Application of K-tree to Document Clustering
Masters of IT by Research (IT60)

Chris De Vries
Supervisor: Shlomo Geva
Associate Supervisor: Peter Bruza

June 23, 2010
“The biggest difference between time and space is that you can’t reuse time.”
- Merrick Furst

“With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.”
- Attributed to John von Neumann by Enrico Fermi

“Computers are good at following instructions, but not at reading your mind.”
- Donald Knuth

“We can only see a short distance ahead, but we can see plenty there that needs to be done.”
- Alan Turing
Acknowledgements

Many thanks go to my principal supervisor, Shlomo, who has put up with me arguing with him every week in our supervisor meeting. His advice and direction have been a valuable asset in ensuring the success of this research. Much appreciation goes to Lance for suggesting the use of Random Indexing with K-tree as it appears to be a very good fit. My parents have provided much support during my candidature. I wish to thank them for proof reading my work even when they did not really understand it. I also would not have made it to SIGIR to present my work without their financial help. I wish to thank QUT for providing an excellent institution to study at and awarding me a QUT Masters Scholarship. SourceForge have provided a valuable service by hosting the K-tree software project and many other open source projects. Their commitment to the open source community is valuable and I wish to thank them for that. Gratatitude goes out to other researchers at INEX who have made the evaluation of my research easier by making submissions for comparison. I wish to thank my favourite programming language, python, and text editor, vim, for allowing me to hack code together without too much thought. It has been valuable for various utility tasks involving text manipulation. The more I use python, the more I enjoy it, apart from its lacklustre performance. One can not expect too much performance out of a dynamically typed language. Although, the performance is not needed most of the time.

External Contributions

Shlomo Geva and Lance De Vine have been co-authors on papers used to produce this thesis. I have been the primary author and written the majority of the content. Shlomo has proof read and edited the papers and in some cases made changes to reword the work. Lance has integrated the semantic vectors java package with K-tree to enable Random Indexing. He also wrote all the content in the “Random Indexing Example” section, including the diagram. Otherwise, the content has been solely produced by myself.
Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature

Date
# Contents

1 Introduction ................................................. 9  
1.1 K-tree .................................................. 10  
1.2 Statement of Research Problems ............................ 10  
1.3 Limitations of Study ....................................... 11  
1.4 Thesis Structure ......................................... 11  

2 Clustering .................................................. 12  
2.1 Document Clustering ....................................... 12  
2.2 Reviews and comparative studies ........................... 13  
2.3 Algorithms ................................................. 14  
2.4 Entropy constrained clustering .............................. 14  
2.5 Algorithms for large data sets .............................. 16  
2.6 Other clustering algorithms ................................ 20  
2.7 Approaches taken at INEX ................................. 22  
2.8 Summary ................................................... 23  

3 Document Representation .................................... 24  
3.1 Content Representation ..................................... 24  
3.2 Link Representation ....................................... 24  
3.3 Dimensionality Reduction .................................. 25  
3.3.1 Dimensionality Reduction and K-tree .................... 25  
3.3.2 Unsupervised Feature Selection .......................... 26  
3.3.3 Random Indexing ....................................... 26  
3.3.4 Latent Semantic Analysis ............................... 26  
3.4 Summary ................................................... 27  

4 K-tree ....................................................... 28  
4.1 Building a K-tree ......................................... 31  
4.2 K-tree Example ............................................. 33  
4.3 Summary ................................................... 33  

5 Evaluation .................................................. 37  
5.1 Classification as a Representation Evaluation Tool ........... 38  
5.2 Negentropy ............................................... 38  
5.3 Summary ................................................... 39
6 Document Clustering with K-tree

6.1 Non-negative Matrix Factorisation ........................................ 42
6.2 Clustering Task .................................................................. 43
6.3 Summary ......................................................................... 45

7 Medoid K-tree

7.1 Experimental Setup ..................................................... 47
7.2 Experimental Results .................................................. 48
  7.2.1 CLUTO ............................................................. 48
  7.2.2 K-tree ............................................................. 49
  7.2.3 Medoid K-tree .................................................... 49
  7.2.4 Sampling with Medoid K-tree ................................ 50
7.3 Summary ......................................................................... 57

8 Random Indexing K-tree

8.1 Modifications to K-tree .................................................. 58
8.2 K-tree and Sparsity ....................................................... 59
8.3 Random Indexing Definition ....................................... 60
8.4 Choice of Index Vectors ............................................... 60
8.5 Random Indexing Example ........................................... 60
8.6 Experimental Setup ..................................................... 61
8.7 Experimental Results .................................................. 62
8.8 INEX Results ............................................................. 63
8.9 Summary ......................................................................... 63

9 Complexity Analysis

9.1 k-means ....................................................................... 67
9.2 K-tree ........................................................................ 68
  9.2.1 Worst Case Analysis ............................................... 68
  9.2.2 Average Case Analysis ......................................... 71
  9.2.3 Testing the Average Case Analysis ......................... 72
9.3 Summary ......................................................................... 73

10 Classification

10.1 Support Vector Machines .......................................... 74
10.2 INEX ....................................................................... 75
10.3 Classification Results .................................................. 75
10.4 Improving Classification Results ................................. 76
10.5 Other Approaches at INEX ......................................... 77
10.6 Summary ..................................................................... 79

11 Conclusion

11.1 Future Work ............................................................... 80
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>K-tree Legend</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Empty 1 Level K-tree</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>1 Level K-tree With a Full Root Node</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>2 Level K-tree With a New Root Node</td>
<td>32</td>
</tr>
<tr>
<td>4.5</td>
<td>Leaf Split in a 2 Level K-tree</td>
<td>32</td>
</tr>
<tr>
<td>4.6</td>
<td>2 Level K-tree With a Full Root Node</td>
<td>33</td>
</tr>
<tr>
<td>4.7</td>
<td>3 Level K-tree With a New Root Node</td>
<td>33</td>
</tr>
<tr>
<td>4.8</td>
<td>Inserting a Vector into a 3 Level K-tree</td>
<td>34</td>
</tr>
<tr>
<td>4.9</td>
<td>K-tree Performance</td>
<td>35</td>
</tr>
<tr>
<td>4.10</td>
<td>Level 1</td>
<td>35</td>
</tr>
<tr>
<td>4.11</td>
<td>Level 2</td>
<td>36</td>
</tr>
<tr>
<td>4.12</td>
<td>Level 3</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>Entropy Versus Negentropy</td>
<td>39</td>
</tr>
<tr>
<td>5.2</td>
<td>Solution 1</td>
<td>40</td>
</tr>
<tr>
<td>5.3</td>
<td>Solution 2</td>
<td>40</td>
</tr>
<tr>
<td>6.1</td>
<td>K-tree Negentropy</td>
<td>42</td>
</tr>
<tr>
<td>6.2</td>
<td>Clusters Sorted By Purity</td>
<td>44</td>
</tr>
<tr>
<td>6.3</td>
<td>Clusters Sorted By Size</td>
<td>45</td>
</tr>
<tr>
<td>6.4</td>
<td>K-tree Breakdown</td>
<td>46</td>
</tr>
<tr>
<td>7.1</td>
<td>Medoid K-tree Graphs Legend</td>
<td>50</td>
</tr>
<tr>
<td>7.2</td>
<td>INEX 2008 Purity</td>
<td>51</td>
</tr>
<tr>
<td>7.3</td>
<td>INEX 2008 Entropy</td>
<td>52</td>
</tr>
<tr>
<td>7.4</td>
<td>INEX 2008 Run Time</td>
<td>53</td>
</tr>
<tr>
<td>7.5</td>
<td>RCV1 Purity</td>
<td>54</td>
</tr>
<tr>
<td>7.6</td>
<td>RCV1 Entropy</td>
<td>55</td>
</tr>
<tr>
<td>7.7</td>
<td>RCV1 Run Time</td>
<td>56</td>
</tr>
<tr>
<td>8.1</td>
<td>Random Indexing Example</td>
<td>61</td>
</tr>
<tr>
<td>8.2</td>
<td>Purity Versus Dimensions</td>
<td>66</td>
</tr>
<tr>
<td>8.3</td>
<td>Entropy Versus Dimensions</td>
<td>66</td>
</tr>
<tr>
<td>9.1</td>
<td>The k-means algorithm</td>
<td>69</td>
</tr>
<tr>
<td>9.2</td>
<td>Worst Case K-tree</td>
<td>71</td>
</tr>
</tbody>
</table>
List of Tables

6.1 Clustering Results Sorted by Micro Purity . . . . . . . . . . . . . 43
6.2 Comparison of Different K-tree Methods . . . . . . . . . . . . . 44
8.1 K-tree Test Configurations . . . . . . . . . . . . . . . . . . . . . 63
8.2 Symbols for Results . . . . . . . . . . . . . . . . . . . . . . . . . 64
8.3 A: Unmodified K-tree, TF-IDF Culling, BM25 . . . . . . . . . . . 64
8.4 B: Unmodified K-tree, Random Indexing, BM25 + LF-IDF . . . . 64
8.5 C: Unmodified K-tree, Random Indexing, BM25 . . . . . . . . . . 65
8.6 D: Modified K-tree, Random Indexing, BM25 + LF-IDF . . . . . 65
8.7 E: Modified K-tree, Random Indexing, BM25 . . . . . . . . . . . 65
9.1 UPDATEMEANS Analysis . . . . . . . . . . . . . . . . . . . . . . . 68
9.2 EUCLIDEAN DISTANCESQUARED Analysis . . . . . . . . . . . . . 68
9.3 NEARESTNEIGHBOURS Analysis . . . . . . . . . . . . . . . . . . . 68
9.4 K-MEANS Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . 70
10.1 Classification Results . . . . . . . . . . . . . . . . . . . . . . . . . 76
10.2 Classification Improvements . . . . . . . . . . . . . . . . . . . . . 77
Digital collections are growing exponentially in size as the information age takes a firm grip on all aspects of society. As a result Information Retrieval (IR) has become an increasingly important area of research. It promises to provide new and more effective ways for users to find information relevant to their search intentions.

Document clustering is one of the many tools in the IR toolbox and is far from being perfected. It groups documents that share common features. This grouping allows a user to quickly identify relevant information. If these groups are misleading then valuable information can accidentally be ignored. Therefore, the study and analysis of the quality of document clustering is important. With more and more digital information available, the performance of these algorithms is also of interest. An algorithm with a time complexity of $O(n^2)$ can quickly become impractical when clustering a corpus containing millions of documents. Therefore, the investigation of algorithms and data structures to perform clustering in an efficient manner is vital to its success as an IR tool.

Document classification is another tool frequently used in the IR field. It predicts categories of new documents based on an existing database of (document, category) pairs. Support Vector Machines (SVM) have been found to be effective when classifying text documents. As the algorithms for classification are both efficient and of high quality, the largest gains can be made from improvements to representation.

Document representations are vital for both clustering and classification. Representations exploit the content and structure of documents. Dimensionality reduction can improve the effectiveness of existing representations in terms of quality and run-time performance. Research into these areas is another way to improve the efficiency and quality of clustering and classification results.

Evaluating document clustering is a difficult task. Intrinsic measures of quality such as distortion only indicate how well an algorithm minimised a similarity function in a particular vector space. Intrinsic comparisons are inherently limited by the given representation and are not comparable between different representations. Extrinsic measures of quality compare a clustering solution to a “ground truth” solution. This allows comparison between different approaches. As the “ground truth” is created by humans it can suffer from the fact that
not every human interprets a topic in the same manner. Whether a document belongs to a particular topic or not can be subjective.

1.1 K-tree

The K-tree algorithm is a scalable and dynamical approach to clustering. It is a hierarchical algorithm inspired by the $B^+$-tree that has been adapted for multi-dimensional data. The tree forms a nearest neighbour search tree where insertions follow the nearest cluster at each level of the tree. In the tree building process the traditional k-means clustering algorithm is used to split tree nodes into two clusters. The hierarchy of clusters is built in a bottom-up fashion as data arrives. The K-tree algorithm is dynamic and adapts to data as it arrives.

Many existing clustering algorithms assume a single shot approach where all data is available at once. The K-tree differs because it can adapt to data as it arrives by modifying its tree structure via insertions and deletions. The dynamic nature and the scalability of the K-tree are of particular interest when applying it to document clustering. Extremely large corpora exist for document clustering such as the World Wide Web. These collections are also frequently updated.

1.2 Statement of Research Problems

This thesis addresses research problems in document representation, document classification, document clustering and clustering algorithms.

XML documents are semi-structured documents that contain structured and unstructured information. The structure is represented by XML markup that forms a hierarchical tree. Content is available as unstructured text that is contained within the nodes of the tree. Exploiting the additional information in semi-structured documents may be able to improve classification and clustering of documents. Therefore, it is a goal of this research is to encode structured information from XML documents in representations for use with classification and clustering algorithms. It is envisaged that this will improve the quality of the results.

The K-tree algorithm has never been applied to document clustering. Another research problem is determining the applicability of the K-tree to document clustering.

The K-tree algorithm offers excellent run-time performance at slightly lower distortion levels than the k-means and TSVQ algorithms. Therefore, it is a goal of this thesis to improve the quality of clusters produced by the K-tree. The scalability and dynamic properties of the tree must be retained when improving the algorithm.

The complexity of the K-tree algorithm has not been examined in detail. This thesis will perform a detailed time complexity analysis of the algorithm.

Feature selection for supervised machine learning is a well understood area. Selecting features in an unsupervised manner where no category labels are available poses a harder problem. This thesis will propose unsupervised feature selection approaches specifically for document representations.
1.3 Limitations of Study

The analysis of the K-tree algorithm will be limited to the context of document clustering. Although testing the K-tree in other fields would be worthwhile, it is not feasible within the scope of the project.

1.4 Thesis Structure

Chapter 2 introduces clustering in general. It looks at clustering algorithms and their application to document clustering. It specifically focuses on scalable clustering algorithms.

Many machine learning algorithms work with vector space representations of data. Chapter 3 discusses representation of documents using content and structure for use with K-trees and SVMs. Dimensionality reduction is discussed with respect to vector space representations.

Chapter 4 introduces the K-tree algorithm. It defines and motivates the data structure and algorithm. An example of building a K-tree is illustrated and performance is compared to the popular k-means algorithm.

Evaluation of document clustering and classification has taken place via the INEX 2008 XML Mining track. This is a collaborative forum where researchers compare results between different methods. Chapter 5 explores evaluation in detail.

Chapter 6 discusses the use of the K-tree algorithm to perform document clustering at INEX 2008. The quality of clustering produced by the K-tree algorithm is compared to other approaches.

The K-tree algorithm has been adapted to exploit the sparse nature of document vectors. This resulted in the Medoid K-tree described in Chapter 7.

Chapter 8 describes the combination of the K-tree algorithm and Random Indexing for large scale document clustering in collections with changing vocabulary and documents.

The average and worst case time complexity of the K-tree algorithm are introduced and explained in Chapter 9.

Chapter 10 discusses classification of documents at INEX 2008. The results are compared to other approaches.
Chapter 2

Clustering

Clustering is a form of data analysis that finds patterns in data. These patterns are often hard for humans to identify when they are in high dimensional space. The constant increase in computing power and storage has allowed analysis of large and high dimensional data sets that were previously intractable. This makes for an interesting and active field of research. Many of the drivers for this type of analysis stem from computer and natural sciences.

Review articles present a useful first look at clustering practices and offer high level analyses. Kotsiantis and Pintelas [48] state that clustering is used for the exploration of inter-relationships among a collection of patterns resulting in homogeneous clusters. Patterns within a cluster are more similar to each other than they are to a pattern belonging to a different cluster [37]. Clusters are learnt in an unsupervised manner where no a priori labelling of patterns has occurred. Supervised learning differs because labels or categories are associated with patterns. It is often referred to as classification or categorisation. When a collection is clustered all items are represented using the same set of features. Every clustering algorithm learns in a slightly different way and introduces biases. Algorithms will often behave better in a given domain. Furthermore, interpretation of the resulting clusters may be difficult or even entirely meaningless.

Clustering has been applied to fields such as information retrieval, data mining, image segmentation, gene expression clustering and pattern classification [37]. Due to the use of clustering in different domains there are many different algorithms. They have unique characteristics that perform better on certain problems.

2.1 Document Clustering

The goal of document clustering is to group documents into topics in an unsupervised manner. There is no categorical or topical labelling of documents to learn from. The representations used for document clustering are commonly derived from the text of documents by collecting term frequency statistics. These text representations result in high dimensional, sparse document by term matrices whose properties can be explained by Zipf distributions [37] in term occurrence.
Recently there has been a trend towards exploiting semi-structured documents [23]. This uses features such as XML tree structure and document to document link graphs to derive data from documents to determine their topic. Different document representations are introduced in Section 3.

2.2 Reviews and comparative studies

Material published in this area aims to cover an extensive range of algorithms and applications. The articles often represent cluster centers themselves by collating similar and important documents in a given area. They can also be viewed as a hub that links work together and points the way to more detail.

“Data clustering: a review” [37] provides an extensive review of clustering that summarises many different algorithms. It focuses on motivations, history, similarity measures and applications of clustering. It contains many useful diagrams for understanding different aspects of clustering. It is useful in gaining an understanding of clustering as a whole.

Kotsiantis and Pintelas [48] summarise the latest and greatest in clustering techniques and explain challenges facing the field. Not all data may exhibit clusterable tendencies and clusters can often be difficult to interpret. Real world data sets often contain noise that causes misclassification of data. Algorithms are tested by introducing artificial noise. Similarity functions, criterion functions, algorithms and initial conditions greatly affect the quality of clustering. Generic distance measures used for similarity are often hard to find. Pre-processing data and post-processing results can increase cluster quality. Outlier detection is often used to stop rare and distinct data points skewing clusters. The article covers many different types of clustering algorithms. Trees have quick termination but suffer from their inability to perform adjustments once a split or merge has occurred. Flat partitioning can be achieved by analysing the tree. Density based clustering looks at the abundance of data points in a given space. Density based techniques can provide good clustering in noisy data. Grid based approaches quantise data to simplify complexities. Model based approaches based on Bayesian statistics and other methods do not appear to be very effective. Combining different clustering algorithms to improve quality is proving to be a difficult task.

Yoo and Hu [79] compare several document clustering approaches by drawing on previous research and comparing it to results from the MEDLINE database. The MEDLINE database contains many corpora with some containing up to 158000 documents. Each document in MEDLINE has Medical Subject Heading (MeSH) terms. MeSH is an ontology first published by the National Library of Medicine in 1954. Additionally, terms from within the documents are mapped onto MeSH. These terms from the MeSH ontology are used to construct a vector representation. Experiments were conducted on hierarchical agglomerative, partitional and Suffix Tree Clustering (STC) algorithms. Suffix Trees are a widely used data structure for tracking n-grams of any length. Suffix Tree Clustering can use this index of n-grams to match phrases shared between documents. The results show that partitional algorithms offer superior performance and that STC is not scalable to large document sets. Within partitional algorithms, recursive bisecting algorithms often produce better clusters. Various measures of cluster quality are discussed and used to measure results. The paper also dis-
cusses other related issues such as sensitivity of seeding in partitional clustering, the “curse of dimensionality” and use of phrases instead of words.

Jain et. al. [37] have written a heavily cited overview of clustering algorithms. Kotsiantis and Pintelas [48] explicitly build upon the earlier work [37] by exploring recent advances in the field of clustering. They discuss advances in partitioning, hierarchical, density-based, grid-based, model based and ensembles of clustering algorithms. Yoo and Hu [79] provide great insight to various clustering algorithms using real data sets. This is quite different from the theoretical reviews in Jain et. al. and, Kotsiantis and Pintelas. Yoo and Hu provide more practical tests and outcomes by experimenting on medical document data sets. Their work is specific to document clustering. The K-tree algorithm fits into the hierarchical class of clustering algorithms. It is built bottom-up but differs greatly from traditional bottom-up hierarchical methods.

2.3 Algorithms

Jain et. al. [37] classify clustering algorithms into hierarchical, partitional, mixture-resolving and mode-seeking, nearest neighbour, fuzzy, artificial neural network, evolutionary and search-based. Hierarchical algorithms start with every data point as a cluster. Closest data points are merged until a cluster containing all the points is reached. This constructs a tree in bottom-up manner. Alternatively the tree can be constructed top-down by recursively splitting the set of all data points. Partitional algorithms split the data points into a defined number of clusters by moving partitions between the points. An example of a partitional algorithm is k-means. Mixture-resolving and mode-seeking procedures are drawn from one of several distributions where the goal is to determine the parameters of each. Most work assumes the individual components of the mixture density are Gaussian. Nearest neighbour algorithms work by assigning clusters based on nearest neighbours and threshold for neighbour distance. Fuzzy clustering allows data points to be associated with multiple clusters in varying degrees of membership. This allows clusters to overlap each other. Artificial neural networks are motivated by biological neural networks [37]. The weights between the input and output nodes are iteratively changed. The Self Organising Map (SOM) is an example of a neural network that can perform clustering. Evolutionary clustering is inspired by natural evolution [37]. It makes use of evolutionary operators and a population of solutions to overcome local minima. Exhaustive search-based techniques find optimal solutions. Stochastic search techniques generate near optimal solutions reasonably quickly. Evolutionary [13] and simulated annealing [46] are stochastic approaches.

2.4 Entropy constrained clustering

Research in this area aims to optimise clusters using entropy as a measure of quality. Entropy is a concept from information theory that quantifies the amount of information stored within a message. It can also be seen as a measure of uncertainty. An evenly weighted coin has maximum entropy because it is entirely uncertain what the next coin toss will produce. If a coin is weighted to land on heads more often, then it is more predictable. This makes the outcome
more certain because heads is more likely to occur. Algorithms that constrain entropy result in clusters that minimise the amount of information in each cluster. For example, all the information of documents relating to sky diving occur in one cluster.

Rose [64] takes an extensive look at deterministic annealing in relation to clustering and many other machine learning problems. Annealing is a process from chemistry that involves heating materials and allowing them to cool slowly. This process improves the structure of the material, thus improving its properties at room temperature. It can overcome many local minima to achieve the desired results. The k-means clustering algorithm often converges in local minima rather than finding the globally optimal solution. The author shows how he performs simulated annealing using information and probability theory. Each step of the algorithm replaces the current solution with a nearby random solution with a probability that is determined by a global temperature. The temperature is slowly decreased until an appropriate state has been reached. The algorithm can increase the temperature, allowing it to overcome local minima. The author discusses tree based clustering solutions and their problems.

ENTS [7] is a tree structured indexing system for vector quantisation inspired by AVL-trees. It differentiates itself by being more adaptive and dynamic. Internal nodes of the tree are referred to as decision nodes and contain a linear discriminant function and two region centres. The tree is constructed by recursively splitting the input space in half. The linear discriminant function is chosen such that it splits space in two while maximising cross entropy. Errors can occur when performing a recursive nearest neighbour search. This error occurs when the input vector exists in no-man’s land, an area around the splitting plane.

Tree Structured Vector Quantisation recursively splits an entire data set of vectors in two using the k-means algorithm. The first level of the tree splits the data in half, the second level splits each of these halves into quarters and so on. Tree construction is stopped based on a criteria such as cluster size or distortion. Rose [65] addresses the design of TSVQ using entropy to constrain structure. Initial algorithms in this area perform better than other quantisers that do not constrain entropy. However, these approaches scale poorly with the size of the data set and dimensionality. The research analyses the Generalised Breiman-Friedman-Olshen-Stone (GBFOS) algorithm. It is used to search for the minimum distortion rate in TSVQ that satisfies the entropy constraint. It has drawbacks that cause suboptimal results by blindly ignoring certain solutions. The design presented in this paper uses a Deterministic Annealing (DA) algorithm to optimise distortion and entropy simultaneously. DA is a process inspired by annealing from chemistry. DA considers data points to be associated in probability with partition regions rather than strictly belong to one partition. Experiments were conducted involving GBFOS and this proposed design. The new design produced significantly better quality clusters via a measure of distortion.

Wallace and Kanade [77] present research to optimise for natural clusters. Optimisation is performed by two steps. The first step is performed by a new clustering procedure called Numerical Iterative Hierarchical Clustering (NIHC) that produces a cluster tree. The second step searches for level clusters having a Minimum Description Length (MDL). NIHC starts with an arbitrary cluster tree produced by another tree based clustering algorithm. It iteratively transforms
the tree by minimising the objective function. It is shown that it performs better than standard agglomerative bottom-up clustering. It is argued that NIHC is particularly useful when there are not clearly visible clusters in the data. This occurs when the clusters appear to overlap. MDL is a greedy algorithm that takes advantage of the minimum entropy created by NIHC to find natural clusters.

Research in this area is a specialist area of clustering investigating optimisation of entropy. There are several explanations why entropy constrained clustering algorithms are not more popular. They are computationally expensive. Information theory is not a commonly studied topic and belongs to the fields of advanced mathematics, computer science and signal processing.

2.5 Algorithms for large data sets

Clustering often takes place on large data sets that will not fit in main memory. Some data sets are so large they need to be distributed among many machines to complete the task. Clustering large corpora such as the World Wide Web poses these challenges. Song et al. [72] propose and evaluate a distributed spectral clustering algorithm on large data sets in image and text data. For an algorithm to scale it needs to complete in a single pass. A linear scan algorithm will take \( O(n) \) time resulting in a set of clusters, whereas creating a tree structure will take \( O(n \log n) \) time. Both of these approaches will cluster in a single pass. Many cluster trees are inspired by balanced search trees such as AVL-tree and B\textsuperscript{+}-tree. The resulting cluster trees can also be used to perform an efficient nearest neighbour search.

BIRCH [81] uses the Cluster Feature (CF) measure to capture a summary of a cluster. The CF is comprised of a threshold value and cluster diameter. The algorithm performs local, rather than global scans and exploits the fact that data space is not uniformly occupied. Dense regions become clusters while outliers are removed. This algorithm results in a tree structure similar to a B\textsuperscript{+}-tree. Nodes are found by performing a recursive nearest neighbour search. BIRCH is compared to CLARANS [58] in terms of run-time performance. It is found that BIRCH is significantly faster. Experiments show that BIRCH produced clusters of higher quality on synthetic and image data in comparison to CLARANS.

Nearest neighbour graphs transform points in a vector space into a graph. Points in a vector space are vertexes and are connected to their \( k \) nearest neighbours via edges in the graph. The edges of the graph can be weighted with different similarity measures. The same theory behind nearest neighbour classification also applies to clustering. Points that lie within the same region of space share similar meaning. Finding nearest neighbours can be computationally expensive in high dimensional space. A brute force approach requires \( O(n^2) \) distance comparisons to construct a pair-wise distance matrix. Each position \( i,j \) of the pair-wise distance matrix represents the distance between points \( i \) and \( j \). Approaches to \( k \)NN search such as kd-tree tend to fall apart at greater than 20 dimensions [16]. The K-tree algorithm may be useful as an approximate solution to the \( k \)NN search problem but investigation of these properties is beyond the scope of this thesis.

Chameleon [43] uses a graph partitioning algorithm to find clusters. It op-
erates on a sparse graph where nodes represent data items and weighted edges represent similarity between items. The sparse graph representation allows it to scale to large data sets. Another advantage is that it does not require the use of metrics. Other similarity measures can be used that do not meet the strict definition of a metric. This algorithm uses multiple cluster similarity measures of inter-connectivity and closeness to improve results while still remaining scalable. Chameleon is qualitatively compared to DBSCAN \[26\] and CURE \[33\] using 2D data. The clusters are clearly visible in the 2D data and Chameleon appears to find these clusters more accurately than DBSCAN or CURE.

CLARANS \[58\] is a clustering algorithm that was developed to deal with terabytes of image data from satellite images, medical equipment and video cameras. It uses nearest neighbour graphs and randomised search to find clusters efficiently. The algorithm restricts itself to a sub graph when searching for nearest neighbors. The paper also discusses different distance measures that can be used to speed up clustering algorithms while only slightly increasing error rate. Experimental results show that the CLARANS algorithm produces higher quality results in the same amount of time as the CLARA \[45\] algorithm.

CURE \[33\] is a hierarchical algorithm that adopts a middle ground between centroid based and all point extremes. Traditional clustering algorithms favour spherical shapes of similar size and are very fragile to outliers. CURE is robust when dealing with outliers and identifies non-spherical clusters. Each cluster is represented by a fixed number of well scattered points. The points are shrunk towards the centroid of each cluster by a fraction. This becomes the representation of the clusters. The closest clusters are then merged at each step of the hierarchical algorithm. It is proposed that it is less sensitive to outliers because the shrinking phase causes a dampening effect. It also uses random sampling and partitioning to increase scalability for large databases. During processing the heap and kd-tree data structures are used to store information about points. The kd-tree data structure is known to have difficulty with data in high dimensional space as it requires \(2^d\) data points in \(d\) dimensional space to gather sufficient statistics for building the tree \[10\]. This renders the CURE algorithm useless for high dimensional data sets such as those in document clustering. Experimental results show that CURE produces clusters in less time than BIRCH. The clustering solutions of CURE and BIRCH are qualitatively compared on 2D data sets. CURE manages to find clusters that BIRCH can not.

The DBSCAN \[26\] algorithm relies on a density based notion of clusters. This allows it to find clusters of arbitrary shape. The authors suggest that the main reason why humans recognise clusters in 2D and 3D data is because density of points within a cluster are much higher than outside. The algorithm starts from an arbitrary point and determines if nearest neighbour points belong to the same cluster based on the density of the neighbours. It is found that DBSCAN is more effective than CLARANS \[58\] at finding clustering of arbitrary shape. The run-time performance is found to be 100 times faster than CLARANS.

iDistance \[36\] is an algorithm to solve the \(k\) Nearest Neighbour (\(k\)NN) search problem. It uses \(B^+\)-trees to allow for fast indexing of on disk data. Points are ordered based on their distance from a reference point. This maps the data into a one dimensional space that can be used for \(B^+\)-trees. The reference points are chosen using clustering. Many \(k\)NN search algorithms, including iDistance, partition the data to improve search speed.

K-tree \[30\] is a hybrid of the \(B^+\)-tree and k-means clustering procedure. It
supports online dynamic tree construction with properties comparable to the results obtained by Tree Structured Vector Quantisation (TSVQ). This is the original and only paper on the K-tree algorithm. It discusses the approach to clustering taken by k-means and TSVQ. The K-tree has all leaves on the same level containing data vectors. In a tree of order \( m \), all internal nodes have at most \( m \) non-empty children and at least one child. The number of keys is equal to the number of non-empty children. The keys partition the space into a nearest neighbour search tree. Construction of the tree is explained. When new nodes are inserted their position is found via a nearest neighbour search. This causes all internal guiding nodes to be updated. Each key in an internal node represents a centre of a cluster. When nodes are full and insertion occurs, nodes are split using the k-means clustering procedure. This can propagate to the root of the tree. If the root is full then it also splits and a new root is created. Experimental results indicate that K-tree is significantly more efficient in run-time than k-means and TSVQ.

O-Cluster \[57\] is an approach to clustering developed by researchers at Oracle. Its primary purpose is to handle extremely large data sets with very high dimensionality. O-Cluster builds upon the OptiGrid algorithm. OptiGrid is sensitive to parameter choice and partitions the data using axis-parallel hyper planes. Once the partitions have been found then axis parallel projections can occur. The original paper shows that the error rate caused by the partitioning decreases exponentially with the number of dimensions making it most effective in highly dimensional data. O-Cluster uses this same idea and also uses statistical tests to validate the quality of partitions. It recursively divides the feature space creating a hierarchical tree structure. It completes with a single scan of the data and a limited sized buffer. Tests show that O-Cluster is highly resistant to uniform noise.

Song et. al. \[72\] present an approach to deal with the scalability problem of spectral clustering. Their algorithm, called parallel spectral clustering alleviates scalability problems by optimising memory use and distributing computation over compute clusters. Parallelising spectral clustering is significantly more difficult than parallelising k-means. The dataset is distributed among many nodes and similarity is computed between local data and the entire set that mimises disk I/O. The authors also use a parallel eigensolver and distributed parameter tuning to speed up clustering time. When testing the Matlab implementation of this code it was found that it performed poorly when requiring a large number of clusters. It could not be included in the comparison of k-means and K-tree by De Vries and Geva \[20\] where up to 12,000 clusters were required. However, the parallel implementation was not tested. The authors report near linear speed increases with up to 32 node compute clusters. They also report using more than 128 nodes is counter productive. Experimental results show that this specialised version of spectral clustering produces higher quality clustering than the traditional k-means approach in both text and image data.

Ailon et. al. \[4\] introduce an approximation of k-means that clusters data in a single pass. It builds on previous work by Arthur et. al. \[6\] by using the seeding algorithm proposed for the k-means++ algorithm to provide a bi-criterion approximation in a batch setting. This is presented as the k-means# algorithm. The work extends a previous divide-and-conquer strategy for streaming data \[32\] to work with k-means++ and k-means#. This results in an approximation guarantee of \( O(e^{\alpha} \log k) \) for the k-means problem, where \( \alpha \approx \log n / \log M \), \( n \)
is the number of data points and $M$ is the amount of memory available. The authors state this is the first time that an incremental streaming algorithm has been proven to have approximation guarantees. A seeding process similar to k-means++ or k-means# could be used to improve the quality of K-tree but is beyond the scope of this thesis.

Berkhin et. al. [10] perform an extensive overview of clustering. The papers section on scalability reviews algorithms such as BIRCH, CURE and DIGNET. The author places scalable approaches into three categories, incremental, data squashing and reliable sampling. DIGNET performs k-means without iterative refinement. New vectors pull or push centroids as they arrive. The quality of an incremental algorithm is dependent on the order in which the data arrives. BIRCH is an example of data squashing that removes outliers from data and creates a compact representation. Hoeffding and Chernoff bounds are used in CURE to reliably sample data. These bounds provide a non-parametric test to determine the adequacy of sampling.

Scalable clustering algorithms need to be disk based. This is to deal with main memory sizes that are a fraction of the size of the data set. The K-tree algorithm [30] is inspired by the B*-tree which is often used in disk based applications such as relational databases and file systems. BIRCH, CURE, O-Cluster and iDistance [31, 33, 35, 57] have disk based implementations.

CURE claims to overcome problems with BIRCH. BIRCH only finds spherical clusters and is sensitive to outliers. BIRCH finds spherical clusters because it uses the Cluster Feature measure that uses cluster diameter and a threshold to control membership. CURE addresses outliers by introducing a shrinking phase that has a dampening effect.

CURE and CLARANS use random sampling of the original data to increase scalability. BIRCH, CURE, ENTS, iDistance and K-tree use balanced search trees to improve performance. Chameleon and CLARANS uses graph based solutions to scale to large data sets. All of these approaches have different advantages. Unfortunately there are no implementations of these algorithms made available by the authors. This makes an evaluation of large scale clustering algorithms particularly difficult.

Many of the researchers in this area talk of the “curse of dimensionality”. It causes data points to be nearly equidistant, making it hard to choose nearest neighbours or clusters. Dimensionality reduction techniques such as Principal Component Analysis [36], Singular Value Decomposition [21] and Wavelet Transforms [71] are commonly used.

All of the papers in this area compare their results to some previous research. Unfortunately there is no standard set of benchmarks in this area. The DS1, DS2 and DS3 synthetic data sets from BIRCH [31] have been reused in papers on Chameleon and WaveCluster [43, 71]. These datasets are 2D making them useless for indicating quality on high dimensional data sets.

Chameleon and CLARANS are based on graph theory. The basic premise is to cut a graph of relations between nodes resulting in the least cost. Generally the graphs are nearest neighbour graphs. Sparse graph representations can be used to further increase performance.
2.6 Other clustering algorithms

This section reviews research that does not fall into entropy constrained or large data sets categories. These algorithms still provide useful information in relation to the K-tree algorithm. Clustering covers many domains and research has resulted in many different approaches. This section will look at a sample of other clustering research that is relevant to the K-tree.

Many clustering problems belong to the set of NP-hard computational problems that are at least as computationally expensive as NP-complete problems. NP-hard problems can also be in NP-complete but in the case of the k-means problem it is not. The halting problem is another well known problem that is NP-hard but not in NP-complete. While finding the globally optimal solution to the k-means problem is NP-hard, the k-means algorithm approximates the optimal solution by converging to local optima and is thus an approximation algorithm. It is desirable to prove the optimality guarantees of approximation algorithms. For example, an approximation algorithm may be able to guarantee that the result it produces is within 5 percent of the global optimum. The k-means algorithm initialised with randomised seeding has no optimality guarantees [6]. The clustering solution it produces can be arbitrarily bad.

Arthur and Vassilvitskii [6] propose a method to improve the quality and speed of the k-means algorithm. They do this by choosing random starting centroids with very specific probabilities. This allows the algorithm to achieve approximation guarantees that k-means cannot. The authors show that this algorithm outperforms k-means in accuracy and speed via experimental results. It often substantially outperforms k-means. The experiments are performed on four different data sets with 20 trials on each. To deal with the randomised seeding process, a large number of trials are chosen. Additionally, the full source code for the algorithm is provided. Finding the exact solution to the k-means problem is NP-hard but it is shown that the k-means++ approximation algorithm is \(O(\log k)\) competitive. The proofs cover many pages in the paper and the main technique used is proof by induction. The algorithm works by choosing effective initial seeds using the \(D^2\) weighting.

Cheng et. al. [12] present an approach to clustering with applications in text, gene expressions and categorical data. The approach differs from other algorithms by dividing the tree top-down before performing a bottom-up merge to produce flat clustering. Flat clustering is the opposite of tree clustering where there is no recursive relationship between cluster centers. The top-down divisions are performed by a spectral clustering algorithm. The authors have developed an efficient version of the algorithm when the data is represented in document term matrix form. Conductance is argued to be a good measure for choosing clusters. This argument is supported by evidence from earlier research on spectral clustering algorithms. Merging is performed bottom-up using k-means, min-diameter, min-sum and correlation clustering objective functions. Correlation clustering is rejected as being too computationally intensive for practical use, especially in the field of information retrieval.

Lamrous and Taileb [51] describe an approach to top down hierarchical clustering using k-means. Other similar methods construct a recursive tree by performing binary splits. This tree allows splits to occur resulting in between two and five clusters. The k-means algorithm is run several times while changing the parameter \(k\) from two to five. The resulting clusters are compared for goodness
using the Silhouette criterion \cite{66}. The result with the best score is chosen, thus
providing the best value for \(k\). Running k-means several times as well as the
Silhouette function is computationally expensive. Therefore, it is recommended
that it only be applied to the higher levels of the tree when used with large data
sets. This is where it can have most impact because the splits involve most of
the data. Experiments are performed using this algorithm, bisecting k-means
and sequential scan. The algorithm presented in this paper performs best in
terms of distortion.

Oyzer and Alhajj \cite{59} present a unique approach to creating quality clusters.
It uses evolutionary algorithms inspired by the same process in nature. In the
methodology described, multiple objectives are optimised simultaneously. It is
argued that humans perform decision problems in the same way and therefore
the outcomes should make more sense to humans. As the algorithm executes,
multiple objective functions are minimised. The best outcome is chosen at
each stage by the measure of cluster validity indexes. The common k-means
partitioning algorithm has a problem. The numbers of desired clusters, \(k\), needs
to be defined by a human. This is error prone even for domain experts that
know the data. This solution addresses the problem by integrating it into the
evolutionary process. A limit for \(k\) needs to be specified and a competition takes
place between results with one to \(k\) clusters.

Fox \cite{27} investigates the use of signal processing techniques to compress the
vectors representing a document collection. The representation of documents
for clustering usually takes the form of a document term matrix. There can
be thousands of terms and millions of records. This places strain on com-
puter memory and processing time. This paper describes a process to reduce
the number of terms by using the Discrete Cosine Transform. According to
the F-measure metric, it has no reduction in quality. The vector compression
is performed in three steps. Firstly, an uncompressed vector representing the
whole document corpus is obtained. Next, DCT is applied to this vector to
find the lower frequency sub-bands that account for the majority of the energy.
Finally, compressed document vectors are created by applying the DCT to un-
compressed document vectors thus leaving only the lower sub-bands identified
in the previous step.

Dhillon et. al. \cite{24} state that kernel k-means and spectral clustering are
able to identify clusters that are non-linearly separable in input space. The
authors give an explicit theoretical relation between the two methods that had
only previously been loosely related. This leads to the authors developing a
weighted kernel k-means algorithm that monotonically decreases the normalised
cut. Spectral clustering is shown to be a specialised case of the normalised cut.
Thus, the authors can perform a method similar to spectral clustering without
having to perform computational expensive eigenvalue based approaches. They
apply this new method to gene expression and handwriting clustering. The
results are found to be of high quality and computationally fast. Methods such
as these can be used to improve the quality of results in K-tree. However,
applying kernel k-means to K-tree is outside the scope of this thesis.

Banerjee et. al. \cite{8} investigate the use of Bregman divergences as a distortion
function in hard and soft clustering. A distortion function may also be referred
to as a similarity measure. Hard, partitional or flat clustering algorithms split
data into disjoint subsets where as soft clustering algorithms allow data to have
varying degree of membership in more than one cluster. Bregman divergences
include loss functions such as squared loss, KL-divergence, logistic loss, Mahalanobis distance, Itakura-Saito distance and I-divergence. Partitional hard clustering using Mutual Information \cite{25} is seen as a special case of clustering with Bregman divergences. The authors prove there exists a unique Bregman divergence to every regular exponential family. An exponential family is a class probabilistic distributions that share a particular form. The authors also show that any Bregman divergence can be simply plugged into the k-means algorithm and retain properties such as guaranteed convergence, linear separation boundaries and scalability. Huang \cite{35} find KL-divergence to be effective for text document clustering by comparing it several other similarity measures on many data sets.

The research presented in this section is recent. Some of these works \cite{12, 59, 27, 6} combine multiple algorithms or similarity measures. This is a good way to discover efficient and accurate methods by combining the best of new and previous methods.

2.7 Approaches taken at INEX

Zhang et. al. \cite{80} describe data mining on web document as one of the most challenging tasks in machine learning. This is due to large data sets, link structure and unavailability of labelled data. The authors consider the latest developments in Self Organising Maps (SOM) called the Probability Mapping Graph SOM (PMGraphSOM). The authors argue that most learning problems can be represented as a graph and they use molecular chemistry as a compelling example where atoms are vertexes and atomic bonds are edges. Usually graphs are flattened onto a vectorial representation. It is argued that this approach loses information and it is better to work with the graph directly. Therefore, the authors explain the PMGraphSOM and how it works directly with graphs. The authors improved their original work and significantly outperformed all other submissions at INEX. Unfortunately, the SOM method is particularly slow and took between 13 and 17 hours to train on this relatively small dataset.

Kutty et. al. \cite{49} present an approach for building a text representation that is restricted by exploiting frequent structure within XML trees. The reduced representation is then clustered with the k-way algorithm. The hypothesis that drives this approach is that the frequent sub-trees contained within a collection contain meaningful text. This approach allows terms to be selected that offer a small decrease in cluster quality. However, the approach highlighted in Section 3.3.2 is much simpler and provided better quality results as per the INEX evaluation. Additionally, it does not rely on any information except the term frequencies themselves.

Tran et. al. \cite{74} exploit content and structure in the clustering task. They also use Latent Semantic Analysis (LSA) to find a semantic representation for the corpora. The authors take a similar approach to Kutty et. al. \cite{49} where the number of terms are reduced by exploiting XML structure. This reduction of terms helps the computational efficiency of the Singular Value Decomposition (SVD) used in LSA. The authors claim that LSA works better in practice does not hold for this evaluation. The BM25 and TF-IDF culled representation used by De Vries and Geva \cite{19} outperforms the LSA approach.

De Vries and Geva \cite{19} investigate the use of K-tree for document clustering.
This is explained in detail in Section 6.

2.8 Summary

This section reviewed many different approaches in clustering. The K-tree algorithm appears to be unique and there are many different approaches in the literature that could be applicable to increase run-time performance or quality. This thesis has improved the algorithm in Sections 7 and 8.
Document Representation

Documents can be represented by their content and structure. Content representation is derived from text by collecting term frequency statistics. Structure can be derived from XML, document to document links and other structural features. Term weightings such as TF-IDF and BM25 were used to represent content in a vector space for the INEX collection. This representation is required before classification and clustering can take place as SVMs and K-tree work with vector space representations of data. The link structure of the Wikipedia was also mapped onto a vector space. The same Inverse Document Frequency heuristic from TF-IDF was used with links.

3.1 Content Representation

Document content was represented with TF-IDF [68] and BM25 [63]. Stop words were removed and the remaining terms were stemmed using the Porter algorithm [62]. TF-IDF is determined by term distributions within each document and the entire collection. Term frequencies in TF-IDF were normalised for document length. BM25 works with the same concepts as TF-IDF except that it has two tuning parameters. The BM25 tuning parameters were set to the same values as used for TREC [63], \( K_1 = 2 \) and \( b = 0.75 \). \( K_1 \) influences the effect of term frequency and \( b \) influences document length.

3.2 Link Representation

Links have been represented as a vector of weighted link frequencies. This resulted in a document-to-document link matrix. The row indicates the origin and the column indicates the destination of a link. Each row vector of the matrix represents a document as a vector of link frequencies to other documents. The motivation behind this representation is that documents with similar meaning will link to similar documents. For example, in the current Wikipedia both car manufacturers BMW and Jaguar link to the Automotive Industry document. Term frequencies were simply replaced with link frequencies resulting in LF-IDF. Link frequencies were normalised by the total number of links in a document.
LF-IDF link weighting is motivated by similar heuristics to TF-IDF term weighting. In LF-IDF the link inverse document frequency reduces the weight of common links that associate documents poorly and increases the weight of links that associate documents well. This leads to the concept of stop-links that are not useful in classification. Stop-links bare little semantic information and occur in many unrelated documents. Consider for instance a document collection of the periodic table of the elements, where each document corresponds to an element. In such a collection a link to the “Periodic Table” master document would provide no information on how to group the elements. Noble gases, alkali metals and every other category of elements would all link to the “Periodic Table” document. However, links that exist exclusively in noble gases or alkali metals would be excellent indicators of category. Year links in the Wikipedia are a good example of a stop-link as they occur with relatively high frequency and convey no information about the semantic content of pages in which they appear.

3.3 Dimensionality Reduction

Dimensionality reduction is used in conjunction with clustering algorithms for different reasons. The most obvious reason is to reduce the number of dimensions required for objects. This allows clustering algorithms to execute in a more efficient manner. It also allows the resolution of the “curse of dimensionality”. Points in high dimensional space appear equidistant from each other. When they are projected into a lower dimensional space, patterns are easier to distinguish and clustering algorithms can result in more meaningful results. LSA exploits properties of dimensionality reduction to find hidden relationships between words in large corpora of text.

CURE and CLARANS use random sampling and random search to reduce the cardinality of the data. In this case, careful selection closely approximates the original result on all of the data. This type of reduction does not involve projections into lower dimensional space. It culls some of the data set.

Wavelet Transforms, Discrete Cosine Transforms, Principal Component Analysis, Independent Component Analysis, Singular Value Decomposition and Non Negative Matrix Factorisation all project data into a reduced dimensionality space. These techniques have been used in a variety of different fields such as signal processing, image processing and spatial databases.

3.3.1 Dimensionality Reduction and K-tree

Dimensionality reduction techniques have been used with K-tree because it performs well with dense representations. If a collection has one million terms, then the means in the top level of the tree are likely to contain many of these one million terms. Between all the means, they must contain all one million terms. This is computationally expensive, even when using a sparse representation because the most frequently accessed means are in the root of the tree. Every vector inserted into the tree has to be compared to these means containing a large number of terms. The use of some sparse representations and the cosine similarity measure can reduce this to the intersection of the terms contained in the mean and document vector. However, the terms are updated at every
insertion, causing repeated additions of new terms into a sparse representation. Further more, each weight for each term has to be updated at every single insertion.

3.3.2 Unsupervised Feature Selection

An unsupervised feature selection approach has been used to select features from both text and link representations. This was initially conceived to deal with the memory requirements of using a dense representation with K-tree. The algorithm works by calculating a feature’s rank. This rank is determined by the summation of a column vector in an object by feature matrix. In this case the objects are documents. The rank is determined by the sum of all weights for each feature in all documents. This ranking function produces a higher value when the feature is more important. Only the top $n$ features are kept in the matrix and the rest are discarded. Submissions to INEX initially used the top 8000 features [19] and this approach was analysed at many different dimensions by De Vries et. al. [18]. Papapetrou et. al. [60] use a similar method in their modified k-means algorithm for text clustering in peer to peer networks. This feature selection exploits the power law distributions that occur in term frequencies [83] and vertex degree in a document link graph [78]. When using this approach with text representations it has been referred to as “TF-IDF culling”.

3.3.3 Random Indexing

The Singular Value Decomposition used by LSA is a computationally expensive approach to dimensionality reduction and is not suitable to large scale document clustering. A similar outcome can be achieved by projecting high dimensional data onto a randomly selected lower dimensional sub-space. This is the essence of how Random Indexing works. This is analysed in more detail in Section 8.

3.3.4 Latent Semantic Analysis

Latent Semantic Analysis (LSA) is a tool for natural language processing. It analyses relationships between documents and the terms they contain. This process results in a set of concepts derived from the documents and terms. Landauer and Dumais [52] explain LSA in a theoretical context. They use experimentation to validate the conclusions they draw. It is shown that LSA can acquire knowledge in text at a rate comparable to school children. The process induces global knowledge indirectly from local co-occurrence data in a large body of representative text. The article emphasises that a large body of text is required. Previous researchers have proposed that humans have some inherent knowledge and that it requires to be “turned on” by hints and contemplation. This is Plato’s solution to Plato’s problem. Plato’s problem was first described by Chomsky [15]. This paper disagrees with this model and suggests that the brain actually performs LSA. It states that the brain does not know anything in advance and can learn from hidden relationships in the text. LSA is applied by the use of Singular Value Decomposition (SVD). The paper suggests that many domains contain large numbers of weak interrelations. When these interrelations are exploited, learning is improved by the process of inference. SVD is a means
of induction, dimensional optimisation and factor analysis. The article explains that SVD finds these interrelations. It also discusses many different areas of psychology and cognition, relating them to LSA and other artificial learning techniques.

Deerwester et. al. [21] is the original work on Latent Semantic Analysis and provides many detailed workings. It discusses many of the problems with traditional information retrieval methods. Terms such as synonymy and polysemy are defined. Synonymy is the fact that there are many ways to refer to the same idea. Polysemy is the fact that words have one or more meanings and often change when used in different contexts or by different people. It is proposed that LSA finds hidden meaning between words in a document. LSA finds the underlying semantic structure in the data and allows users to retrieve data based on concepts rather than key words alone. LSA surpasses previous works in terms of scalability and quality. It is performed by the mathematical process of Singular Value Decomposition. SVD allows the arrangement of the space to reflect major associative patterns. Smaller and less important influences are ignored. SVD is performed on a document term matrix and breaks the data into linearly independent components. The derived dimensions can be thought of as artificial concepts.

Much of the work surrounding LSA is very theoretical and discusses how humans acquire information from text [21, 52]. The mathematics that supports the process existed before the concept of LSA. These articles are particularly useful in understanding the motivation for performing LSA. The common theme in LSA research is that it discovers meaning the same way humans do. Unfortunately most experimental results do not show this convincingly. LSA may be a simplification of a more complicated process.

3.4 Summary

In this section a link representation called LF-IDF was introduced. A simple and effective approach to dimensionality reduction called TF-IDF culling was defined. This work was presented by De Vries in Geva [19] in a peer reviewed paper at INEX 2008.
Chapter 4

K-tree

K-tree \cite{Geva2000} \cite{Katz2000} is a height balanced cluster tree. It was first introduced in the context of signal processing by Geva \cite{Geva2000}. The algorithm is particularly suitable to clustering of large collections due to its low complexity. It is a hybrid of the B$^+$-tree and k-means algorithms. The B$^+$-tree algorithm is modified to work with multi-dimensional vectors and k-means is used to perform node splits in the tree.

Unlike partitional algorithms such as k-means, the K-tree does not require that the number of clusters be specified upfront. The numbers of clusters that exist in the tree depends on the tree order $m$ and the number of vectors inserted. The tree order $m$ is a parameter specified by the user of the tree.

K-tree builds a nearest neighbour search tree over a set of real valued vectors $X$ in $d$ dimensional space.

$$X \subset \mathbb{R}^d$$ \hspace{1cm} (4.1)

It is inspired by the B$^+$-tree where all data records are stored in leaf nodes. Tree nodes, $n$, consist of a sequence of (vector $v$, child node $c$) pairs of length $l$. The tree order, $m$, restricts the number of vectors stored in any node to between one and $m$.

$$1 \leq l \leq m$$ \hspace{1cm} (4.2)

$$n = \langle (v_1, c_1), ..., (v_l, c_l) \rangle$$ \hspace{1cm} (4.3)

The length of a node is denoted by a leading $#$ and the ith pair of (vector $v$, child node $c$) pairs in the sequence is denoted by $n_i$.

$$\#n = l$$ \hspace{1cm} (4.4)

$$n_1 = \text{head}(n)$$ \hspace{1cm} (4.5)

The key function returns a vector from one of the pairs, $r$, contained within a node. The key is either a search key in the nearest neighbour search tree (i.e. a cluster centre) or a vector inserted into the tree in a leaf node.

$$\text{key}(r) = \text{first}(r)$$ \hspace{1cm} (4.6)

The child function returns a node, $n$, from one of the pairs, $r$, contained within a node. The child node is the sub-tree associated with the search key in the

$$\text{child}(n) = r$$
pair. In the case of leaf nodes this is a null reference representing termination of the tree structure.

\[
child(r) = second(r)
\]  
\[(4.7)\]

The tree consists of two types of nodes, leaf and internal. \( N \) is the set of all nodes in a K-tree, including both leaf and internal nodes. Leaf nodes, \( L \), are a subset of all nodes in the tree, \( N \).

\[
L \subset N
\]  
\[(4.8)\]

Leaf nodes, \( L \), contain the data vectors that were inserted into the tree and contain a null child reference.

\[
L = \{n \in N : \sum_{i=1}^{\#n} key(n_i) \in X \land child(n_i) = null\}
\]  
\[(4.9)\]

\( W \) represents the set of all cluster centres in a K-tree. The cluster centres are also in \( d \) dimensional space.

\[
W \subset \mathbb{R}^d
\]  
\[(4.10)\]

Internal nodes, \( I \), are a subset of all nodes in the tree.

\[
I \subset N
\]  
\[(4.11)\]

Internal nodes, \( I \), contain cluster centres and reference to a child node.

\[
I = \{n \in N : \sum_{i=1}^{\#n} key(n_i) \in W \land child(n_i) \neq null\}
\]  
\[(4.12)\]

A cluster centre (vector) is the mean of all data vectors contained in the leaves of all descendant nodes (i.e. the entire cluster sub-tree). This follows the same recursive definition of a \( B^+ \)-tree, where each tree is made up of a set of smaller sub-trees. Upon construction of the tree, a nearest neighbour search tree is built in a bottom-up manner by splitting full nodes using k-means \[55\] where \( k = 2 \). As the tree depth increases it forms a hierarchy of “clusters of clusters” from the root to the above-leaf level. The above-leaf level contains the finest granularity cluster vectors. Each leaf node stores the data vectors pointed to by the above-leaf level. The efficiency of K-tree stems from the low complexity of the \( B^+ \)-tree algorithm, combined with only ever executing k-means on a relatively small number of vectors, defined by the tree order, and by using a small value of \( k \). The constraints placed on the tree are relaxed in comparison to a \( B^+ \)-tree. This is due to the fact that vectors do not have a total order like real numbers.

\( B^+ \)-tree of order \( m \)

1. All leaves are on the same level.

2. Internal nodes, except the root, contain between \( \lceil \frac{m}{2} \rceil \) and \( m \) children.

3. Internal nodes with \( n \) children contain \( n - 1 \) keys, partitioning the children into a search tree.

4. The root node contains between 2 and \( m \) children. If the root is also a leaf then it can contain a minimum of 0.
K-tree of order \( m \)

1. All leaves are on the same level. Leaf nodes contain data vectors.

2. Internal nodes contain between one and \( m \) children. The root can be empty when the tree contains no vectors.

3. Codebook vectors (cluster representatives) act as search keys.

4. Internal nodes with \( n \) children contain \( n \) keys, partitioning the children into a nearest neighbour search tree.

5. The level immediately above the leaves form the codebook level containing the codebook vectors.

The leaf nodes of a K-tree contain real valued vectors. The search path in the tree is determined by a nearest neighbour search. It follows the child node associated with the nearest vector. This follows the same recursive definition of a B\(^+\)-tree where each tree is made up of a smaller sub-tree. Any similarity measure can be used for vectors in a K-tree. However, the choice of similarity measure will affect the tree's ability to perform as a nearest neighbour search tree.

K-tree achieves its efficiency through execution of the high cost k-means step over very small subsets of the data. The number of vectors clustered during any step in the K-tree algorithm is determined by the tree order (usually \( \ll 1000 \)) and it is independent of collection size. It is efficient in updating the collection while maintaining clustering properties through the use of a nearest neighbour search tree that directs new vectors to the appropriate leaf node.

The K-tree forms a hierarchy of clusters. This hierarchy supports multi-granular clustering where generalisation or specialisation is observed as the tree is traversed from a leaf towards the root or vice versa. The granularity of clusters can be decided at run-time by selecting clusters that meet criteria such as distortion or cluster size.

The K-tree algorithm is well suited to clustering large document collections due to its low time complexity. The time complexity of building K-tree is \( O(n \log n) \), where \( n \) is the number of bytes of data to cluster. This is due to the divide and conquer properties inherent to the search tree. De Vries and Geva \[19, 20\] investigate the run-time performance and quality of K-tree by comparing results with other INEX submissions and CLUTO \[12\]. CLUTO is a popular clustering tool kit used in the information retrieval community. K-tree has been compared to k-means, including the CLUTO implementation, and provides comparable quality and a marked increase in run-time performance. However, K-tree forms a hierarchy of clusters and k-means does not. Comparison of the quality of the tree structure will be undertaken in further research. The run-time performance increase of K-tree is most noted when a large number of clusters are required. This is useful in terms of document clustering because there are a huge number of topics in a typical collection. The on-line and incremental nature of the algorithm is useful for managing changing document collections. Most clustering algorithms are one shot and must be re-run when new data arrives. K-tree adapts as new data arrives and has the low time complexity of \( O(\log n) \) for insertion of a single document. Additionally, the tree structure
also allows for efficient disk based implementations when the size of data sets exceeds that of main memory.

The algorithm shares many similarities with BIRCH [81] as both are inspired by the B+-tree data structure. However, BIRCH does not keep the inserted vectors in the tree. As a result, it cannot be used for a nearest neighbour search tree. This makes precise removal of vectors from the tree impossible. K-tree is also related to Tree Structured Vector Quantisation (TSVQ) [28]. TSVQ recursively splits the data set, in a top-down fashion, using k-means. TSVQ does not generally produce balanced trees.

4.1 Building a K-tree

Figure 4.1 displays the meaning of the symbols used in diagrams illustrating the building process. The example is a K-tree of order 3 \((m = 3)\). The K-tree is constructed dynamically as data vectors arrive. Initially the tree contains a single empty root node at the leaf level (Figure 4.2). Vectors are inserted via a nearest neighbour search, terminating at the leaf level. The root of an empty tree is a leaf, so the first \(m\) data vectors are stored in the root, at which point the node becomes full. When the \(m + 1\) vector arrives, the root is split using k-means where \(k = 2\), clustering all \(m + 1\) vectors into two clusters (Figure 4.3).

The two centroids that result from k-means are then promoted to become the centroids in a new root. The vectors associated with each centroid are placed into a child node. This promotion process has created a new root and two leaf nodes in the tree (Figure 4.4). The tree is now two levels deep. Insertion of a new data vector follows a nearest neighbour search to find the closest centroid in the root. The vector is inserted into the associated child. When a new vector is inserted the centroids are updated recursively along the nearest neighbour search path, all the way back to the root node. The propagated means are weighted by the number of data vectors contained beneath them. This ensures that any centroid in the K-tree is the mean vector of all the data vectors contained in the associated sub-tree. This insertion process continues, splitting leaves when they become full (Figure 4.5), until the root node itself becomes full. K-means is then run on the root node containing centroids (Figure 4.6). The vectors in the new root node become centroids of centroids (Figure 4.7). As the tree grows, internal and leaf nodes are split in the same manner. The process of promotion can potentially propagate to cause a full root node, at which point the construction of a new root follows and the tree depth is increased by one. At all times the tree is guaranteed to be height balanced. Although the tree is always height balanced nodes can contain as little as one vector. In this case the tree will contain many more levels than a tree where each node is full. Figure 4.8 illustrates inserting a vector into a three level K-tree. The filled black vector represents the vector inserted into the tree and the dashed lines represent the parts of the tree read during nearest neighbour search.

Figure 4.9 compares k-means performance with K-tree where \(k\) for k-means is determined by the number of codebook vectors. This means that both algorithms produce the same number of document clusters and this is necessary for a meaningful comparison. The order, \(m\), for K-tree was 50. Each algorithm was run on the 8000 dimension BM25 vectors from the INEX 2008 XML mining track.
Figure 4.1: K-tree Legend

Figure 4.2: Empty 1 Level K-tree

Figure 4.3: 1 Level K-tree With a Full Root Node

Figure 4.4: 2 Level K-tree With a New Root Node

Figure 4.5: Leaf Split in a 2 Level K-tree
4.2 K-tree Example

Figures 4.10, 4.11 and 4.12 are K-tree clusters in two dimensions. 1000 points were drawn from a random normal distribution with a mean of 1.0 and standard deviation of 0.3. The order of the K-tree, m, was 11. The grey dots represent the data set, the black dots represent the centroids and the lines represent the Voronoi tessellation of the centroids. Each of the data points contained within each tile of the tessellation are the nearest neighbours of the centroid and belong to the same cluster. It can be seen that the probability distribution is modelled at different granularities. The top level of the tree is level 1. It is the coarsest grained clustering. In this example it splits the distribution in three. Level 2 is more granular and splits the collection into 19 sub-clusters. The individual clusters in level 2 can only be arrived at through a nearest neighbour association with a parent cluster in level 1 of the tree. Level 3 is the deepest level in the tree consisting of cluster centroids. The fourth level is the data set of vectors that were inserted into the tree.

4.3 Summary

In this chapter K-tree algorithm was introduced and defined. An example of building a K-tree was presented. Examples of K-tree clusters were displayed in two dimensional synthetic data.
Figure 4.8: Inserting a Vector into a 3 Level K-tree

leaf nodes contain the data vectors

nodes above the leaves contain codebook vectors

nodes above the codebook level are clusters of clusters

root node

level 1

level 2

level 3

the dashed parts represent the nearest neighbour search

the filled black vector followed the nearest neighbour search path
K–tree and k–means Performance Comparison
INEX XML Mining Collection

Figure 4.9: K-tree Performance

Figure 4.10: Level 1
Figure 4.11: Level 2

Figure 4.12: Level 3
Evaluation of classification and clustering techniques has taken place via the INEX 2008 XML Mining track [23]. This is a collaborative evaluation forum where researchers from different disciplines evaluate supervised and unsupervised learning tasks in XML Information Retrieval. An 114,366 document subset of the 2006 XML Wikipedia is used and results are compared against a ground truth set of labels. The labels are provided by the track organisers and are extracted from Wikipedia portals. Each portal represents a topic. The labels are available for download from the INEX website [11]. To download any of the INEX data registration is required. The track aims to explore machine learning for semi-structured documents. In 2008 the INEX XML Mining track started investigating the link structure of the Wikipedia. Participants make submissions without complete knowledge of the ground truth. The results are then released, allowing a comparison between different methods.

Document cluster quality has been assessed by comparing clusters produced by algorithms to a ground truth. The ground truth consists of a single label for each document from 15 categories and allows measures such as purity and entropy to be calculated. Purity and entropy have been used by Zhao and Karypis [82] to measure document cluster quality. These are common evaluation metrics used for clustering.

For the clustering evaluation at INEX every participant had to submit a clustering solution with 15 clusters. The purity evaluation metric is only comparable when the same number of clusters are compared. The more clusters there are, the more likely they are to be pure.

The classification task had a 10 percent training set released for participants to train their classifier. The classifiers then predict labels for documents in a 90 percent test set. Classification is evaluated via the recall metric. This is simply the accuracy of the classifier. It is the proportion of labels that the classifier correctly predicted.

The collaborative nature of the INEX evaluation allows for greater confidence in the performance of different approaches. A comparison against many different methods is completed in a time that would not be possible by a single researcher. It provides means to further increase the confidence of empirical results found as the research by different groups occurs independently.
5.1 Classification as a Representation Evaluation Tool

Classification is an easy to execute and understand method for evaluation of representations for Information Retrieval. The problem is very well defined and has meaningful, unambiguous metrics for evaluation. The goal of clustering is to search for “something interesting”, where in comparison, the goal of classification is to predict classes that are pre-defined. Lewis [53] also argues that text classification is a much cleaner determination of text representation properties when compared to standard AdHoc evaluations. While De Vries et. al. [18] have some evidence that the exact trend in representation performance does not carry over from classification to clustering, it provides an easy and well defined first step for testing a new representation. Further investigation of classification as a tool for representation evaluation for Information Retrieval is certainly an appealing area of research.

5.2 Negentropy

The negentropy metric has been devised to provide an entropy based metric that gives a score in the range of 0 to 1, where 0 is the worst solution and 1 is the best solution. The score always falls in this range no matter how many labels exist in the ground truth. The negentropy metric measures the same system property as information entropy [70]. It correlates with the goals of clustering where 1 is the best score and 0 is the worst possible score.

The purity measure is calculated by taking the most frequently occurring label in each cluster. Micro purity is the mean purity weighted by cluster size and macro is the unweighted arithmetic mean. Taking the most frequently occurring label in a cluster discards the rest of the information represented by the other labels. Negentropy is the opposite of information entropy [70]. If entropy is a measure of uncertainty associated with a random variable then negentropy is a measure of certainty. Thus, it is better when more labels of the same class occur together. When all labels are evenly distributed across all clusters the lowest possible negentropy is achieved.

Negentropy is defined in Equations 5.1, 5.2 and 5.3. $D$ is the set of all documents in a cluster. $X$ is the set of all possible labels. $l(d)$ is the function that maps a document $d$ to its label $x$. $p(x)$ is the probability for label $x$. $H(D)$ is the negentropy for document cluster $D$. The negentropy for a cluster falls in the range $0 \leq H(D) \leq 1$ for any number of labels in $X$. Figure 5.1 shows the difference between entropy and negentropy. While they are exact opposites for a two class problem, this property does not hold for more than two classes. Negentropy always falls between zero and one because it is normalised. Entropy is bounded by the number of classes. The difference between the maximum value for negentropy and entropy increase when the number of classes increase.

$$l(d) = \{(d_1, x_1), (d_2, x_2), \ldots, (d_{|D|}, x_{|D|})\} \quad (5.1)$$

$$p(x) = \frac{|\{d \in D : x = l(d)\}|}{|D|} \quad (5.2)$$
The difference between purity and negentropy can easily be demonstrated with an artificial four class problem. There are six of each of the labels A, B, C and D. For each cluster in Solution 1 purity and negentropy is 0.5. For each cluster in Solution 2 the purity is 0.5 and the negentropy is 0.1038. Entropy differentiates between the solutions and indicates Solution 1 is better because of a larger negentropy score (0.5 > 0.1038). Purity makes no differentiation between the two solutions and each solution scores 0.5. If the goal of document clustering is to group similar documents together then Solution 1 is clearly better because each label occurs in two clusters instead of four. The grouping of labels is better because they are less spread. Figures 5.2 and 5.3 show Solutions 1 and 2.

5.3 Summary

In this section the evaluation of machine learning via the collaborative INEX evaluation forum was discussed. Classification was motivated as an unambiguous evaluation method for representations in IR. A normalised variant of entropy named negentropy was introduced, defined and motivated.
Cluster | Label Counts
--- | ---
1 | A=3, B=3
2 | A=3, C=3
3 | B=3, D=3
4 | C=3, D=3

Figure 5.2: Solution 1

Cluster | Label Counts
--- | ---
1 | A=3, B=1, C=1, D=1
2 | B=3, C=1, D=1, A=1
3 | C=3, D=1, A=1, B=1
4 | D=3, A=1, B=1, C=1

Figure 5.3: Solution 2
Chapter 6

Document Clustering with K-tree

K-tree was used to create clusters using the INEX 2008 XML Mining collection. Documents were represented as 8000 dimension BM25 weighted vectors. Thus, clusters were formed using text only. This representation was used because it was the most effective text representation in the classification task. The K-tree was constructed using the entire collection. Cluster membership was determined by comparing each document to all centroids using cosine similarity. The track required a submission with 15 clusters but K-tree does not produce a fixed number of clusters. Therefore, the codebook vectors were clustered using k-means++ [6] where \( k = 15 \). The codebook vectors are the cluster centroids that exist above the leaf level. This reduces the number of vectors used for k-means++, making it quick and inexpensive. As k-means++ uses a randomised seeding process, it was run 20 times to find the solution with the lowest distortion. The k-means++ algorithm improves k-means by using the \( D^2 \) weighting for seeding. Two other submission were made that represent different levels of a K-tree. A tree of order 100 had 42 clusters in the first level and a tree of order 20 had 147 clusters in the second level. This made for a total of three submissions for K-tree.

Negentropy was used to determine the optimal tree order. K-tree was built using the documents in the 10 percent training set from the classification task. A tree was constructed with an order of 800 and it was halved each time. Negentropy was measured in the clusters represented by the leaf nodes. As the order decreases the size of the nodes shrinks and the purity increases. If all clusters became pure at a certain size then decreasing the tree order further would not improve negentropy. However, this was not the case and negentropy continued to increase as the tree order decreased. This is expected because there will usually be some imperfection in the clustering with respect to the labels. Therefore, the sharp increase in negentropy in a K-tree below an order of 100 suggests that the natural cluster size has been observed. This can be seen in Figure 6.1. The “left as is” line represents the K-tree as it is built initially. The “rearranged” line represents the K-tree when all the leaf nodes have been reinserted to their nearest neighbours without modifying the internal nodes.

Negentropy was calculated using the 10 percent training set labels provided on clusters for the whole collection. This was used to determine which order
of 10, 20 or 35 fed into k-means++ with $k = 15$ was best. A tree of order 20 provided the best negentropy.

### 6.1 Non-negative Matrix Factorisation

NMF factorises a matrix into two matrices where all the elements are $\geq 0$. If $V$ is a $n \times m$ matrix and $r$ is a positive integer where $r < \min(n, m)$, NMF finds two non-negative matrices $W_{n \times r}$ and $H_{r \times m}$ such that $V \approx WH$. When applying this process to document clustering, $V$ is a term document matrix. Each column in $H$ represents a document and the largest value represents its cluster. Each row in $H$ is a document cluster and each column is a document. Each row in $W$ is a term cluster and each column is a term.

The projected gradient method was used to solve the NMF problem [5] at INEX 2008. $V$ was a $8000 \times 114366$ term document matrix of BM25 weighted terms. The algorithm ran for a maximum of 70 iterations. It produced the $W$ and $H$ matrices. Document cluster membership was determined by the maximum value in each column of $H$. Term cluster membership can be determined by the maximum value in each column in $W$. NMF was run with $r$ at 15, 42 and 147 to match the submissions made with K-tree. NMF was run to aid in the comparison of K-tree to other methods.
### Table 6.1: Clustering Results Sorted by Micro Purity

<table>
<thead>
<tr>
<th>Name and Authors</th>
<th>Clusters</th>
<th>Micro Purity</th>
<th>Macro Purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>De Vries and Geva [19] (K-tree)</td>
<td>15</td>
<td>0.4949</td>
<td>0.5890</td>
</tr>
<tr>
<td>Tran et al. [74] (QUT LSK 3)</td>
<td>15</td>
<td>0.4928</td>
<td>0.5307</td>
</tr>
<tr>
<td>Kutty et al. [49] (QUT Entire collection 15)</td>
<td>15</td>
<td>0.4880</td>
<td>0.5051</td>
</tr>
<tr>
<td>De Vries and Geva [19] (NMF)</td>
<td>15</td>
<td>0.4732</td>
<td>0.5371</td>
</tr>
<tr>
<td>Tran et al. [74] (QUT LSK 1)</td>
<td>15</td>
<td>0.4518</td>
<td>0.5594</td>
</tr>
<tr>
<td>Tran et al. [74] (QUT LSK 4)</td>
<td>15</td>
<td>0.4476</td>
<td>0.4948</td>
</tr>
<tr>
<td>Tran et al. [74] (QUT LSK 2)</td>
<td>15</td>
<td>0.4442</td>
<td>0.5201</td>
</tr>
<tr>
<td>Zhang et al. [80] (Hagenbuchner)</td>
<td>15</td>
<td>0.3774</td>
<td>0.2586</td>
</tr>
</tbody>
</table>

#### 6.2 Clustering Task

Every team at INEX 2008 submitted at least one solution with 15 clusters. This allows for a direct comparison between different approaches. It only makes sense to compare results where the number of clusters are the same due to the purity metric used in evaluation. The submission of 15 clusters also matches the number of categories in the ground truth.

The K-tree performed well according to the macro and micro purity measures in comparison to the rest of the field. The difference in macro and micro purity for the K-tree submissions can be explained by the uneven distribution of cluster sizes. Figure 6.4 shows that many of the higher purity clusters are small. Macro purity is simply the average purity for all clusters. It does not take cluster size into account. Micro purity does take size into account by weighting purity in the average by the cluster size. Three types of clusters appear when splitting the x-axis in Figure 6.4 in thirds. There are very high purity clusters that are easy to find. In the middle there are some smaller clusters that have varying purity. The larger, lower purity clusters in the last third are hard to distinguish. Figures 6.2 and 6.3 show clusters sorted by purity and size. K-tree consistently found higher purity clusters than other submissions. Even with many small high purity clusters, K-tree achieved a high micro purity score. The distribution of cluster size in K-tree was less uniform than other submissions. This can be seen in Figure 6.3. It found many large clusters and many small clusters, with very few in between.

The K-tree submissions were determined by cosine similarity with the centroids produced by K-tree. The tree has an internal ordering of clusters as well. A comparison between the internal arrangement and cosine similarity is listed in Table 6.2. This data is based on a K-tree of order 40. Levels 1, 2 and 3 produced 17, 397 and 7384 clusters respectively. Level 3 is the above leaf or codebook vector level. The “left as is” method uses the K-tree as it is initially built. The rearranged method uses the K-tree when all vectors are reinserted into the tree to their nearest neighbour. The cosine method determines cluster membership by cosine similarity with the centroids produced. Nodes can become empty when reinserting vectors. This explains why levels 2 and 3 in the rearranged K-tree contain less clusters. Using cosine similarity with the centroids improved purity in almost all cases.
<table>
<thead>
<tr>
<th>Method</th>
<th>Clusters</th>
<th>Micro</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>left as is</td>
<td>17</td>
<td>0.4018</td>
<td>0.5945</td>
</tr>
<tr>
<td>left as is</td>
<td>397</td>
<td>0.5683</td>
<td>0.6996</td>
</tr>
<tr>
<td>left as is</td>
<td>7384</td>
<td>0.6913</td>
<td>0.7202</td>
</tr>
<tr>
<td>rearranged</td>
<td>17</td>
<td>0.4306</td>
<td>0.6216</td>
</tr>
<tr>
<td>rearranged</td>
<td>371</td>
<td>0.6056</td>
<td>0.7281</td>
</tr>
<tr>
<td>rearranged</td>
<td>5917</td>
<td>0.7174</td>
<td>0.7792</td>
</tr>
<tr>
<td>cosine</td>
<td>17</td>
<td>0.4626</td>
<td>0.6059</td>
</tr>
<tr>
<td>cosine</td>
<td>397</td>
<td>0.6584</td>
<td>0.7240</td>
</tr>
<tr>
<td>cosine</td>
<td>7384</td>
<td>0.7437</td>
<td>0.7286</td>
</tr>
</tbody>
</table>

Table 6.2: Comparison of Different K-tree Methods

Figure 6.2: Clusters Sorted By Purity
6.3 Summary

K-tree was introduced as a tool for document clustering for the first time. The results show that it is well suited for the task. It produces good quality clustering solutions and provides excellent performance in comparison to k-means when a large number of clusters are required. This work has been published in a peer reviewed publication at INEX 2008 by De Vries and Geva [19].
Figure 6.4: K-tree Breakdown
Medoid K-tree

The K-tree algorithm was initially designed to work with dense vectors. This causes space and efficiency problems when dealing with sparse document vectors. We will highlight the issue using the INEX 2008 XML Mining collection \cite{22}. It contains 114,366 documents and 206,868 stopped and stemmed terms. TF-IDF culling is performed by ranking terms. A rank is calculated by summing all weights for each term. The 8000 terms with the highest rank are selected. This reduced matrix contains 10,229,913 non-zero entries. A document term matrix using a 4 byte float requires 3.4 GB of storage. A sparsely encoded matrix with 2 bytes for the term index and 4 bytes for the term weighting requires 58.54 MB of storage. It is expected that sparse representation will also improve performance of a disk based implementation. More data will fit into cache and disk read times will be reduced.

Unintuitively, a sparse representation performs worse with K-tree. The root of the K-tree contains means of the whole collection. Therefore, the vectors contain all terms in the collection. This makes them dense. When building a K-tree, most of the time is spent near the root of the tree performing nearest neighbour searches. This explains why the sparse representation is slower. The most frequently accessed vectors are not sparse at all.

We propose an extension to K-tree where all cluster centres are document exemplars. This is inspired by the k-medoids algorithm \cite{41}. Exemplar documents are selected by choosing the nearest documents to the cluster centres produced by k-means clustering. Internal nodes now contain pointers to document vectors. This reduces memory requirements because no new memory is allocated for cluster centres. Run time performance is increased because all vectors are sparse. The trade-off for this increased performance is a drop in cluster quality. This is expected because we are throwing away information. Cluster centres in the tree are no longer weighted and are not updated upon insertion.

7.1 Experimental Setup

Performance and quality is compared between CLUTO \cite{45} and K-tree. The k-means and repeated bisecting k-means algorithms were chosen from CLUTO. The medoid K-tree was also used to select 10 percent of the corpus for sampling.
This sample was used to construct a K-tree. The resulting K-tree was used to perform a nearest neighbour search and produce a clustering solution. In all cases the K-tree order was adjusted to alter the number of clusters at the leaf level. CLUTO was then run to match the number of clusters produced. All algorithms were single threaded.

The INEX 2008 XML Mining corpus and a selection of the RCV1 corpus used by Song et. al. were clustered. The INEX collection consists of 114,366 documents from Wikipedia with 15 labels. The subset of the RCV1 corpus consists of 193,844 documents with 103 industry labels. Both corpora were restricted to the most significant 8000 terms. This is required to fit the data in memory when using K-tree with a dense representation.

Micro averaged purity and entropy are compared. Micro averaging weights the score of a cluster by its size. Purity and entropy are calculated by comparing the clustering solution to the labels provided. Run-times are also compared.

### 7.2 Experimental Results

The experimental results contain quality and run-time performance figures for five approaches. Two of the approaches are standard k-means approaches from the CLUTO toolkit. These are the k-means and repeated bisecting k-means methods. The repeated bisecting k-means approach is the same as earlier approaches described as Tree Structured Vector Quantisation (TSVQ). The other three approaches are K-tree based approaches. The original K-tree algorithm is used. The Medoid K-tree algorithm is used by itself. The Medoid K-tree is also used to sample 10 percent of the collection for use with the original K-tree algorithm. Figure 7.1 describes the meaning of symbols used in the plots. Figures show quality relationships and Figures and 7.7 show run-time performance.

#### 7.2.1 CLUTO

The k-means approach in CLUTO provides the best quality results as it is the only algorithm that performs global analysis at the required number of clusters. The repeated bisecting k-means approach produces results that are lower quality than the k-means approach. In this case the distortion measured in RMSE and the “ground truth” based metrics of purity and entropy are approximately aligned. This would suggest that the culled BM25 representation used for the INEX collection and the culled TF-IDF representation used for the RCV1 collection are meaningful in terms of document similarity. However, the TF-IDF representation does not maintain this relationship when comparing repeated bisecting k-means and K-tree on the RCV1 collection. The K-tree produces higher distortion clusters than repeated bisecting k-means but produces similar quality results in terms of purity and entropy as seen in Figures and 7.6. This is also contrary to previous results found by the authors of the CLUTO toolkit, Steinbach et. al., who found that the repeated bisecting k-means approach found the highest quality clusters. This could be explained by the number of documents contained in the respective collections. The number of documents contained in the collections used by Steinbach et. al. range from 690 to 3204 documents. The INEX and RCV1 collections contain 114,366 and 193,844
documents respectively. A lack of adequate statistics in the small collections could be favouring one approach to another. The conclude this affirmatively further experimentation is required. Subsets of the INEX and RCV1 collections could be sampled to match the size of the document collections presented by Steinbach et. al [73]. Then similar experimental conditions can be replicated to determine if this is the cause.

7.2.2 K-tree

K-tree produces large numbers of clusters in significantly less time than the standard approaches in CLUTO. The graphs show an early poor run-time performance of K-tree. This has been artificially inflated by choosing a large tree order. One can choose a smaller tree order and find a smaller number of clusters higher in the tree. The computational cost of k-means dominates when choosing a large K-tree order. Note that the k-means algorithms in K-tree and CLUTO differ. K-tree runs k-means to convergence using dense vectors. CLUTO stops after a specified number of iterations and uses sparse vectors.

The K-tree produces lower quality results than the k-means methods in CLUTO. In this experiment it is exaggerated because CLUTO uses cosine similarity and K-tree uses Euclidean distance. It is well known that cosine similarity is a better distance measure for document similarity [35]. Unfortunately, this fact was realised after the publication of these results. While the K-tree produces results close to the k-means methods the Medoid K-tree does not.

As can be seen in the graphs, the K-tree increases in performance as the tree order is made smaller, thus producing more clusters at the codebook level. This is an artificial result because a K-tree with a small order can be built and clusters higher in the tree used for more coarse grained clustering. This would result in a flat line for K-tree at different numbers of clusters because it can produce many granularities of clustering in a single pass. One could argue that k-means would need to be run at all granularities produced by K-tree. This would make k-means performance appear even worse.

The increased performance of K-tree at a small tree order can be explained by the fact that as the tree order becomes smaller, the search tree has more influence on computation time than k-means. The k-means algorithm converges quickly on a small number of vectors. Favouring the search tree favours a lower asymptotic complexity. A single lookup in the tree is logarithmic where as k-means is linear.

The difference in quality between the K-tree and the CLUTO methods may not be significant enough for humans to tell the difference in clustering results. This is just conjecture and the necessary experiments with humans involved in the loop are far beyond the scope of this thesis.

7.2.3 Medoid K-tree

The Medoid K-tree produces even poorer quality results than the original K-tree but it speeds the process up by a factor of 10 by exploiting the extremely sparse nature of document vectors. The most notable performance difference between CLUTO and K-tree exists when a large number of clusters are required. The poor quality of the Medoid K-tree can be explained by the sparsity of the document vectors. The document examplars in the root of the tree are unlikely
to contain vocabulary that overlaps with other document vectors and are therefore orthogonal to many other document vectors. When the document vectors in the root are orthogonal to documents being inserted then the placement of a document does not have meaning because they share no terms in common. Thus, the Medoid K-tree solves the sparsity problem in terms of speed but sacrifices a large amount of quality in return. This algorithm would not likely be particularly useful in practice due to the poor quality of results.

7.2.4 Sampling with Medoid K-tree

A 10 percent sample of well spread documents can be attained by selecting the codebook vectors in the Medoid K-tree of the appropriate order. As the nodes in the tree are approximately half full this requires a tree of order $25 (m = 25)$. Once the subset has been chosen the original K-tree algorithm is used to build a tree. The tree is then used to assign all documents in the collection to one of the finest grained clusters. This approach achieves higher quality clusters than the Medoid K-tree algorithm. This suggests it is better to randomly sample large collections rather than use an extremely lossy approach such as the Medoid K-tree. The end results is higher quality clusters given the same amount of computational resources. Further experimentation is required to determine if the Medoid K-tree selects better documents than random sampling.
Figure 7.2: INEX 2008 Purity
Figure 7.3: INEX 2008 Entropy
Figure 7.4: INEX 2008 Run Time
Figure 7.5: RCV1 Purity
Figure 7.6: RCV1 Entropy
Figure 7.7: RCV1 Run Time
7.3 Summary

The Medoid K-tree was introduced and able to exploit sparse representations of documents for a marked increase in performance. This work was published in a peer reviewed paper by De Vries and Geva \[20\] at SIGIR 2009.

Using the Medoid K-tree further increases run-time performance over the original K-tree but sacrifices quality. Sampling with the Medoid K-tree and producing final clusters with the original K-tree produces clusters of higher quality than the Medoid K-tree. All of the K-tree based solutions provide significant performance gains when large numbers of clusters are required. This leaves a trade off between performance and quality. The original K-tree algorithm provides a reasonable trade off where as the Medoid K-tree sacrifices too much quality. Further more, the K-tree algorithm makes clustering tractable when large number of clusters are required. The k-means and repeated bisecting k-means approaches can take days to run where the K-tree is done in minutes. Further research needs to run these experiments where the K-tree uses the cosine similarity distance measure rather than Euclidean distance. It would also be worthwhile performing the same comparison to the Random Indexing (RI) K-tree in Chapter 8.
Chapter 8

Random Indexing K-tree

Random Indexing (RI) \textsuperscript{[67]} is an efficient, scalable and incremental approach to the word space model. Word space models use the distribution of terms to create high dimensional document vectors. The directions of these document vectors represent various semantic meanings and contexts.

Latent Semantic Analysis (LSA) \textsuperscript{[21]} is a popular word space model. LSA creates context vectors from a document term occurrence matrix by performing a Singular Value Decomposition (SVD). Dimensionality reduction is achieved through projection of the document term occurrence vectors onto the subspace spanned by the vectors with the largest singular values in the decomposition. This projection is optimal in the sense that it minimises the variance between the original matrix and the projected matrix \textsuperscript{[31]}. In contrast, Random Indexing first creates random context vectors of lower dimensionality, and then combines them to create a term occurrence matrix in the dimensionally reduced space. Each term in the collection is assigned a random vector, and the document term occurrence vector is then a superposition of all the term random vectors. There is no matrix decomposition and hence the process is efficient.

The RI process is conceptually very different from LSA and does not have the same least squares optimality properties. The context vectors used by RI should optimally be orthogonal. Nearly orthogonal vectors can be used and have been found to perform similarly \textsuperscript{[11]}. These vectors can be drawn from a random Gaussian distribution. The Johnson and Linden-Strauss lemma \textsuperscript{[39]} states that if points are projected into a randomly selected subspace of sufficiently high dimensionality, then the distances between the points are approximately preserved. The same topology that exists in the higher dimensional space is reflected in the lower dimensional randomly selected subspace. Consequently, RI offers low complexity dimensionality reduction while still preserving the topological relationships amongst document vectors.

The time complexity of K-tree depends on the length of the document vectors. K-tree insertion incurs two costs, finding the appropriate leaf node for insertion and k-means invocation during node splits. It is therefore desirable to operate with lower dimensional vector representation.

The combination of RI with K-tree is a good fit. Both algorithms operate in an on-line and incremental mode. This allows it to track the distribution of
data as it arrives and changes over time. K-tree insertions and deletions allow flexibility when tracking data in volatile and changing collections. Furthermore, K-tree performs best with dense vectors, such as those produced by RI.

8.1 Modifications to K-tree

The K-tree algorithm was modified for use with RI. This modified version will be referred to as “Modified K-tree” and the original K-tree will be referred to as “Unmodified K-tree”.

All the document vectors created by RI are of unit length in the modified K-tree. Therefore, all centroids are normalised to unit length at all times. The k-means used for node splits in K-tree was changed to use randomised seeding and restart if it did not converge within six iterations. The process always converged quickly in experiments; although it is possible to constrain the number of restarts we did not find this to be necessary.

The original K-tree algorithm does not modify any of the centroids. They are simply the means of the vectors they represent. The k-means implementation runs to complete convergence and seeds centroids via perturbation of the global mean. To create two seeds the global mean is calculated and then the two seeds are created by moving away from the mean in opposite directions.

8.2 K-tree and Sparsity

K-tree was originally designed to operate with dense vectors. When a sparse representation is used, performance degrades even though there is significantly less data to process. The clusters in the top levels of the tree are means of most of the terms in the collection and are not sparse at all. The algorithm updates cluster centres along the insertion path in the tree. Since document vectors have very high dimensionality this becomes a very expensive process.

The Medoid K-tree [20] extended the algorithm to use a sparse representation and replace centroids with document examples. This improved run-time performance and decreased memory usage. Unfortunately it decreased quality when using sparse document vectors. The document examples in the root of the tree were almost orthogonal to new documents being inserted. The documents were unlikely to have meaningful overlap in vocabulary. This was also explained in Section 7.

The approach taken by De Vries and Geva at INEX 2008 [19] is a simple approach to dimensionality reduction or feature selection. It is called TF-IDF culling and it is performed by ranking terms. A rank is calculated by summing all weights for each term. The weights are the BM25 weight for each term in each document. This can also be explained as the sum of the column vector in the document by term matrix. The top $n$ terms with the highest rank are selected, where $n$ is the desired dimensionality. This works particularly well with term occurrences due to the Zipf law distribution of terms [83]. The collection frequency of a term is inversely proportional to its rank according to collection frequency. Most of the term weights are contained in the most frequent terms. This was also explained in Section 3.3.2.
8.3 Random Indexing Definition

In RI, each dimension in the original space is given a randomly generated index vector. The index vectors are high dimensional, sparse and ternary. Sparsity is controlled via a seed length that specifies the number of randomly selected non-zero dimensions. Ternary vectors consist of randomly distributed +1 and -1 values in the non-zero dimensions.

In the context of document clustering, RI can be viewed as a matrix multiplication of a document by term matrix, \( D \), and a term by index-vector matrix, \( I \). Alternatively, \( I \) can be referred to as a random projection matrix. Each row vector in \( D \) represents a document, each row vector in \( I \) is an index vector, \( n \) is the number of documents, \( t \) is the number of terms and \( r \) is the dimensionality of the reduced space. \( R \) is the reduced matrix where each row vector represents a document,

\[
D_{n \times t} I_{t \times r} = R_{n \times r}.
\]

(8.1)

RI has several advantages. It can be performed incrementally and on-line as data arrives. Any document can be indexed (i.e. encoded as an RI vector) independently from all other documents in the collection. This eliminates the need to build and store the entire document by term matrix. Additionally, newly encountered dimensions (terms) in the document collection are easily accommodated without having to recalculate the projection of previously encoded documents. In contrast, SVD requires global analysis where the number of documents and terms are fixed. The time complexity of RI is also very attractive. It is linear in the number of terms in a document and independent of collection size.

8.4 Choice of Index Vectors

The index vectors used in RI were chosen to be sparse and ternary. Ternary index vectors for RI were introduced by Achlioptas [3] as being well suited for database environments. The primary concern of sparse index vectors is reducing time and space complexity. Bingham and Mannila [11] run experiments indicating that sparse index vectors do not affect the quality of results. This is not the only choice when creating index vectors. Kanerva [40] introduces binary spatter codes. Plate [61] explores Holographic Reduced Representations that consist of dense vectors with floating point values.

8.5 Random Indexing Example

In practice, to construct a document vector, the document vector is initially set to zero, and then the sparse index vector for each term in the document is added to the document vector. The weight of the added term index vector may be determined by TF-IDF or another weighting scheme. When all terms have been added, the document vector is normalised to unit length. There is no need to explicitly form the random projection matrix in Equation\(8.1\) up-front. The random index vectors for each term can be generated and stored as they
are first encountered. The fact that each index vector is sparse means that the vectors use less memory to store and are faster to add.

The effect of this approach is that each document will have a particular signature that can be compared with other documents via cosine similarity. The document signature is thus a vector on the unit hyper-sphere.

In the simple scenario in Figure 8.1, the index vectors for the four words travel, mars, space and telescope, are added to the document vector as they are encountered in the text of the document. Afterwards, the document should be normalised.

The sparse index vectors can be efficiently stored by simply storing the position of the non-zero entries with the sign of the position indicating whether it is one or negative one.

8.6 Experimental Setup

Experiments have been run to measure the quality difference between various configurations of K-tree. Table 8.1 lists all the configurations tested.

The following conditions were used when running the experiments.

1. Each K-tree configuration was run a total of 20 times.

2. The documents were inserted in a different random order each time K-tree is built.

3. If RI was used, the index vectors were generated statistically independently each time the K-tree was built.

4. For each K-tree built, k-means++ [6] was run 20 times on the codebook vectors to create 15 clusters.

5. All document vectors were unitised after performing dimensionality reduction.

The conditions listed above resulted in 400 measurements for each K-tree configuration. For each of the 20 K-trees built, k-means++ was run 20 times. The repetition of the experiments is to measure the variance caused by the
Assessment of clustering quality is based on the INEX 2008 XML Mining track. The set of 114,366 documents, belonging to 15 classes were used to evaluate clustering quality of INEX submissions. The cluster labels are taken from the Wikipedia itself. K-tree generates clusters in an unsupervised manner, and it is not necessarily going to produce 15 clusters at a particular level in the tree. In order to re-use the INEX test collection, it was necessary to post process the K-tree and to reduce a cluster level in the tree to 15 clusters by using k-means++. Note that this is a low cost operation involving only a small number of vectors, which is not required in an ordinary application. It is done for the sole purpose of producing comparable results with the INEX benchmark data. The same approach was taken at INEX 2008 by De Vries and Geva [19]. For a comparison of entropy and purity to be meaningful they have to be measured on the same number of clusters.

Micro averaged purity and entropy are compared. Micro averaging weights the score of a cluster by its size. Purity and entropy are calculated by comparing the clustering solution to the labels provided. A higher purity score indicates a higher quality solution because the clusters are more pure with respect to the ground truth. A lower entropy score indicates a higher quality solution because there is more order with respect to the ground truth.

8.7 Experimental Results

Tables 8.3, 8.4, 8.5, 8.6 and 8.7 contain results for the K-tree configurations listed in Table 8.1. Table 8.2 lists the meaning of the symbols used. Figures 8.2 and 8.3 are graphical representations of the average micro purity and entropy.

The unmodified K-tree using TF-IDF culling and BM25 had unexpected results as seen in Table 8.3. The average micro purity and entropy peaked at 400 dimensions. Performing this dimensionality reduction at these lower dimensions had not been performed before. This is an interesting and unexpected result and future experiments will need to determine if the phenomenon occurs in different corpora.

Improvements in micro purity have been tested for significance via t-tests. The null hypothesis is that both results come from the same distribution with the same mean. In this case they are not significantly different. If the null hypothesis is rejected then the difference is statistically significant.

The modifications made to K-tree for use with RI had a significant impact. The unmodified K-tree and modified K-tree were compared. Specifically, configurations B and D, and configurations C and E were tested against each other. All dimensions were compared against each other. The improved performance of the modified K-tree was statistically significant for all dimensions (100 vs 100, 200 vs 200 and so on) with a p-value of 0 or extremely close to 0 ($p < 1 \times 10^{-100}$).

The modified K-tree using RI was tested with two representations. Configurations D and E were tested at all dimensions. The null hypothesis was rejected at all dimensions except 10000. This means that BM25 performed significantly better than the BM25 + LF-IDF representation at all dimensions except 10000. At 10000 dimensions the difference was not considered statistically significant with a p-value of 0.3. The increased performance of this representation in clas-
sification did not apply to clustering when using RI. The LF-IDF representation may be interfering with the BM25 representation and approaches such as reducing the weight of LF-IDF in the RI process or performing RI separately on each representation and then concatenating the reduced vectors may improve performance. Running k-means on the full sparse vectors will also indicate if RI is responsible for this. Further experimentation is required to provide more evidence for this result.

The unexpected results in configuration A were tested against the best RI configuration, E. The highest average at 400 dimensions in configuration A was tested against all dimensions in configuration E (400 vs 100, 400 vs 200, 400 vs 400, 400 vs 1000 and so on). The RI K-tree, configuration E, became statistically more significant at 2000 dimensions with a p-value of $1.48 \times 10^{-6}$ and thus rejected the null hypothesis. For dimensions 4000 through 10000, the performance difference was statistically significant, with a p-value of 0 in all cases. Thus, RI K-tree improves results, even over the unexpected high results of configuration A, by embedding the original 200,000 dimensional term space into at least a 2000 dimension reduced space.

### 8.8 INEX Results

The INEX XML Mining track is a collaborative evaluation forum where research teams improve approaches in supervised and unsupervised machine learning with XML documents. Participants make submissions and the evaluation results are later released.

The RI K-tree in configuration E performs on average at a comparable level to the best results submitted to the INEX 2008 XML Mining track. The top two results from the track had a micro purity of 0.49 and 0.50. These are not average scores for the approaches but the best results participants found. The RI K-tree in configuration E had a maximum micro entropy of 0.55. This is 10 percent greater than the INEX submissions.

### 8.9 Summary

The RI K-tree was introduced as an attractive approach to clustering large and frequently changing document collections. It was able to improve the results over the original K-tree algorithm. This work has been peer review and published in the proceedings on ADCS 2009 by De Vries et. al. [13].

---

<table>
<thead>
<tr>
<th>ID</th>
<th>K-tree</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Unmodified</td>
<td>TF-IDF Culling, BM25</td>
</tr>
<tr>
<td>B</td>
<td>Unmodified</td>
<td>RI, BM25 + LF-IDF</td>
</tr>
<tr>
<td>C</td>
<td>Unmodified</td>
<td>RI, BM25</td>
</tr>
<tr>
<td>D</td>
<td>Modified</td>
<td>RI, BM25 + LF-IDF</td>
</tr>
<tr>
<td>E</td>
<td>Modified</td>
<td>RI, BM25</td>
</tr>
</tbody>
</table>

Table 8.1: K-tree Test Configurations
## Table 8.2: Symbols for Results

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>Average Micro Entropy</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Standard Deviation of $\alpha$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Average Micro Purity</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Standard Deviation of $\gamma$</td>
</tr>
</tbody>
</table>

Table 8.3: A: Unmodified K-tree, TF-IDF Culling, BM25

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2.6299</td>
<td>0.0194</td>
<td>0.3981</td>
<td>0.0067</td>
</tr>
<tr>
<td>200</td>
<td>2.4018</td>
<td>0.0207</td>
<td>0.4590</td>
<td>0.0085</td>
</tr>
<tr>
<td>400</td>
<td>2.2762</td>
<td>0.0263</td>
<td>0.4814</td>
<td>0.0093</td>
</tr>
<tr>
<td>800</td>
<td>2.2680</td>
<td>0.0481</td>
<td>0.4768</td>
<td>0.0155</td>
</tr>
<tr>
<td>1000</td>
<td>2.2911</td>
<td>0.0600</td>
<td>0.4703</td>
<td>0.0192</td>
</tr>
<tr>
<td>2000</td>
<td>2.3302</td>
<td>0.0821</td>
<td>0.4569</td>
<td>0.0254</td>
</tr>
<tr>
<td>4000</td>
<td>2.3751</td>
<td>0.1103</td>
<td>0.4401</td>
<td>0.0331</td>
</tr>
<tr>
<td>8000</td>
<td>2.3868</td>
<td>0.1068</td>
<td>0.4402</td>
<td>0.0300</td>
</tr>
<tr>
<td>10000</td>
<td>2.3735</td>
<td>0.1062</td>
<td>0.4431</td>
<td>0.0306</td>
</tr>
</tbody>
</table>

Table 8.4: B: Unmodified K-tree, Random Indexing, BM25 + LF-IDF

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3.0307</td>
<td>0.0149</td>
<td>0.3093</td>
<td>0.0045</td>
</tr>
<tr>
<td>200</td>
<td>2.9295</td>
<td>0.0206</td>
<td>0.3300</td>
<td>0.0079</td>
</tr>
<tr>
<td>400</td>
<td>2.7962</td>
<td>0.0379</td>
<td>0.3648</td>
<td>0.0143</td>
</tr>
<tr>
<td>800</td>
<td>2.6781</td>
<td>0.0718</td>
<td>0.3921</td>
<td>0.0236</td>
</tr>
<tr>
<td>1000</td>
<td>2.6509</td>
<td>0.0842</td>
<td>0.3959</td>
<td>0.0260</td>
</tr>
<tr>
<td>2000</td>
<td>2.6315</td>
<td>0.1262</td>
<td>0.3908</td>
<td>0.0345</td>
</tr>
<tr>
<td>4000</td>
<td>2.6380</td>
<td>0.1451</td>
<td>0.3860</td>
<td>0.0356</td>
</tr>
<tr>
<td>8000</td>
<td>2.6371</td>
<td>0.1571</td>
<td>0.3844</td>
<td>0.0382</td>
</tr>
<tr>
<td>10000</td>
<td>2.6302</td>
<td>0.1540</td>
<td>0.3876</td>
<td>0.0385</td>
</tr>
<tr>
<td>Dimensions</td>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>$\gamma$</td>
<td>$\delta$</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>100</td>
<td>0.0404</td>
<td>0.0104</td>
<td>0.0500</td>
<td>0.0004</td>
</tr>
<tr>
<td>200</td>
<td>0.0428</td>
<td>0.0120</td>
<td>0.0540</td>
<td>0.0010</td>
</tr>
<tr>
<td>400</td>
<td>0.0477</td>
<td>0.0166</td>
<td>0.0600</td>
<td>0.0020</td>
</tr>
<tr>
<td>800</td>
<td>0.0560</td>
<td>0.0240</td>
<td>0.0700</td>
<td>0.0030</td>
</tr>
<tr>
<td>1000</td>
<td>0.0620</td>
<td>0.0306</td>
<td>0.0770</td>
<td>0.0040</td>
</tr>
<tr>
<td>2000</td>
<td>0.0686</td>
<td>0.0342</td>
<td>0.0841</td>
<td>0.0056</td>
</tr>
<tr>
<td>4000</td>
<td>0.0746</td>
<td>0.0380</td>
<td>0.0912</td>
<td>0.0080</td>
</tr>
<tr>
<td>8000</td>
<td>0.0800</td>
<td>0.0420</td>
<td>0.0986</td>
<td>0.0110</td>
</tr>
<tr>
<td>10000</td>
<td>0.0850</td>
<td>0.0460</td>
<td>0.1060</td>
<td>0.0130</td>
</tr>
</tbody>
</table>

Table 8.5: Unmodified K-tree, Random Indexing, BM25

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.0404</td>
<td>0.0104</td>
<td>0.0500</td>
<td>0.0004</td>
</tr>
<tr>
<td>200</td>
<td>0.0428</td>
<td>0.0120</td>
<td>0.0540</td>
<td>0.0010</td>
</tr>
<tr>
<td>400</td>
<td>0.0477</td>
<td>0.0166</td>
<td>0.0600</td>
<td>0.0020</td>
</tr>
<tr>
<td>800</td>
<td>0.0560</td>
<td>0.0240</td>
<td>0.0700</td>
<td>0.0030</td>
</tr>
<tr>
<td>1000</td>
<td>0.0620</td>
<td>0.0306</td>
<td>0.0770</td>
<td>0.0040</td>
</tr>
<tr>
<td>2000</td>
<td>0.0686</td>
<td>0.0342</td>
<td>0.0841</td>
<td>0.0056</td>
</tr>
<tr>
<td>4000</td>
<td>0.0746</td>
<td>0.0380</td>
<td>0.0912</td>
<td>0.0080</td>
</tr>
<tr>
<td>8000</td>
<td>0.0800</td>
<td>0.0420</td>
<td>0.0986</td>
<td>0.0110</td>
</tr>
<tr>
<td>10000</td>
<td>0.0850</td>
<td>0.0460</td>
<td>0.1060</td>
<td>0.0130</td>
</tr>
</tbody>
</table>

Table 8.6: Modified K-tree, Random Indexing, BM25 + LF-IDF

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.0404</td>
<td>0.0104</td>
<td>0.0500</td>
<td>0.0004</td>
</tr>
<tr>
<td>200</td>
<td>0.0428</td>
<td>0.0120</td>
<td>0.0540</td>
<td>0.0010</td>
</tr>
<tr>
<td>400</td>
<td>0.0477</td>
<td>0.0166</td>
<td>0.0600</td>
<td>0.0020</td>
</tr>
<tr>
<td>800</td>
<td>0.0560</td>
<td>0.0240</td>
<td>0.0700</td>
<td>0.0030</td>
</tr>
<tr>
<td>1000</td>
<td>0.0620</td>
<td>0.0306</td>
<td>0.0770</td>
<td>0.0040</td>
</tr>
<tr>
<td>2000</td>
<td>0.0686</td>
<td>0.0342</td>
<td>0.0841</td>
<td>0.0056</td>
</tr>
<tr>
<td>4000</td>
<td>0.0746</td>
<td>0.0380</td>
<td>0.0912</td>
<td>0.0080</td>
</tr>
<tr>
<td>8000</td>
<td>0.0800</td>
<td>0.0420</td>
<td>0.0986</td>
<td>0.0110</td>
</tr>
<tr>
<td>10000</td>
<td>0.0850</td>
<td>0.0460</td>
<td>0.1060</td>
<td>0.0130</td>
</tr>
</tbody>
</table>

Table 8.7: Modified K-tree, Random Indexing, BM25
Figure 8.2: Purity Versus Dimensions

Figure 8.3: Entropy Versus Dimensions
This chapter defines the time complexity analysis of the K-tree algorithm. As the K-tree algorithm uses the k-means algorithm, time complexity analysis of the k-means algorithm is also required.

9.1 k-means

If k-means is limited to a given number of iterations, $I$, it becomes linear with all inputs. Some variables need to be defined to show this. The dimensionality of the vectors being clustered is $d$. $C$ is a $k \times d$ 2-dimensional array of real numbers where $k$ is the desired number of clusters. $C$ can also be referred to as the set of cluster centres, centroids or means. The vectors in $C$ can be initialised by randomly selecting $k$ vectors from $V$ or using a more intelligent initialisation procedure. $V$ is a $v \times d$ 2-dimensional array of real numbers where $v$ is the number of vectors being clustered. $V$ can also be referred to as the set of vectors.

By inspection of the algorithm in Figure 9.1 it is easy to see that it is linear with respect to $I$, $c$, $v$ and $d$. To analyse the complexity of k-means we will investigate each of the procedures in Figure 9.1. The exact number of operations have been listed in Tables 9.1, 9.2, 9.3 and 9.4. Array index, assignment and arithmetic operators are considered operations. Loop counters are ignored.

The time cost function of k-means for a fixed number of iterations is listed in Table 9.1. As $v, k, d, I \rightarrow \infty$, $vkdI$ dominates the growth of this function giving it the asymptotic time complexity of $O(vkdI)$.

The use of k-means in the K-tree algorithm is limited to operating on vectors in a single node. The size of a node is controlled by the tree order $m$. Therefore, every time k-means is run it will take no longer than a fixed limit. For the analysis on K-tree the k-means algorithm running time can be treated as a constant because the number of vectors $v$ is always fixed to $m+1$. The number of clusters $k$ is fixed to two and the dimensionality $d$ of all the vectors are the same. The algorithm may converge before the maximum number of iterations specified. The algorithm listed in Table 9.1 does not check for convergence and assumes a worst case running time of the maximum number of iterations. However, in practice it has been observed that k-means converges very quickly when
clustering a small number of vectors into two clusters. Arthur et. al. [5] have performed a smoothed complexity analysis that explains the quick convergence of k-means theoretically.

### 9.2 K-tree

The K-tree algorithm is inspired by the B⁺-tree to provide an efficient nearest neighbour search tree.

#### 9.2.1 Worst Case Analysis

The k-means algorithm is always run on \( m + 1 \) vectors. In the worst case it will form two clusters of size \( m \) and 1. Given a topology and insertion order

<table>
<thead>
<tr>
<th>Line</th>
<th>Operations</th>
<th>Times Executed</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>( v )</td>
<td>( v )</td>
</tr>
<tr>
<td>4</td>
<td>1 + 7( d ) + 1</td>
<td>( vk )</td>
<td>( vk(7d + 2) = 7vkd + 2vk )</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>( vk )</td>
<td>( vk )</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>( vk )</td>
<td>( 2vk )</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>( vk )</td>
<td>( vk )</td>
</tr>
</tbody>
</table>

| Total |                  |                | \( 7vkd + 6vk + v \) |

Table 9.3: NearestNeighbours Analysis
K-Means$(I, C, V)$
1. $N = \text{NearestNeighbours}(C, V)$
2. for $i = 1$ to $I$
   3. $\text{UpdateMeans}(C, V, N)$
4. $N = \text{NearestNeighbours}(C, V)$
5. return $N$

NearestNeighbours$(C, V)$
1. for $v = 1$ to $V$.length
   2. nearestDistance = $\infty$
   3. for $k = 1$ to $C$.length
      4. distance = $\text{EuclideanDistanceSquared}(C[k], V[v])$
      5. if distance < nearestDistance
         6. $N[v] = k$
         7. nearestDistance = distance
8. return $N$

EuclideanDistanceSquared$(X, Y)$
1. distance = 0
2. for $i = 1$ to $X$.length
   3. difference = $X[i] - Y[i]$
   4. distance = distance + (difference $\times$ difference)
5. return distance

UpdateMeans$(C, V, N)$
1. for $k = 1$ to $C$.length
   2. clusterSize$[k] = 0$
   3. for $d = 1$ to $C[k]$.length
      4. $C[k, d] = 0$
5. for $v = 1$ to $V$.length
   6. for $d = 1$ to $V[v]$.length
      7. $n = N[v]$
      8. clusterSize$[n] = \text{clusterSize}[n] + 1$
      9. $C[n, d] = C[n, d] + V[v, d]$
10. for $k = 1$ to $C$.length
   11. for $d = 1$ to $C[k]$.length
      12. $C[k, d] = C[k, d] / \text{clusterSize}[k]$

Figure 9.1: The k-means algorithm
that causes each new vector inserted to always reach the same leaf node and
k-means performs the worst possible splits, a K-tree can consist of many nodes
containing only one vector and have a large depth. Every node along the path
to the node where all insertions occur will have to be full, except the root which
contains at least two vectors. This has been illustrated in a K-tree of order four
in Figure 9.2.

The height of the tree in the worst case can be calculated by Equation 9.1
where \( h \) is the tree height, \( v \) is the number of vectors inserted into the tree and
\( m \) is the tree order. For example, a tree where \( v = 1000000 \) and \( m = 50 \) will
have \( h = 20409 \). Thus, the tree height is only a constant factor smaller than
the number of vectors inserted into the tree. This formula has been derived by
the observation made in Figure 9.2 where one leaf node is full and all others
contain one vector. Each new level where \( h > 1 \) contributes \( m - 1 \) leaf nodes
once it becomes full.

\[
h = \left\lceil \frac{v - m}{m - 1} \right\rceil + 1 \quad (9.1)
\]

Now that the height of a worst case K-tree can be determined, the number
of splits, \( s \), can be determined. The number of splits can be calculated by a
summation derived from the observation in Figure 9.2 that each new level has
created \( l - 1 \times m - 1 \) splits where \( l \) is the level number in the tree. Note that
splits never occur on the root, they are only promoted to the root. Once a split
it performed on the root, it is no longer the root as the two means are promoted
to the new root. This is why the summation runs from \( 2 \to h \).

\[
s = \sum_{l=2}^{h} (l - 1)(m - 1) \quad (9.2)
\]

\[
s = \sum_{k=1}^{n} k = \frac{1}{2} n(n + 1) = \Theta(n^2) \quad (9.3)
\]

As \( h \) is \( O(n) \), the summation in Equation 9.2 fits the summation in Equation
9.3 which is \( \Theta(n^2) \). As \( v \) vectors have to be inserted which is \( O(n) \) and have

<table>
<thead>
<tr>
<th>Line</th>
<th>Operations</th>
<th>Times Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1 + 7vkd + 6vk + v )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>3</td>
<td>( 10vd + 7kd + 2k )</td>
<td>( I )</td>
</tr>
<tr>
<td>4</td>
<td>( 1 + 7vkd + 6vk + v )</td>
<td>( I )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1 + 7vkd + 6vk + v )</td>
</tr>
<tr>
<td>3</td>
<td>( I(10vd + 7kd + 2k) = 10vdI + 7kdI + 2kI )</td>
</tr>
<tr>
<td>4</td>
<td>( I(1 + 7vkd + 6vk + v) = I + 7vkdI + 6vkI + vI )</td>
</tr>
</tbody>
</table>

Total  \( 7vkdI + 6vkI + 10vdI + 7kdI + 7vkd + 2kI + 6vk + vI + v + I + 1 \)

Table 9.4: K-Means Analysis
Figure 9.2: Worst Case K-tree

to on average visit $\frac{h}{2}$ nodes which is $O(n)$ leaving the insertion process with an $O(n^2)$ complexity. So the worst case running time of K-tree is $O(n^2)$.

A topology that causes a K-tree to form like this would be bizarre. All points inserted always have to be nearest to the left hand side of the tree as per Figure 9.2. In all experiments run thus far it has never been observed that K-tree has an $O(n^2)$ time complexity and has always run faster than the k-means algorithm. It is likely that an average case analysis will provide a more realistic analysis.

9.2.2 Average Case Analysis

The average case will be considered to consist of a K-tree where every node is on average half full. The average fill rate of a node will be denoted by $t$ where $t = \frac{m}{2}$. Therefore the analysis becomes similar to the worst case for the B$^+$-tree. The worst case complexity of building a B$^+$-tree is well known to be $O(n \log n)$ where $n$ is the number of records indexed by the tree. As with the analysis of k-means, $n$ could be replaced by $vd$ where $v$ is the number of vectors, $d$ the dimensionality of the data. Alternatively, $n$ could be considered the number of
Figure 9.3: Average Case K-tree

bytes used by a representation, thus having the same effect as \( vd \) for a dense representation.

Figure 9.3 shows an average case K-tree where every node contains on average \( t \) vectors. Level 1 contains 1 node. Level 2 contains \( t \) nodes. Level 3 contains \( t^2 \) nodes. Level 4 contains \( t^3 \) nodes and so on. Equation 9.4 gives the average expected height of a K-tree indexing \( v \) vectors. Equation 9.5 gives the average expected splits in a K-tree of height \( h \). Given that there are \( v \) vectors to insert and each will take approximately \( \log t \) time to insert it is easy to see this will take \( O(n \log n) \) time.

\( h \approx \log_t v \) \hspace{1cm} (9.4)

\( s \approx \sum_{l=2}^{h} t^{(l-1)} \) \hspace{1cm} (9.5)

9.2.3 Testing the Average Case Analysis

The average case complexity of K-tree has been tested using the INEX 2009 corpus consisting of 2,666,190 Wikipedia documents. The documents have been represented as 1000 dimensional Random Indexed (RI) dense vectors. A K-tree where \( m = 40 \) was built. The execution time does not include RI or the loading time of the dataset from disk. The interest is in the running time of the K-tree algorithm. The time elapsed since beginning of the tree building process was sampled every 26661 documents resulting in 100 measurements. These measurements are represented by the dots labelled “INEX09” in Figure 9.4. The Matlab curve fitting toolbox was used to find a constant, \( c \), that gives \( f(n) = c(n \log n) \) a best fit to the data where \( n \) is the x-axis value and \( t = \frac{m}{2} \). It found that the best fit was \( c = 0.00007494 \) where \( RMSE = 13.88 \). The resulting K-tree was 6 levels deep with an average of 16.76 vectors per node.
9.3 Summary

This chapter defined the time complexity analysis of the K-tree algorithm. The average case analysis was supported by empirical evidence when fitting execution times to the average case complexity.
Automated text classification into topical categories dates back to the 1960s [69]. Up until the late 1980s it was common for experts to manually build classifiers by means of knowledge-engineering approaches. With the growth of the Internet, advances in machine learning and significant increases in computing power, there was an increased interest in text classification. Supervised machine learning techniques automatically build a classifier by “learning” from a set of (document, category) pairs. This eliminated the costly expert manpower needed in the knowledge-engineering approach [69].

With the infiltration of the information age into all aspects of society, supervised learning techniques such as text classification are starting to show their weaknesses. Just as knowledge-engineering approaches were costly compared to supervised learning approaches, supervised learning approaches are expensive compared to unsupervised approaches. Manpower is required to label documents with topical categories. The quality and quantity of the provided categories greatly determine the quality of the classifiers. Now that the data deluge has begun producing billions of documents on the Internet, unsupervised learning techniques are much more practical and effective [34]. Supervised learning is still useful in situations where there are large quantities of high quality category information available. It is also useful as an evaluation tool for different representations. This is discussed in Section 5.1.

Text classification is used in many contexts such as document filtering, automated meta-data generation, word sense disambiguation and population of hierarchical catalogues of Web resources. It can be applied in any situation requiring document organisation or selective dispatching [69]. Joachims [38] describes text classification as a key technique for handling and organising text data for tasks such as classification of news stories, finding interesting information on the WWW and to guide a user’s search through hypertext. Text classification is seen as the meeting point of machine learning and information retrieval.

The goal of text classification is to learn a function that predicts if a document belongs to a given category. This function is also called the classifier, model or hypothesis [69]. The categories are symbolic labels with no additional information regarding to their semantic meaning. The assignment of documents to
categories is determined by features extracted from the documents themselves, thus deriving the semantic meaning from the documents, rather than the labels.

10.1 Support Vector Machines

Joachims [38] highlights theoretically why SVMs are an excellent choice for text classification. The author finds that empirical results support the theoretical findings. The SVM is a relatively new learning method that is well-founded in terms of computational learning theory and are very open to theoretical understandings and analysis [38].

SVMs are based on the Structural Risk Minimisation principle that finds a hypothesis that guarantees the lowest true error. The true error is the probability that the hypothesis will make an error on an unseen and randomly selected test example [38].

SVMs are very universal learners that learn a linear threshold function. However, kernel functions can be used to learn polynomial classifiers, radial basis function (RBF) networks and three-layer sigmoid neural networks [38]. They can learn independent of the the dimensionality of the feature space, thus allowing them to generalise even when there are many features [38]. This makes them particularly effective for text classification where many features exist. For example, the INEX 2008 XML Mining collection has approximately 200,000 features based on text alone. Document vectors are sparse and SVMs work well with sparse representations. Furthermore, most text classification problems are linearly separable [38].

Joachims [38] compares SVMs to Bayes, Rocchio, C4.5 and $k$NN classifiers. It is found that SVMs perform better than all other classifiers with both polynomial and RBF kernels. Furthermore, SVMs train faster than all other methods except for C4.5 which trains in a similar amount of time. Additionally, SVMs do not require any parameter tuning as they can find good parameters automatically.

10.2 INEX

Classification of documents in the INEX 2008 XML Mining collection was completed using an SVM and content and link information. This approach allowed evaluation of the different document representations. It allowed the most effective representation to be chosen for clustering.

SVM\textsuperscript{multiclass} [75] was trained with TF-IDF, BM25 and LF-IDF representations of the corpus. BM25 and LF-IDF feature vectors were concatenated to train on both content and link information simultaneously. Submissions to INEX were made only using BM25, LF-IDF or both because BM25 out performed TF-IDF.

10.3 Classification Results

Table 10.1 lists the results for the classification task. They are sorted in order of decreasing recall. Recall is simply the accuracy of predicting labels for
documents not in the training set. Concatenating the link and content representations did not drastically improve performance. Further work has been subsequently performed to improve classification accuracy. This further work is listed in the table with the “improved” term.

### 10.4 Improving Classification Results

Several approaches have been carried out to improve classification performance. All features were used for text and links, where the official INEX 2008 results were culled to the top 8000 features using the unsupervised feature selection approach described in Section 3.3.2. Links were classified without LF-IDF weighting. This was to confirm LF-IDF improved results. Document length normalisation was removed from LF-IDF. It was noticed that many vectors in the link representation contained no features. Therefore, inbound links were added to the representation. For $i$, the source document and $j$, the destination document, a weight of one is added to the $i, j$ position in the document-to-document link matrix. This represents an outbound link. To represent an inbound link, $i$ is the destination document and $j$ is the source document. Thus, if a pair of documents both link to each other they receive a weight of two in the corresponding columns in their feature vectors. Links from the entire Wikipedia were inserted into this matrix. This allows similarity to be associated on inbound and outbound links outside the XML Mining subset. This extends the $114,366 \times 114,366$ document-to-document link matrix to a $114,366 \times 486,886$ matrix. Classifying links in this way corresponds to the idea of hubs and authorities in HITS [47]. Overlap on outbound links indicates the document is a hub. Overlap on inbound links indicates the document is an authority.

<table>
<thead>
<tr>
<th>Name</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>De Vries and Geva [19] (improved text and links)</td>
<td>0.8372</td>
</tr>
<tr>
<td>De Vries and Geva [19] (improved links)</td>
<td>0.7920</td>
</tr>
<tr>
<td>De Vries and Geva [19] (improved text)</td>
<td>0.7917</td>
</tr>
<tr>
<td>Gery et. al. [29] (Expe 5 tf idf T5 10000)</td>
<td>0.7876</td>
</tr>
<tr>
<td>Gery et. al. [29] (Expe 3 tf idf T4 10000)</td>
<td>0.7874</td>
</tr>
<tr>
<td>Gery et. al. [29] (Expe 1 tf idf TA)</td>
<td>0.7874</td>
</tr>
<tr>
<td>De Vries and Geva [19] (Vries text and links)</td>
<td>0.7849</td>
</tr>
<tr>
<td>De Vries and Geva [19] (Vries text only)</td>
<td>0.7798</td>
</tr>
<tr>
<td>Chidlovskii [14] (Boris inex tfidf1 sim 0.38.3)</td>
<td>0.7347</td>
</tr>
<tr>
<td>Chidlovskii [14] (Boris inex tfidf sim 037 it3)</td>
<td>0.7340</td>
</tr>
<tr>
<td>Chidlovskii [14] (Boris inex tfidf sim 034 it2)</td>
<td>0.7310</td>
</tr>
<tr>
<td>Gery et. al. [29] (Expe 4 tf idf T5 100)</td>
<td>0.7231</td>
</tr>
<tr>
<td>Kaptein and Kaamps [41] (Kaptein 2008NBscoresv02)</td>
<td>0.6981</td>
</tr>
<tr>
<td>Kaptein and Kaamps [41] (Kaptein 2008run)</td>
<td>0.6979</td>
</tr>
<tr>
<td>de Campos et. al. [17] (Romero nave bayes)</td>
<td>0.6767</td>
</tr>
<tr>
<td>Gery et. al. [29] (Expe 2.tf idf T4 100)</td>
<td>0.6771</td>
</tr>
<tr>
<td>de Campos et. al. [17] (Romero nave bayes links)</td>
<td>0.6814</td>
</tr>
<tr>
<td>De Vries and Geva [19] (Vries links only)</td>
<td>0.6233</td>
</tr>
</tbody>
</table>

Table 10.1: Classification Results
links indicates the document is an authority. The text forms a 114,366×206,868 document term matrix when all terms are used. The link and text representation were combined using two methods. In the first approach text and links were classified separately. The ranking output of the SVMs were used to choose the most appropriate label. We call this SVM by committee. Secondly, both text and link features were converted to unit vectors and concatenated forming a 114,366×693,754 matrix. Table 10.2 highlights the performance of these improvements.

The new representation for links has drastically improved performance from a recall of 0.62 to 0.79. It is now performing as well as text based classification. However, the BM25 parameters have not been optimised. This could further increase performance of text classification. Interestingly, 97 percent of the correctly labelled documents for text and link classification agree when performing the SVM by committee approach to combining representations. To further explain this phenomenon, a histogram of cosine similarity of text between linked documents was created. Figure 10.1 shows this distribution for the links in the XML Mining subset. Most linked documents have a high degree of similarity based on their text content. Therefore, it is valid to assume that linked documents are highly semantically related. By combining text and link representations we can disambiguate many more cases. This leads to an increase in performance from 0.7920 to 0.8372 recall. The best results for text, links and both combined, performed the same under 10 fold cross validation using a randomised 10 percent train and 90 percent test split.

### 10.5 Other Approaches at INEX

Gery et. al. [29] use a traditional SVM approach using a bag of words Vector Space Model (VSM). The also introduce two new feature selection criteria Category Coverage (CC) and Category Coverage Entropy (CCE). These measures allow the index to be reduced from approximately 200,000 terms to 10,000 terms while gaining a slight increase in classifier accuracy. De Vries and Geva [19] achieved very similar results using SVMs and a bag of words representation. Kaptein and Kaamps [41] use link information in the Naive Bayes model to try and improve classification accuracy. A baseline using term frequencies is
built. Adding link information on top of the term frequencies only marginally increases classification accuracy. De Vries and Geva [19] proposed why this is so by analysing the cosine similarity of linked documents in the INEX 2008 XML Mining collection. Documents that are linked are likely to have high similarity according to their content. Kaptein and Kaamps [41] analyse where classification errors occurred. They concluded the link information is not available or too noisy in the cases where content information fails to classify documents.

de Campos et al. [17] have used a new method for link based classification using Bayesian networks at INEX 2008. The model considers the class of documents connected by the Wikipedia link graph. This group investigated the utility of link information for classification of documents. A matrix is constructed that is determined by the probability that a document from category i links to category j. The probability along the main diagonal is high, thus suggesting that links are a good indication of category. The authors suggest that the XML structure in the 2006 INEX XML Wikipedia dump is of no use for classification. In their previous work at INEX it actually decreased the accuracy of document classification. The performance of this model is similar to Kaptein and Kaamps [41] who also use the Naive Bayes model to classify documents using links.

Chidlovskii [14] investigated semi-supervised learning approaches for the classification task at INEX 2008. Their approach uses content and structure to build a transductive categoriser for unlabelled document in the link graph. Their method also ensures good scalability on sparse graphs. The authors have
used four different sources of information. These include

- content, the words that occur in a document
- structure, the set of XML tags, attributes and their values
- links, the navigational links between documents in the collection
- metadata, information present in the infobox in a document.

These alternative representations are used to build different transductive categorisers for content and structure. The edges in the graph are weighted by the content and structure information. Label expansion is used to exploit the structure of the link graph while taking the content and structure weights into account. While this is an interesting approach it does not perform as well as SVMs on a bag of words representation. However, it does outperform the Naive Bayes models that use the link graph. The author has suggested that better graph generation and node similarity measures may lead to increased performance.

Interestingly, no one clustered content or structural information to remove noise from the data. This has been shown to work well with text classification [50] and may also work for link graphs.

Given the INEX 2008 XML Mining collection, category information and the test and train split, Naive Bayes classifiers perform poorly. Given that the collection is small and the categories are extracted from a subset of the Wikipedia it could be suggested that these results are artificial. The English Wikipedia now contains three million documents. All this extra information can easily be exploited and it has been argued by [9] that classification algorithm performance converges given enough data. As there is an abundance of information available for IR tasks it would be interesting to perform these same experiments on the full Wikipedia with a much larger test set. It can be suggested that SVMs perform better on the INEX 2008 XML Mining collection because they make an excellent bias, variance trade-off [56] and reach an optimal state quickly [76].

### 10.6 Summary

The classification approach presented in this section has performed well when compared to other researchers at INEX 2008 as listed in Table 10.1. The LF-IDF representation was combined with traditional text representations to boost classification performance. This work was published in a paper by De Vries and Geva [19] at INEX 2008.
Chapter 11

Conclusion

This thesis has applied the K-tree algorithm to document clustering and found it to be a useful tool due to its excellent run-time performance and quality comparable to other algorithms. It also presented two new modifications to the K-tree algorithm in Medoid K-tree and Random Indexing K-tree. The Medoid K-tree improved the run-time performance of the K-tree by exploiting the sparse nature of document vectors and using them as cluster representatives. However, the loss in quality introduced by the Medoid K-tree was considered too large for practical use. Random Indexing K-tree provided a means to improve the quality of clustering results and adapt well with the dynamic nature of the K-tree algorithm. It also reduced memory requirements when compared to previous approaches using dense vectors with TF-IDF culling.

The time complexity of the K-tree algorithm has been analysed. Both worst case and average case complexities were defined. It was confirmed via empirical evidence that the average case reflects actual performance in a document clustering setting.

This research has found K-tree to be an excellent tool for large scale clustering in Information Retrieval. It is expected that these properties will also be applicable to other fields with large, high dimensional data sets.

The LF-IDF representation was introduced, explained and found to be useful for improving classification results when combined with text representations. This exploits structured information available as a document link graph in the INEX 2008 XML Mining collection. Unfortunately, this representation did not improve the results of clustering when combined with a text representation using Random Indexing.

The simple and effective dimensionality reduction technique of TF-IDF culling was able to exploit the power law distribution present in terms. This simple approach based on term frequency statistics performed better in the INEX 2008 XML mining evaluation than other approaches that exploited the XML structure to select terms for clustering.

This work has resulted in papers published at INEX 2008 [19], SIGIR 2009 [20] and ADCS 2009 [18].
11.1 Future Work

Recommended future directions for this work include research problems relating to the K-tree algorithm, document representation, unsupervised feature selection and further evaluation of the approaches presented.

The K-tree algorithm needs to be defined in a formal language such as Z notation. This will provide an unambiguous definition of the algorithms and data structures.

The K-tree algorithm has research problems to be addressed when creating disk based and parallel versions of the algorithm. Issues such as caching and storage mechanisms need to be solved in a disk based setting. Ensuring the algorithm scales and is safe in a concurrent execution environment poses many research problems. It is expected that this will require fundamental changes to the way the algorithm works.

There still exist research problems in exploiting XML structure for document clustering. The document link graph in the INEX 2008 XML Mining collection did not prove useful using the LF-IDF approach. Other structured information in the XML mark-up may improve clustering quality.

The unsupervised feature selection in Chapter 3 are specific to term and link based document representations. Investigation of more general unsupervised feature selection techniques that work with any representation would be useful.

The Medoid K-tree experiments using the INEX 2008 and RCV1 collections conducted in Chapter 7 can be completed with different configurations of K-tree. The cosine similarity measure can be used instead of the Euclidean distance measure in the K-tree. It is expected that this will further increase the quality of the K-tree results. Other measures such as Kullback-Leiber divergence could be investigated for use with the K-tree. The RI K-tree also needs to be compared in this setting to give an overall comparison of the approaches to the CLUTO methods.
Bibliography


