2006-04-17

Clustering of Database Query Results

Kristine Jean Daniels
Brigham Young University - Provo

CLUSTERING OF DATABASE QUERY RESULTS

by

Kristine Jean Daniels

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Kristine Jean Daniels

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Christophe Giraud-Carrier, Chair
Date	Dan Olsen
Date	Phillip Windley

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Kristine Jean Daniels in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Christophe Giraud-Carrier
Chair, Graduate Committee

Accepted for the Department

Parris Egbert
Graduate Coordinator

Accepted for the College

Thomas Sederberg
Dean, College of Physical and Mathematical Sciences

ABSTRACT

CLUSTERING OF DATABASE QUERY RESULTS

Kristine Jean Daniels

Department of Computer Science

Master of Science

Increasingly more users are accessing database systems for interactive and exploratory data retrieval. While performing searches on these systems, users are required to use broad queries to get their desired results. Broad queries often result in too many items forcing the user to spend unnecessary time sifting through these items to find the relevant results. This problem, of finding a desired data item within many items, is referred to as "*information overload*". Most users experience information overload when viewing these database query results.

This thesis shows that users information overload can be reduced by clustering database query results. A hierarchical agglomerative clustering algorithm is used to cluster the query results. The reduction of users information overload is evaluated using Chakrabarti et al information overload cost model. Empirical results show that users are able to find more relevant information as well as experiencing a reduction in information overload.

ACKNOWLEDGMENTS

I would like to thank all those who supported me in this pursuit. I am grateful to Dr. Giraud-Carrier for his mentoring and advice. Without his help this work would not have been possible.

I also would like to thank my mother, husband, and children for their constant support and patience in the completion of this work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today many users are accessing database systems for interactive and exploratory data retrieval [5, 6, 7]. Examples of these systems are real-estate databases (e.g. MSN House and Home [8]), genealogical databases (e.g. Family Search[9]), and citation reference databases (e.g. CiteSeer [10]). While performing searches on these systems, users are required to use broad queries to get their desired results. Broad queries will often result in too many items, forcing the user to spend unnecessary time sifting through a number of items to find the relevant results. The user then has a final step to identify the non-redundant, useful data. This problem of finding a desired data item within many items, is referred to as, "*information overload*".

Searches occasionally return duplicate data in multiple forms, adding to the problem. This is especially visible in citation data, as they vary greatly in their style and representation. This difference in style and representation can be found in the author's names: abbreviations, volumes are sometimes missing, and the order of the attributes can also differ. Some of the differences in citations are visible in Figure 1.1. The same data represented in different forms increases the difficulty a user can have in finding information. When viewing the citations next to each other it is apparent they refer to the same article, but when they are separated in a list the user could think they are viewing new information when they have already looked at this data.

For example, suppose a user is looking for all of the citations from the author Dan R Olsen Jr. from Brigham Young University on CiteSeer (see Figure 1.2). If the users search terms were "Dan R. Olsen" then all of the references of "D. R. Olsen", "Dan Olsen", and "D. Olsen" would be left out. A broader search must be

```
Citation 1:
    Giraud-Carrier, C., & Hilario, M. (Eds.). (1998).
    Ecml'98 workshop notes - upgrading learning to the meta-level: Model
    selection and data transformation. Technical Report CSR-98-02
    Technische Universit at Chemnitz, Fakultat fur Informatik.

Citation 2:
    C. Giraud-Carrier and M. Hilario, editors. ECML'98
    Workshop Notes - Upgrading Learning to the Meta-Level: Model
    Selection and Data Transformation. Technical Report CSR-98-02
    Technische Universitat Chemnitz, Fakultat fur Informatik, 1998.
```

Figure 1.1: Two Representations of the Same Data

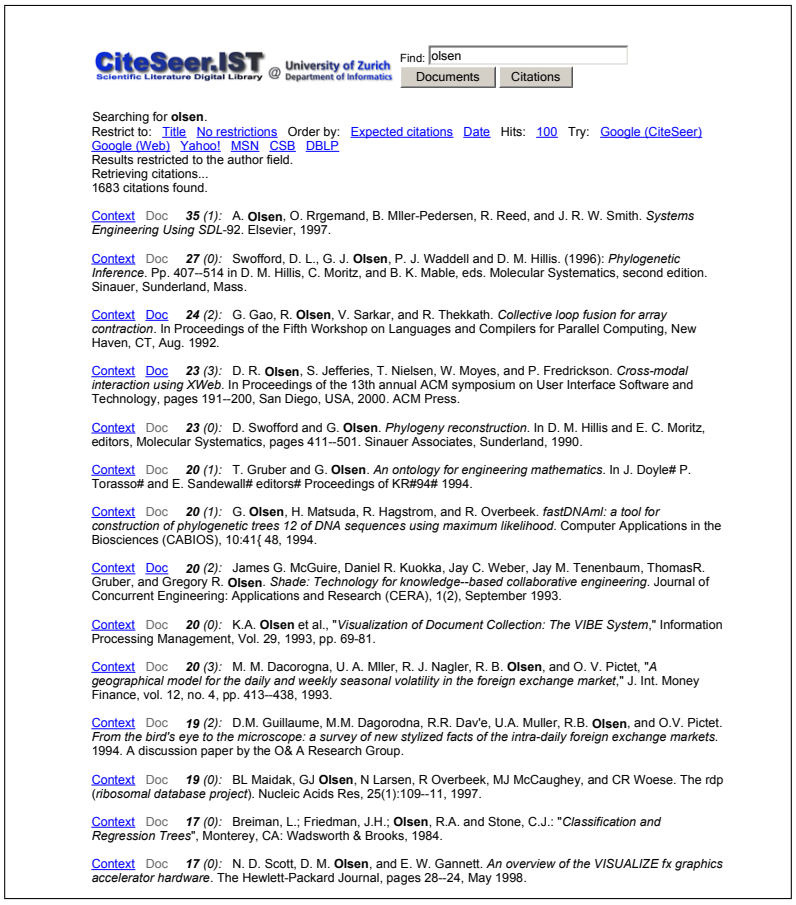
done using "Olsen" to find all citations, resulting in 1,679 items returned, these were not all written by Dan R. Olsen Jr. Further review shows that within the first 100 results there were 9 variations of this author's name, preventing the user from doing a narrower search.

The user can experience information overload for several reasons. One reason is that a user may not be certain what they are looking for. For example if they are looking for a house in Las Vegas but they do not know which neighborhood. A second reason is that a user does not use the advanced search option. This may be because they usually do not know how, it is too complicated, or involves more effort than the user is willing to put forth. A third reason for information overload is a narrower search may eliminate desired results, as shown by the example with Dr. Olsen.

In addition to database query results, Internet text searches have the same problem of information overload. Internet text search, also known as Information Retrieval, has three primary techniques to avoid information overload including the use of ranking, clustering, and categorization.

Ranking returns results in a ranked order, displaying the most relevant results first and the least relevant results last. Ranking can be based on any number of criteria, Google [11], for example, ranks webpages by the number of links from the highest number of pages ranked high by the service, while another search engine may rang according to how much money the company has paid.

Categorization places results in pre-defined categories allowing the user to



CiteSeer.IST Scientific Literature Digital Library @ University of Zurich Department of Informatics

Find: olsen Documents Citations

Searching for **olsen**.
Restrict to: Title No restrictions Order by: Expected citations Date Hits: 100 Try: Google (CiteSeer) Google (Web) Yahoo! MSN CSB DBLP
Results restricted to the author field.
Retrieving citations...
1683 citations found.

Context Doc ***35 (1):*** A. **Olsen**, O. Rrgemand, B. Mller-Pedersen, R. Reed, and J. R. W. Smith. *Systems Engineering Using SDL*-92. Elsevier, 1997.

Context Doc ***27 (0):*** Swofford, D. L., G. J. **Olsen**, P. J. Waddell and D. M. Hillis. (1996): *Phylogenetic Inference*. Pp. 407--514 in D. M. Hillis, C. Moritz, and B. K. Mable, eds. Molecular Systematics, second edition. Sinauer, Sunderland, Mass.

Context Doc ***24 (2):*** G. Gao, R. **Olsen**, V. Sarkar, and R. Thekkath. *Collective loop fusion for array contraction*. In Proceedings of the Fifth Workshop on Languages and Compilers for Parallel Computing, New Haven, CT, Aug. 1992.

Context Doc ***23 (3):*** D. R. **Olsen**, S. Jefferies, T. Nielsen, W. Moyes, and P. Fredrickson. *Cross-modal interaction using XWeb*. In Proceedings of the 13th annual ACM symposium on User Interface Software and Technology, pages 191--200, San Diego, USA, 2000. ACM Press.

Context Doc ***23 (0):*** D. Swofford and G. **Olsen**. *Phylogeny reconstruction*. In D. M. Hillis and E. C. Moritz, editors, Molecular Systematics, pages 411--501. Sinauer Associates, Sunderland, 1990.

Context Doc ***20 (1):*** T. Gruber and G. **Olsen**. *An ontology for engineering mathematics*. In J. Doyle# P. Torasso# and E. Sandewall# editors# Proceedings of KR#94# 1994.

Context Doc ***20 (1):*** G. **Olsen**, H. Matsuda, R. Hagstrom, and R. Overbeek. *fastDNAml: a tool for construction of phylogenetic trees 12 of DNA sequences using maximum likelihood*. Computer Applications in the Biosciences (CABIOS), 10:41{ 48, 1994.

Context Doc ***20 (2):*** James G. McGuire, Daniel R. Kuokka, Jay C. Weber, Jay M. Tenenbaum, ThomasR. Gruber, and Gregory R. **Olsen**. *Shade: Technology for knowledge--based collaborative engineering*. Journal of Concurrent Engineering: Applications and Research (CERA), 1(2), September 1993.

Context Doc ***20 (0):*** K.A. **Olsen** et al., "*Visualization of Document Collection: The VIBE System*," Information Processing Management, Vol. 29, 1993, pp. 69-81.

Context Doc ***20 (3):*** M. M. Dacorogna, U. A. Mller, R. J. Nagler, R. B. **Olsen**, and O. V. Pictet, "*A geographical model for the daily and weekly seasonal volatility in the foreign exchange market*," J. Int. Money Finance, vol. 12, no. 4, pp. 413--438, 1993.

Context Doc ***19 (2):*** D.M. Guillaume, M.M. Dagorodna, R.R. Dav'e, U.A. Muller, R.B. **Olsen**, and O.V. Pictet. *From the bird's eye to the microscope: a survey of new stylized facts of the intra-daily foreign exchange markets.* 1994. A discussion paper by the O& A Research Group.

Context Doc ***19 (0):*** BL Maidak, GJ **Olsen**, N Larsen, R Overbeek, MJ McCaughey, and CR Woese. The rdp (*ribosomal database project*). Nucleic Acids Res, 25(1):109--11, 1997.

Context Doc ***17 (0):*** Breiman, L.; Friedman, J.H.; **Olsen**, R.A. and Stone, C.J.: "*Classification and Regression Trees*", Monterey, CA: Wadsworth & Brooks, 1984.

Context Doc ***17 (0):*** N. D. Scott, D. M. **Olsen**, and E. W. Gannett. *An overview of the VISUALIZE fx graphics accelerator hardware*. The Hewlett-Packard Journal, pages 28--24, May 1998.

Figure 1.2: CiteSeer Database Query Results

select categories that may be relevant, ignoring the ones that are not. A webpage on cross trainer shoes could go into Running, Manufacturing, or Sports Medicine pre-defined categories. All of these are legitimate, depending on the context.

Clustering puts similar items in groups based on some similarity metric. Labeling clusters is based on the contents of the clusters. This allows the user to identify the cluster or grouping that most closely matches what they are looking for. Carrot2 [12] and Clusty [13] are both engines that cluster web search results by placing items in clusters based on the contents of the search. A search on the keyword "jaguar", clusters the found websites based on the content of the websites. Possible clusters could be car, clubs, and big cats.

Research articles tend to concentrate on clustering, categorizing and ranking results of Internet text search, and not on database query results. Only in the last couple of years, has research been done on ranking and categorization of query results. In contrast, to our knowledge no work has been done about clustering the results from database queries.

The information retrieval clustering techniques are based on properties that do not allow them to be directly applied to database query clustering, however, there are principles that can be transferred. Information retrieval clustering concentrates on documents containing large amounts of unstructured text while database clustering only deals with tuples containing small amounts of structured data. The different structure and amount of data, prevent the information retrieval techniques from being used on database query results, despite this fact some of the philosophies for defining, evaluating and creating clusters can be. These include cluster structures, cluster labeling, and cluster distances.

This thesis focuses on the clustering of query results. In the proposed clustering method, we process the returned results from a database query and cluster the results based on a pair-wise distance. The clusters are then labeled and presented to the user. Organizing the tuples in clusters reduces the users information overload allowing them to spend less time sifting and sorting through the results to find the information they need allowing more time to process relevant data.

We present our results as follows. In Chapter 2 we discuss existing approaches and similar research. In Chapter 3 we describe clustering, the goals of clustering, and the hierarchical clustering algorithm. Hierarchical clustering algorithm require a threshold. In Chapter 4 we describe an algorithm to learn the threshold. In Chapter 5 we present our experimental analysis of clustering database query results to reduce information overload. In Chapter 6 we give concluding remarks.

Chapter 2

Related Works

The problem of reducing information overload has been investigated in a number of previous works. Some of these works follow.

Kang and Kim's paper classifies queries for web document retrieval to solve the problem of information overload due to short user queries [14]. The user query is classified into one of three types of tasks: topic relevance task, homepage finding task, or a service finding task. Depending on the task, a different information emphasis is presented to the user. This solution is for web pages searches and not database query results.

In another paper Calado and Cristo's predict document categories based on the Web link structure [15]. Link information alone can improve classification when compared against a content-based classifier. The highest classification results for Web collections happened when link structure was combined with content-based methods.

In 2003, Zhang and Yang's researched classification of documents when there are a small number of features, mislabeled documents and rare positive examples [16]. In a similar paper Zeng and He's cluster web search results with readable names [17]. The algorithm first extracts and ranks salient phrases as candidate cluster names, learned from the training data. It then assigns documents to relevant salient phrases to form clusters. Lastly, the candidate clusters are merged to generate the final clusters. Clustering has been effective for displaying web search results and should be just as effective in database query results.

In a recent paper Hammouda and Kamel present two key parts to successful document clustering. The first part is the incremental construction of a phrase-based

index, not a single-term index. The second part is an incremental document-clustering algorithm that maximizes the tightness of clusters by watching the pair-wise document similarity distribution inside clusters. These two components lead to improved Web document clustering results [18].

Information retrieval (IR) technologies provide rich functionality for text search, feature extraction, and scoring methods while database technologies do not. Conversely, database (DB) technologies handle structured data and query optimizations while information retrieval does not. IR and DB technologies have critical gaps that the other fills. Chaudhuri and Ramakrishnan's paper has proposed research to create a platform that integrates their technologies [19]. This integration contains many challenges and has yet to be implemented as a universal infrastructure.

There are a few systems that have scored database records for ranking. One system is the QUIQ system [20] built for customer support applications. This system scores data and text fields for similarity search. This similarity score was only designed for individual field inspection, and does not consider the entire data contents of a tuple. In addition, its functionality cannot be extended outside its target domain. Another paper, presenting research from Nambiar and Kambhampati [21], provides a ranked set of results by finding similar queries to identify relevant answers to a given user query.

Categorization of database query results differs from Internet text searches in that categorization of Internet text searches occur by placing results in pre-defined categories based on how they were previously tagged. These categorizations are independent of the query and are susceptible to skewed results. Categorization of database query results occur by placing results in categories based on the attribute values [22]. Clustering combines items based on similarity with no pre-defined structure.

There are no known papers on clustering database query results. There is one known research article on categorization of database query results [22]. This paper validates that intelligent automatic categorization of database query results can reduce the problem of information overload to the user significantly, up to three orders of magnitude less by creating a labeled hierarchical category structure based

on the contents of the tuples in the answer set. The sub-categorization is created through a literal string search of the values in the resulting tuples. Labeling and literal string search does not allow for variations in spelling or data representation for the categorical attributes.

Frequently data contains typographical errors, or multiple representations, like abbreviations, so an exact comparison is not sufficient in determining if an entity belongs to a category. A cost model was created to measure the information overload users will experience. This cost model will be used in the evaluation of information overload for this thesis. This categorization did not handle large datasets well; the hierarchical structure was unable to be viewed by the users.

The distinguishing aspects of this thesis from the papers above is that (a) it addresses the challenges that string based attributes pose to clustering database query results (b) it presents a unique implementation and approach to clustering database query results by using similarity measures with a termination based hierarchical agglomerative clustering, and (c) it does not use pre-defined categories in placing items in cluster or in labeling the cluster for database query results. This thesis shows that a user's information overload can be reduced from clustering database query results.

Chapter 3

Clustering

This chapter explains clustering and the clustering algorithm used to cluster the database query results. The chapter begins with giving a brief definition of clustering. It then explains possible different cluster shapes, the need for clustering, the requirements of clustering, and the different types of clustering structures. Then, the chapter gives a metric to calculate the quality of a cluster. Finally, it explains hierarchical agglomerative clustering and the similarity metric and cluster labeling method used to cluster the data.

Clustering is the process of partitioning a set of objects into groups called clusters. A cluster is a group of the same or similar elements gathered or occurring closely together [23]. Clusters help users understand the structure or natural groupings in a data set.

Data clustering groups data into unique sets of objects. Clustering this data creates many different types of clusters, some clusters will be well separated while other clusters will be of arbitrary shape. Well-separated clusters contain distinct sets of points that are easily partitioned. Arbitrary shaped clusters contain points that are not easily partitioned: S-shaped data or random data (see Figure 3.1).

Clustering has been used in various ways in many disciplines. Marketing has used clustering to discover distinct groups in their customer bases. The clustering is then used to target individuals in specific marketing programs. Clusters of land-use are used to identify areas of similar land-use from an earth observation database. Insurance policy underwriters use clustering to identify groups of higher than average claim cost. City planning has used clustering to identify houses according to their

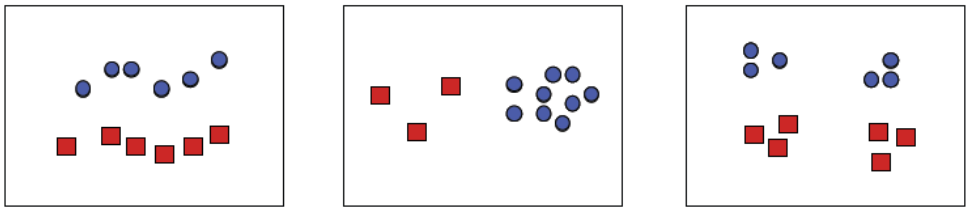


Figure 3.1: Different Shapes of Clusters, Figure Adapted From [1]

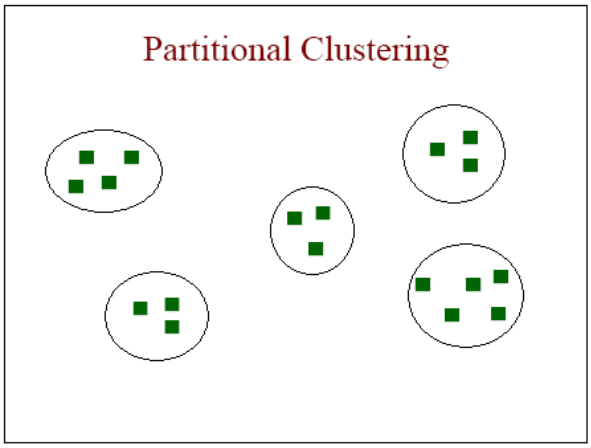


Figure 3.2: Partitional Clustering, Figure Adapted From [2]

value, geographic location, and house type. Geologists have used clustering to study the correlation between earthquake epicenters and continent faults, etc.

The multi-faceted use of clustering has lead to many clustering algorithm requirements. Clustering algorithms should be able to scale, deal with different types of attributes, and discover clusters with arbitrary shape. They should also be able to deal with noise and outliers, be insensitive to order of input records and have minimal requirement for domain knowledge with input parameters.

There are two types of clustering structures: partitional and hierarchical. A few partitional clustering methods described briefly include: the density-based

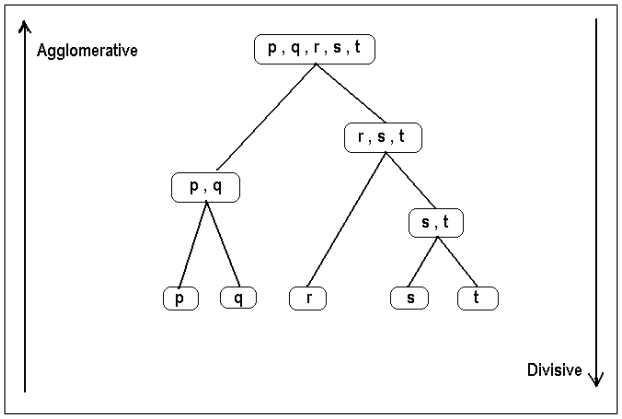


Figure 3.3: Divisive and Agglomeartive, Hierarchical Clustering, Figure Adapted From [3]

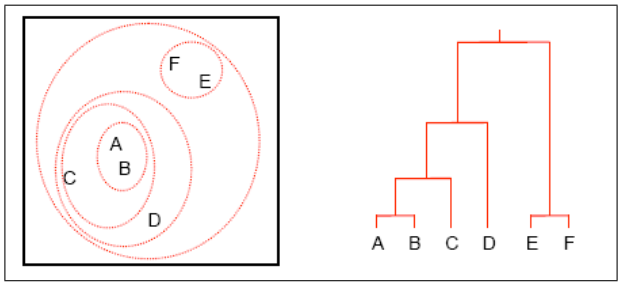


Figure 3.4: Hierarchical Clustering, Figure Adapted From [4]

method, the grid-based method, and the model-based methods. Partitional clustering clusters the data into non-overlapping subsets such that each data item is in exactly one cluster (see Figure 3.2). The number of clusters and partitioning criterion is decided beforehand. Examples of partitional clustering algorithms are k-means and k-medoids. Although the k-means algorithms is relatively efficient and terminates at a local optimum, it also has weaknesses. Some of the weaknesses include: the number of clusters must be specified in advance, the algorithms are unable to handle noisy data or data with outliers, not suitable to discover clusters with non-convex shapes, doesn't handle categorical data and is only applicable when a mean is defined, and is sensitive to starting positions.

Hierarchical clusters are clusters that are concentrically combined where each data item is in many clusters (see Figure 3.4). Hierarchical clustering can either be divisive or agglomerative. Divisive clustering starts with every data item in one cluster. The cluster is then divided until each data item is in a cluster by itself thereby creating a nested clustering. Agglomerative clustering begins with each data item in its own cluster, and then combines, based on similarity, until all items are in one cluster (see Figure 3.3). A distance method is used as clustering criteria. The number of clusters does not need to be specified, but a termination condition needs to be specified; the algorithm terminates when the required termination condition is reached. Nodes are merged that have the least dissimilarity which is computed using the dissimilarity matrix and a cluster distance. Eventually all nodes belong to the same cluster. (see Figure 3.5). The height of the bar indicates the degree of dissimilarity with in the cluster. At any point in the hierarchical structure a proximity bound can be imposed creating a partitional clustering of the data. Essentially taking a slice of the tree where data is not combined, if the pair-wise similarity exceeds the proximity bound. This is known as Partitional Clustering from a Hierarchical Clustering (see Figure 3.6). A partitional clustering can be generated from a hierarchical clustering by "cutting" the tree at some level. Disadvantages of hierarchical clustering is that it has a quadratic time complexity so it does not scale well, and what was done previously can never be undone.

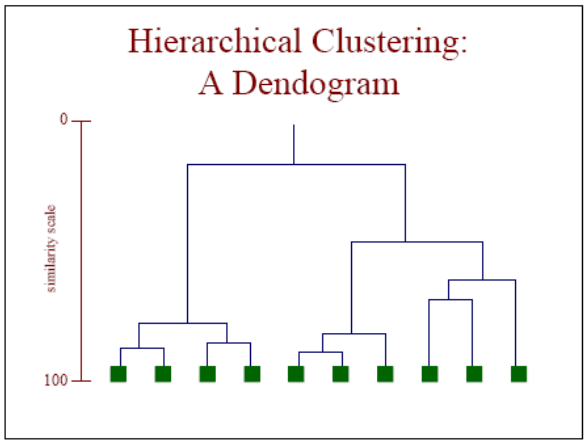


Figure 3.5: Hierarchical Clustering, Adapted Figure From [2]

Density-based clusters contain dense regions of points separated from other regions of high density by low-density regions. Density-based clusters are used when clusters are irregular or intertwined. Density based algorithms create clusters from connectivity and density functions and are good with noisy data, but to their disadvantage, they are represented graphically, can be slow, and do not cluster border or outlier points.

Grid-based algorithms have a multiple-level granularity structure. The algorithm can parallelize easily, but only creates cluster boundaries that are either horizontal, vertical, and has no ability to detect diagonal boundaries. It can also automatically find subspaces of the highest dimensionality but the simplicity of the methods degrade the accuracy of the clustering.

Model-based algorithms hypothesize which model is best for each cluster. This algorithm assumes that the attributes are independent of each other when in fact a correlation may exist.

Clustering is typically unsupervised. In unsupervised clustering the labels or the number of labels is not known and discovered at run time. Recently clustering research has introduced semi-supervised clustering where a small amount of labeled data is used to direct the clustering [24].

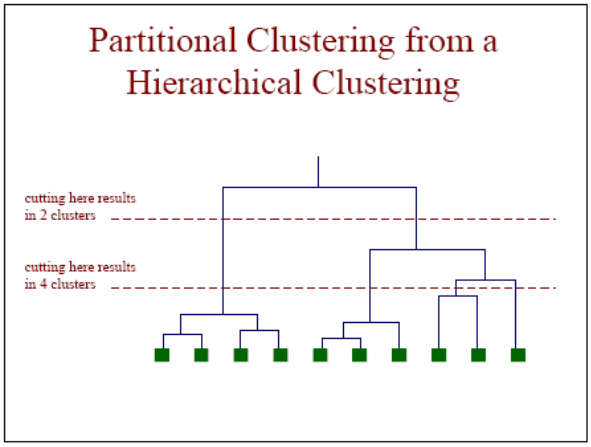


Figure 3.6: Partitional Clustering from a Hierarchical Clustering

3.1 Cluster Quality

The measure of a good cluster is based on intra-class and inter-class similarity. The intra-class similarity should be high, while the inter-class similarity should be low. In other words, data items contained in a cluster should be similar to each other, but not similar to items contained in other clusters. Cluster analysis is highly subjective since it is hard to define "similar enough" or "good enough" in clusters. The quality of a clustering depends on the similarity measure used by the method, its implementation, and its ability to discover some or all of the hidden cluster patters.

In order to measure the quality of clusterings a measure of clustering quality is required. The F-measure is frequently used to measure clustering quality. The F-measure follows. Let $C_1, \ldots, C_n$ be the correct target clusters (as per the provided labels) and $HC_1, \ldots, HC_m$ be the computed clusters (as per the complete-link agglomerative hierarchical clustering algorithm). Then:

$$P(i,j) = \frac{|C_i \cap HC_j|}{|HC_j|}$$

$$R(i,j) = \frac{|C_i \cap HC_j|}{|C_i|}$$

$$F(i,j) = \frac{2.P(i,j).R(i,j)}{P(i,j) + R(i,j)}$$

$$F(i) = \max_{j=1}^{m} F(i,j)$$

$$F = \sum_{i=1}^{n} \frac{|C_i|}{N} F(i)$$

In summary, the F-measure, is how accurately the clusters are formed.

3.2 Hierarchical Agglomerative Clustering

Hierarchical agglomerative clustering starts with each data item in its own cluster and then organizes the data as a set of nested clusters within hierarchical tree, also known as a dendrogram. Clusters that have the smallest distance are then merged iteratively in a pair-wise manner until all data items belong to the same cluster. Hierarchical agglomerative clustering is used to combine data items into clusters to obtain a reduction in information overload. This clustering algorithm was chosen to reduce user's information overload. It was chosen since it does not require the number of clusters to be previously specified. A high-level view of the clustering algorithm is in Figure 3.7.

1. Assign each data item to its own cluster
2. Compute pairwise distances between clusters
3. Merge the two closest clusters, i.e., that have the smallest distance
4. If more than one cluster is left, go to step 2

Figure 3.7: Generic Hierarchical Agglomerative Clustering Algorithm

There are three different ways to compute the distance between clusters (Step 2): complete-link, single-link, and average-link. Complete-link measures the cluster distance as the maximum pairwise distance between the items in two different clusters. Single-link computes the minimum pairwise distance between the items in two different clusters while average-link computes the average pairwise distance. The clustering approach used is independent of the cluster distance measure. We choose

to illustrate it on the complete-link approach. Complete-link has been shown to work well on a variety of clustering tasks such as data sets containing non-isotropic clusters, including well-separated, chain-like, and concentric clusters [25].

3.3 Partitional Clustering from a Hierarchical Agglomerative Clustering

Most partitional clustering algorithms require the number of desired clusters to be set a priori. Not only is this somewhat counter-intuitive, it is also difficult except in the simplest of situations. By contrast, hierarchical clustering may create partitions with varying numbers of clusters. The actual final partition depends on a threshold placed on the similarity measure used. Given a cluster quality metric, one needs to calculate an appropriate threshold. Chapter 4 demonstrates how the threshold can be learned.

In hierarchical agglomerative clustering, the number of clusters is not specified a priori. It is clear, however, that in practice, the hierarchical agglomerative clustering algorithm should be stopped before all data items are in the same cluster. In other words, step 4 in Figure 3.7 should be replaced with some condition on the cluster distances, i.e., a threshold value, T, at which the algorithm stops merging clusters.[1] The value of T, in turn, becomes the critical parameter of the algorithm since it directly affects the contents and density of the clusters. A threshold value too high will result in a large number of sparse clusters. A threshold value too low will result in very few dense clusters.

Selecting the most appropriate threshold value for a specific clustering task is non trivial. Guessing is generally not reasonable, while proceeding by trial-and-error (i.e., picking different threshold values and looking at the clustering results) is computationally prohibitive and prone to local optima. It is possible to learn a near-optimal threshold value efficiently through the use of a small subset of labeled data.

Partitional Clustering from a Hierarchical Agglomerative Clustering Algorithm

[1] Of course, one could also let the algorithm complete and apply the threshold to the full dendogram to find the corresponding partition. This is clearly less efficient however.

is the clustering algorithm used to cluster the query results. This algorithm is similar to the algorithm in Figure 3.7. It differs in that it contains a termination condition dependent on the threshold value. This algorithm is in Figure 3.8.

1. Assign each data item to its own cluster
2. Compute pairwise distances between clusters
3. Merge the two closest clusters if their distance is less than the threshold
4. If more clusters to merge, go to step 2

Figure 3.8: Generic Partitional Clustering from a Hierarchical Agglomerative Clustering Algorithm

3.4 Similarity Metric

Approximate string matching is fundamental to text processing, because we live in an error-prone world. Any spelling correction program must be able to identify the closest match for any text string not found in a dictionary. The distance between tuple variables is typically calculated on the backend of the database in order to organize the data in clusters for quicker query retrieval rates. It is also used to determine if two tuples are equal. This thesis is clustering the results from query results. The results from these queries are represented as strings. Due to the inconsistency in string data, the similarity measure that is going to be used is a distance measure.

A Distance Measure is a function that associates a non-negative numeric value with (a pair of) sequences, with the idea that a short distance means greater similarity. Distance measures usually satisfy the mathematical axioms of a metric. A distance measure is a type of scoring function.

The distance between objects is based on the similarity between the two objects. There are many different similarity metrics available to calculate the similarity between two data items. The Euclidean distance is typically used. This metric was

not chosen since it is not intended to find the distance between strings. A string metric was desired since the majority of query results contain data with strings. Some string metrics are Jaro-Winkler, Q-Grams, Levenshtein, and Jaccard Similarity [26].

The q-grams similarity was chosen as the similarity metric of choice because it is frequently used as a 'fuzzy' search in databases and allow queries that do not exactly match. It also performed well in preliminary experiments. Although the q-gram was chosen for this implementation, any composite similarity metric could be used.

A q-gram is a sequence of q characters of the original string. The q-grams of length q = 3 for string "olsen" are {##o, #ol, ols, lse, sen, en$,n$$}. Where '#' and '$' indicate the beginning and end of the string. A string with length L will have L+ q-1 q-grams. Similar strings have many q-grams in common. SimMetrics is an open source java library of Similarity and Distance Metrics [26]. The SimMetrics implementation of the q-gram metric will be used.

The two strings 'tuple' and 'tupel' are similar but not exact. The q-grams for 'tuple' when q = 3 is {##t, #tu, tup, upl, ple, le$,e$$}. The q-grams for 'tupel' are {##t, #tu, tup, upe, pel, el$,l$$}. These strings have 3 q-grams in common.

The data items in these clusters are tuples of results from querying the database. The tuple is represented as a data object that can be compared against another data object and generate a similarity score. Tuples returned from database query results contain text-based data tags and non-text data tags. Non-text data like references to external documents or the number of bedrooms in a house for sale; will be treated as textual data. The tuple is evaluated as a whole to reduce the requirement for input parameters, pre-processing.

3.5 Cluster Labeling

Clusters need to be labeled, to let the user know the contents of the clusters, without inspecting all of the data items within the cluster. There has been research in the Information Retrieval (IR) community that has concentrated on labeling clusters [17]. The clusters given in this thesis are simply labeled with the author names that

were contained with in the clustering. The cluster label was created after the data was clustered based on the contents of the cluster. Although this method was used any other method could be used. Cluster labeling is out of the scope of this thesis.

Chapter 4

Learning the Threshold in Hierarchical Agglomerative Clustering

This chapter presents an efficient technique, based on the notion of semi-supervised learning, to set the threshold T in the context of complete-link hierarchical agglomerative clustering. Semi-supervised clustering is essentially clustering (i.e., unsupervised learning) with some level of supervision imposed, in the form of a small amount of labeled data. Although the proposed solution is applicable to any form of hierarchical clustering, we choose to illustrate it here on the complete-link version because it "produces tightly bound or compact clusters...[and] from a pragmatic viewpoint, it has been observed that the complete-link algorithm produces more useful hierarchies in many applications than the single-link algorithm." [27].

A recent survey of unsupervised and semi-supervised clustering is in [28]. Several researchers have made attempts at semi-supervised or constrained clustering in the specific context of hierarchical agglomerative clustering.

- Kim and Lee present a 3-step semi-supervised algorithm for clustering documents [29]. The first step generates fine-grained clusters using a complete-link agglomerative hierarchical threshold-based clustering to create clusters. The threshold value is set to a small non-zero value. The second step is the supervised step, where the algorithm receives relevance-feedback from the user: positive if the documents go together, negative if they do not go together. Finally, the third step assigns an initial cluster to the nearest documents. As in

our approach, clustering performance is evaluated using the F-measure. However, although the algorithm is semi-supervised, the threshold parameter is not learned, but set independent of the data.

- Wagstaff and Cardie propose the introduction of instance-level must-link and cannot-link constraints to the clustering process [30]. Results with their algorithm show that clustering accuracy increases while runtime decreases. Interestingly, their proposed constraints would be implied naturally if the threshold were learned from labeled data class structure as is the case in our approach.

- Zhao and Karypis introduce a new class of clustering algorithms, known as constrained agglomerative algorithms, that combine the features of both partitional and agglomerative algorithms [31]. Their experiments show that the new algorithms lead to better hierarchical solutions than either agglomerative or partitional algorithms alone. Absent from their algorithms, a semi-supervised threshold would provide a high-level view of the clustering structure and prevent agglomerative errors introduced during the initial merging decision, where a large number of equally good merging alternatives for each cluster are available.

- Davidson and Ravi indicate that small amounts of labeled data can improve the cluster purity in hierarchical clustering [24]. Their so-called δ constraint requires the distance between any pair of points in two different clusters to be at least δ, a kind of inter-cluster threshold reciprocal to T. Despite recognizing the importance of setting the value of δ adequately, no attempt is made at finding a "good" value for it. Experiments are based on δ arbitrarily set to 5 times the average distance between a pair of points. The constraint can nevertheless reduce computation time by a factor of 2 to 4.

4.1 Threshold Learning Algorithm

Here we use a novel approach to semi-supervised hierarchical agglomerative clustering by learning the threshold and allowing the clustering to be directed by a

small set of labeled data. One natural side effect of our approach is that, as one would hope, different threshold values — and hence, partitions — are obtained for different sets of data and different representations of the same data. As an illustration, consider the simple example of Figure 4.1.

```
Citation 1:
    <author>Domingos, P.</author><date>(1996b).</date>
    <title>Using partitioning to speed up specific-to-
    general rule induction.</title><booktitle>In
    Proceedings of the AAAI-96 Workshop on Integrating
    Multiple Learned Models,</booktitle><pages>pp.
    29-34.</pages>

Citation 2:
    Domingos, P. Using partitioning to speed up
    specific-to-general rule induction. In Proceedings
    of the AAAI-96 Workshop on Integrating Multiple
    Learned Models, pp. 29-34. (1996b).
```

Figure 4.1: Two Representations of the Same Data

Citation 1 and Citation 2 each contain a citation to the same paper. These citations are representative of the datasets to which they belong. Citation 1 contains tagged data while Citation 2's data is not tagged. As a result, the threshold for these datasets will likely be different. In particular, one would expect the threshold for Citations 1's dataset to be higher since all citations in the dataset share the same tags.

In semi-supervised clustering, labeled data is used along with the unlabeled data to constrain clustering. As pointed out in [32], "a set of classified points implies an equivalent set of pairwise constraints." Such constraints are essentially must-link and cannot-link constraints, both expressed through labels. Data items that have the same label imply must-link constraints, while data items with different labels imply cannot-link constraints.

Labeled data takes a data item and labels which cluster it should be in. Must-link constraints require that two data items must be in the same cluster while cannot-link constraints require that two data items should not be in the same cluster.

In most approaches to semi-supervised learning, constraints are used directly to affect the clustering process. In our approach, constraints play a more indirect role. First, the labeled data is used to discover a good threshold value (with respect to the constraints). Then that threshold value is used when clustering the complete dataset. Hence, the learned threshold implicitly captures the constraints of the labeled data and indirectly enforces them on the rest of the data.

In order to compare different threshold values (or corresponding clusterings), we require some measure of clustering quality. Given the form of our constraints, we choose the F-measure.

In overview, the threshold learning algorithm works as follows. Given a collection C of data items, we draw a relatively small random sample S from C and label its elements. We then proceed to cluster S repeatedly with varying threshold values. The threshold value that results in the greatest F-measure is retained and used to cluster C. The basic assumption of our semi-supervised approach is that fast local optimization over S will direct the threshold to a value that creates a global partition of C with good quality clusters. A high-level view of the threshold learning algorithm is in Figure 4.2.

1. Select a random subset, S, of C
2. Label all the items in S
3. d_{min} = smallest pairwise distance in S
4. d_{max} = largest pairwise distance in S
5. For $t = d_{min} + s$ to $d_{max} - s$ in increment of s
   a. Cluster S with threshold t
   b. Compute resulting F-measure
6. Return the threshold value(s) with the highest F-measure

Figure 4.2: Threshold Learning Algorithm

The algorithm examines values at regular intervals (step size s) between the

smallest pairwise distance, d_{min}, and the largest pairwise distance, d_{max}, among data items in S. We purposely leave out the extreme values in step 5 of the algorithm as they correspond to what can be regarded as degenerate clusterings (i.e., either one where each data item is in its own cluster or one where all data items are lumped together in a single cluster). The value of s is discussed later.

There is an implicit assumption here that C is relatively large. It is evident that for small C's one can efficiently use the threshold learning algorithm on C rather than S (provided all of C is labeled). Indeed, one may have to do so as the size of S in such cases would be insufficient to provide statistical confidence in any inference from S to C. On the other hand, there is a clear trade-off between the need for statistical significance and the cost of labeling data items. In practice, the less data to label, the better.

In order to find an empirical value for the size of S, we conducted experiments on a few UCI datasets [33] of varying sizes (here, 214 for Glass, 958 for Tic-Tac-Toe and 1728 for Car). Figure 4.3 shows the evolution of the value of the threshold learned by the above algorithm as the size of S increases. In each graph, the 100% lines correspond to the range (i.e., lower bound and upper bound) of threshold values that produce the highest F-measure on the complete datasets.

As mentioned above, our objective is to come up with a good local estimate of the global threshold value, i.e., the value when $S = C$. As can be observed from the graphs, in most cases, the curves tend to start with smaller values, but rapidly "settle" within or close to the optimal bounds. Based on these results, we choose $|S| = 50$ in our experiments.

Note that due to inevitable variance, the learning algorithm may have to be executed more than once and resulting learned threshold values averaged. In our experiments, we resample and learn 10 times. There is obviously additional computational cost involved in this repetition and it is only justifiable if the overall cost remains substantially lower than the cost of learning with the complete dataset. Recall that the computational complexity of the complete-link hierarchical agglomerative clustering algorithm is $\mathrm{O}(n^2 \log n)$, which can easily be seen to be also the complexity

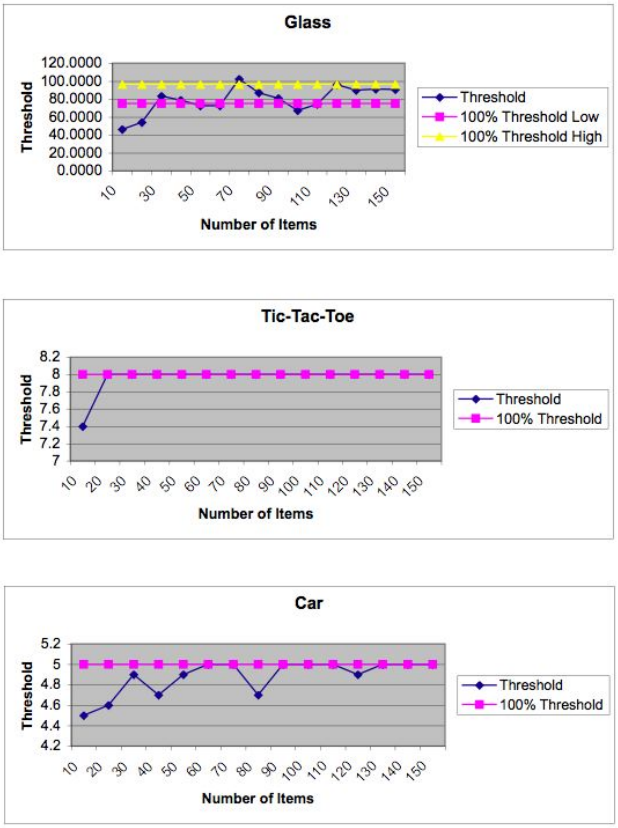


Figure 4.3: Learned Threshold Value vs. Size of Sample

of our learning algorithm. The relationship between the time, t_S, it takes to execute the learning algorithm on sample S and the time, t_C, it takes to execute it on the complete dataset C is therefore given by:

$$t_S = \frac{|S|^2 \log |S|}{|C|^2 \log |C|} t_C$$

Now, as we have just empirically shown, $|S| = 50$ gives good threshold values. For most datasets of interests, 50 is a small fraction of the total number of data items, generally less than 20%. In other words, $|S| < 0.2|C|$ and it follows that:

$$t_S < 0.04 \frac{\log(0.2|C|)}{\log |C|} t_C < 0.04\, t_C$$

Hence, even as we run the threshold learning algorithm 10 times on S, the total execution time is still a fraction of the time required to run it on C. Since the size of S is fixed, the computational gain is larger as C gets larger.

4.2 Learning Threshold Results

In order to evaluate our semi-supervised threshold learning approach, we conduct experiments on several datasets from the UCI repository [33]. Note that since these are actually classification tasks, all data items are already labeled with a class value. For simplicity, we treat these labels as cluster labels. In more realistic settings, one would rely on a human (or other) expert to label the sample. Clearly, labels are ignored during clustering and only used to assess cluster quality with the F-measure. Details of the selected datasets are in Table 4.1.[1]

Dataset	**Inp. Type**	**Size**	**# Classes**
Iris	Continuous	150	3
Glass	Continuous	214	6
Ionosphere	Continuous	351	2
Soybean	Nominal	683	19
Breast-W	Continuous	699	2
Vehicle	Continuous	846	4
Tic-Tac-Toe	Nominal	958	2
Car	Nominal	1728	4
Segment	Continuous	2310	7
Kr-vs-Kp	Nominal	3196	2

Table 4.1: Experimental Datasets

To avoid the issue of heterogeneous distance measures, all datasets have inputs of a single type. This allows the standard Euclidean distance to be computed for continuous inputs, and a Hamming-like distance for nominal inputs. The step size for the threshold learning algorithm is set to $s = \frac{d_{max} - d_{min}}{40}$ for continuous inputs and $s = 1$ for nominal inputs. The LingPipe's version 2.1.1 implementation of the complete-link hierarchical agglomerative clustering algorithm was used[34].

[1]For Segment, the location and neighborhood size attributes have been removed as they seem irrelevant to the task and tend to dominate the distance values thus creating artificial sparseness.

Table 4.2 shows the target threshold's value or range, along with the learned threshold. The target threshold is obtained by running the threshold learning algorithm on the entire dataset. Observed ranges are the result of plateaux in the F-measure vs. threshold value curve. The learned threshold, on the other hand, is an average over 10 runs, and if the maximum F-measure forms a plateau, the threshold value is taken to be the low end of that plateau (i.e., the smallest threshold value corresponding to the maximum F-measure). The numbers in parenthesis are standard deviations.

Dataset	**Target** T	**Learned** T
Iris	3.37 - 3.90	3.49 (1.32)
Glass	75.22 - 96.43	75.53 (38.14)
Ionosphere	9.50	6.77 (1.51)
Soybean	11	9 (1.7)
Breast-W	22.10 - 25.35	21.24 (2.65)
Vehicle	221.27 - 287.13	219.54 (99.22)
Tic-Tac-Toe	8	8 (0)
Car	5	5 (0.3)
Segment	150.98	45.23 (21.76)
Kr-vs-Kp	23	18 (1.02)

Table 4.2: Learned vs. Target Threshold

With the exception of Ionosphere and Segment, learned thresholds generally fall within or close to the range of the target thresholds. Table 4.3 shows the respective F-measures and numbers of clusters. In each cell, the first number corresponds to the target threshold value and the second to the learned threshold value.

In most cases, the difference in F-measure from clustering with the learned threshold vs. clustering with the target threshold is relatively small. However, for some datasets, such small differences may hide substantial differences in the number of clusters. Discrepancies may be explained in part by the underlying sparseness of the data (e.g., Segment), by Hamming cliffs (e.g., Kr-vs-kp) or by a large number of classes, leading to under-representation of constraints in the labeled sample (e.g., Soybean). Despite these limitations, the above experiments suggest that the thresholded

Dataset	F-measure	# Clusters
Iris	0.83 / 0.83	3 / 3
Glass	0.76 / 0.76	4 / 4
Ionosphere	0.67 / 0.63	2 / 15
Soybean	0.72 / 0.67	38 / 61
Breast-W	0.92 / 0.93	2 / 2.2
Vehicle	0.48 / 0.48	6 / 6
Tic-Tac-Toe	0.19 / 0.20	19 / 19.5
Car	0.17 / 0.22	32 / 28.3
Segment	0.54 / 0.49	19 / 56
Kr-vs-Kp	0.51 / 0. 33	2 / 10

Table 4.3: F-measure and Number of Clusters

hierarchical agglomerative clustering algorithm performs at a level close to optimal and at a fraction of the cost.

4.3 Summary

Unlike partitional clustering, hierarchical agglomerative clustering does not require the number of clusters to be set a priori. However, hierarchical agglomerative clustering does not produce a single clustering but a series of nested clusterings, which does leave open the problem of choosing the preferred one among them. This choice is typically implemented via a threshold value on the distance function, which acts as the iterative merging process' stopping criterion. Selecting an appropriate threshold value is generally beyond the reach of human capability.

As presented here an automatic method to discover the most appropriate value of the threshold in complete-link hierarchical agglomerative clustering is effective. The two-stage, semi-supervised clustering method uses a small subset of labeled data items to find a threshold value that maximizes the F-measure. It then subsequently clusters the complete dataset up to that threshold value, thus automatically discovering the number of clusters. Empirical results indicate that the quality of clustering obtained using the learned threshold value are near-optimal.

Chapter 5

Experimental Evaluation

The experiments performed for this thesis were based on a hierarchical agglomerative clustering. This clustering combined with generating a learned threshold and using q-grams to compute the pair-wise distance between tuples, showed a reduction in information overload.

5.1 Quantitative Analysis

This section describes in detail the methodology and results of our approach in solving the problem of information overload. The amount of user information overload is evaluated using an information overload cost model. The experiments compared our clustering method with an existing categorization algorithm and against no processing at all. These experiments demonstrate that clustering with a semi-supervised hierarchical agglomerative clustering is able to decrease users information overload.

5.1.1 Information Overload Cost Model

Chakrabarti et al developed an analytical cost model to estimate information overload faced by users. It is this information overload cost model that we use to calculate the user's information overload. This cost model allows the measurement of information overload a user will experience given a particular arrangement of data. This allows our algorithm results to be compared with existing query processing algorithms. The cost model is based on the assumption that the time spent finding the relevant tuples is proportional to the number of items the user examines. The

cost model demonstrates that there is a strong correlation between calculated average user exploration cost and actual user exploration cost.

The results from a one-level categorization or a clustering of query results can be viewed as a tree. With each cluster or category as children of the root of the tree. The cost model calculates the cost of a tree, T. The tree contains three levels: level 0 is the root, level 1 is the category labels and level 2 contains the tuples. An arbitrary node in T is referred to as C. The calculation of the cost is recursive. When calculating the cost of a node, C, all children nodes also must be calculated. So calculating the cost of C when C is the root of the tree, calculates the cost of the entire tree. The cost model is

$$Cost_{ALL}(C) = P_w(C) * |tset(C)| + (1 - P_w(C)) * (K * n + \sum_{i=1}^{n} P(C_i) * Cost_{ALL}(C_i))$$

The $Cost_{ALL}(C) = |tset(C)|$ and $P_w(C) = 1$ when C is a leaf node. The cost model contains five variables: $P_w(C)$, *tset(C)*, K, n, and P(C). $P_w(C)$ and P(C)are probabilities calculated from the aggregate knowledge of previous user behavior. The previous user behavior is from the the log of SQL query strings. If past users have issued queries with contents equal to the category name then the probability the current user will explore the category increases.

'SHOWTUPLES' or 'SHOWCAT' is an option provided to the user. 'SHOWTUPLES' shows the user all of the tuples in a category. 'SHOWCAT' shows the user the categories within a category. $P_w(C)$ is the probability that the user decides the option 'SHOWTUPLES' after inspecting the category label. The $P_w(C)$ is very low since the user is very likely to view the categories labels for the root. P(C) is the probability that the user viewing category C explores the category with either 'SHOWTUPLES' or 'SHOWCAT'. This probability is based on past user queries.

tset(C) is the number of tuples in the category C. A category typically had an average of 1 to 3 tuples. K is the cost of examining a category label relative to the cost of examining a data tuple. This value was set to 0.25 since the size of the tuples were about four times the size of a label. The time it takes to examine a category label is less than the time it takes to examine a tuple. n is the number of category

labels, or children that the current category C contains. This value is equal to one except for the root.

In order to calculate the information overload the cost model requires information from a log of query strings to calculate the probabilities within the cost model. The cost model is able to calculate the cost for any tree structure. Thereby, it can be used for evaluating clustering and categorization tree, since both of these can be viewed as tree structures. The cost of not processing the results is equal to the number of tuples returned from a query, assuming a cost of one for each tuple examined, so the cost of a query returning 300 tuples is 300.

Chakrabarti et al information overload cost model was developed for the purpose of evaluating and creating their categorization of query results algorithm. Chakrabarti et al categorization algorithm creates their categorization based on the cost to the user. Database attributes are presented to the user in the categorization based on their cost.

Chakrabarti's categorization performs well, however, it does require a log of SQL queries. A log of SQL query strings is tied to access of the database which is typically not shared. Because of this, the MSN House and Home database that was used to evaluate their cost and categorization model, could not be obtained for a direct comparison. An alternative was found using the BibFinder database [35]. A log of queries was ascertained and used as the base for all of the experiments. The query log consists of 181,677 query strings representing real searches conducted by users on the BibFinder web site. The attributes used in the queries consisted of author, title, year published, and conference/journal published.

5.1.2 Experiment Overview

Following the same process outlined by Chakrabarti, each query from the log of queries was treated as a separate user exploration. A subset Q, of 100 queries is then treated as a set of user explorations over which the cost of each query is calculated and averaged across Q. This set Q and the average cost of the results is the basis for comparison between algorithms. Figure 5.1 outlines the process used to

calculate the cost of each methodology. For cross validation purposes the procedure was repeated for 8 mutually disjoint sets of Q.

```
1.  Execute 100 queries.
2.  For each query
3.     Create Clustering of Query Results
4.     Create Categorization of Query Results
5.     Calculate the cost for clustering
6.     Calculate the cost for categorization
7.     Calculate the cost for no processing
```

Figure 5.1: User Explorations

A dataset was labeled from the BibFinder database. The threshold learning algorithm was then executed. A threshold value of 0.686 was learned using the technique outlined in Chapter 4. This value was then used to cluster the data.

The users information overload is reduced on data sets from databases containing many strings that are prone to having multiple representations of the same data, prone to typographical errors, and contain many attributes values. Since clustering data reduces the volume of information presented to the user at any one time it reduces information overload. The average number of items per set is in Figure 5.2. It reveals the weakness of the Categorization algorithm in handling strings. The number of partitions is very close to the number of data items. Where the number of partitions for clustering is almost double that of the categorization algorithm.

The following experiments were evaluated using the cost all model. Figure 5.3 is the estimated cost of the users to exploration to find every tuple that is relevant. The chart shows that with each set of 100 explorations the user had less information overload than with the categorization and substantially less than with no processing of the data.

The formula relies strongly on partition labels being created from past queries that the user issues. User queries usually consist of a one word whereas the cluster

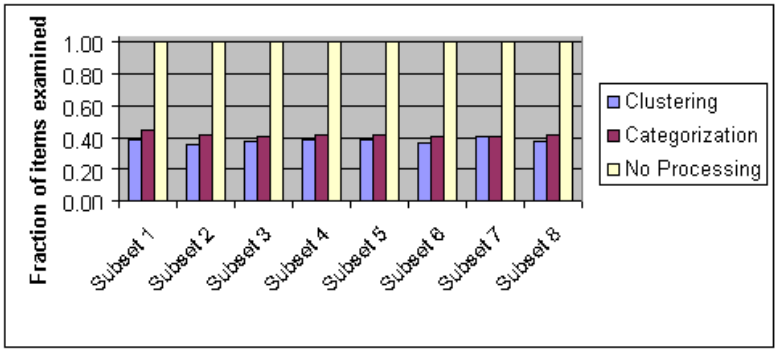


Figure 5.2: Percentage of Items Examined

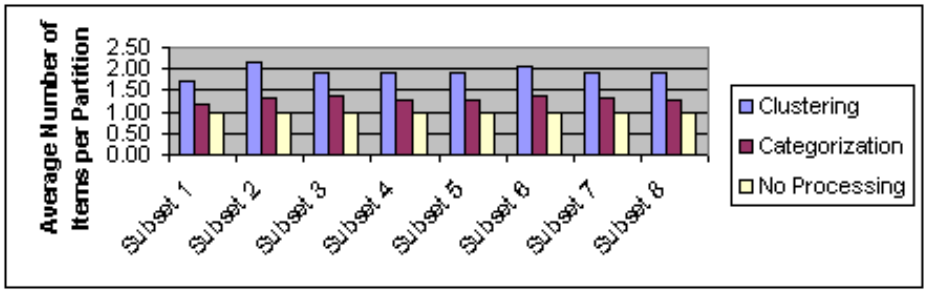


Figure 5.3: Average Number of Item per Partition

label was a combination of the authors contained within the cluster. The users information overload could have been reduced even more if less descriptive cluster labels were used. The cost model uses the cluster labels to calculate information overload. The cluster label is considered efficient if it matches the user query. The advantages to clustering over categorization increases as the number of data items increase (Figure 5.4).

Our experimental results validate the thesis of this paper that similarity, threshold-based, hierarchical agglomerative clustering can reduce the problem of information overload.

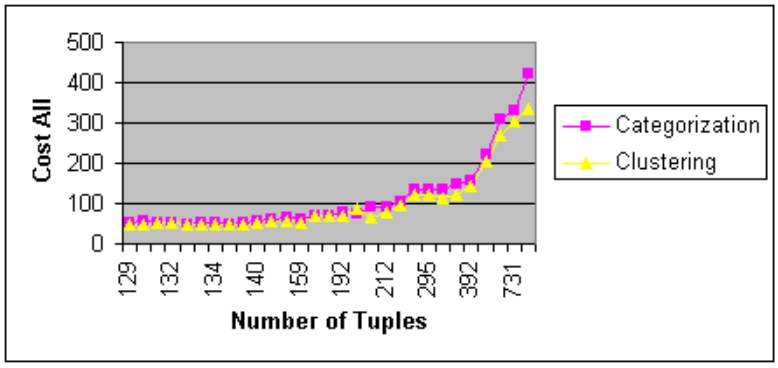


Figure 5.4: Number of Tuples vs Cost

5.2 Qualitative Analysis

Determining the best clustering usually depends on a certain amount of subjectivity. The true value of this clustering algorithm lies in whether it actually reduces the user's information overload. Although the algorithm performed well on the mathematical cost models, there is still a need for a human element to indicate if the algorithm is helpful to the user. The human element is subjective by nature but provides great feedback as to how effective it really is. As a result, we performed some qualitative experiments. After executing the query and clustering the data; the results were presented to users and their feedback received.

The user study involved 4 subjects (research lab members). The experiments clustered the data results from the search "Giraud-Carrier" on the CiteSeer website. The dataset consisted of 67 citations, and after executing the clustering algorithm, 8 clusters were created. The threshold value used was 0.686. This was the value learned on the BibFinder database, the BibFinder database and the Citeseer database are very similar. As expected, the citations clustered around common authors and subjects or some common theme like a conference or year of publishing. Visual inspection also verified that each contained similar items, but differed from items contained in other clusters. A subset of the clustering is in Figure 5.5.

Input was then received from the users to whether these groupings made sense

or not. The observations of users indicated that the clustering appeared reasonable.

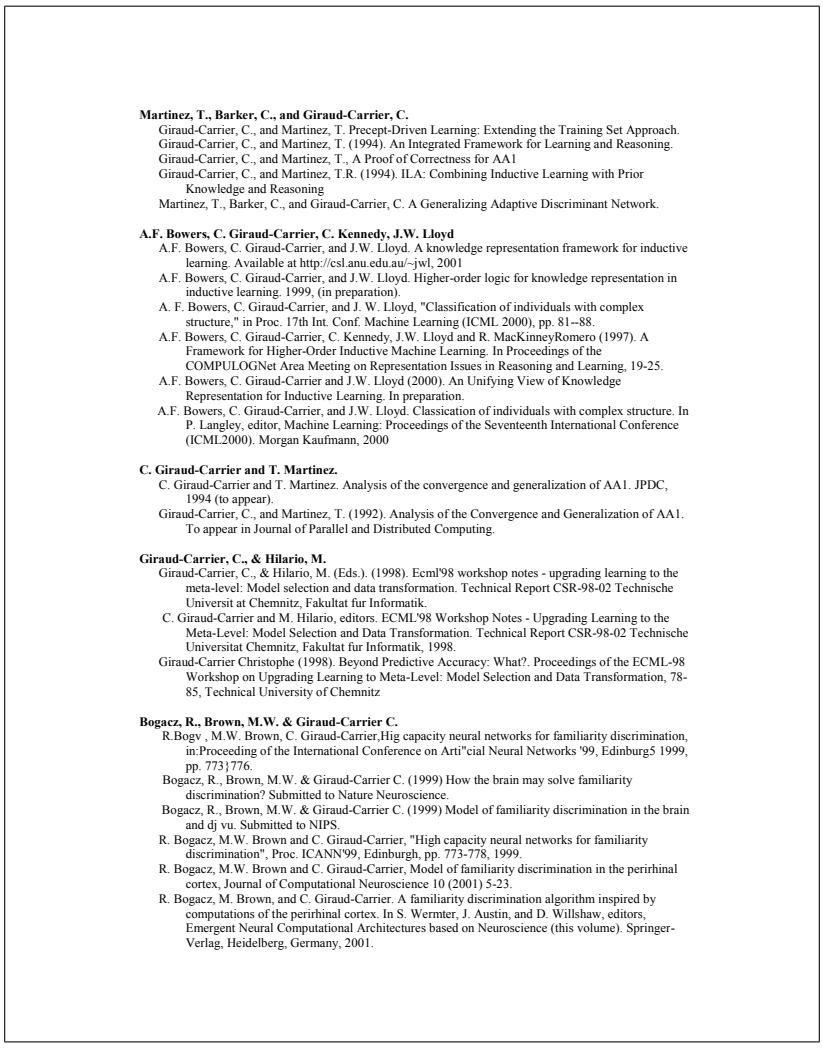


Figure 5.5: Clustering From Giraud-Carrier Clustering

In addition to the citation query, a query for 'Kristine' was executed on the IMDB database (an internet movie database)[36]. This query resulted in 303 results and 39 clusters after running the results through the HAC algorithm. The threshold used was the same as the above example. The labels for this particular clustering were manually added to make better sense of the groupings, (as this is not the focus

of the thesis and it does not alter the actual cluster contents).

An examination of the clustering shows that those individuals with common attributes where dynamically clustered together. Primarily, individuals with the same job titled were paired up. However, the clusters also exposed other commonalities, such as a writer and actress that were in the same movie, a group that were all on TV shows, and yet another cluster identified actresses who were in foreign films. While manual categorization of the 'Job Title' attribute would have given some of this functionality, it is impossible for it to detect these other cited commonalities.

User feedback did indicate that better labeling of the cluster would enhance the ability to find a particular item. As this is not the primary focus of this thesis, we leave this as a area of improvement in future works.

This qualitative exercise of the clusters show that information was reduced. This once again validates that clustering query results is able to provide users information without overwhelming them with to much information.

Chapter 6

Conclusion

Interactive and exploratory data retrieval systems are increasingly being used to access database systems. Queries to these systems often retrieve too many answers. Users then need to spend time sifting and sorting through this information to find the data they require. This problem, of finding a desired information within many items, is often refereed to as "*information overload*".

This thesis presents a clustering of query results to address this problem. This solution dynamically create clusters using a semi-supervised hierarchical agglomerative clustering algorithm. Experimental results show that our algorithm results in reduced information overload. We also present an algorithm to learn the threshold in hierarchical clustering. This learning algorithm is applicable in other areas besides query results.

Future work could include trying various similarity metrics in clustering the results to see how the metric effects the cluster contents. In addition, parsing the query string by separating the author and title may yield different clusters. Another future work could be labeling the clusters with different labeling algorithms and then evaluating different labeling techniques on the users information overload.

Bibliography

[1] Teg Grenager, "Natural language prcessing text clustering, em", http://www.stanford.edu/ grenager/papers/FA05l.pdf, 2005.

[2] Mark Craven, "Clustering gene expression data, http://www.biostat.wisc.edu/ craven/776/lecture14.pdf", 2002.

[3] Wei Wang, "Clustering", http://www.cs.unc.edu/Courses/comp290-090-s06/Lecturenotes/clustering2.pdf, 2006.

[4] Periklis Andritsos, "Scalable clustering of categorical data and applications", http://www.aueb.gr/users/ion/sdep/slides/slides_andritsos_20_12_2005.pdf, 2006.

[5] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das, "DBXplorer: A system for keyword-based search over relational databases", in *ICDE*, 2002, pp. 5–16.

[6] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis, "Automated ranking of database query results", in *CIDR*, 2003.

[7] Vagelis Hristidis and Yannis Papakonstantinou, "DISCOVER: Keyword search in relational databases", in *VLDB*, 2002, pp. 670–681.

[8] MSN House and Home, "http://houseandhome.msn.com/", 2006.

[9] Family Search, "http://www.familysearch.org/".

[10] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence, "Citeseer: An automatic citation indexing system", in *DL'98: Proceedings of the 3rd ACM International Conference on Digital Libraries*, 1998, pp. 89–98.

[11] Google, "http://www.google.com".

[12] Carrot^2, "http://www.carrot2.org/website/xml/index.xml".

[13] Clusty, "http://clusty.com/".

[14] In-Ho Kang and Gil-Chang Kim, "Query type classification for web document retrieval", in *SIGIR*, 2003, pp. 64–71.

[15] Pável Calado, Marco Cristo, Edleno Silva De Moura, Nivio Ziviani, Berthier A. Ribeiro-Neto, and Marcos André Gonçalves, "Combining link-based and content-based methods for web document classification", in *Proceedings of CIKM-03, 12th ACM International Conference on Information and Knowledge Management*, New Orleans, US, 2003, pp. 394–401, ACM Press, New York, US.

[16] Jian Zhang and Yiming Yang, "Robustness of regularized linear classification methods in text categorization", in *SIGIR*, 2003, pp. 190–197.

[17] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma, "Learning to cluster web search results", in *SIGIR*, 2004, pp. 210–217.

[18] Khaled M. Hammouda and Mohamed S. Kamel, "Efficient phrase-based document indexing for web document clustering", *IEEE Trans. Knowl. Data Eng*, vol. 16, no. 10, pp. 1279–1296, 2004.

[19] Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum, "Integrating DB and IR technologies: What is the sound of one hand clapping?", in *CIDR*, 2005, pp. 1–12.

[20] Navin Kabra, Raghu Ramakrishnan, and Vuk Ercegovac, "The QUIQ engine: A hybrid IR DB system", in *ICDE*, 2003, p. 741.

[21] Ullas Nambiar and Subbarao Kambhampati, "Providing ranked relevant results for web database queries", in *WWW (Alternate Track Papers & Posters)*, 2004, pp. 314–315.

[22] Kaushik Chakrabarti, Surajit Chaudhuri, and Seung won Hwang, "Automatic categorization of query results", in *SIGMOD Conference*, 2004, pp. 755–766.

[23] Dictionary.com, "http://www.dictionary.com", 2006.

[24] I. Davidson and S.S. Ravi, "Agglomerative hierarchical clustering with constraints: Theoretical and empirical results", in *Proceedings of the Ninth European Conference on Principles and Practice of Data Mining and Knowledge Discovery (PKDD'05)*, 2005, pp. 59–70.

[25] Jain, Murty, and Flynn, "Data clustering: A review", *CSURV: Computing Surveys*, vol. 31, 1999.

[26] Sam Chapman, "Simmetrics version 1.1", http://www.dcs.shef.ac.uk/s̃am/simmetrics.html, 2004.

[27] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data clustering: A review", *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[28] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and semi-supervised clustering: A brief survey", A Review of Machine Learning Techniques for Processing Multimedia Content. Report of the MUSCLE European Network of Excellence, 2004.

[29] H. Kim and S. Lee, "A semi-supervised document clustering technique for information organization", in *Proceedings of the Ninth ACM International Conference on Information and Knowledge Management*, 2000, pp. 30– 37.

[30] K. Wagstaff and C. Cardie, "Clustering with instance-level constraints", in *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000, pp. 1103–1110.

[31] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithms for document datasets", in *Proceedings of the Eleventh ACM International Conference on Information and Knowledge Management*, 2002, pp. 515–524.

[32] M. Bilenko, S. Basu, and R.J. Mooney, "Integrating constraints and metric learning in semi-supervised clustering", in *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004, pp. 81–88.

[33] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz, "UCI repository of machine learning databases [http://www.ics.uci.edu/∼mlearn/mlrepository.html]", Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[34] LingPipe, "http://alias-i.com/lingpipe", Version 2.1.1, 2005.

[35] Zaiqing Nie, Subbarao Kambhampati, and Thomas Hernandez, "Bibfinder/statminer: Effectively mining and using coverage and overlap statistics in data integration", in *VLDB*, 2003, pp. 1097–1100.

[36] IMDB, "An internet movie database", http://www.imdb.com, 2006.