where the prototype is said to be the best representative of a category.

Before we delve into our prototype-based approach, we should clarify the nature of a category, its representative prototypes and how we can try to improve already well-known approaches.

3.1.1 Categories and Their Features

Aristotle was among the first to start categorizing nature in hierarchies. He defined a category in terms of a sufficient amount of necessary features. In his classical view, the concept bird consist of a set of necessary and sufficient attributes like for instance wings, can fly, beak, legs. More specifically we can say that the different semantic features are common to several different lexemes in the category, but the members of this category have at least one distinguishing feature as well [33]. In the componential analysis the objects are divided into binary oppositional components, stressing the complementary features of the objects in a given category. By stressing the complementary features we indicate that shared features are more important for an object when categorizing it. It is possible to see a document the same way. As an object that contains different meanings, thus being categorized on the basis of binary decisions. However, as Wittgenstein first pointed out, this classical approach does not suffice. According to [33] he mentions in his Philosophical Investigations that the term "game" is defined not by a set of necessary and sufficient features, but by

> "a complicated network of similarities overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail"

This suggests that a document is to be classified by familiarities, or resemblance structures. This is later adapted by Rosch [5, 33], and we will use the Roscharian sense of Wittgensteins investigations. The objects of a category do not need to have all features in common. A document that completely lacks any resemblance to a given document in a category, can still be correctly categorized as a member of this category. We need to create a representative that accommodates the demands given by Wittgenstein.

What do these deliberations imply for prototype-based approaches to text categorization? It certainly raises the following question: if we can't just use the set of all common features as a prototype, what should we use?

3.1.2 Categories and Their Representation

Categorization is a problem cognitive psychologists have dealt with for many years [29]. The basic principle for creating categories are cognitive economy and the perceived world structure. Here, cognitive economy means that the function of categories is to provide maximum information with the least cognitive effort. The principle of perceived world structure means that the perceived world is not an unstructured set of arbitrary and unpredictable attributes. These two principles are directly translatable for us: cognitive economy is nothing but the computational processing costs and world structure is the vector position in the n-dimensional term space.

Let us take a closer look at some of the underlying ideas. Tests show that there is strong agreement about what counts as the best exemplar of a particular category [33]. For example, most people consider the object "chair" to be the most typical instance of the category "furniture". As we noted above, peripheral category members can be accommodated, because it is not necessary for any one member to possess all the features of the prototype. Thus, despite being flightless, an ostrich can still be classified as a bird, since it possesses other bird-like features.

The median representative for a category has been proposed as a suitable candidate for a prototype within a number of cognitive theories, including Rosch's theory of prototypes [33]. Rosch argues that the task for a category system is to provide maximum information with the least effort. Significantly often, there seems to be a common idea used by people when describing the prototype for a category. Moreover, informants were able to describe in great detail the middle-ranking items in a given taxonomy. This led Rosch to the conclusion that these middle ranking objects were the prototypes around which the taxonomies are based.

Rosch's view was later attacked by, amongst others, Kleiber [33] who claimed that a prototype does not necessarily have to be a real object, but, rather, can be regarded as a mental representation of the best characteristics of a given category. After all, a sparrow, although categorized as a bird, does not define the meaning of bird as such. This indicates that we instead of picking an actual document as the best representative, we compose a document from the documents in the training set which belongs to the given category. Our categories, then, are in need of good informants, that is, a system which can both describe a document and its categories. As we will see later, the TF/IDF weighting scheme is such an informant for the individual documents.

There are, moreover, a few known issues in prototype theory [5, 33] that we need to consider. First, we assume that each category in the prototype theory adopts an idiosyncratic range of criterial features. A problem with this can be to measure distance for documents in the outermost edges of the category. The possibility of having to overcome even more problems are present as our documents sometimes need to be assigned more than one category. Furthermore, decisions about the number and type of features to be included in a prototype are by no means straightforward. And although certain features appear to be more central than others, it has often proved difficult to establish which ones take priority when we make decisions about category membership.

3.2 A New Approach Based on Dissimilarities

With this in mind we implement an, amongst others, psychologically motivated theory for finding similarity by measuring distance between two objects. We will measure the distance between co-occurring weights in the vectors and we will implement different versions of distance checking. The distance between objects in n-dimensional space are summed up and given in a dissimilarity score. First, we will find our prototype by averaging the dissimilarity scores between the individual test documents and all the training documents. By averaging the dissimilarity scores of all training documents in each of the given categories we find our prototypes, our mean documents and their dissimilarity scores. The second way we will make our prototypes is by applying the Rocchio prototypes in our dissimilarity algorithms. And lastly we will apply a simple mean prototype in the same algorithms. After ranking the dissimilarity scores for each of the later given algorithms we choose the least dissimilar score and assign the category for this document.

However in accordance with what we mentioned previously, a member of a category does not necessarily have to have all the quintessential features of a category and this constitutes one of the big problems in choosing an algorithm for making a category representative.

Our underlying assumption is that, when trying to find the similarity score we basically look at the intersection, meaning we calculate only on the weights occurring in both the test and training document. When measuring the distance, we calculate the union between the word weights in two document, that is, weights which also occur in only one of the documents. Of course, where both the test and the learned object, whether it is a prototype vector or a training vector, have a given weight above zero, one of them will be subtracted from the other and their positive result will be added to the sum of the dissimilarity score. For instance, if we have weight n which only occur in the test vector, the value of the weight will be subtracted from the neutral value of the non-existent weight of the prototype vector. However instead of lowering the dissimilarity score we have to add the positive value of the weight. We will choose the category which has the least dissimilar score or the categories which are below a certain threshold of dissimilarity.

We first use the Rocchio vectors but instead of choosing to look for similar features we look for dissimilar features, and, then, by finding the least dissimilar category, we find our match. The promising angle here is that this approach already has a weighted difference between the features of the individual categories and the rest of the categories, thus reducing the number of weights included.

Recall from 2.2 on page 14 that the Rocchio prototype given as

$$\vec{c_k} = (\alpha\cdot)\frac{1}{|C_k|}\cdot\sum_{i\in C_k} w_{ik} - (\beta\cdot)\frac{1}{|\overline{C}_k|}\cdot\sum_{i\in\overline{C}_k}\overline{w}_{ik}$$

Because the Rocchio prototype is designed first and foremost for similarity measures, we will try to adapt an approach that has been utilized in cognitive psychology by, amongst others, S.K. Reed. He defines the prototype in [27] as

> "a prototype P of a category is the central tendency and can be defined as that pattern which has for each component X_m the mean value of the mth component of all other patterns in that category.

In our context we translate and formalize our *simple mean centroid vector* into:

$$\vec{c_k} = \frac{1}{|C_k|}\sum_{i\in C}\vec{d_i}$$

which is nothing more than a centroid vector c_k for category k obtained by the sum of each co-occurring weights i in all the documents in the category k and averaging by the number of documents in the category $|C_k|$.

We do not modify the prototypes' relative value in relation to the other categories, as in the Rocchio prototype. The reason lies in that in our distance measurer the thought is to look at all the included weights in the category vector is a crucial factor when we want to find correct categories.

3.2.1 Dissimilarity Metrics

Now that we have decided to use a notion of dissimilarity as part of our text categorization algorithm, the first question is: how do we measure dissimilarity? In this section we propose two dissimilarity metrics: the Minkowski and the Canberra metrics.

The distance checking is formalized as the *Minkowski metric* [19, 12, 27] where the intra-dimensional weights are subtractive, and the inter-dimensional weights are additive. By intra-dimensional weights we mean that when we check for the distance between two vectors one of the co-occurring weights are subtracted from the other. The absolute value is then added to the final score, which makes up the inter-dimensional addition.

$$D_{jk} = \left(\sum_{i=1}^{n}|x_{ij} - x_{ik}|^{\gamma}\right)^{\frac{1}{\gamma}},$$

where $\gamma \geq 1$ and D_{jk} is the sum of the dissimilarity between vectors that have already been classified x_k and the vector x_j to be classified, for set of weights $i = 1, ..., n$. The result will always be positive as we take the absolute value of the difference.

When $\gamma = 1$ the algorithm is called Manhattan metric whereas $\gamma = 2$ is referred to as Euclidian metric. For values $\gamma > 2$ it is referred to as Supermum metric.

In addition to the Minkowski dissimilarity algorithm we will implement and test the *Canberra metric* with different realizations thereof. This metric has a simple normalizer. This means for us that for each test document we sum up the difference between the vectors co-occurring weights over the sum of the co-occurring weights [8].

$$D_{jk} = \sum_{i=1}^{n}\frac{|\vec{d_{ij}} - \vec{c_{ik}}|}{(|\vec{d_{ij}}| + |\vec{c_{ik}}|)}$$

where D_{jk} is the dissimilarity score between vector document d_j and prototype vector for the category c_k for all the weights i in range $[1, .., n]$. The score will also have to be divided by n to ensure the that the score lies between 0 and 1[12]

Prototypes which have more documents would normally cover a larger term-space area. If we do not normalize the vectors before processing them in our dissimilarity system it would most probably influence the end score for our dissimilarities, and thereby strongly influence the end result for the performance of our system. We aim to reduce the potential influence this has by using a normalization of vector length. We do this so that all vectors have the same influence and possibility to be correctly categorized. We will do this to all vectors, regardless of whether they are document vectors or vectors created from our meta documents, that is prototypes. We will basically focus on two types of vector length normalization. The first way is to convert the vector scores to unit length using the so-called *Versor normalization*:

$$\vec{d} = \frac{\vec{d_i}}{\sum_{i=1}^{n} \vec{d_i}}$$

Earlier experiments [27, 12, 8] have often focused on a normalization method that differentiates somewhat better on the inter-size of the weights in the vectors and moreover the intra-size, meaning each of the internal weights relative size. This is generally done using the *cosine normalization*, defined as

$$\vec{D} = \frac{\vec{D}_w}{\|\vec{D}\|}$$

3.3 Summarizing the Test Scenarios

To conclude this chapter we provide a brief overview of the various set-ups and scenarios that we will using for testing later in the thesis. Our main tests will consist of two main processes, as illustrated in Figure 3.1.
In accordance with figure 3.1 we will basically apply three ways of testing the dissimilarity algorithms. Testing is done by way of both the Minkowski metric and the Canberra metric with vectors in n-dimensional term space, that is shown by following the skip pattern in figure 3.1, we will then test the distance by using a prototype derived from the Rocchio algorithm. Lastly we will test by using a simple mean prototype. For all sets we will use a non-normalized set, a versor normalized set and a set for the cosine normalization for both test and training set. The Minkowski metric will also be tested with different values for γ. We also check if there are differences in the results for set which include categories with $2 \leq$ documents and $50 \leq$ documents.

1. In Canberra metric
    - apply Rocchio prototype for
        - untreated set

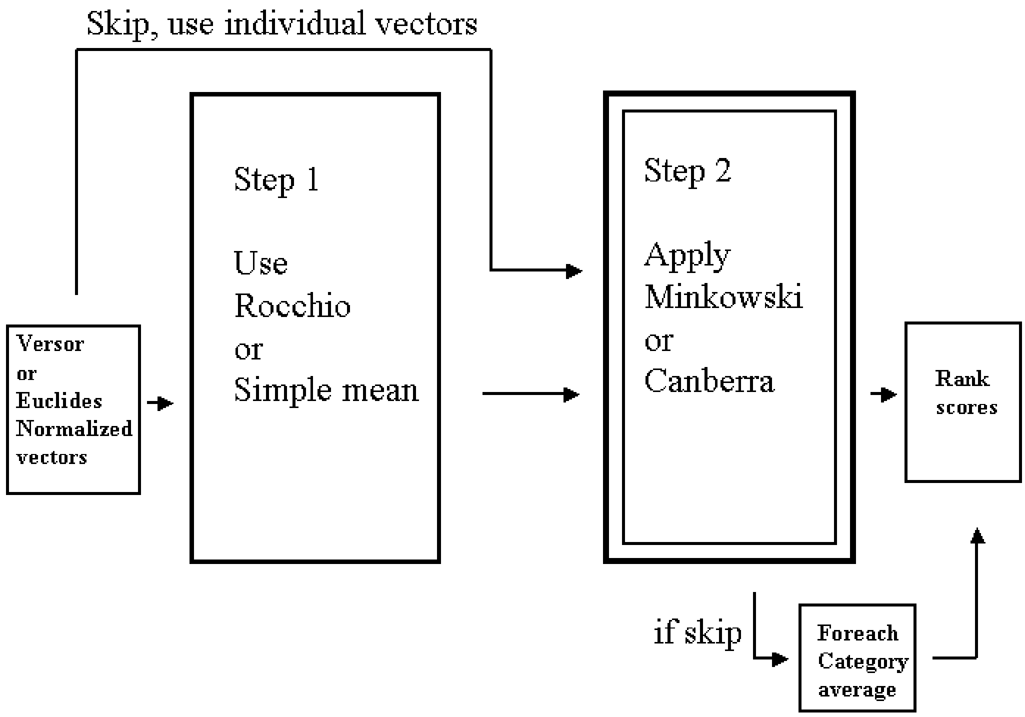

Figure 3.1: New tests

  - Versor normalized set
  - Cosine normalized set

- skip step 1, use untreated documents, then average end-score for each category for
  - untreated set
  - Versor normalized set
  - Cosine normalized set
- apply simple mean centroid for
  - untreated set
  - Versor normalized set
  - Cosine normalized set

2. In Minkowski metric, and realizations therein,
   - apply Rocchio prototype for
     - untreated set

  - Versor normalized set
  - Cosine normalized set
- skip step 1, use untreated documents, then average end-score for each category for
  - untreated set
  - Versor normalized set
  - Cosine normalized set
- apply simple mean centroid for
  - untreated set
  - Versor normalized set
  - Cosine normalized set

Before we can test our metrics, we need to explain our experimental set-up. This involves the document collection, the way we will represent the documents, as well as ways of measuring the performance of our classification methods. All this, and more, will be done in the following chapter.

Chapter 4

Testing the Approaches

Now that we have presented the basic ideas of our approach to text categorization, the next item on our agenda is to evaluate our approach. As theoretical studies do not provide any indication of the effectiveness of the classification methods and their optimizations, this has to be determined by empirical testing. In any case, empirical testing provides a number of benefits over theoretical complexity. It directly gives resource consumption, in terms of computation time and memory use. It factors in all the pieces of the system, not just the basic algorithm itself.

Empirical testing can be used not only to compare different systems, but also to tune a system with parameters that can be used to modify its performance. Moreover, it can be used to show what sort of inputs the system handles well, and what sort of inputs the system handles poorly. In this chapter we present an outline of our testing methodology.

To be able to test and evaluate our results we need to utilize a set of texts that:

- has the desired format, or at least can be modified into such a format
- can be divided into
  - a set for training our systems
  - a set for testing our systems performance
- contains enough documents for us to make a reliable assumption and judgement on the performance of our system
- is already categorized relatively well, which will enable us to use the earlier results as a basis for judging the correctness of our tests
- has enough contextual information, meaning that we can easily extract useful information about the behaviour of our system

4.1 The Reuters Collection

For the evaluation and test of different categorizing approaches we will therefore use a widely acknowledged [30, 14, 26, 11, 23] and distributed corpus originally used for information retrieval purposes. Namely, the Reuters-21578 collection [16]. In recent years the collection has been modified to fit text categorization purposes.

The documents in the Reuters-21578 collection appeared as Reuters newswire stories in 1987. All the documents are indexed manually and they were made available for information retrieval research purposes in 1990. The files where further modified by D.D. Lewis and Peter Schoemaker and in 1996 a SGML tagged version came as the version with less ambiguity [16]. Lewis and Schoemaker found 595 documents which were exact duplicates of other documents in the set. In addition to the removal of these duplicates, all documents were given a new identifier. The resulting collection is known today as Reuters-21578. The collection is distributed in 22 files, each of the files has 1000 documents except the last which has 578. The documents are formatted with SGML labels to identify the different documents category, title, text, places, people and topics.

4.1.1 Format

There are features in the current version of the Reuters collection that are superfluous for our TC purposes; one can think of features such as the inclusion of topic names, which is directly harmful for the reliability of the experiments. This indicates the necessity for a slight modification of the Reuters-21578 version into a version more suitable for our TC purposes. We deleted unlabelled documents, SGML tags and divided the rest of the documents into a training set and a test set.

The training set was used to create a dictionary. A deeper explanation on this will follow in the following section (Section 4.2 on page 28). We also extracted the topics from the sets, identifying the different documents and their categories as seen in the example later on this section. The latter was done for control purposes for our systems.

The assigned documents have labels like

```
<TOPICS><D>earn</D><D>acq</D></TOPICS>
```

to show which categories the document has been assigned. This is an example document to show how the original documents are structured:

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN"
CGISPLIT="TRAINING-SET" OLDID="5555" NEWID="12">
<DATE>26-FEB-1987 15:19:15.45</DATE>
<TOPICS><D>earn</D><D>acq</D></TOPICS><PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE> <ORGS></ORGS> <EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES> <UNKNOWN> &#5;&#5;&#5;F
```

```
&#22;&#22;&#1;f0773&#31; reutes u f BC-OHIO-MATTRESS-&lt;OMT>-M
02-26 0095</UNKNOWN> <TEXT>&#2;

<TITLE>OHIO MATTRESS &lt;OMT> MAY HAVE LOWER 1ST QTR NET</TITLE>
<DATELINE> CLEVELAND, Feb 26 -</DATELINE>

<BODY>Ohio Mattress Co said its first quarter, ending February
28, profits may be below the 2.4 mln dlrs, or 15 cts a share,
earned in the first quarter of fiscal 1986.  The company said
any decline would be due to expenses [...]  including
conducting appraisals, in connection with the acquisitions.
Reuter &#3; </BODY>
</TEXT>
</REUTERS>
```

To format the documents for TC purposes, we had to re-shape them into a structure where the whole set had a unique document identifier, headline and document body. The headline was made part of the document body. In addition, end-of document markers were inserted.

Here's an example of such a cleaned-up document:

```
BEGDOCID_1 OHIO MATTRESS &lt;OMT> MAY HAVE LOWER 1ST QTR NET

Ohio Mattress Co said its first quarter, ending February 28,
profits may be below the 2.4 mln dlrs, or 15 cts a share, earned
in the first quarter of fiscal 1986.  The company said any decline
would be due to expenses [...]  including conducting appraisals,
in connection with the acquisitions.
Reuter
ENDDOC
```

4.1.2 Splitting the collection

Before we could do any experiments, further preparations had to be carried out. For instance, we had to split our collection into documents used for training and for testing. The rules are as follows:

1. we did not use categories (or documents belonging to such categories) which contained at most one document, presuming that the mentioned document only had one or fewer categories assigned to it.

2. we secured that both the training set and test set contained at least one document from each category.

3. the rest of the documents were randomly spread between training set and test set with a preference to divide the sets into 80% and 20%, respectively

To see if there is a difference in the performance between categories containing a small number of documents and categories that contains more documents, we made 2 additional sets. In addition to the split where categories contained 2 or more documents, let's call it the "2-split", we made a set which is split the same way as the previously mentioned set, only this time the criteria for being included is that each of the included categories has to contain at least 50 documents, and the documents have to be a member of one of these categories. We now have a 2-split and a 50-split. Note that these documents come from the same Reuters-21578 set, but to secure the 80%–20% split we have to randomly redistribute the documents.

In the 2-split set the test set contained in the end 2008 documents. These documents had an average of 1.3 categories assigned to them, that makes 2548 required categories for these documents. There were in fact only 339 documents that have more than one category assigned to them. In the training set there were 7847 documents which also had an average of 1.3 categories. The 50-split set, also has 1.3 categories assigned to each document, on average. The different sets were used to see if the basis for categorization was influenced by the number of documents in the categories.

4.2 Document Representation

To be able to compare the different documents, we transformed them into feature vector representations [10] that is, into vectors of weighted word-counts. We followed the following criteria to extract features from the training documents;

- Punctuation marks are separated from words
- Number and punctuation marks are removed
- All words are converted to lowercase
- Prepositions, conjunctions, auxiliary verbs, articles etc. are removed [23]
- Words are replaced with their morphological root form.

Let's make the point mentioned above a bit more concrete; words that have the same word-morphology count as the same word, and we chop the affix of the word stems. Words like

`economics`

and

`economy`

will have the same representation

`econom`

The features derived from our training set of documents were used to create a dictionary containing word stems, word identifiers, and number of times a word occurs in the training set.

Word stem	Word id	No. of occurrences
econom	12	1922
offer	899	1897
industr	317	1893
quarter	758	1842
incr	18	1835

Table 4.1: Example dictionary.

At a later stage, the dictionary was to be used as a means for calculating the relative importance of the occurrence of words in all documents.

Once the dictionary has been created, the individual document vectors are generated. A vector space model represent documents as vectors of weights in n-dimensional space and this, in fact, constitutes our data representation model. This method needs little preprocessing of the data, which is a benefit when we are going to process large amounts of data [13]

Definition 3 *Term frequency*, denoted as tf_{ij}, is the number of times term t_i occurs in document d_j.

Inverse document frequency, the relative occurrence of the term t_i in the whole training set, is denoted as $idf_i = \log\left(\frac{|N|}{n_i}\right)$, where $|N|$ is the number of documents in the training set, and n_i is the number of documents in the training set that term t_i occurs in.

We determine the *weight* w_{ij} of term t_i in document d_j by

$$w_{ij} = tf_{ij} \cdot idf_i$$

The *vector* for document d_j consists of the set of weight from 1 to n as

$$\vec{v_j} = (w_1j, ..., w_nj)$$

Earlier research shows that more sophisticated methods of representing the data, although perhaps intuitively more appealing, have performed worse [18], especially when using noun-phrases instead of words [23].

The result of our weighting scheme is that a term is important if it occurs frequently in one document, and it has a lower frequency in other documents. On the other hand if it also occurs frequently in other documents the weight will be low due to a low inverse document frequency. The inverse document frequency is a useful feature because it helps to lower the relative importance a word if the word seem to be a common term amongst the training documents.

4.3 Choosing Categories

After these preparations, the documents are ready to be compared to other other documents. To find the category that we want to assign to a particular document, we use similarity measures represented by, for instance, the kNN approach, the Rocchio algorithm, or one of our new approaches. Below, we will see an example of how everything is computed. But first we need to show how we select the categories. We use two methods for choosing categories for each test document:

- The first is to simply add the similarity scores from each of the ranked documents to each individual category.
- The second is to, for each of the categories, sum the score of $1/k$, where k is the number rank in the ranked list.

In both approaches the highest scoring category is assigned to the test document. If there is a tie, then all the tied categories are picked.

Because the documents in our document collection have an average of 1.3 categories assigned to them, we need to make an algorithm that will also select categories that are not the top scoring ones. We could just pick two or more categories from the list, but this would probably lead to worse overall performance since the average number of categories for each document is less than two. The choice of how many categories a document is assigned to is a given percent on the basis of the highest scoring category.

4.4 A Worked-Out Example

In this section we take a toy document collection and give a fairly detailed account of the way the documents are represented and classified. Suppose we have a set of four documents as given in Table 4.2, where the training set consist of documents d_1, d_2, d_3, and d_4 is our test set. We can see that the split here is 75% and 25%.

Set	Documents	
Training set	$d_1 =$	Some odd
	$d_2 =$	Some even are more
	$d_3 =$	Some even say examples are odd
Test set	$d_4 =$	Even some more say even examples

Table 4.2: Example documents.

The dictionary is displayed in Table 4.3; it is based on our training set. For reasons of presentation, we have not applied word stemming, and have, instead, included the whole word.

Word	Word number	Number of docs.
Some	1	3
odd	2	2
even	3	2
more	4	1
say	5	1
are	6	2
examples	7	1

Table 4.3: Example dictionary.

In Table 4.4 our documents are represented in a word occurrence matrix where term t_i, i term-identifier and d_j denotes the document, j the document identifier.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
$d_1 =$	1	1					
$d_2 =$	1		1	1		1	
$d_3 =$	1	1	1		1	1	1
$d_4 =$	1		2	1	1		1

Table 4.4: Term frequency matrix.

The term frequency for each document is listed in Table 4.4; the inverse document frequency of a term is found by the logarithm of the total number of documents in the training set divided by the number of documents in training set where $t_i \neq 0$. Note that the vectors in Table 4.4 are not length normalized; they partly serve as an example of how longer vectors have an advantage over smaller vectors. If these documents had been longer, then the similarity scores would potentially be very high. The method would also favour longer documents as they trivially have more words in them.

Table 4.5 shows us that the vector $\vec{v}_4$, the test document has these values: $\vec{v}_4 = \langle 1, 0, 2.44, 1.73, 1.73, 0, 1.73 \rangle$. The default value of a word is 0 so that a word that does not occur just sets the value to 0.
We will pairwise match all document vectors in the training set with our test document vector. It will be done by multiplying the weights $w_1, \ldots, w_7$ for the test document vector $\vec{v}_4$ with the co-occurring weights in all the vectors from the training set. By summing up the scores for each of the calculations and rank them according to their score. The comparison that achieves the highest summed score is the document from which our test vector inherits its category. This is the kNN approach.

From Table 4.6 and for $k = 1$ (that is find only the nearest neighbour)

Doc	WordNo	TF	IDF	$\vec{v}_j$
d_1	t_1	1	1	1
	t_2	1	1.22	1.22
d_2	t_1	1	1	1
	t_3	1	1.22	1.22
	t_4	1	1.73	1.73
	t_6	1	1.22	1.22
d_3	t_1	1	1	1
	t_2	1	1.22	1.22
	t_3	1	1.22	1.22
	t_5	1	1.73	1.73
	t_6	1	1.22	1.22
	t_7	1	1.73	1.73
d_4	t_1	1	1	1
	t_3	2	1.22	2.44
	t_4	1	1.73	1.73
	t_5	1	1.73	1.73
	t_7	1	1.73	1.73

Table 4.5: Calculating weights.

Doc.Number	w_1	w_2	w_3	w_4	w_5	w_6	w_7	Result
d_4	1	0	2.44	1.73	1.73	0	1.73	
d_1	1	1.22	0	0	0	0	0	
$\vec{v}_4 \bullet \vec{v}_1$	1	0	0	0	0	0	0	1
d_2	1	0	1.22	1.73	0	1.22	0	
$\vec{v}_4 \bullet \vec{v}_2$	1	0	2.98	2.99	0	0	0	6.97
d_3	1	1.22	1.22	0	1.73	1.22	1.73	
$\vec{v}_4 \bullet \vec{v}_3$	1	0	2.98	0	2.99	0	2.99	9.96

Table 4.6: kNN similarity scores.

$(\vec{v}_4 \bullet \vec{v}_3) > (\vec{v}_4 \bullet \vec{v}_2)$ and $(\vec{v}_4 \bullet \vec{v}_3) > (\vec{v}_4 \bullet \vec{v}_1)$ then $\vec{v}_4$ will adapt the category of $\vec{v}_3$.

Table 4.7 shows an example of how the Rocchio prototype is computed. For the Rocchio classifier the vectors exists inare given in Table 4.5. The training documents are by definition already classified. Vector $\vec{v}_1$ has category c_1 and $\vec{v}_2$ and $\vec{v}_3$ has c_2 as category. We will compute for c_2. Note that we will here use values for $\alpha = 1$ and $\beta = 1$.

$$w'_{ck} = \alpha \frac{1}{|R_c|} \sum_{i \in R_c} w_{ik} - \beta \frac{1}{|\overline{R}_c|} \sum_{i \in \overline{R}_c} w_{ik}.$$

If $c_j < 0$, then c_j is set to 0.

For column 2, the weights for c_2 are d_2 and d_3 summed and divided by the number of documents in the category. In column 3 the weight of the rest of training vectors, also called the *negative weight*, in this case $\vec{v}_1$, is calculated.

Prototype vector for category 2			
	$\alpha \cdot \frac{1}{\lvert R_c \rvert} \sum_{i \in R_c} w_{ik}$	$-\beta \cdot \frac{1}{\lvert \overline{R_c} \rvert} \sum_{i \in \overline{R_c}} w_{ik}$	w'_{ck}
w_1	1	1	0
w_2	.61	1.22	$-.61$ set to 0
w_3	1.22	0	1.22
w_4	.87	0	.87
w_5	.87	0	.87
w_6	1.22	0	1.22
w_7	.87	0	.87

Table 4.7: Rocchio scores.

This calculation would have been done for all the predefined categories. The similarity would then have been computed the same way as for the kNN, however instead of using the vectors from the training set, we would utilize the new prototype vectors.

Now our attention moves to dissimilarities. We already have the vectors from table 4.5. One of our new classifiers, the Minkowski Metric will be exemplified here. The goal is to find the least dissimilar category for each of the test documents. We have talked about two basic ways of computing dissimilarities (see illustration in section 3.1 on page 23):

1. For each of the documents in the training set, compute the average dissimilarity for each of the categories by first measuring the distance between the test vector and all the training vectors and then average the distance for each of the categories.

2. First make a prototype for each of the categories, and then compute the dissimilarities

We go for the second option.

Assume the mean centroid,

$$\vec{C} = \frac{1}{|R|} \sum_{t \in R} \vec{d},$$

which, for category c_1 is trivially the same as d_1, as there is only one document in this category. For c_2 the average score for the weights in the vectors $\vec{v}_2$ and $\vec{v}_3$ are their sum for the co-occurring weights, divided by the number of documents in this particular category.The values for the weights is the values for $w_1, \ldots, w_7$ in the second column in Table 4.7.

For the Manhattan metric we look for the least dissimilar prototype:

$$\min_i \{ \sum_{w=1}^{n} |\vec{d_4} - \vec{c_i}| \}$$

We also need dissimilarities between the vector for document 4, $\vec{v}_4$ and the simple mean centroid vector for category 2, $\vec{C}_2$ is smaller than the dissimilarity between vector for document 4, $\vec{v}_4$ and simple mean centroid vector for category 2, $\vec{C}_1$, thus category 2 is assigned to d_4.

Doc. No.	w_1	w_2	w_3	w_4	w_5	w_6	w_7	sum diss.
d_4	1	0	2.44	1.73	1.73	0	1.73	
c_1	1	1.22	0	0	0	0	0	
$\vec{v}_4 \bullet \vec{c}_1$	0	1.22	2.44	1.73	1.73	0	1.73	7.12
c_2	1	.61	1.22	.87	.87	1.22	.87	
$\vec{v}_4 \bullet \vec{c}_2$	0	.61	1.22	.87	.87	1.22	.87	5.66

Table 4.8: Dissimilarity scores.

An example for the Canberra metric would not look very different in this case. Each of the dissimilarity scores would be the average dissimilarity score between two weights.

In conclusion, we have seen that all of our systems will categorize document 4 in category 2.

4.5 Performance Measures

The conclude this chapter, we discuss the performance measures that we will use in our experiments.

After computing similarities and dissimilarities we need a method for measuring the correctness of our findings. The primary goal is not only to find the categories originally assigned to the test documents, but also to not include categories which are not assigned to the test documents originally. To be able to reliably measure the performance of the different systems, we use widely accepted methods for measuring [31]:

- precision
- recall
- f-score

These will be explained shortly.

For each document we look for categories to assign, also called category ranking. Each of our classification algorithms returns, for each of the test documents in the given test set, a ranked list of categories; recall and precision can then be computed at any threshold on this ranked list. We can classify the categories above threshold four ways. For binary classifiers a two-ways contingency table for each category is used to evaluate the assignments. The conventional performance measures as recall and precision are defined and computed from this table. They are calculated for each of the categories where the category has a document which is a member of the test set.

In Table 4.9 the correctness of how the test documents are assigned, or have categories assigned to them are counted in relation to the original assignment.

		Original assignment	
New assignment		Yes is correct	No is correct
	Assigned Yes	a	b
	Assigned No	c	d

Table 4.9: Process of assignment.

Here

- cell a counts categories correctly assigned to the test set
- cell b counts categories incorrectly assigned to the test set
- cell c counts categories incorrectly rejected from the test set
- cell d counts categories correct rejected from the test set

From these scores we calculate the recall, precision and f-score, in the following manner:

$$\text{Recall} = \frac{\text{a}}{\text{a} + \text{c}},$$

where we assume $a + c > 0$, otherwise recall is undefined. Recall measures the algorithm's performance relative to the original assigned documents. For instance, if there are 2000 categories originally assigned to the different test documents, and the given approach has 1500 categories correctly assigned to the test set, then our recall score will be .75. By itself this may seem like a reasonable score, but if these 1500 correct categories comes from 9000 categories assigned all together, then the score is not impressive.

Therefore the precision measure is introduced.

$$\text{Precision} = \frac{\text{a}}{\text{a} + \text{b}},$$

where we assume $a + b > 0$, otherwise precision is undefined. Using the example mentioned above, this would give us, from the 9000 assigned categories, where 1500 were correctly assigned, a precision score of .16. This score indicates that although we found a lot of correct categories, we picked too many. Moreover it indicates that if we only examine the recall it might be somewhat misleading.

Recall and precision scores usually exhibit a trade-off in the classifier, when adjusting parameter internally. We would like to find a score which gives us one score from the precision and recall score. If we use interpolated "break-even point" (BEP) [31], which is the average of the precision and recall score, then the BEP will not always reflect the true behaviour of our classifier. That is, if the precision and recall score are far apart, as our example, we do not think of .46 as a reasonable performance for our system. To find a single numbered measure that more accurately reflects what we think is correct we turn to Van Rijsbergen's *f-score* [31].

$$\text{f-score} = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}.$$

Our example will then give us a score of .26, which seems like a more reasonable performance indicator for a system that picked 9000 categories instead of 2000. As a further illustration, note that if we had a recall score of .99, which basically meant to sacrifice the precision to an extent of .0001, then instead of getting a BEP score of .50 we will get a f-score of .0002.

Chapter 5

Results

In this chapter we present our experimental results. We start out but setting the baseline scores, by looking at the kNN and Rocchio classifiers. After that, in Section 5.3, we present results obtained using systems based on dissimilarities. To help the reader keep track of the many options we consider, a brief overview at the start of Section 5.3.

5.1 Results for kNN

Picked categories. For kNN we chose categories from the ranked list two ways as we mentioned in Chapter 3 on page 17. The first way to assign categories is the $\frac{1}{k}$. The results are seen in Table 5.1 on page 38. This way of finding the appropriate category adapts each of the categories for the chosen k nearest neighbour documents. Then simply sum, for each category, all the $\frac{1}{k}$ values from the ranked training documents, where the category rank is represented. k is the rank from the similarity scores. In practice this will mean for $k = 4$, that if the 1 nearest neighboring ($k = 1$) document has "lead" as the category, and the correct category is "grain", then the next three training documents will have to have "grain" as one of their categories to be able to correctly assign grain to our test document as $\frac{1}{2} + \frac{1}{3} + \frac{1}{4} > \frac{1}{1}$. The category(ies) which has the best scores will be assigned to the document.

The results shows that the 2-split set peak for the f-score at $k = 10$. The f-score is here 0.81178. The precision is here 0.8939 whereas the recall is 0.74. This may indicate that many of the found documents are correct, but that we should try to pick more categories. This is done by looking at the final best scoring category, and then by looking for categories that performed almost as good as the best scoring category.

Our intuition of having to pick more categories is confirmed when we delve deeper into the results. All in all there, for the 2-split and $k = 10$, are 1820 out of 2036 found categories correctly assigned to the test documents. These test documents would, to have a f-score of 1, need 2448 documents found and correctly assigned. To remedy, or more precisely to check, if there is a significant difference in kNN's ability to classify with a higher probability of correctness when the categories have a minimum of 50 documents we utilize the 50-split. Different approaches perform comparably when categories are sufficiently common [32].

split	specified	k	precision	recall	f-score
2	0	10	0.8939	0.74346	0.81178
2	30%	20	0.84855	0.8125	0.83013
50	0	100	0.87479	0.71903	0.7893
50	20%	150	0.8229	0.78334	0.80264

Table 5.1: kNN, $\frac{1}{k}$ as category selector.

In Table 5.1 and Table 5.2, column one denotes from which set the results come. Second column is what kind of percent frame was used to choose categories relative to the top-scoring category. 0 denotes that only the top scoring category has been chosen. The column labelled with k indicates how many nearest neighbours that were used.

The same way that we showed the scores for $\frac{1}{k}$ we display the scores for the categories chosen by summing the similarity scores for each of the categories represented in the ranked training vectors. Highest scoring category, as seen in Table 5.2 is assigned to the document. If there is a tie, all the categories in the tie will be assigned to the document.

split	specified	k	precision	recall	f-score
2	flat	8	0.8896	0.74429	0.8105
2	20%	8	0.85733	0.7953	0.8252
50	flat	75	0.88632	0.7285	0.7997
50	10%	75	0.8483	0.761	0.80228

Table 5.2: kNN, highest similarity score category selector.

This approach (Table 5.2) seems to be performing just as well as the $\frac{1}{k}$ approach. However, the performance decreases relatively rapid as k increases. An example of the decrease is illustrated by the graph in Figure 5.1. This is probably due to the fact that the Reuters collection has 27 (not including the 18 categories that only contain one or no document) categories which has five or less documents in them. This will inevitable affect results when k reaches larger number since the similarity scores run out of documents from the correct category. Extensive categories will then, naturally, be preferred over categories with a smaller set of documents because the documents from this particular category becomes exhausted.

We have seen that one of the problems for kNN is that our documents are not supposed to have only one category. It is not necessary to choose either this or that category, it may be that both are correct. As indicated we tried to include the top scoring category and the categories that has a score which is almost as good. We did this by including categories that did not differ more than a given percentage from the top-ranked category score. This approach shows, as seen in Table 5.2, a score of about $2-3\%$ better than to pick only the top scoring category. It has a best f-score of 0.83 for 30% diversion, weighted $k = 2$. The problem is that this particular matrix shows signs of fluctuation compared to $\frac{1}{k}$ approach also for 30%. This has a top score of 0.83 for k = 20. What is interesting is that the lowest score for this approach is 0.8, which is about 0.06 better that the lowest scoring 30% weighted k. This, of course, has to be included when making a decision about which approach is preferred in the end.
When looking at the graphs in Figures 5.1 and 5.2, it is evident that the set containing two or more documents per category performs slightly better than its opponent here. However, the better performance is at a lower value for k, and the set where each category has fifty or more documents seem overall to be performing more steadily.

This experiment shows an important development in that it seems to indicate that as the minimum number of documents per category increases, the number of neigbours that has to be included in the similarity score increases. For the test set where there are two documents or more per category, we see a slight decline in performance as the number of neighbours increase.

5.2 Results for the Rocchio prototype

The Rocchio algorithm is designed to first create a prototype representative per category, and then rank these representatives according the similarity scores. Then adapt the most similar category to the test document. As expected the best results came when choosing only the most similar category. It has an f-score of .3. However if when we look a bit deeper into the results it seems like the Rocchio is distinguishing quite well between the different super-categories [16]. When we perform a vector length normalization on the prototype it self, we note an overwhelming increase in performance as seen when comparing the graphs in Figure 5.3.

For the prototypes we select categories by assigning the best similarity score and the scores relative to that score.

In Table 5.3, we can see the performance of sets that are not normalized. When we then turn our attention to Table 5.4 we can see a noticeably difference for the cosine normalized set. We have also included score for the 50-split. The score was, as seen in Table 5.4 slightly better for the

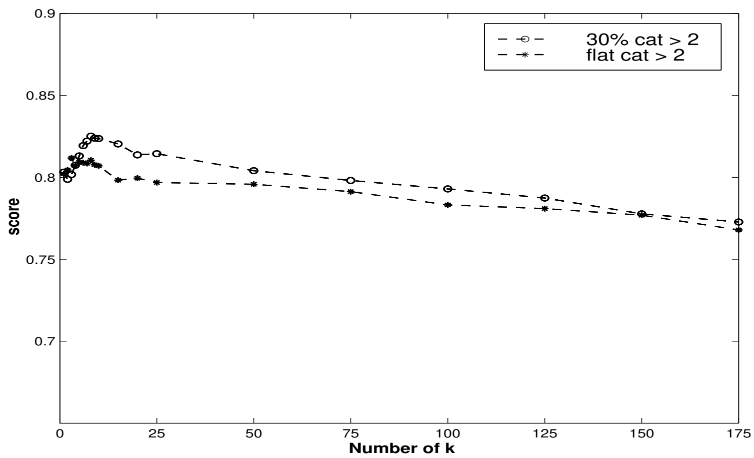

Figure 5.1: kNN graph for cats > 2.

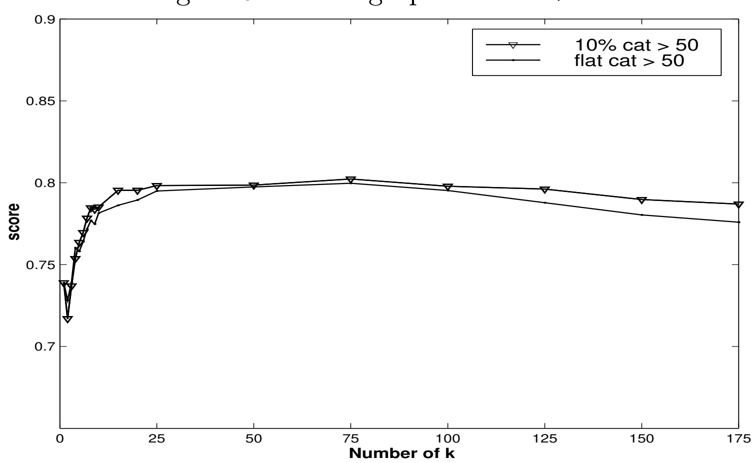

Figure 5.2: kNN graph for cats > 50.

Percent	a + c	a + b	a
0	2446	2008	687
10	2446	2819	810
20	2446	4136	982

Table 5.3: Not normalized Rocchio similarities.

normalized set evaluated.

As mentioned, there is only a slight improvement in the performance for categories holding more than 50 documents. We note that earlier research [14] has indicated, although not in comparison with smaller sets and with a different kind of document representation, a better performance for

Percent	a + b	a	Rec	Prec	f-score
0	2578	1480	.61	.57	.58
10	6289	1926	.79	.31	.44
20	15262	2126	.87	.13	.24
50 > 0	2542	1447	.64	.57	.60

Table 5.4: Normalized Rocchio similarities.

large set. By large category we here mean well over 100 documents. The reservation lies, for us, in the incomparable results this test showed as it computes the accuracy instead of computing precision and recall. Computing accuracy has pitfalls for sets where the number of categories are large and the number of documents per category is low.

The difference between normalized set and an unmodified prototype is illustrated graphically in Figure 5.3.

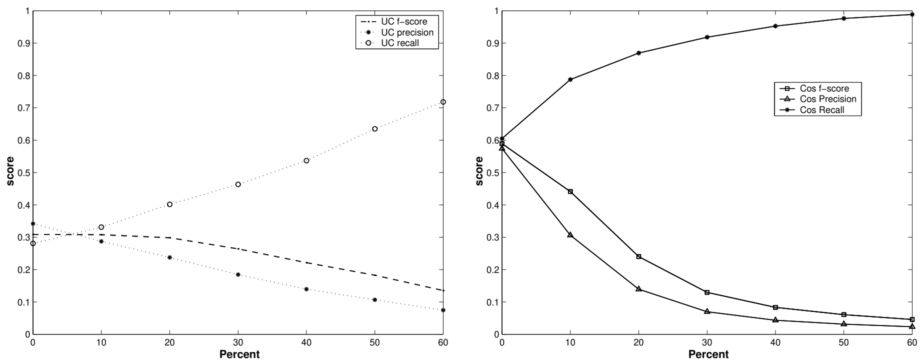

Figure 5.3: Results for Rocchio.

Regarding the coding in Tables 5.3 and 5.4 and the graphs in Figure 5.3 we note that Table 5.3 has not been normalized, whereas Table 5.4 has been cosine normalized. From the graph we can see that, as expected due to the average number of categories in the test set being 1.3, the Rocchio algorithm has its best performance when we only pick the best scoring categories. The result also provides strong indications that it is necessary to normalize the similarity scores due to the noise introduced by the prototypes internal size difference.

5.3 Results for Dissimilarities

We have tested and received results from dissimilarity systems for different sets and document normalization. Included are also some other interesting scores, not only the better performing ones. These results can serve well to explain the nature of the selection process and in what way the better performing algorithms are successful in choosing categories for the different documents representation sets. First we have a closer look at the results for the Canberra algorithm. Due to the incorporated normalization we also tested it for a non-normalized set.

Note that the method for choosing more categories are basically the same as for the kNN approach. For each of the test documents we choose categories in the ranked list by their score relative to the least dissimilar score. It is, however, useless to select by including via the rank as the categories themselves are ranked. If we were to include more categories, based on the rank, we would always get the same number of categories for each of the test documents. Due to the nature of the dissimilarity scores we choose not to increase by 10% at the time but by 1%.

The tables consist of columns where

- "set" is the explanation of what set we tested the algorithms
  - "All" conveys that all the vectors from the training set are used as basis for comparison, and the prototype is then calculated as described in Figure 3.1 on page 23
  - "Roc" is then that the system use the Rocchio prototypes as basis for comparison
  - "Ctr" denotes results from the purely mean Centroid vectors
- "nrm" describes if we used document length normalization and which one we used. These are coded as
  - "NoN" is sets where no document length normalization has been applied
  - "Ver" denotes the Versor vector size normalization
  - "Ecl" indicates that we have a set normalized by the Euclides vector size normalization
- "Ref" is this particular result reference number for referral later in the text
- "split" explains how many documents there was in each per category for the category to be counted as valid.
- "Rec" is the recall
- "prec" is the precision
- "f-scr" is the f-score.

- for the Minkowski metric we also include %/R. It denotes the per cent or number of Ranking (R) that are included in the results the same way as for the Rocchio similarity score
- we also include a value for γ, as given in the formula 3.2.1 on page 3.2.1, which is the variable included in the algorithm.

For the dissimilarity score for the new prototypes we have also looked into the performance the system had on the ranking of individual categories. The reason to this was that we needed more information on how, and why the metric performed. We evaluated, for $4-6$ different categories, the average rank in documents that were supposed to have the particular category above threshold for selection and compare them to the average rank for the categories for documents where they are not supposed to be above threshold. These categories are fundamentally different in that they contain everything from 2 to over 700 documents. This is a potentially crucial point in that some of our systems might prefer smaller or larger categories. We will also see details of how the size of the categories included in the test set can influence the performance.

The results for these tests are tables where

- "Categ" denotes the category that are measured and the mentioned categories are:
  - "lit"
  - "austdlr"
  - "ipi"
  - "alum"
  - "money-supply"
  - "acq"
  - "earn"
- "a+c" is the number of documents in the test set that are supposed to have the particular category assigned to it
- "avr. for a+c" is the average rank the category in fact had when it was supposed to be assigned to the test document
- "avr. for rest" is the average rank the category had when it was not supposed to be assigned to the test document

Note that both "lit" and "austdlr" have less than 50 documents in them, so they will not have any average value when we test for sets which has 50 or more documents in them.

5.3.1 Canberra

First, we have tested the Canberra metric and compared all the weights in the test-set vectors with the weights in training-set vectors, and then averaged the dissimilarity scores for each category.

CANBERRA MEASURES							
Set	Nrm.	Ref.	Cat	a	Rec	Prec	f-scr
All	NoN	2	2	9	0.004	0.004	0.004
		3	50	1022	0.45	0.51	0.48
	Ver	4	2	10	0.004	0.005	0.005
		5	50	942	0.417	0.47	0.44
	Ecl	6	2	10	0.004	0.005	0.005
		7	50	956	0.42	0.48	0.45

Table 5.5: Results from the Canberra metric.

In Ref. 2 we use an unweighted set and we note that the performance gives no grounds for optimism. But if we look a bit deeper below the surface we can see that Ref. 3, from the same dissimilarity scores as 2, only here we used the 50-split, the system suddenly finds over a thousand correct categories. For Ref. 4 and 5 we have the Versor normalization, and we observe that the same way as Ref. 2 and 3 it performs better when only including categories with fifty or more documents in them. The first issue we notice is the enormous difference between the f-scores for the dissimilarity scores where the categories which has over 2 documents per category and over 50 documents per category are included. In fact if we look at Ref. 2 we can see that it has an impressive lack of ability to choose the correct categories. Compared to Ref. 3, which are test done in the same context, but with the 50-split. The correctly assigned documents increase by over 1000. This may suggest that, as this is a test done on a non-normalized set, the Canberra metrics built in normalizer works to a certain degree, but that it, as expected might not be a very good length normalizer. The same goes for sets that are Versor normalized (Ref. 4 and 5) and Cosine normalized (5 and 6). Mind you, these scores are done on sets where we average the sum of the individual document dissimilarity scores for each category.

It is also worth noting that the scores suggest a certain degree of noise from the categories containing a less number of documents, due to the fact that the general performance increase as much as it does when the number of documents in the categories increase. This might not be the ideal way of categorizing on sets that has big differences in how many documents each category contain, specially when there is a very low number of documents per category.

The results from Ref. 2 and 3 were not normalized before the test. Whereas the result for Ref. 2 has an amazingly poor performance at 0.004, the result for Ref. 3 is better. Not very good, but there is a significant difference compared to Ref. 2. If we look at the internal results from the same categories as mentioned above, we immediately locate a possible reason, namely interference from categories that has fewer documents in them.

Note that the assumption is that categories containing more documents, in general, has more weights.

Catg.	a+c	avr. for a+c	avr. rest
lit	1	1	9.4
austdlr	1	2	2.22
ipi	8	33.75	57.7
alum	13	45.08	57.19
money-supply	22	34.14	66.93
acq	449	51.82	74.78
earn	709	31.55	77.73
made for Ref. 2.			
ipi	8	1.38	10.39
alum	13	3.39	8.91
money-supply	22	2.41	18.82
acq	449	6.62	26
earn	709	4.59	28.94
made for Ref. 3.			

Table 5.6: Internal results for Ref. 2 and Ref. 3.

The categories “lit” and “austdlr” have either 1 or 2 documents in the test set. All in all the category austdlr has 3 documents in it, which means that there are only 1 or 2 documents in the training set. For Ref. 2 it correctly assigns the category to the correct test vector, however it seem like the extremely low number of documents in it affects the rest of the categorization task. It is probably due to the fact that the dissimilarity measures on all the weights. That makes it vulnerable to categories with few number of weights in them. By not using categories which has less than 50 documents increases the correctly assigned documents by 950 (Ref. 3) strongly supports this hypothesis.We can also see the consistency in being vulnerable to categories that contains fewer weights in them by looking at the relations between Ref. 4 and 5 and Ref. 6 and 7 respectively. They have different document length normalization, but still the same phenomena occurs.

Figure 5.4 on page 46 illustrates quite well a situation where a skewed

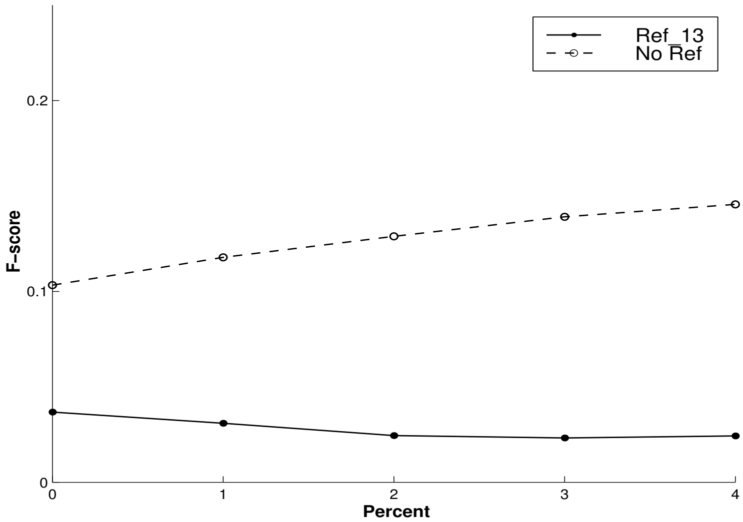

Figure 5.4: Graph for the mean centroid in the Canberra system.

document set can pose problems in the categorization task. It looks on the surface here as if one of the graphs, No Ref, performs somewhat better than the other. The other Ref. 13 has a performance which obviously is bad with a best f-score of 0.04. The No ref. seems to be performing somewhat better, however it does fundamentally the same mistake as Ref. 13, only it picks a category which occur more often.

It is also evident from our testing that this approach with the median category representatives is not worth the effort, as we did not get any promising results from the tests. The latter may be due to the fact that there is such a big difference in number of weights between test vectors and the prototype vectors. Different realizations of the built-in vector length normalizer weight different aspects of the vectors as more important. However it seems like there is too much difference when weights that occur in both vectors are compared to weights that only occur in one of the vectors. If we set each of the weights that the vectors do not have in common to 1, the categories dissimilarity score are only marginally different. The difference seem often go in favour of the smaller categories. This is not the case for the Minkowski algorithm

5.3.2 Minkowski

One of the motivations for testing the Minkowski approach is to see if it is possible to categorize the documents with a better f-score than the kNN. In particular, we want to try to adapt an approach that performs better in a prototype context. We developed and adapted the psychological approach where, instead of looking for similarities, we we look at differences, and then we choose the least dissimilar category or categories. We first tested our Rocchio prototypes to see if we were able to perform better than when looking for similarities.

For the Minkowski scores we use mostly the same tables as before, only there is an addition namely that %/R denotes the per cent or number of Ranking (R) that are included in the results. 0 % means that only the least dissimilar category is chosen, else it is chosen by including from the ranked categories on the basis of the percent deviation from the least dissimilar score.

First, we look at the Rocchio prototype score. For the Versor length normalization we can superficially see that there is only a .2 difference between the test for categories > 2 documents (Ref. 15.1), and > 50 documents (Ref. 16.1). The more interesting score here is that its peak performance is when including all categories where the rank score is not higher than 3% (Ref. 15.2 and 16.2) of the lowest. We can see that the amount of found and correct categories increase by 500 when including all categories that has not a higher dissimilarity score than 3% from the score of the least dissimilar category. This is quite unusual, as we will see later, and it shows a good ability to discriminate the weights with a higher margin.

Earlier we have seen that by increasing the number of categories the precision dropped rapidly, but up to the mentioned 3% the precision is stable and the recall increases, which leads to a peak for the 2-split and 50-split tests with an f-score of .37 and .39 respectively. Even though this may seem like an acceptable score, if we compare it to the earlier mentioned scores we see that it is not a very good score compared to similarity or to dissimilarities scores.
Tests, in Table 5.7, for both the Versor and the cosine vector length normalization, on the Rocchio vectors performs for the Minkowski algorithm, as we see in the table above, stable but not impressively compared to earlier shown results from both similarity and dissimilarity scores. The algorithm does not, on the surface, seem to be noticeable affected with interference from vectors which had fewer weights. A possible and plausible explanation is that the Rocchio vectors has been normalized, and that the relative impact for weights not occurring in both the test vector and the prototype vector does not affect the end dissimilarity score as much as for the Canberra metric. However, as we delve under the surface of the scores, the Minkowski for Rocchio does nothing more than disclose a weakness in the compatibility of the prototype and Minkowski metric. Let's try to understand why.

When looking at the scores we see in Table 5.8 that the results comes from picking the "earn" category as the least dissimilar category regardless of it being the correct choice for the test vector or not. It prefers the category, the same way as we discussed for Figure 5.4. There is a tendency to move

MINKOWSKI MEASURES for Rocchio								
γ	Nrm	Ref.	%/R	split	a	Rec	Prec	f-scr
$\gamma = 1$	Ver	15.1	0%	2	709	0.29	0.36	0.32
		16.1	0%	50	708	0.31	0.36	0.33
		15.2	3%	2	1262	0.52	0.3	0.38
		16.2	3%	50	1261	0.56	0.31	0.4
	Ecl	17	2%	2	1160	0.47	0.29	0.36
		18	2%	50	1158	0.51	0.3	0.38
$\gamma = 2$		19.1	0%	2	163	0.07	0.08	0.07
		19.2	1%	2	1724	0.71	0.019	0.03
		20.1	0%	50	288	0.13	0.15	0.14
		20.2	1%	50	2231	0.99	0.03	0.06

Table 5.7: Results form Minkowski for Rocchio prototypes.

Catg.	a+c	avr. for a+c	avr. rest
lit	1	76	80.13
austdlr	1	82	82.2
ipi	8	20.625	27.37
alum	13	28.77	31.55
money-supply	22	8.41	14.11
acq	449	2	2.0006
earn	709	1	1.0008
made for Ref. 15.			
ipi	8	20.63	27.4
alum	13	28.77	31.4
money-supply	22	8.41	14.12
acq	449	2	2
earn	709	1	1
made for Ref. 16.			

Table 5.8: Internal results for Ref. 15 and 16.

in the right direction for the categories that are supposed to be categorized, however it is not enough to make it interesting. We suspect that one the reasons for this is that the Rocchio algorithm is designed for the opposite task, namely to look for similarities, and that trying to apply it in the context of dissimilarities obviously lead to misclassification.

Catg.	a+c	avr. for a+c	avr. rest
lit	1	1	77.088
austdlr	1	81	82.19
ipi	8	5.63	17.42
alum	13	2.31	23.54
money-supply	22	9.68	22.81
acq	449	38.6	49.46
earn	709	33.93	51.51
made for Ref. 19.			
ipi	8	1.25	10.52
alum	13	1.38	14.64
money-supply	22	3	13.95
acq	449	30.06	34.18
earn	709	23.98	35.84
made for Ref. 20.			

Table 5.9: Internal results for Ref. 19 and 20.

To push this point one step further, we can look at scores for the same categories for Ref. 19 and Ref. 20 where $r = 2$. The tests for the cosine normalized Rocchio came back with a significantly worse f-score, however we can still argue that it would be more interesting to investigate these scores more closer than the Ref. 15 and 16 due to the fact that there is more movement in the average category rank.

As for the Canberra metric, we also need to look at the dissimilarity scores for tests of each of the vectors for the training set.
As for the some of the results in the Canberra metric, we notice immediately that the average dissimilarity score for each of categories for the Minkowski metric also seem to be influenced by noise. The difference between noise, as mentioned in the Canberra metric, and what is characterized as unpromising results is that the noise still has an overall improvement of the ranking of categories, while the mentioned unpromising results do not seem to, generally speaking, improve the average rank for most of the categories. The tests were done for both Versor- and cosine normalized training set where $\gamma = 1, 2, 3$. They were measured for both 2-split and 50-split.

For $\gamma = 1$ the best scoring normalization were the Versor, for the 50-split where the f-score is .5. The score displays that we get over half of the assigned and wanted categories correct. Finding if there are more correct categories to include in the results is done by looking at, and including, the 1% of the categories within the neighboring dissimilarity scores. It shows that the scores for this calculation is marginal. When choosing categories within 1% of the lowest score it more than doubles the number of categories.

MINKOWSKI MEASURES for All								
γ	Nrm	Ref.	%/R	Split	a	Rec	Prec	f-scr
	Ver	21	0%	2	11	0.005	0.006	0.005
$\gamma = 1$		22	0%	50	1087	0.48	0.54	0.51
	Ecl	23	0%	2	11	0.005	0.006	0.005
		24	0%	50	766	0.34	0.38	0.36
	Ver	25	0%	2	10	0.004	0.005	0.005
$\gamma = 2$		26	0%	50	197	0.09	0.1	0.09
	Ecl	27	0%	2	14	0.006	0.007	0.006
		28	0%	50	929	0.41	0.46	0.43
	Ver	29	0%	2	10	0.004	0.005	0.005
$\gamma = 3$		30	0%	50	177	0.078	0.088	0.082
	Ecl	31	0%	2	13	0.005	0.006	0.006
		32	0%	50	587	0.26	0.29	0.28

Table 5.10: Results from Minkowski for full sets.

Catg.	a+c	avr. for a+c	avr. rest
lit	1	1	9.36
austdlr	1	2	2.32
ipi	8	26.25	56.16
alum	13	30.39	60.29
money-supply	22	24.64	63.12
acq	449	54.92	79.3
earn	709	29.77	82
made for Ref. 21.			
ipi	8	1.375	10.4
alum	13	2.15	11.86
money-supply	22	1.86	16.99
acq	449	8.68	30.44
earn	709	5.23	33.15
made for Ref. 22.			

Table 5.11: Internal results for Ref. 21 and Ref. 22.

Even though the scores are marginal we can see from the example categories that Ref. 22 does in fact seem to move correct categories towards

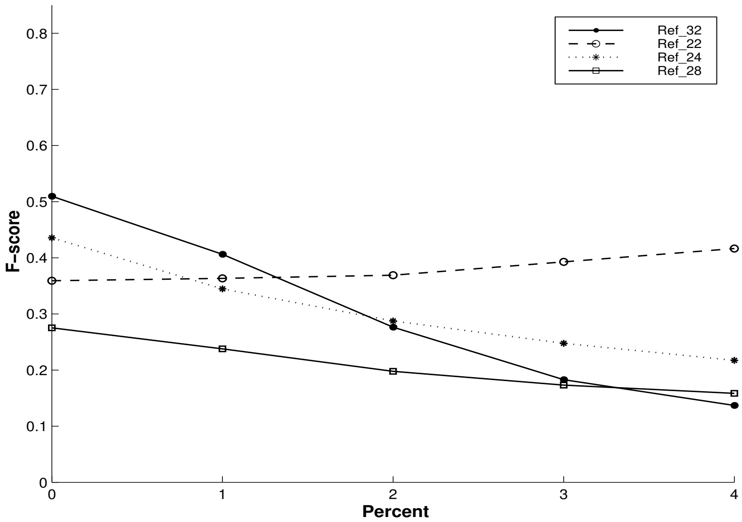

Figure 5.5: Graphs for Ref. 22, 24, 28 and 32.

correct ranking. However it does give the impression of being preferring categories that has fewer weights all together in the categories. Probably due to the nature of the Versor normalization where all documents are of unit length. Moreover it still outperforms the cosine normalized version of $\gamma = 1$ (Ref. 24). But looking at the cosine normalized sets for $\gamma = [2, 3]$ it is noticeable that the Versor version is outperformed. This might have something to do with the mentioned marginal score. Here we see that the sets which are cosine normalized is not by far as marginally calculated. When checking for categories the top f-score for Ref. 24 is when including 5% of the averaged categories dissimilarity scores.

In Figure 5.5 it is clear that the scores for Ref. 24 do not have the same high peak as for Ref. 22. Ref. 24 does not have its peak before 5%, which means that the scores discriminate each of the scores better. However it is natural to assume that the score would not increase more as the system at this point has picked a few categories more than are wanted for the documents. Ref. 28 has its top f-score at 0%, but it on contrary to Ref. 22 it holds better the f-scores. The problem is of course that Ref. 22 is the best scoring approach for these tests.

There is a fundamental difference between the way the similarity approach and dissimilarity approach searches for a category. Namely that the dissimilarity takes the mean average dissimilarity score for each of the category and then adapts the category, whereas the kNN adapts the closest documents category. To see if there is a significant difference we tried to look for the least dissimilar document, and then adapt its category. If we picked the four least dissimilar documents we received a f-score of .60. However we did not at all exhaust these tests due to the fact that we want to look for prototypes that has a lower CPU cost. If we compare these results to the kNN and the Rocchio it is accurate to say that there still are differences for the scores, but it is still not big enough to call the scores so far uninteresting. As the earlier mentioned dissimilarity results only in aspects can compare with the results in the similarity scores we continue the search for a improvement of some sort.

Simple Mean Prototype One of the problems with the approach where we look for dissimilarities by calculating each of the documents dissimilarity and then average the dissimilarity score for each of the categories is the same as for the kNN; it is computationally more expensive then for instance the Rocchio approach. In addition to what we have been through so far we have tested to find the least dissimilar score for the categories where the score is a dissimilarity score between a normalized version of the test set and a normalized version of the simple mean for each of the categories. That is we made a simple mean prototype vector for each of the categories. Then we did the dissimilarity scores.

MINKOWSKI MEASURES for CENTROID

γ	Nrm	Ref.	%/R	split	a	Rec	Prec	f-scr
	Ver	33	0%	2	1232	0.5	0.6	0.6
$\gamma = 1$		34	0%	50	1473	0.65	0.76	0.70
	Ecl	35	0%	2	772	0.32	0.38	0.35
		36	0%	50	760	0.33	0.4	0.36
	Ver	37	0%	2	1378	0.65	0.8	0.71
$\gamma = 2$		38	0%	50	1565	0.69	0.81	0.74
	Ecl	39	0%	2	1619	0.66	0.8	0.72
		40	0%	50	1614	0.71	0.84	0.77
	Ver	41	0%	2	1378	0.56	0.69	0.62
$\gamma = 3$		42	0%	50	1480	0.65	0.77	0.7
	ecl	43	0%	2	1431	0.59	0.72	0.64
		44	0%	50	1411	0.62	0.73	0.67

Table 5.12: Results for simple mean centroid.

The scores are substantially better than before in the dissimilarity scores. We can see that the scores are not only sporadically better, but consistently better. Both the 2-split and 50-split have almost equally good scores. We

Catg.	a+c	avr. for a+c	avr. rest
lit	1	1	54.02
austdlr	1	17	42.06
ipi	8	2.25	29.03
alum	13	4	44.95
money-supply	22	2.8	22.77
acq	449	3.45	39.46
earn	709	1.65	34.8
made for Ref. 33.			
ipi	8	1.5	14.34
alum	13	2.85	21.93
money-supply	22	1.32	10.95
acq	449	2.04	18.28
earn	709	1.28	16.88
made for Ref. 34.			
lit	1	1	63.13
austdlr	1	9	55.92
ipi	8	1.875	28.1
alum	13	2.15	39.91
money-supply	22	2	24.67
acq	449	1.36	14.58
earn	709	2.03	37.07
made for Ref. 39.			
ipi	8	1.75	17.61
alum	13	2.08	24.31
money-supply	22	1.68	15.31
acq	449	1.22	9.12
earn	709	1.76	21.03
made for Ref. 40.			

Table 5.13: Internal results for Ref. 33, 34, 39 and 40.

can see from the internal categorization of our test categories that for Ref. 33 has a mishap for the smaller category “austdlr”. This approach seem to consistently rank both categories with fewer documents, the medium sized category and the larger categories quite well.

By nrt we mean, “Not Referred to”, and that these results are from experiments that we have not presented yet, however they serve as a good examples, on how the scores vary under certain circumstances. In figure 5.6 we illustrate how a selection similar to that of the kNN where we pick, in

addition to the least dissimilar centroid representative, also the least but 1 dissimilar prototype. That is, the "percent" in Figure 5.6 also means that in addition to the least dissimilar, we also picked the next, or the two next, and so on. It illustrates, for Ref. 40 and its ranked Ref. 40 that if we pick the two lest dissimilar documents, that is making the average number of categories per document two, then the score falls, but not as much as expected, because in this process it manages to pick up over 300 new correct categories. It is not enough to keep the good, but it does indicate that there are possibilities to scoop up these by adjusting variables even more. The same illustration for Ref. 41 and its ranked version shows less success. The interesting part here is that it is not the ranked version that decreases its f-score so radically — it is Ref. 41 itself. It seems like the algorithm's ability to discriminate scores is in fact so low that by including the categories that has a dissimilarity score that is only one percent deviation from the least dissimilar score it assigns over 17000 new categories to the test set. Only 300 are also correct.
Figure 5.7 illustrates the different normalization development in variations over Minkowski system. That is over different values for γ. In this particular graph. What is interesting here is that all examples have their peak at $\gamma = 2$. The next point is to see that the absolute peak here at f-score = .77 is Ref. 40. We can see that, whereas the cosine normalized set, where each category included has over fifty documents, has an f-score under .40 at $\gamma = 1$ the Versor normalization performs more steadily. It is also notable that the categories where we have less than fifty documents do not seem to influence the performance as much as the other tests. This can also be seen in a context of traditional use of the Rocchio classifier where it usually does not

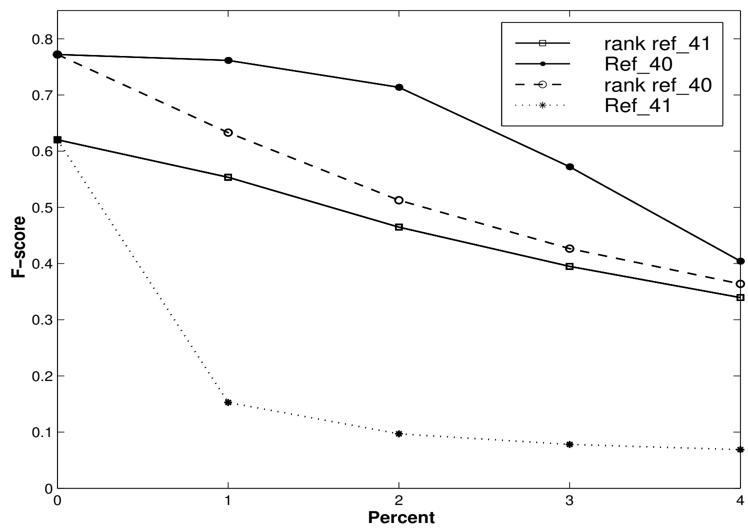

Figure 5.6: Graphs for Ref. 40, nrt, 41, nrtII

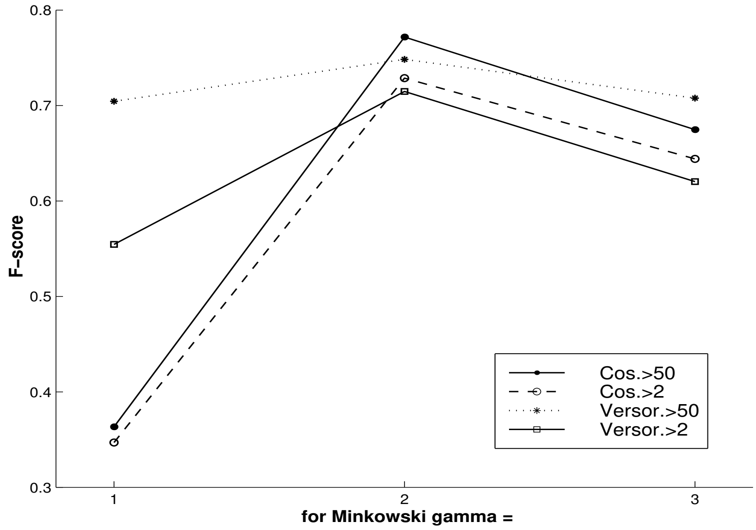

Figure 5.7: Best results from mean centroid in Minkowski.

perform very well on smaller categories [31, 14].

All in all, the Minkowski is a quite well performing categorization algorithm for the mean centroid. The Manhattan metric for cosine normalization has the lowest f-score. The lower score is probably due to the lower relative impact of non-conjunctive dissimilarity scores. When we use the Euclidean version of the Minkowski metric this is adjusted and the higher dissimilarity scores will have a bigger relative impact on the overall score for this category.

Chapter 6

Discussion and Conclusion

In this, the final chapter of the thesis, we summarize our findings, try to understand and explain them, and identify a number of avenues for further work.

6.1 Summary and Main Findings

In this thesis we have presented different approaches to to text categorization. The famous kNN classifier, the Rocchio prototype algorithm was introduced as a basis for comparison. We have tried to create an approach which performs better than the kNN. However we have as earlier discussion showed found that there might me reasons to delve deeper into the materia of dissimilarities, or even more so the cross approach algorithms.

We have implemented two new approaches where instead of finding the closer documents and adapt their categories, we measured the distance between vector in N-dimensional space. We received extremely varied results from these experiments.

There was two algorithms implemented namely the Canberra metric and variations of Minkowski metric[1].

All in all, we have seen that some approaches have a clear advantage over other techniques. We have drawn the following conclusions based on the results from the experiments:

- For the kNN approach:
  - the 2-split has a higher peak for the general f-score
  - the 50-split set has a generally more steady performance for the f-score
  - 50-split peaks at $k = 75$ which indicates that the k value should be increased with larger sets

[1] Variations called: Manhattan, Euclidian and Supermum

- For the Rocchio-based approach:
  - Rocchio applied in the Minkowski metric also performs best when applied with categories that has more than 50 documents
  - The subtraction of negative training in Rocchio prototype is not only a crucial loss for us when we want to utilize the Minkowski metric. It also supports the notion that we should include support for the documents that do not have the necessary and crucial features, but only resemblance structures.
- For the Canberra metric:
  - In our tests the Canberra metric needs a less skewed set, and performs at its best when categories has more than 50 documents in them
  - the built in normalizer does not affect or contribute in any positive way
- For the Minkowski metric:
  - performs well, both with the simple mean centroid and the set where we use all document vectors and then find the average distance. Especially we did see a good performance with the Cosine normalization and Euclidian configuration of the Minkowski metric.

Overall, then, the Minkowski was the most successful one of the new approaches. However, the Canberra metric had better scores for tests on collections with no length normalization to reduce the relative impact of high dimensional vectors. This was of course due to the metrics built in normalization.

The algorithms were both tested with different sets where we had Versor, Euclidian and no vector length normalization. The Euclidian normalization was definitely the better performing in these tests.

By comparing results from a standard within the field of text categorization, namely the kNN, we found that our approaches in part could measure and compete with highly recognized approaches within the field of text categorization. Because our centroid prototype is computationally less expensive we can conclude that it is well worth further research.

6.2 Digging Deeper

Earlier work [26] states that prototypes are computationally less expensive than for instance kNN. It is not hard to see why. Anyway, for a document to be assigned to a category it must have some traits in common with already categorized documents, which in turn indicates that for each document vector that is added to the corpus the number of dimensions added is more if the document vector is a stand-alone vector than for it to be included in a category where we know it has at least some weights in common. It is also natural to assume that the bigger the corpus the fewer added weights per category. For our dissimilarity scores we looked at what the machine had to do. If we assume that the average number of weights for each vector is 100, then comparing all the test vectors to all the training vectors (as we would have to in the kNN approach), would lead to approximately 1500000000 calculations. For the prototypes the number of calculations is about, taking into account that each prototype would have more than 100 weights, 150 weights per category, we would have to do 26000000 calculations.

For our tests this leaves us with processing times for the psychological prototype and on the average cat, the machine use on average 9.5 sec (on a standard PC) to process each test document, while on for the dissimilarity scores the machine use approximately one second per test document. The difference in processing time would probably increase as we add more test documents.

We have seen that the Canberra system had problems categorizing when the number of weights per category vector varied too much. As expected, it had a tendency to prefer smaller categories because each of the weights not included in both the test vector and the training vector are assigned the dissimilarity score one. Bigger categories have of course a tendency to have more weights, and the average dissimilarity score will increase with the number of weights not in the conjunction between two vectors. This becomes less important when the number of weights in the vectors are more balanced. Still, we would expect that the scores would be somewhat better if the number of documents per category were slightly more distributed or even increased for the whole set. Compared to Minkowski the performance is not very interesting.

Why is the Minkowski system using the simple mean prototype performing better than both the Rocchio similarity, and the mean dissimilarity score for each category? It seems that, for the mean average, by normalizing each of the vectors and then averaging the dissimilarity scores, each of the documents has the same relative impact of the mean of the category. This implies that documents that are smaller, and have lower number of weights, will have the a higher impact on the dissimilarity scores, even though this effect should be minimized when using the Euclidian document length normalization. When we make a simple mean prototype for each of the categories, then the documents receiving higher scores for the weight have a bigger say in determining where the prototype is placed in n-dimensional space.
We can see an illustration of how a document can be assigned the wrong category in figure 6.1. The average distance for test document D_1 from the correct category C_1 is longer than for the incorrect category C_2, which means

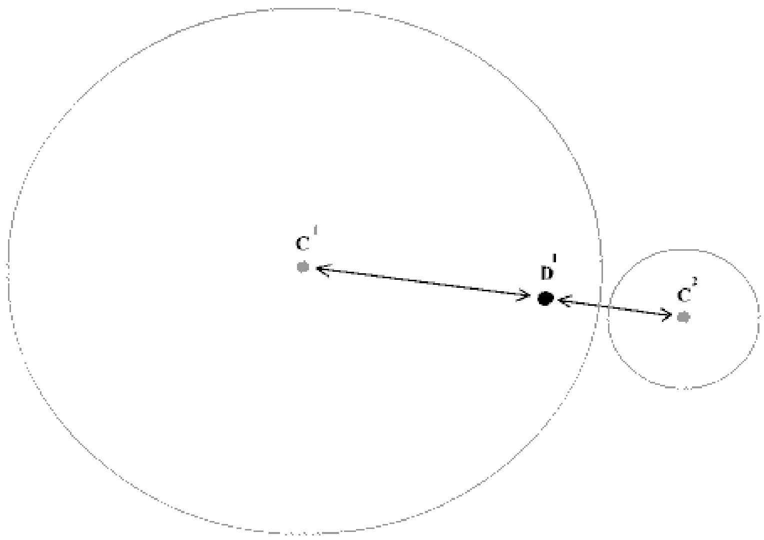

Figure 6.1: How an incorrect category is chosen

that the incorrect category is chosen. When calculating the simple mean prototype the notion is that more correct weights are given more attention and that bigger documents, or document clusters with a higher density will have more weight when deciding place for the centroid in the category. We have seen, in the tests, that it beats all the other new tests, it beats the Rocchio similarity and it can nearly be compared to the performance of the kNN similarity approach when it comes to precision and recall. It has an overall good performance for sets where the number of documents per category varies a great deal. It performs well on all categories and it seems like the performance is relatively invariant over different variations in the Minkowski system. The peak is definitely for the set cosine document length normalization and the Euclidian variation of Minkowski algorithm.

If this is the case for the centroid prototype, then why does the Rocchio vector perform relatively poorly in the different dissimilarity scores? We know that the Rocchio vectors are designed for similarity scores, and that they have, at least compared to the dissimilarity scores, alright scores there. Although it has been highly specialized through consistent research, it does not seem to be able to justify and uphold its reputation from information retrieval as legendary in any way in text categorization. It is probably because by including the algorithm into the dissimilarity scores we really try to join two radically, and fundamentally different algorithms. The Rocchio prototype is weighted against, and normalized relative to the other categories. This seem to make the normalization of the vectors unstable relative to the normalized test vectors. It will not have the same effect on similarity scores as one of the more crucial factor there, is the fact that there is weights from the same dimension, not necessarily their score. At least not in the same degree.

When first comparing Rocchio similarity approach with the Minkowski dissimilarity approach, it returns the notion of the dissimilarity being less complex, from an implementation point of view. On this set it is definitely a better performing approach. There have been earlier research where Rocchio came quite well out from the test, however this test computed with accuracy, which for different reasons are a bad way for to measure the performance. The reason being the categories varies as much as they do in our set.

The two of the best performing algorithms, the kNN, and the simple mean Minkowski, performs, both well, and almost equal for both the 2-split and 50-split. This imply that it, for each of the vectors, has a reasonable vector length normalization, and that each of the weights seem to be given, for their respective categories, an evenly and justifiably distribution relatively to the internal weighting scheme. It seem to be a pattern that correctly are imposed on a more complex reality. It does assist in mediating a perception for the system in such a way, that it perceives the test and training vectors correctly and thereby guiding the system to a correct response. Further more it also imply that a well performing and balanced algorithm will perform well also for smaller categories.

Given the nature of the prototypes, why does not Rocchio perform as well the mean centroid? In the sets we have seen, for variations of the Minkowski system that the mean centroid outperformed the Rocchio prototype. In the process of making the Rocchio centroid we find the relative weight against the rest of the documents weight for the same feature. It decrease the domain of the prototype, and the number of features to be classified on. However in our set there are some categories that has only two (or fifty) documents per category. When a centroid has many weights, as for instance “earn”, then there is a better chance of having more weights that co-occur, that is, the chance for a decreased dissimilarity score heightened by the number of weights in the centroid. More notably so for skewed sets.

6.3 Future Work

Here are a number of suggestions for further work:

- It would be natural to look closer at each of the categories to look for reasons why these particular test documents are or are not assigned correct categories. Since the correct categories are assigned by humans that it might be worth taking a closer look at which documents are classified correct / incorrectly, and why.

- Develop an environment that choose system on the basis of the corpus ahead. This implies that we would use kNN for small sets, but for larger set it would be naturally to choose and assign categories by the use of the mean prototypical vector and by the dissimilarity scores.

- It would also be wise to include tests on different methods for selecting weights in a vector. To be more strict, and not include weights that have a low value would probably save even more computation. There are different methods currently available today [26, 11, 24, 6, 4, 1, 20].

- A re-consideration and further development of the current weight encoding system where one incorporated and suggested solutions on current problems. Problems such as contextual use of terms, where equal word-stems has different meanings in different contexts and problems with the use of synonyms. I think there is a huge potential when looking at how to be able to find the reminder, or at least increase the f-score, of the correct categories.

- All our experiments were based on the Reuters collection. To further validate our findings, additional experiments with other document collections are called for.

- Further compare our best performing new method (Minkowski with centroids) to the kNN approach, especially with a view to understanding when we have to give up kNN because it becomes prohibitively expensive from a computational point of view.

Bibliography

[1] Alexa. Computer-assisted analysis methodology in the social sciences. Zuma-Arbeitsbericht 97/07, Zuma, Mannheim, October 1997.

[2] Ribeiro-Neto Baeza-Yates. *Modern Information Retrieval*. Addison Wesley, 1999.

[3] G. Salton C. Buckley and J. Allan. The effect of adding information in a relevance feedback environment. Technical report.

[4] Cohen. Learning rules that classify e-mail. Technical report, AT & T Laboratories.

[5] Connell. Categories, concepts and co-occurrence: Modeling categorization effects with lsa. Master's thesis, Division of informatics at University of Edinburgh, 2000.

[6] Eui-Hong Han G. Karypis. Concept indexing -a fast dimensionality reducing algorithm with applications to document retreival & categorization. Technical report, University of Minnesota, Dept. of Computer Science / Army HPC Research Center, March 2000.

[7] Henry Gleitman. *Basic Psychology*. University of Pennsylvania, 1992.

[8] A.D. Gordon. *Classification*. 2nd edition edition, 1982. monographs on statistics and applied probability.

[9] C. Maccing H. Solitze. Foundations of statistical natural language processing. Technical report, MIT Press, 1999.

[10] K. L. Low H. T. Ng, W. B. Goh. Feature selection, perceptron learning, and usability case study for text categorization. Technical report, Ministry of Defense, Gombak Drive, Singapore.

[11] Kumar Han, Karpyris. Text categorization using weight adjusted k-nearest neighbour classification. Technical report, Dept. of CS, University of Minnesota.

[12] D.J. Hand. *Discrimination and classification*. Wiley Series in probability and Mathematical Statistics.

[13] E. J. H. Kerckhoffs H.C.M.de Kroon, T.M. Mitchell. Improving learning accuracy in information filtering. Technical report, Carnegie Mellon University, School of Computer Science, Pittsburg, USA and Delft University of Technology, Faculty of of Technical Mathermatics and Informatics, Delft, The Netherlands.

[14] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. *Logic J. of the IGPL*, 1998.

[15] Ho Lam. Using a generalized insance set for automatic text categorization. Technical report, The Chinese UNiversity of HongKong.

[16] David D. Lewis. Distribution 1.0 readme file (v1.2) for reuters-21578. AT&T Labs - Research, 1997.

[17] T. Mitchell. *Machine Learning*. McGraw Hill, NY, US, 1996.

[18] C. Monz. Computational semantics and information retrieval. Technical report, ILLC, UvA.

[19] Each Penn. Psychological anthropology. Theoretical Statement for doctoral exam, 1999.

[20] Ellen Rilhoff and Wendy Lehnert. Information extraction as a basis for high-precision text classification. Acm, University of Massachusetts, 1994.

[21] L. M. Rocha. Evidence sets: Modeling subjective categories. Technical report, Complex Systems Modeling Team, Los Alamos National Laboratory, New Mexico.

[22] Sebastiani. Machine learning in automated text categorization. Technical report, Consigilo Nazionale delle Rieche, Italy.

[23] F. Sebastiani. A tutorial on automated text categorization. Technical report, Instituto di Elaborazione dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy.

[24] Karypis Shankar. Weight adjustment schemes for a centroid based classifier. Technical report, Dept. of Computer Science, University of Minnesota.

[25] Buckley Singhal, Mitra. Learning routing queries in a query zone. Technical report, AT & T Labs Research, Dept. ofCS, Cornell University, Sabir Research.

[26] David B. Skalak. Prototype selection for composite nearest neighbour classifiers. Technical report, Dept. of CS. University of Massachusetts, 1995.

[27] S.K.Reed. Pattern recognition and categorization. *Journal Cognitive Psychology*, (3):pages $382 - -407$, 1972.

[28] Kumar Steinbach, Karypis. A comparison of document clustering techniques. Technical report, Dept. of Computer Science and Engineering, University of Minnesota.

[29] Takkinen. An adaptive approach to text categorization and understanding. Technical report, Dept. of Computer and Information Science, Linkoping, 1995.

[30] Yiming Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval.

[31] Yiming Yang. An evaluation of statistical approaches to text categorization. *Logic J. of the IGPL*, 1998.

[32] Yiming Yang and Xin Liu. A re-examination of text categorization methods.

[33] Michaela Zitzen. On the efficiency of prototype theoretical semantics. http://ang3−11.phil-fak.uni-duesseldorf.de/ ang3/LANA/Zitzen.html.