Analysis and Comparison of Efficient Techniques of Clustering Algorithms in Data Mining

Shiv Pratap Singh Kushwah, Keshav Rawat, Pradeep Gupta

Abstract—This paper presents the comparison of data mining algorithms for clustering. These algorithms are among the most influential data mining algorithms in the research community. With each algorithm, we provide a description of the algorithm, discuss the impact of the algorithm, and review current and further research on the algorithm. These algorithms cover classification, clustering, statistical learning, association analysis, and link mining, which are all among the most important topics in data mining research and development.

Index Terms— cluster, data mining, clustering method, k-mean.

I. INTRODUCTION

Cluster analysis divides data into meaningful or useful groups (clusters). If meaningful clusters are the goal, then the resulting clusters should capture the "natural" structure of the data. For example, cluster analysis has been used to group related documents for browsing, to find genes and proteins that have similar functionality, and to provide a grouping of spatial locations prone to earthquakes. However, in other cases, cluster analysis is only a useful starting point for other purposes, e.g., data compression or efficiently finding the nearest neighbors of points. Whether for understanding or utility, cluster analysis has long been used in a wide variety of fields: psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining.

The scope of this paper is modest: to provide an introduction to cluster analysis in the field of data mining, where we define data mining to be the discovery of useful, but non-obvious, information or patterns in large collections of data. Much of this paper is necessarily consumed with providing a general background for cluster analysis, but we also discuss a number of clustering techniques that have recently been developed specifically for data mining.

.

II. K- MEAN ALGORITHM

The k-means algorithm is a simple iterative method to partition a given dataset into a user- specified number of clusters, k. This algorithm has been discovered by several researchers across different disciplinesGray and Neuhoff [6] provide a nice historical Back ground for k-means placed in the larger context of hill-climbing algorithms. The algorithm operates on a set of d-dimensional vectors, $D = \{xi \mid i = 1, \ldots, N\}$, where $xi \in d$ denotes the ith data point. The algorithm is initialized by picking k points in d as the initial k cluster representatives or "centroids". Techniques for selecting these initial seeds include sampling at random from the dataset, setting them as the solution of clustering a small subset of the data or perturbing the global mean of the data k times. Then the algorithm iterates between two steps till convergence:

Step 1: Data Assignment. Each data point is assigned to its closest centroid, with ties broken arbitrarily. This results in a partitioning of the data.

Step 2: Relocation of "means". Each cluster representative is relocated to the center (mean) of all data points assigned to it. If the data points come with a probability measure (weights), then the relocation is to the expectations (weighted mean) of the data partitions.

The algorithm converges when the assignments (and hence the cj values) no longer change. The algorithm execution is visually depicted in Fig. 1. Note that each iteration needs N × k comparisons, which determines the time complexity of one iteration. The number of iterations required for convergence varies and may depend on N , but as a first cut, this algorithm can be considered linear in the dataset size.

One issue to resolve is how to quantify "closest" in the assignment step. The default measure of closeness is the Euclidean distance, in which case one can readily show that the non-negative cost function,

$$\sum_{i=1}^{N}\left(\underset{j}{\operatorname{argmin}}||\mathbf{x_i}-\mathbf{c_j}||_2^2\right)$$

will decrease whenever there is a change in the assignment or the relocation steps, and hence convergence is guaranteed in a finite number of iterations. The greedy-descent nature of k-means on a non-convex cost also implies that the convergence is only to a local optimum, and indeed the algorithm is typically quite sensitive to the initial centroid locations. Figure 2 1illustrates how a poorer result is obtained for the same dataset as in Fig. 1 for a different choice of the three initial centroids. The local minima problem can be countered to some extent by running the algorithm multiple times with different initial centroids, or by doing limited local search about the converged solution.

Manuscript received August 08, 2012.

Shiv Pratap Singh Kushwah, Department of CSE/IT, ITM Universe, Gwalior, India

Keshav Rawat, Department of CSE/IT, ITM Universe, Gwalior, India

Pradeep Gupta, Department of CSE/IT, GEC, Gwalior, India

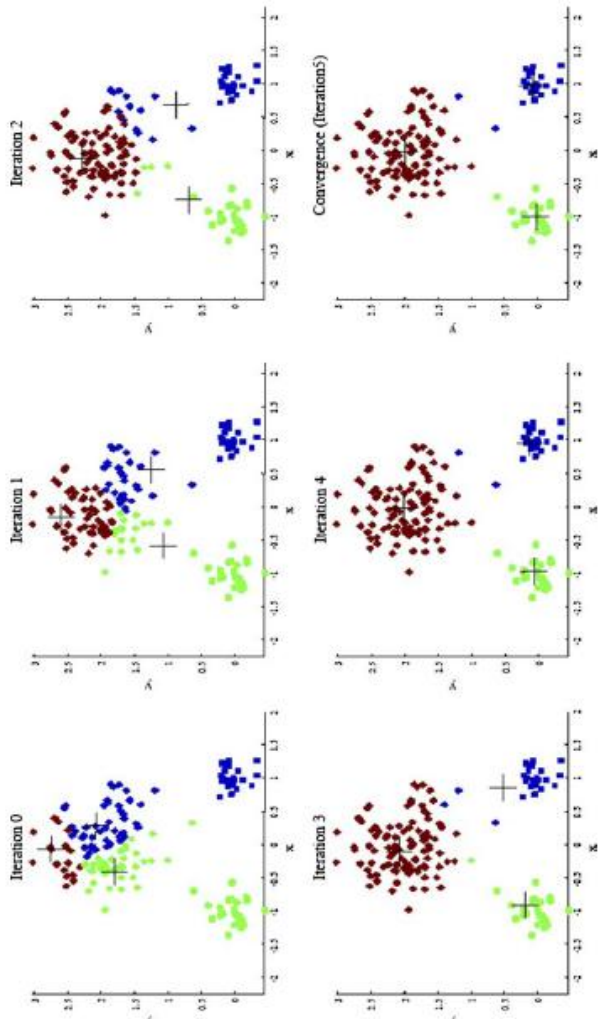

Fig. 1 Changes in cluster representative locations (indicated by '+' signs) and data assignments (indicated by color) during an execution of the k-means algorithm

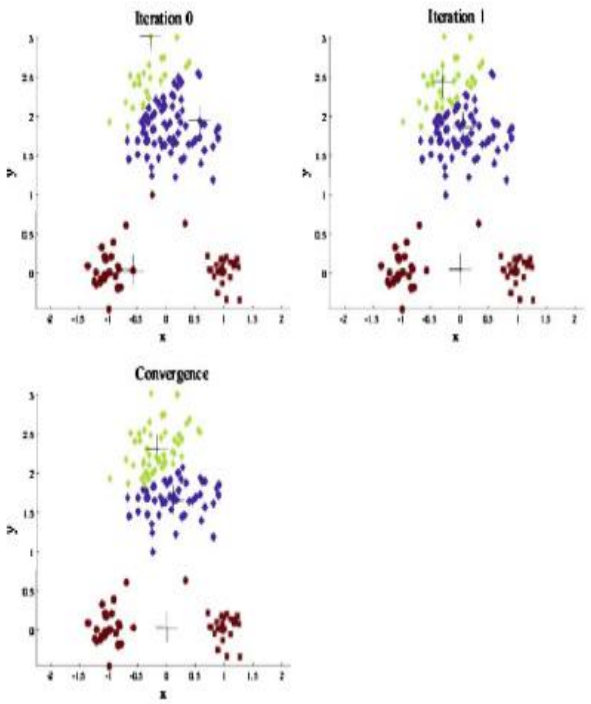

Fig. 2 Effect of an inferior initialization on the k-means results

In addition to being sensitive to initialization, the k-means algorithm suffers from several other problems. First, observe that k-means is a limiting case of fitting data by a mixture of k Gaussians with identical, isotropic covariance matrices (= σ 2 I), when the soft assignments of data points to mixture components are hardened to allocate each data point solely to the most likely component. So, it will falter whenever the data is not well described by reasonably separated spherical balls, for example, if there are non-covex shaped clusters in the data. This problem may be alleviated by rescaling the data to "whiten" it before clustering, or by using a different distance measure that is more appropriate for the dataset. For example, information-theoretic clustering uses the KL-divergence to measure the distance between two data points representing two discrete probability distributions. It has been recently shown that if one measures distance by selecting any member of a very large class of divergences called Bregman divergences during the assignment step and makes no other changes, the essential properties of k-means, including guaranteed convergence, linear separation boundaries and scalability, are retained [2]. This result makes k-means effective for a much larger class of datasets so long as an appropriate divergence is used. k-means can be paired with another algorithm to describe non-convex clusters. One first clusters the data into a large number of groups using k-means. These groups are then agglomerated into larger clusters using single link hierarchical clustering, which can detect complex shapes. This approach also makes the solution less sensitive to initialization, and since the hierarchical method provides results at multiple resolutions, one does not need to pre-specify k either.

The cost of the optimal solution decreases with increasing k till it hits zero when the number of clusters equals the number of distinct data-points. This makes it more difficult to (a) directly compare solutions with different numbers of clusters and (b) to find the optimum value of k. If the desired k is not known in advance, one will typically run k-means with different values of k, and then use a suitable criterion to select one of the results. For example, SAS uses the cube-clustering-criterion, while X-means adds a complexity term (which increases with k) to the original cost function (Eq. 1) and then identifies the k which minimizes this adjusted cost. Alternatively, one can progressively increase the number of clusters, in conjunction with a suitable stopping criterion. Bisecting k-means achieves
this by first putting all the data into a single cluster, and then recursively splitting the least compact cluster into two using 2-means. The celebrated LBG algorithm [6] used for vector quantization doubles the number of clusters till a suitable code-book size is obtained. Both these approaches thus alleviate the need to know k beforehand.

The algorithm is also sensitive to the presence of outliers, since "mean" is not a robust statistic. A preprocessing step to remove outliers can be helpful.

III. KNN: K-NEAREST NEIGHBOR CLASSIFICATION

One of the simplest, and rather trivial classifiers is the Rote classifier, which memorizes the entire training data and performs classification only if the attributes of the test object match one of the training examples exactly. An obvious drawback of this approach is that many test records will not be classified because they do not exactly match any of the training records. A more sophisticated approach, k-nearest neighbor (kNN) classification [4,10], finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood. There are three key elements of this approach: a set of labeled objects, e.g., a set of stored records, a distance or similarity metric to compute distance between objects, and the value of k, the number of nearest neighbors. To classify an unlabeled object, the distance of this object to the labeled objects is computed, its k-nearest neighbors are identified, and the class labels of these nearest neighbors are then used to determine the class label of the object.

Figure 3 provides a high-level summary of the nearest-neighbor classification method. Given a training set D and a test object x = (x , y), the algorithm computes the distance (or similarity) between z and all the training objects (x, y) ∈ D to determine its nearest-neighbor list, Dz . (x is the data of a training object, while y is its class. Likewise, x is the data of the test object and y is its class.)

Once the nearest-neighbor list is obtained, the test object is classified based on the majority class of its nearest neighbors:

$$\text{Majority Voting: } y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i),$$

where v is a class label, yi is the class label for the ith nearest neighbors, and I (·) is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

Input: D, the set of k training objects, and test object $z = (\mathbf{x}', y')$

Process:

Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every object, $(\mathbf{x}, y) \in D$.

Select $D_z \subseteq D$, the set of k closest training objects to z.

Output: $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$

Fig. 3 The k-nearest neighbor classification algorithm

3.1 Issue with kNN:

There are several key issues that affect the performance of kNN. One is the choice of k. If k is too small, then the result can be sensitive to noise points. On the other hand, if k is too large, then the neighborhood may include too many points from other classes. Another issue is the approach to combining the class labels. The simplest method is to take a majority vote, but this can be a problem if the nearest neighbors vary widely in their distance and the closer neighbors more reliably indicate the class of the object. A more sophisticated approach, which is usually much less sensitive to the choice of k, weights each object's vote by its distance, where the weight factor is often taken to be the reciprocal of the squared distance: wi = 1/d(x , xi)2 . This amounts to replacing the last step of the kNN algorithm with the following:

$$\text{Distance-Weighted Voting: } y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} w_i \times I(v = y_i).$$

The choice of the distance measure is another important consideration. Although various measures can be used to compute the distance between two points, the most desirable distance measure is one for which a smaller distance between two objects implies a greater likelihood of having the same class. Thus, for example, if kNN is being applied to classify documents, then it may be better to use the cosine measure rather than Euclidean distance. Some distance measures can also be affected by the high dimensionality of the data. In particular, it is well known that the Euclidean distance measure become less discriminating as the number of attributes increases. Also, attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes. A number of schemes have been developed that try to compute the weights of each individual attribute based upon a training set [5].

3.2 Impact of kNN:

KNN classification is an easy to understand and easy to implement classification technique. Despite its simplicity, it can perform well in many situations. In particular, a well known result by Cover and Hart [3] shows that the the error of the nearest neighbor rule is bounded above by twice the Bayes error under certain reasonable assumptions. Also, the error of the general kNN method asymptotically approaches that of the Bayes error and can be used to approximate it.

KNN is particularly well suited for multi-modal classes as well as applications in which an object can have many class labels. For example, for the assignment of functions to genes based on expression profiles, some researchers found that kNN outperformed SVM, which is a much more sophisticated classification scheme [9].

3.3 Current and future research

Although the basic kNN algorithm and some of its variations, such as weighted kNN and assigning weights to objects, are relatively well known, some of the more advanced techniques for kNN are much less known. For example, it is typically possible to eliminate many of the stored data objects, but still retain the classification accuracy of the kNN classifier. This is known as 'condensing' and can greatly speed up the classification of new objects [7]. In addition, data objects can be removed to improve classification accuracy, a process known as "editing" [13]. There has also been a considerable amount of work on the application of proximity graphs (nearest neighbor graphs, minimum spanning trees, relative neighborhood graphs, Delaunay triangulations, and Gabriel graphs) to the kNN problem. Recent papers by Toussaint [11,12], which emphasize a proximity graph viewpoint, provide an overview of work

addressing these three areas and indicate some remaining open problems.

IV. THE PRIORI ALGORITHM

One of the most popular data mining approaches is to find frequent itemsets from a transaction dataset and derive association rules. Finding frequent itemsets (itemsets with frequency larger than or equal to a user specified minimum support) is not trivial because of its combinatorial explosion. Once frequent itemsets are obtained, it is straightforward to generate association rules with confidence larger than or equal to a user specified minimum confidence. Apriori is a seminal algorithm for finding frequent itemsets using candidate generation [1]. It is characterized as a level-wise complete search algorithm using anti-monotonicity of itemsets, "if an itemset is not frequent, any of its superset is never frequent". By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order.

Let the set of frequent itemsets of size k be Fk and their candidates be Ck . Apriori first scans the database and searches for frequent itemsets of size 1 by accumulating the count for each item and collecting those that satisfy the minimum support requirement. It then iterates on the following three steps and extracts all the frequent itemsets.

1. Generate Ck+1 , candidates of frequent itemsets of size k + 1, from the frequent itemsets of size k.
2. Scan the database and calculate the support of each candidate of frequent itemsets.
3. Add those itemsets that satisfies the minimum support requirement to Fk+1 .

The Apriori algorithm is shown in Fig. 3. Function apriori-gen in line 3 generates Ck+1 from Fk in the following two step process:

1. Join step: Generate R K +1 , the initial candidates of frequent itemsets of size k + 1 by taking the union of the two frequent itemsets of size k, Pk and Q k that have the first k – 1 elements in common.

$$R_{K+1} = P_k \cup Q_k = \{item_1, \ldots, item_{k-1}, item_k, item_{k'}\}$$
$$P_k = \{item_1, item_2, \ldots, item_{k-1}, item_k\}$$
$$Q_k = \{item_1, item_2, \ldots, item_{k-1}, item_{k'}\}$$

where, iteml < item2 < · · · < item k < item k .

2. Prune step: Check if all the itemsets of size k in Rk+1 are frequent and generate Ck+1 by removing those that do not pass this requirement from Rk+1 . This is because any subset of size k of Ck+1 that is not frequent cannot be a subset of a frequent itemset of size k + 1.

Function subset in line 5 finds all the candidates of the frequent itemsets included in transaction t. Apriori, then, calculates frequency only for those candidates generated this way by scanning the database.

It is evident that Apriori scans the database at most kmax+1 times when the maximum size of frequent itemsets is set at kmax .

The Apriori achieves good performance by reducing the size of candidate sets (Fig. 4). However, in situations with very many frequent itemsets, large itemsets, or very low min- imum support, it still suffers from the cost of generating a huge number of candidate sets and scanning the database repeatedly to check a large set of candidate itemsets. In fact, it is necessary to generate 2100 candidate itemsets to obtain frequent itemsets of size 100.

Algorithm 1 *Apriori*

F_l=(Frequent itemsets of cardinality 1);
for($k = 1; F_k \neq \phi; k++$) do begin
  C_{k+1} = apriori-gen(F_k); //New candidates
  for all transactions $t \in$ Database do begin
    C'_t = subset(C_{k+1}, t); //Candidates contained in t
    for all candidate $c \in C'_t$ do
      $c.count++$;
    end
    $F_{k+1} = \{C \in C_{k+1} \quad | c.count \geq$ minimum support $\}$
  end
end
Answer $\cup_k F_k$;

Fig. 4 Apriori algorithm

4.1 The impact of the algorithm

Many of the pattern finding algorithms such as decision tree, classification rules and clustering techniques that are frequently used in data mining have been developed in machine learning research community. Frequent pattern and association rule mining is one of the few exceptions to this tradition. The introduction of this technique boosted data mining research and its impact is tremendous. The algorithm is quite simple and easy to implement. Experimenting with Apriori-like algorithm is the first thing that data miners try to do.

4.2 Current and further research

Since Apriori algorithm was first introduced and as experience was accumulated, there have been many attempts to devise more efficient algorithms of frequent itemset mining. Many of them share the same idea with Apriori in that they generate candidates. These include hash-based technique, partitioning, sampling and using vertical data format. Hash-based technique can reduce the size of candidate itemsets. Each itemset is hashed into a corre- sponding bucket by using an appropriate hash function. Since a bucket can contain different itemsets, if its count is less than a minimum support, these itemsets in the bucket can be removed from the candidate sets. A partitioning can be used to divide the entire mining problem into n smaller problems. The dataset is divided into n non-overlapping partitions such that each partition fits into main memory and each partition is mined separately. Since any itemset that is potentially frequent with respect to the entire dataset must occur as a frequent itemset in at least one of the partitions, all the frequent itemsets found this way are candidates, which can be

checked by accessing the entire dataset only once. Sampling is simply to mine a random sampled small subset of the entire data. Since there is no guarantee that we can find all the frequent itemsets, normal practice is to use a lower support threshold. Trade off has to be made between accuracy and efficiency. Apriori uses a horizontal data format, i.e. frequent itemsets are associated with each transaction. Using vertical data format is to use a different format in which transaction IDs (TIDs) are associated with each itemset. With this format, mining can be performed by taking the intersection of TIDs. The support count is simply the length of the TID set for the itemset. There is no need to scan the database because TID set carries the complete information required for computing support.

The most outstanding improvement over Apriori would be a method called FP-growth (frequent pattern growth) that succeeded in eliminating candidate generation [8]. It adopts a divide and conquer strategy by (1) compressing the database representing frequent items into a structure called FP-tree (frequent pattern tree) that retains all the essential information and (2) dividing the compressed database into a set of conditional databases, each associated

with one frequent itemset and mining each one separately. It scans the database only twice. In the first scan, all the frequent items and their support counts (frequencies) are derived and they are sorted in the order of descending support count in each transaction. In the second scan, items in each transaction are merged into a prefix tree and items (nodes) that appear in common in different transactions are counted. Each node is associated with an item and

its count. Nodes with the same label are linked by a pointer called node-link. Since items are sorted in the descending order of frequency, nodes closer to the root of the prefix tree are shared by more transactions, thus resulting in a very compact representation that stores all the necessary information. Pattern growth algorithm works on FP-tree by choosing an item in the order of increasing frequency and extracting frequent itemsets that contain the

chosen item by recursively calling itself on the conditional FP-tree. FP-growth is an order of magnitude faster than the original Apriori algorithm.

V. CONCLUSION

Data mining is a broad area that integrates techniques from several fields including machine learning, statistics, pattern recognition, artificial intelligence, and database systems, for the analysis of large volumes of data. There have been a large number of data mining algorithms rooted in these fields to perform different data analysis tasks.

The K mean approach makes the solution less sensitive to initialization, and since the hierarchical method provides results at multiple resolutions, and K-mean algorithm is also sensitive to the presence of outliers, since “mean” is not a robust statistic. A Knn algorithm is more sophisticated approach, k-nearest neighbor (kNN) classification ,finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood. KNN classification is an easy to understand and easy to implement classification technique. Despite its simplicity, it can perform well in many situations. Apriori is a seminal algorithm for finding frequent itemsets using candidate generation . It is characterized as a level-wise complete search algorithm using anti-monotonicity of item sets. We hope this paper can inspire more researchers in data mining to further explore these algorithms, including their impact and new research issues.

REFERENCES

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB conference, pp 487–499
2. Banerjee A, Merugu S, Dhillon I, Ghosh J (2005) Clustering with Bregman divergences. J Mach Learn Res 6:1705–1749
3. Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inform Theory 13(1):21–27
4. Fix E, Hodges JL, Jr (1951) Discriminatory analysis, nonparametric discrimination. USAF School of Aviation Medicine, Randolph Field, Tex., Project 21-49-004, Rept. 4, Contract AF41(128)-31, February 1951
5. Han E (1999) Text categorization using weight adjusted k-nearest neighbor classification. PhD thesis, University of Minnesota, October 1999
6. Gray RM, Neuhoff DL (1998) Quantization. IEEE Trans Inform Theory 44(6):2325–2384
7. Hart P (1968) The condensed nearest neighbor rule. IEEE Trans Inform Theory 14:515–516
8. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of ACM SIGMOD international conference on management of data, pp 1–12
9. Kuramochi M, Karypis G (2005) Gene Classification using Expression Profiles: A Feasibility Study. Int J Artif Intell Tools 14(4):641–660
10. Tan P-N, Steinbach M, Kumar V (2006) Introduction to data mining. Pearson Addison-Wesley11. Toussaint GT (2002) Proximity graphs for nearest neighbor decision rules: recent progress. In: Interface- 2002, 34th symposium on computing and statistics (theme: Geoscience and Remote Sensing). Ritz-Carlton Hotel, Montreal, Canada, 17–20 April, 2002
11. 12.Toussaint GT (2002) Open problems in geometric methods for instance-based learning. JCDCG 273–283
12. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cyberne 2:408–420