Advanced Data Clustering Methods of Mining Web Documents

Samuel Sambasivam
Azusa Pacific University
Azusa, CA, USA

ssambasivam@apu.edu

Nick Theodosopoulos
London, UK

london1@rocketmail.com

Abstract

The aim of this paper is to evaluate, propose and improve the use of advanced web data clustering techniques, allowing data analysts to conduct more efficient execution of large-scale web data searches. Increasing the efficiency of this search process requires a detailed knowledge of abstract categories, pattern matching techniques, and their relationship to search engine speed.

In this paper we compare several alternative advanced techniques of data clustering in creation of abstract categories for these algorithms. These algorithms will be submitted to a side-by-side speed test to determine the effectiveness of their design. In effect this paper serves to evaluate and improve upon the effectiveness of current web data search clustering techniques.

Keywords: Web mining, database, data clustering, algorithms, web documents.

Introduction

This paper examines the use of advanced techniques of data clustering in algorithms that employ abstract categories for the pattern matching and pattern recognition procedures used in data mining searches of web documents. With the rapid advances in data mining software technology now taking place, website managers and search engine designers have begun to struggle to maintain efficiency in "mining" for patterns of information and user behaviour. Part of the problem is the enormous amount of data being generated, making the search of web document databases in real time difficult. Real-time searching are critical for real time problem solving, high-level documentation searches and prevention of database security breaches.

The analysis of this problem will be followed by a detailed description of weaknesses in data mining methods, with suggestions for a reduction of pre-processing to improve performance of search engine algorithms, and recommendation of an optimum algorithm for this task.

The first investigators who gave a serious thought to the problem of algorithm speed were persons conducting researching in the area of database searches. The field is still in its infancy; most of the tools and techniques used for data mining today come from other related fields such as pattern recognition, statistics and complexity theory. Only recently have the researchers of these various fields been interacting to solve mining and timing issues. (Oliver & Armin, 2002)

Overview of the Methodology

The methodology employed in this paper will be experimental analysis, with the objective of testing the feasibility of abstract category data clustering algorithms for a real world web application. In order to perform this test, a group of five linear time clustering algorithms will be applied to a sample group of online web documents, simulating the activities of a web search engine looking for similar words, phrases or sequences in a large database set of web articles, publications and records. The six techniques compared will be the K-Means, Single Pass, Fractionation, Buckshot, Suffix Tree and AprioriAll clustering algorithms.

The procedure will be to measure the execution time of the test algorithms in clustering data sets consisting of whole documents, excerpts and key words of a fixed quantity and size.

Purpose of the Study

The purpose of this study is to conduct research that will analyze and improve the use of data clustering techniques in creating abstract categories in algorithms, allowing data analysts to conduct more efficient execution of large-scale searches. Increasing the efficiency of the search process requires a detailed knowledge of abstract categories, pattern matching techniques, and their relationship to search engine speed.

Data mining involves the use of search engine algorithms looking for hidden predictive information, patterns and correlations within large databases. The technique of data clustering divides datasets into mutually exclusive groups. The distance between groups is measured with respect to all the available variables, versus variables that are specific predictors, to produce "abstract categories" for analysis. Search engine algorithms and user audit trails are complex, leading to time-consuming quests for specific information. It is anticipated that the proposed study will identify the most efficient and effective data clustering algorithms for this purpose.

Background of the Problem

Data Clustering is a technique employed for the purpose of analyzing statistical data sets. Clustering is the classification of objects with similarities into different groups. This is accomplished by partitioning data into different groups, known as clusters, so that the elements in each cluster share some common trait, usually proximity according to a defined distance measure. Essentially, the goal of clustering is to identify distinct groups within a dataset, and then place the data within those groups, according to their relationships with each other. Document Clustering automatically groups documents into sets. There is no predetermined taxonomy - the taxonomy of the clusters is determined at run time (Han & Kamber, 2005).

One of the main points of document clustering is that it isn't so much a matter of finding the documents off of the web- that's what search engines are for. Search engines do a fantastic job by themselves, when the user has a specific query, and knows exactly what words will get the desired results. But the user gets a ranked list that has questionable relevance because it turns up anything that has the word in it, and the only metric is the number of times that word appears in the document. With a good clustering program, the user will instead be presented with multiple avenues of inquiry organized into broad groups that get more specific as selections are made.

Clustering exploits similarities between the documents to be clustered. The similarity of two documents is computed as a function of distance between the corresponding term vectors for those documents. Of the various measures used to compute this distance, the cosine-measure has proved the most reliable and accurate. (Meyer Zu Eissen & Stein, 2002).

Data clustering algorithms come in two basic types: hierarchical and partitional. In partitions, searches are matched against the designated clusters, and the documents in the highest scoring clusters are returned as a result. When a hierarchy processes a query, it moves down the tree along the highest scoring branches until it achieves the predetermined stopping condition. The sub tree where the stopping condition is satisfied is then returned as the result of the search.

Both strategies rely on variations of the near-neighbour search. In this search, nearness is determined by the similarity measure used to generate the clusters. Cluster search techniques are comparable to direct near-neighbour searches. Both are evaluated in terms of precision and recall. The evidence suggests that cluster search techniques are only slightly better than direct near-neighbour searches and the former can even be less effective than the latter in certain circumstances. Because of the quadratic running times necessary, clustering algorithms are often slow and unsuitable for extremely large volume tasks.

Hierarchical algorithms start with established clusters, and then create new clusters based upon the relationships of the data within the set. Hierarchical algorithms can do this one of two ways, from the bottom up or from the top down. These two methods are known as agglomerative and divisive, respectively. Agglomerative algorithms start with the individual elements in the set as clusters, and then merge them into successively larger clusters. Divisive algorithms start with the entire dataset in one cluster, and then break it up into successively smaller clusters. Because hierarchical algorithms must analyze all the relationships inherent in the dataset, they tend to be costly in terms of time and processing power.

Partitional algorithms determine the clusters at one time, in the beginning of the clustering process. Once the clusters have been created, each element of the dataset is then analyzed and placed within the cluster that it is the closest to. Partitional algorithms run much faster than hierarchical ones, which allows them to be used in analyzing large datasets, but they have their disadvantages as well. Generally, the initial choice of clusters is arbitrary, and does not necessarily comprise all of the actual groups that exist within a dataset. Therefore, if a particular group is missed in the initial clustering decision, the members of that group will be placed within the clusters that are closest to them, according to the predetermined parameters of the algorithm. In addition, partitional algorithms can yield inconsistent results- the clusters determined this time by the algorithm probably won’t be the same as the clusters generated the next time it is used on the same dataset.

Algorithms of this Study

Of the six algorithms under scrutiny in this paper, three are hierarchical and three are partitional. The three hierarchical methods are suffix tree, single pass and AprioriAll.

The suffix tree, as defined by Black (2005), is “a compact representation of a trie (retrieval) corresponding to the suffixes of a given string where all nodes with one child are merged with their parents.” It is a divisive method; it begins with the dataset as a whole and divides it into progressively smaller clusters, each composed of a node with suffixes branching off of it like leaves.

Single-pass clustering, on the other hand, is an agglomerative, or bottom-up, method. It begins with a single cluster, and then analyzes each element in turn to determine if it falls within a current cluster, or places it in a new cluster, depending on the similarity threshold set by the analyst.

The AprioriAll algorithm builds upon a principle called Learning Associations (Giraud-Carrier, 2004). Learning Associations consists of discovering strong associations among transactions’ items, or by extension, among objects’ variables and attributes. Examples include: Customers who bought product A also bought product B, or Children of 4-5 yrs old with younger siblings are more likely to be kind to others. The purpose of association learning is to find rules that meet two user defined conditions: minimum support and minimum confidence.

The three partitional algorithms are k-means, buckshot, and fractionation. K-means derives its clusters based upon longest distance calculations of the elements in the dataset then it assigns each element to the closest centroid (the data point that is the mean of the values in each dimension of a set of multi-dimensional data points). Buckshot partitioning starts with a random sampling of the dataset, then derives the centres by placing the other elements within the randomly chosen clusters. Fractionation is a more careful clustering algorithm which divides the dataset into smaller and smaller groups through successive iterations of the clustering subroutine. Fractionation requires more processing power, and therefore time (Cutting, Karger, Pedersen, & Tukey, 1992).

Clustering is a methodology for more effective search and retrieval functions pertaining to datasets. The principle is simple enough- documents with a high degree of similarity will automatically be sought by the same query. By automatically placing the documents in groups based upon similarity (e.g. clusters), the search is effectively broadened.

The buckshot algorithm is simple in design and intent. It chooses a small random sampling of the documents in the database, and then applies the cluster routine to them. The centres of the clusters generated by the subroutine are returned. This use of a rectangular time clustering algorithms makes the buckshot method fast. The trade off is that buckshot is not deterministic, since it initially relies on a random sampling process. Repeated use of this algorithm can return different clusters than previous searches, although it is maintained that repeated trials generate clusters that are similar in quality to the previous set of clusters (Cutting, et. al. 1992).

Fractionation algorithms find centres by initially breaking the corpus of documents into a set number of buckets of predetermined size. The cluster subroutine then is applied to each bucket individually, breaking the contents of the bucket into yet smaller groups within the bucket. This process is repeated until a set number of groups are found, and these are the k centres.

This method is like building a branching tree from bottom up, with leaves as individual documents and the centres (clusters) as the roots. The best fractionation methods sort the dataset based on a word index key (e.g. based on words in common between two documents).

It is important to note that both buckshot and fractionation rely on a clustering subroutine. Both algorithms are designed to find the initial centres, but rely on a separate algorithm to do the actual clustering of individual documents. This subroutine can itself be agglomerative, or divisive. However, in practice, the subroutine tends to be an agglomerative hierarchical algorithm. Buckshot applies the cluster subroutine to a random sampling of the dataset to determine the centres, while the fractionation algorithm uses repeated applications of the subroutine over groups of fixed size in order to find the centres. Fractionation is considered more accurate, while buckshot is much faster, making it more suitable for searching in real time on the Web (Cutting, et. al. 1992).

The original k-means algorithm presented by MacQueen (1967 cited in Cheung, 2003) has since been refined. The new version can deal with ellipse shaped data clusters as well as ball shaped ones, and does not suffer from the dead unit problem that plagued the earlier k-means algorithm. In addition, the new k-means algorithm performs proper clustering without pre-determining the exact cluster number. In experimental runs, it has proven to be efficient and accurate.

Suffix trees were introduced in a paper by Zamir and Etzioni (1998). It is an incremental algorithm that runs in linear time. Instead of treating a document as a collection of words, Suffix Tree Clustering treats each document as a string. The string analogy allows suffix tree to analyze relationships between word proximities from document to document. Therefore, documents are grouped together by common phrases and words, and delivered in the form of a suffix tree for ease of browsing. This results in a highly relevant hierarchy of documents.

The primary characteristics of AprioriAll algorithm are that it generates all frequent item sets that satisfy the condition of minimum support. All possible rules are then checked against condition of confidence. All rules above that threshold (minimum confidence) are considered significant. Authors claim it to be a true data mining algorithm. (Giraud-Carrier, 2004) One particular advantage of AprioriAll is that it is easy to implement with a sparse matrix and simple sums. However, it is computationally expensive- the run time depends on the minimum support. It is not strictly an association's learner- as it induces rules which are inherently unidirectional. The basic assumption is that the items under consideration are in sets, and are not in a specific order.

Since the primary focus of this paper is on clustering as a means of searching the web, then there are criteria to be considered when this is kept in mind. There are several important criteria to take into account when analyzing algorithms for use on the web (Zamir & Etzioni, 1998).

Relevance: the method ought to produce clusters that group documents relevant to the user's query separately from irrelevant ones.

- Browsable summaries: The user needs to determine at a glance whether a cluster's contents are of interest. We do not want to replace sifting through ranked lists with sifting through clusters. Therefore the method must produce concise and accurate descriptions of the clusters.
- Overlap: Since documents have multiple topics, it is important that documents not be confined to a single cluster.
- Snippet-tolerance: The method ought to produce high quality clusters even when it only has access to the snippets returned by the search engine, as most users are unwilling to wait for the original documents to be downloaded from the web by the system.
- Speed: A patient user might sift through 100 ranked documents in a set of returned results. A clustering method to allow the user to browse through at least an order of magnitude more documents. Therefore the clustering method ought to be able to cluster up to one thousand snippets in a few seconds. For the impatient user, every second counts.
- Incrementality: To save time, the method should start to process each snippet as it is received from the web.

Of the six algorithms under scrutiny, the only one that treats a document as a string is the Suffix Tree. The other five treat each document as a collection of disjointed words- the order of the words has no importance. This causes a lot of important information to be lost in the search process. Phrases have been used to supplement word-based indexing in IR projects. Lexical atoms and syntactic phrases have been shown to improve accuracy without a corresponding cost in recall. Simple statistical approaches have also been used successfully to generate phrases (Zamir & Etzioni, 1998).

Suffix Tree Clustering is organized into three logical steps:

- The first step is document cleaning, where a light stemming algorithm is used to reduce prefixes and suffixes, turn plural form to singular form, and non-word tokens such as numbers and symbols are stripped. The original document strings are preserved so that the full document can be displayed on command.
- The second step is to identify the base clusters. This phase is equated with creating an inverted index of phrases for the document collection. These results in the Suffix Tree (first introduced by Zamir and Etzioni (1998)), a data structure that can be constructed in a time linear to the number of documents in the dataset, and can also be constructed incrementally, as the documents are being read.
- The third step of the process is to combine base clusters that have many similarities between their respective sets.

Since documents can have multiple phrases in common, it is imperative for the algorithm to recognize nearly identical clusters and merge them to prevent a proliferation of redundant clusters. This step is accomplished by merging clusters that have a high level of overlap.

The Suffix Tree algorithm is incremental. Each document is read, cleaned, and then inserted into the appropriate base cluster, or placed in a newly created cluster if the similarity threshold is exceeded. Then each base cluster is recalculated and redistributed according to the same principles of similarity.

The final clusters created by the Suffix Tree Clustering method are scored and sorted by the scores of their base clusters and the amount of overlap. Each cluster is then displayed according to the number of documents it contains and the phrases of the base clusters.

Suffix Tree Clustering algorithm does not require the user to specify the number of clusters before the search- the clusters are produced incrementally according to the dictates of the algorithm. It does require the specification of the similarity threshold. Suffix Tree Clustering has a markedly less sensitive response to the similarity threshold, as opposed to AHC methods. In this way, Suffix Tree Clustering is much more forgiving concerning the initial specifications of the search.

Data Clustering Problems and Solutions

Data Clustering algorithms are basically designed to address one or more problematic factors, such as high dimensionality, efficiency, scalability with data size, sensitivity to noise in the data, and identification of clusters in various shapes. K-means (and other partitioning methods) result in the break-up of genuine clusters. Hierarchical approaches fail to handle convex and elongated shapes of clusters. Density-based algorithms are extremely sensitive to noise in the data (Foss & Zaiane, 2002). A database may constitute up to 30%-40% of Noisy data. Pre-processing data to remove the noise typically takes more time than the execution time of the algorithm (Joshi, 1997).

The need for input parameters is the most pressing problem in clustering methods. Many algorithms, especially hierarchical ones, need the number of clusters to be arbitrarily selected at the beginning of the process. Even when this is not the case, the parameters of the algorithm can greatly influence the outcome of the process.

The basic idea of partitioning is intuitive in nature, and the process of partitioning is generally to achieve an optimal performance through iterative means. All partitioning methods have comparable clustering qualities, and the major difficulties of these methods are:

- the number of clusters to be found needs to be specified before the process, requiring at least some domain knowledge beforehand (which is often unavailable),
- difficulty in determining clusters with a large variance in size, and
- the method is only suitable for conclave clusters (Foss & Zaiane, 2002).

A hierarchical clustering algorithm produces a dendogram representing the nested grouping relationship of objects in the dataset. The main distinguishing characteristic among hierarchical approaches is the measure of similarity between individual clusters and the underlying modelling of the clusters. These algorithms are costly in terms of time and computational powers, so many are initiated with a sampling of the dataset that is then clustered. This leaves the end results as a direct function of the initial sampling. While this certainly helps with the problem of computational speed, if there are many small clusters within the dataset, the sampling is likely to miss them. (Foss & Zaiane, 2002)

Dennis et. al. (cited in Meyer Zu Eissen & Stein, 2002) placed current web searching technology into four broad classifications:

Unassisted keyword search: One or more search times are entered and the search engine returns a ranked list of document summaries. (ex: Google or AltaVista)

Assisted keyword search: The search engine produces results based on the user's initial inquiry. (ex: Vivisimo)

Directory-based Search: Here, the information space is divided into a hierarchy of categories, where the user navigates from broad to specific classes. (ex: yahoo)

Query –by-example: The user selects an interesting document snippet, which is then used as the basis for a new query.

Meyer zu Eissen and Stein (2002) specify the major criteria for a successful search engine. The ideal search engine process has three stages.

- First, an initialization phase according to the plain unassisted keyword search paradigm.
- Second, a categorization phase similar to the directory based search paradigm.
- Third, a refinement phase that may combine aspects from assisted keyword search and query-by-example paradigm.

As shown in the work of Zamir and Etzioni (1998) the feasibility of searching web results is heavily influenced by the ability to preserve accuracy when clustering based only upon the snippets returned to a search engine. In their results, it was found that almost the same accuracy was present in the use of snippets only, as opposed to the whole document. This alleviates the computer from downloading the entire document for the express purpose of measuring similarities with other documents. In general, each algorithm was slightly more accurate when using the entire document, but the accuracy of the snippets was much higher than expected. In the case of the k-means algorithm, it actually performed better using the snippets instead of the whole document.

Sequential pattern mining is the mining of frequently occurring patterns related to time or other sequences (Han & Kamber, 2005).

Sequential Data mining can be confined to the data within a specified time frame. This can be set time allotments, by the month or year, etc. They can also be tied to specific parameters, such as the weeks directly following a hurricane (Han & Kamber, 2005).

The Apriori algorithm is mainly concerned with the mining of sequential patterns, or sequential analysis (Giraud-Carrier, 2004). The basic method to mine sequential patterns (based upon Apriori) is to count all large sequences (including non maximal ones.

The Apriori Algorithm operates in five phases: Sort phase, large item set phase, transformation phase, sequence phase, maximal phase. In the Sort Phase, the algorithm sorts the database with customer ID as major key and transaction time as minor key. Large item set phase is where the algorithm finds large item sets and maps them to integers. Following this is the Transformation phase, where the algorithm deletes non-large item datasets and then transforms them using integer mapping. In the Sequence phase, the algorithm uses the set of large item sets to find the desired sequences. For the Maximal phase, it is necessary to find the maximum sequences among the large item sets.

This phase is sometimes combined with sequence phase, depending on algorithm. When the two are separate, the algorithm wastes a lot of time counting non-maximal sequences which cannot be sequential patterns. A possible solution to this problem is the AprioriSome algorithm (Giraud-Carrier, 2004). It generates candidates for a pass using only the large sequences found in the previous pass. It also employs forward and backward passes.

The facts revealed in the literature review can be summarized in a few key points. Data clustering is an essential step in making web- based searches more user friendly and productive. Data Clustering operates by attempting to measure how alike documents are, then sorting them into groups based upon content. The more accurate a Clustering algorithm is, the slower it works, because it must examine more relationships in greater detail. There are many different algorithms to choose from this study encompasses a review of five of the most successful and popular clustering algorithms.

K-means clustering is a partitional algorithm that works in linear time. It calculates the centres to be used in clustering through an analysis of maximum distance relationships between documents in the dataset. Single pass clustering is an incremental algorithm that develops its clusters through calculations based upon the vector differences ascribed to the documents in the dataset.

Buckshot is a clustering method that takes a random sampling of documents to use as the centres. Fractionation is an algorithm that divides the entire dataset into fixed size buckets that are then each clustered by the clustering subroutine, then the clusters are merged or split according to similarity. Suffix Tree Clustering is a hierarchical algorithm that runs in linear time, and provides an intuitive platform in the form of the suffix tree.

Proposed Solution & Methodology

Speed versus Accuracy

The natural trade-off that exists is between the speed of the algorithm and its accuracy at returning highly relevant results. This is simply because more accurate clustering techniques require more analysis of the data set, which translates into a higher number of calculations to be performed in order to accomplish the task. The more accurate algorithms require so much time to construct the clusters that they are rendered impractical for online real time searching. Fractionation, for instance, performs excellently when allowed to cluster data offline, but the high computational cost would make its usefulness in searching the web suspect.

The suffix tree clustering method provides an avenue for exploration. In previous tests, Suffix Tree Clustering outperformed the other methods in terms of speed (Zamir & Etzioni, 1998). It also provides an intuitive visual for presenting the data, thereby allowing the user to sort through the documents for the most relevant ones pertaining to the search.

Another strong point is that it is incremental- this allows it to begin sorting documents into clusters as soon as they are received, then to add new documents as they come. In the tests from the study, the results were that the clustering program was finished clustering the last files received just .01 seconds after the search engine had delivered them.

Large Datasets and Noisy Data

The sheer size of data is a serious impediment to data mining techniques. Most existing algorithms have not addressed the issue of out of core data structures. Parallel algorithms, and other newer techniques, have begun to address this problem (Joshi, 1997).

The use of snippets seems to detract from the amount of “noise” a document can generate. These excerpts of web documents represent a search engine’s attempt at extracting the distinguishing phrases of a web document. This reduction in noise could have a positive effect on low accuracy techniques, such as k-means and buckshot. In the case of k-means, this streamlines the calculations a great deal because of the reduced number of relational analyses that need to be performed.

The four primary types of web-based search are the unassisted keyword search, the assisted keyword search, the directory search, and the query-by-example. The algorithms presented here fit some or all of these functions. The unassisted keyword search is straightforward, and presents the results returned from the search engine as is, in their ranked positions. This has, proven to be a difficult and time consuming process for sifting through to find specific information on a given subject (hence the need for data clustering).

The assisted keyword search is an advancement in terms of relevancy- it provides links based upon words with similar meanings. This increases the chance of finding the right kind of information, but still leaves the sorting and sifting to the user- there is very little categorization involved, and the search still results in ranked lists. The directory search is much more closely similar to the results of clustering at least. It is a hierarchical approach, allowing the user to begin in broad categories as yielded by the search, then to progressively narrow the search and find smaller groups of more specific documents. The query-by-example is another method that closely simulates the clustering approach. By selecting a snippet and deriving a new search around it, the search becomes iterative, allowing the user to further refine the search based upon the excerpt of a highly relevant site.

When considering these four points, the need for clustering algorithms becomes clearer. The often intimidating number of results that can be found via a search engine can frustrate many users. It also creates a large obstacle to efficient sifting and sorting of data. The major focus of the clustering algorithm is to display those search results in a sensible fashion. By grouping similar documents together, the user is given a much friendlier working environment in which to search.

Based upon the research, it seems that the hierarchical algorithms have a much better platform from which to satisfy those four essential types of web- search. Certainly, the more favourable searches are the directory based and query-by-example, which are both functions that are facilitated by the suffix tree, as a hierarchical, string-based algorithm. While there will certainly be those times when the keyword search is desired, such as when you know the exact title of the document you’re looking for, the more generalized search options are fitting for the average user’s needs.

Since the focus of this paper is to measure the speeds of the various algorithms and produce a conclusion based upon the results, there is little attention paid to the functionality or accuracy of the algorithms. In general, it is known that there is a trade-off between speed and accuracy when using algorithms. In that case, it should be expected that the fastest algorithm is not going to be the most accurate one. Due to the nature of web-based search, however, speed is essential. Therefore, the objective is to find the fastest algorithm, then to find ways to improve its accuracy. Since the introduction of the k-means algorithm, there have been several significant advances and permutations of it that have improved performance and speed. This can also be accomplished with other algorithms as well.

Implementation

The procedure of the experiment was to measure the execution time of the test algorithms in clustering data sets consisting of whole documents, excerpts and key words of a fixed quantity and size. Times were recorded for a series of 10 tests repeated for the same group size and numbers of files, in order to determine the optimum clustering algorithm for real time online web document searches. The dataset consisted of 1000 documents pertaining to classic literature. Each document contained on average 710 words, while the excerpts averaged 50 words. The documents were results of searches for the keywords “classic” and “literature.” The source of the documents was from the internet. Google search engine was used to locate and download them.

The PC was an AMD Athlon 1500+, 60 GB HD, 1024MB Ram, an intel 100Mb network card, ADSL line 2000Mb/s and a home made router/firewall (Smoothwall).

There are three series of tests. Each series is a compilation of the time taken by each of the chosen clustering algorithms to sort the documents of the dataset. The first series consists of each algorithm analyzing the entire content of the document. The second series compares the speed of searches based upon excerpts, or snippets. The third and final series of tests measures the speed of searches based upon keywords only.

Throughout the experiment, the suffix tree consistently outperformed the other algorithms with respect to speed. This proved true for all three levels of the experiment. The second best performer was the buckshot algorithm. The worst performer overall was the fractionation algorithm. Although it still runs at acceptable speeds, it is by far the slowest of the algorithms that see common usage. K-means and Single Pass reside in the middle ground between the two extremes. The following charts show a complete breakdown of each algorithms performance in each series of tests.

Table 1 Full document comparison

Trial No.	k-means	single pass	buckshot	fractionation	suffix tree	AprioriAll
1	142.6s	156.2s	116.8s	221.9s	102.9s	180.1s
2	144.8s	164.9s	113.1s	224.7s	101.5s	180.2s
3	140.5s	159.8s	118.3s	223.2s	97.9s	180.0s
4	144.8s	160.1s	111.5s	218.8s	100.1s	180.3s
5	139.2s	160.2s	110.1s	219.9s	102.2s	179.9s
6	137.2s	158.4s	109.3s	217.1s	103.6s	180.4s
7	137.6s	161.2s	118.9s	220.2s	108.0s	179.8s
8	140.6s	162.4s	115.8s	215.8s	104.4s	180.5s
9	136.7s	165.6s	108.5s	220.4s	103.0s	179.7s
10	142.2s	165.3s	112.4s	220.1s	99.4s	180.6s

Table 1 illustrates the performance of each algorithm when using the entire content of the document for clustering. The suffix tree handled the document load efficiently and beat the nearest competitor by an average of 11.2 seconds. The slowest algorithm, fractionation is more than twice as slow as the suffix tree in finishing the clustering task. In each case, the mean value of the range was within 1 second of the actual calculated average. In this series of tests, the algorithms each performed in accordance with the expectations as presented in the literature.

Table 2 Comparative results

Algorithm	Avg. Value	Range (low-high)	Mean Value
k-means	140.6	136.7- 144.8	140.75
Single pass	161.4	156.2- 165.6	160.9
buckshot	113.5	108.5- 118.9	113.7
fractionation	220.2	215.8- 224.7	220.25
suffix tree	102.3	97.9- 108.0	102.95
AprioriAll	180.1	179.7-180.6	180.1

Table 2 gives a breakdown of the comparative results of the first series of tests. The listing includes the average speed of each algorithm, the range of values from lowest to highest, and the mean value of that range. As expected the means stay close to the average values. The suffix tree came at the lowest average, beating the competition by as much as 117.9 seconds. While k-means and single pass both perform well, they are outperformed by buckshot, which comes in at a close second.

Having now run the first series of tests, it is possible to begin to formulate a basic idea of what the other two series will bring. Keep in mind that the next two series will run in just a fraction of the time of the first test. In each instance, however, the basic pattern established by the initial series of tests should be corroborated by the results of the other tests. Throughout the experiment, there will be a continuous examination of the possible ramifications of the data. It is expected that the other two series of tests are going to produce results in line with the first series. Each series of tests was recorded in the same fashion, with the only metric being speed of the operation.

Table 3 Clustering by excerpts

Trial No.	k-means	single pass	buckshot	fractionation	suffix tree	AprioriAll
1	9.8s	11.0s	8.0s	15.2s	7.2s	13.3s
2	10.0s	11.3s	7.8s	15.5s	7.1s	13.2s
3	10.2s	11.1s	8.1s	15.5s	6.9s	13.4s
4	9.7s	11.2s	7.8s	15.2s	7.1s	13.1s
5	9.7s	11.4s	7.7s	15.6s	7.0s	13.5s
6	9.7s	11.0s	7.7s	15.4s	7.3s	13.0s
7	10.2s	11.4s	8.2s	15.4s	7.4s	13.6s
8	9.9s	11.2s	8.1s	15.2s	7.2s	12.9s
9	9.9s	11.5s	7.7s	15.1s	7.1s	13.7s
10	10.0s	11.4s	7.7s	15.5s	7.0s	12.8s

Table 3 shows the results of the test when applied to the excerpts of the documents. This search highlights certain strengths for the clustering algorithms- the reliance on key phrases instead of the document as a whole has allowed each algorithm to perform much faster. Not surprisingly, however, the suffix tree is still on top with a lead ranging from .8 seconds to 7.3 seconds. Buckshot comes in a strong second place, with an average of 7.9 seconds. Fractionation lags far behind the others in terms of speed- it is still 4 full seconds away from the next in line, the single pass. At this level, the gap between the k-means and buckshot methods becomes more apparent. Also, this type of test is a comparable representation of the sort of search that a librarian or researcher would be performing regularly.

Table 4 Comparative results

Algorithm	Avg. Value	Range	Mean Value
k-means	9.9	9.7-10.2	10.0
single pass	11.4	11.0-11.5	11.3
buckshot	7.9	7.7-8.2	8.0
fractionation	15.4	15.1-15.6	15.4
suffix tree	7.1	6.9-7.4	7.2
AprioriAll	13.3	12.9-13.8	13.3

Table.4 shows the average values again stay very close to the mean value, according to range. In the case of fractionation, the mean and the average coincide. The results continue to place the suffix tree at the top of the list, followed in order by buckshot, k-means, single pass and fractionation. The shortest gap is between the first two, buckshot and the suffix tree, with a mere .8 seconds separating them. The largest gap is between the last two (single pass and fractionation) with a full 4.0 seconds between them.

Table 5 presents the results of the search by keyword series. The small running times are indicative of the simplicity of the search- after all, the algorithm is going to find the keywords in every single document in the dataset. As before, the suffix tree continues to dominate the proceedings, while the fractionation algorithm trails far behind. The relative places of each algorithm that were established in the first two series are maintained, validating the hypothesis. Since the search is faster overall the result difference between the algorithms is noticeably less.

Table 5 Keywords “classic” and “literature”

Trial No.	k-means	single pass	buckshot	fractionation	suffix tree	AprioriAll
1	.40s	.44s	.32s	.60s	.28s	.50s
2	.39s	.44s	.31s	.62s	.28s	.49s
3	.39s	.44s	.32s	.63s	.27s	.51s
4	.41s	.45s	.31s	.61s	.28s	.48s
5	.38s	.46s	.31s	.62s	.28s	.49s
6	.39s	.44s	.30s	.62s	.29s	.51s
7	.38s	.46s	.30s	.61s	.29s	.50s
8	.40s	.45s	.33s	.60s	.28s	.52s
9	.40s	.46s	.31s	.61s	.28s	.49s
10	.39s	.46s	.30s	.62s	.28s	.50s

The patterns that have been seen so far continue to be reinforced in the third series. The relative positions of the algorithms continue to remain stable with respect to one another. The suffix tree continues to be the strongest performer, while fractionation lags behind. The largest gap in this series is between single pass and fractionation, at .16 seconds. The smallest gap is a mere .3 seconds between the suffix tree and buckshot. At this level of operation, the time increments are exceedingly small, but proportionately, they stay within the same ranges as seen in the previous two series of tests. The mean values continue to stay within a short range of the average for each algorithm.

The range of lowest to highest for each algorithm is in the order of .3 seconds, giving an average variance of .15 seconds for this series.

Table 6 Comparative results

Algorithm	Avg. Value	Range	Mean Value
k-means	.39	.38-.41	.395
single pass	.45	.44-.46	.45
buckshot	.31	.30-.33	.315
fractionation	.61	.60-.63	.615
suffix tree	.28	.27-.29	.28
AprioriAll	.50	.48- .52	.50

The tests performed were designed solely to test clustering algorithms in their speed. From the results recorded, it can be seen that the suffix tree is the fastest performing algorithm. In all cases the relative ranking of the algorithms remained the same throughout all three series of tests. The ranking of the algorithms according to average speed is summarized in Table 7. The values given are for the averages of the algorithm in each of the three series of tests. The algorithms are also listed in order from fastest to slowest.

Table 7 Rankings of the six algorithms

Algorithm	1st series	2nd series	3rd series
suffix tree	102.3	7.1	.28
Buckshot	113.5	7.9	.31
k-means	140.6	9.9	.39
single pass	161.4	11.4	.45
AprioriAll	180.1	13.3	.50
fractionation	220.2	15.4	.61

Conclusion of Results

The World Wide Web is huge, widely distributed and global. It is too huge for effective data mining and data warehousing. It has no standards and structure, making it heterogeneous and complex to the point of being prohibitive (Han & Kamber, 2005). The World Wide Web grows and changes rapidly. There is a highly diverse group of users, and thus an equally diverse cross section of topics. Only a relatively small portion of web content is useful to a given user (on the order of 1%).

Web Search engines operate on a few common principles (Han & Kamber, 2005). Index based searches alternately search the web, index the web pages, and build and store huge keyword-based indices. The deficiencies of this approach are twofold: a topic of appreciable breadth is likely to return hundreds or thousands of documents, and many documents that are highly relevant may not contain the keyword of the search, and thereby be missed by the search engine. Web mining includes mining web link structures to identify authoritative web pages, the automatic classification of web documents, building a multilayered web information base, and web-log mining.

The results of the experiment clearly indicate that the suffix tree is the fastest clustering algorithm in terms of real time processing speed. It far outstrips fractionation in terms of speed. It maintains a healthy margin over both k-means and single pass methods. Buckshot runs a close second, but the suffix tree consistently outperforms it as well. The suffix tree also provides an excellent interface for the user, giving the results of the search with an intuitive bent. This allows for quick and easy browsing of categories until the desired objective is obtained.

However, the other algorithms are far from dead. Buckshot is not too far off- with a little work, or possibly a new suffix tree subroutine, the time gap could be closed. There are several versions of the k-means algorithm in existence, and the k-means has certainly improved greatly since its introduction in 1967.

The AprioriAll algorithm has some interesting points, but overall, its slow run time makes it impractical for real time web searches. While other uses of Apriori, like AprioriSome, result in a faster run time, they also reinforce the trade off between time and accuracy that is prevalent in the algorithms being studied.

When weighed against other features, the suffix tree still produces strong results. It has the ability to define similarities through strings of words, instead of just the words themselves. This makes the relevancy of its results much higher than the other algorithms. This, coupled with its speed, make it the ideal algorithm for web-based data clustering.

Earlier in this paper, the criteria for a web-based clustering algorithm were discussed. The suffix tree successfully fits all the criteria as expressed herein. The points are repeated below, with additional commentary on how the Suffix Tree fulfils the requirements of each step.

Relevance: the method ought to produce clusters that group documents relevant to the user’s query separately from irrelevant ones.

This is one of the suffix tree’s strongest points. While every algorithm presumably has this as its goal; the point is that some do it better than others. Some of the older algorithms, such as k-means have been known to produce questionable results when grouping documents according to relevancy.

Browsable Summaries

The user needs to determine at a glance whether a cluster’s contents are of interest. Replacing sifting through ranked lists with sifting through clusters is not desirable. Therefore the method must produce concise and accurate descriptions of the clusters.

This is another aspect that the suffix tree fulfils. The suffix tree provides a concise yet accurate description of each cluster, thereby allowing the user to easily determine whether or not the cluster has relevant documents to the search. While other algorithms do produce clusters that can be browsed through, there is no guarantee that the cluster will contain all the most relevant documents.

Overlap

Since documents have multiple topics, it is important that documents not be confined to a single cluster.

This is also a feature that the suffix tree performs well. Since it relies on strings of words, and it is highly likely that documents that have the same phrase in them will be closely related in content, theme and context. This is important, because without the overlap, it is easy to miss an appropriate search result because the algorithm happened to place it in the wrong cluster.

Snippet-Tolerance

The method ought to produce high quality clusters even when it only has access to the snippets returned by the search engine, as most users are unwilling to wait for the original documents to be downloaded from the web by the system.

The results have shown that snippets are almost as effective as the full documents in the process of clustering accurately. The suffix tree was no exception to this, and still outperformed the others in terms of absolute speed. While entire document searches have shown themselves to be more accurate overall, the degree of relevance delivered by excerpts from documents has been surprisingly high.

Speed

A patient user might sift through 100 ranked documents in a set of returned results. It would be better to use clustering, allowing the user to browse through at least an order of magnitude more documents. Therefore the clustering method ought to be able to cluster up to one thousand snippets in a few seconds. For the impatient user, every second counts.

As the results of this experiment indicate, it is indeed possible for a clustering algorithm to process 1000 documents in just a few seconds when snippets are used. While snippets are in fact not as accurate as the full document, the increases in speed of computation will more than make up for the loss of accuracy. Most users would rather wait seven seconds to get their results than nearly two minutes.

Incrementality

To save time, the method should start to process each snippet as it is received from the web. Of all the algorithms that were tested, the suffix tree is the only one that performed incrementally. This provides it with the advantage of being able to begin the clustering process before all the results have been returned by the search engine. Incrementality also allows the algorithm to process more results as they are returned and place them in the appropriate cluster one at a time. This has provided a significant speed advantage for the Suffix Tree clustering algorithm.

Conclusion

When weighed by these criteria, the suffix tree stands out as not only the fastest data clustering algorithm, but it is also the most accurate and easy to use. While the focus of the research in this paper is based upon the speed of the operation, there is no significant effort to qualitatively investigate the accuracy of the algorithms in producing relevant clusters.

After this study we think a new concept of data mining pattern has to be developed. Search engine capable of automatic data mining, relationship building abilities and “smart engine”. This new concept will be able to accurately find relevant information. This should be the focus of further research. The effectiveness and increased efficiency of “smart engine” will lead to a greater productivity and more satisfying end user experience.

References

Black, P. (2005). *Dictionary of Algorithms and Data Structures*. National Institute of Standards and Technology web site. Retrieved April 23 2005 from: http://www.nist.gov/dads/

Cheung, Y. (2003). k*-means: A new generalized k-means clustering algorithm. *Pattern Recognition Letters, 24*(15), 2883-2893. Retrieved 16 August 2005, from: www.comp.hkbu.edu.hk/~ymc/papers/journal/PATREC_3261_publication_version.pdf

Cutting, D. R., Karger, D. R., Pedersen, J. O., & Tukey, J. W. (1992). Scatter/Gather: A cluster-based approach to browsing large document collections. *Proceedings of the Fifteenth Annual International ACM SIGIR Conference*, June 1992, 318-329. Retrieved August 9 2005, from: http://www2.parc.com/istl/projects/ia/papers/sg-sigir92/sigir92.html

Foss, A., & Zaiane, O.R. (2002). A parameter less method for efficiently discovering clusters of arbitrary shape in large datasets. *Proceedings of the IEEE International Conference on Data Mining* (ICDM'2002), pp 179-186. Retrieved August 2 2005, from: http://doi.ieeecomputersociety.org/10.1109/ICDM.2002.1183901

Giraud-Carrier, C., (2004). From *CS 478- Association Rule Learning.ppt* Department of Computer Sciences, Brigham Young University. Retrieved August 2 2005 from: http://faculty.cs.byu.edu/~cgc/Teaching/CS_478_Su05/Lectures/CS%20478%20-%20Association%20Rule%20Learning.ppt

Han, J., & Kamber, M. (2005). *Data Mining: Concepts and Techniques* (Chapter 9). Intelligent Database Systems Research Lab. School of Computing Science, Simon Fraser University, Canada. Retrieved August 19 2005, from: http://www.cs.sfu.ca/~han/bk/9cmplx.ppt

Joshi, K. P. (1997). *Analysis of Data Mining Algorithms.* University of Minnesota. Retrieved July 25 2005, from: http://www-users.cs.umn.edu/~desikan/research/dataminingoverview.html#apriori

Meyer zu Eissen & Stein. (2002). *Analysis of Clustering Algorithms for Web-based Search*. Paderborn University, pp. 168-178. Retrieved September 2 2005, from: http://www-ai.upb.de/aisearch/pakm02-frame.pdf

Oliver W. & Armin W. (2003). *Data Mining for Ontology Building Data Mining Concepts.* HSR Hochschule Für Technik Rapperwil (University). Retrieved April 1 2005, from: http://informatik.hsr.ch/Content/Gruppen/Doz/jjoller/diplom/DataMiningForOntologyBuilding/DataMining/DataMiningConcepts.pdf

Zamir, O. & Etzioni, O. (1998). Web document clustering: A feasibility demonstration. *Proceedings of ACM SIGIR'98*. Retrieved May 15 2005, from: www.cs.washington.edu/homes/etzioni/papers/sigir98.pdf

Biographies

Dr. Samuel Sambasivam is the chairman of the Department of Computer Science of Azusa Pacific University. Professor Sambasivam has done extensive research, publications, and presentations in both computer science and mathematics. His research interests include optimization methods, expert systems, Fuzzy Logic, client/server, Databases, and genetic algorithms. He has taught computer science and mathematics courses for over 20 years. Professor Sambasivam has run the regional Association for Computing Machinery (ACM) Programming Contest for six years. He has developed and introduced several new courses for computer science majors. Professor Sambasivam teaches

Database Management Systems, Information Structures and Algorithm Design, Microcomputer Programming with C++, Discrete Structures, Client/Server Applications, Advanced Database Applications, Applied Artificial Intelligence, JAVA and others courses. Professor Sambasivam coordinates the Client/Server Technology emphasis for the Department of Computer Science at Azusa Pacific University.

Mr. Nick Theodosopoulos has a proven and successful educational background, along with over 10 year's commercial experience in the IT sector. He is currently working as an independent software consultant serving the financial institutions in the city of London. He has performed several roles including software development, in a variety of languages and operating systems. He specializes in data manipulation, migration and database design. He has recently read a Masters in Computer science at Liverpool University. His current pursuits and interests are database performance tweaking, data striping and enhancing mathematical empirical formulae all to speed up database transaction processes and overall performance.