Clustering & Biclustering:Application In gene Expression Finding

Project report submitted for the partial fulfillment of the requirements for the degree of

Bachelor of Technology
In
Computer Science And Engineering

By

Bijay Das
Sambit Ghosh
Souradeep Bhowmick
Sourav Maji
Souvik Majee

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL

Department of Computer Science & Engineering

At
Government College of Engineering
&
Leather Technology

2018

DECLARATION

It is hereby declared that the work presented in the project report is an original concept except for citations and quotations which have been duly acknowledged. It is also declared that neither this report nor any part of it has been submitted elsewhere for any other degree or award of any kind at any university or any other institutions.

____

(Bijay Das)
Roll: 11200214008

____

(Sambit Ghosh)
Roll: 11200114032

____

(Souradeep Bhowmick)
Roll: 11200114041

____

(Sourav Maji)
Roll: 11200114043

____

(Souvik Majee)
Roll: 11200114044

ACKNOWLEDGEMENT

Successful completion of this work will never be one man's task. It requires hard work in right direction. There are many who have helped to make our experience as a student a rewarding one.

In particular, we express our gratitude and deep regards to our project guide **Dr. Brijesh Srivastava**, first for his valuable guidance, constant encouragement and kind co-operation throughout period of work which has been instrumental in the success of the project.

Bijay Das
Sambit Ghosh
Souradeep Bhowmick
Sourav Maji
Souvik Majee

Department of Computer Science & Engineering
Government College of Engineering and Leather
Technology West Bengal

CERTIFICATE

This is to certify that the project entitled, "**Clustering & Biclustering: Application in gene expression finding**" in partial fulfilment of the requirements for the award of Bachelor of Technology Degree in Information Technology at the Government College of Engineering and Leather Technology, West Bengal is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge the matter embodied in the project has not been submitted to any other University/ Institute for the award of any Degree or Diploma.

"Clustering & Biclustering: Application in gene expression finding"

is the bona fide work of

Bijay Das
Sambit Ghosh
Souradeep Bhowmick
Sourav Maji
Souvik Majee

Supervisor,

(Dr. Brijesh Srivastava)
Assistant Professor,
Dept. Of Computer Science & Engineering.
Government College of Engineering & Leather Technology
West Bengal

INDEX

ABSTRACT

The need to analyze high-dimension biological data is driving the development of new data mining methods.Bi-clustering and clusterings algorithm have been successfully applied to gene expression data to discover local pattern. Bi-clustering has become a popular technique for the study of gene expression data, especially for discovering functionally related gene sets under different subsets of experimental conditions. Clustering has also done a good progress in this field. Most of clustering approaches use a measure or cost function that determines the quality of bi-clusters. In this project we have learned several algorithm in bi-clustering and clustering and will try to analyze the gene expression matrix using those algorithm.

INTRODUCTION

Monitoring large number of genes is a tedious task for now days.The sample of genes may correspond to different time point or different experimental condition.The sample may have come from different organs,from cancerous or healthy tissues.Simply visualizing this kind of large data set called gene expression data or simply expression data is challenging and extracting relevant data or knowledge is a very hard task to do.One of the solution of this problem is that we can separate the similar kind of genes.Thus the total number of data set will be partitioned into smaller number of set.It is always easy to monitor those small gene expression data.

Biclustering and clustering algorithm is one of the successful approach to exploring the data.Using these algorithms, we can partition objects into clusters to maximize within-cluster similarity, or minimize between-cluster similarity, based on a similarity measure. Given a two-dimensional gene expression matrix M with m rows and n columns, in which the each columns corresponds to one condition, and each row corresponds to one gene. Biclustering, allows simultaneously clustering of both rows and columns in the data matrix. This method is useful to capture the genes that are correlated only in a subset of samples. Such clusters are biologically interesting since they not only allow us to capture the correlated genes, but also enable the identification of genes that do not behave similar in all conditions

PROPOSED WORK:

In this project we have given the task of understanding the basic concept of biclustering and clustering and its several algorithm,also its application in gene expression finding.And implements some of the algorithm of biclustering and clustering.

BICLUSTERING

1.INTRODUCTION:-

Bi-Clustring or Co - Clustering is a data mining method that simultaneously clusters both rows and columns of a matrix. Given a set of m samples represent by n-dimensional feature vector, the entire dataset can be represented as m rows in n columns , the biclustering algorithm generates biclusters are subset of rows which exhibit similar behavior across a subset of columns, or vice versa.

2.DEFINITION:-

Biclusters are represented in the literature in different ways, where genes can be found either in rows or columns, and different names refer the same expression sub-matrix.
Let, from nowon, B be a bicluster consisting of a set ***i*** of **|I|** genes and a set *J* of |J|conditions, in which **bij** refers to the expression level of gene *i* under sample *j*. Then B can be represented as follows:

$$\mathscr{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1|J|} \\ b_{21} & b_{22} & \dots & b_{2|J|} \\ \vdots & \vdots & \ddots & \vdots \\ b_{|I|1} & b_{|I|2} & \dots & b_{|I||J|} \end{pmatrix}$$

where the gene G_i is the ith row, i.e.,G_i={ $b_{i,1}$, $b_{i,2}$,..., $b_{i|J|}$}, and condition c_j is the jth column, i.e., c_j={ $b_{1,j}$, $b_{2,j}$,..., $b_{|I|,j}$}.

Genes and samples means in biclusters are frequently used in several evaluation measure definitions. We represent these values as $b_{i,J}$ and $b_{I,j}$ referring to the *i* row (gene) and *j* column (sample) means, respectively. Furthermore, the mean of all the expression values in B is referred to as b_{IJ}.

BICLUSTER TAXONOMY BASED ON GENE EXPRESSION DATA:

The four major classes of biclusters are--

1)biclusters with constant values
2)Biclusters with constant values on rows and columns
3)Biclusters with coherent values on both orws and columns

1.Bicluster with constant values

When a biclustering algorithm tries to find a constant bicluster, the normal way for it is to reorder the rows and columns of the matrix so it can group together similar rows/columns and find biclusters with similar values. This method is OK when the data is tidy. But as the data can be noisy most of the times, so it can’t satisfy us. More sophisticated methods should be used. A perfect constant bicluster is a matrix(I,J) where all values a(i,j) are equal to μ. In real data, a(i,j) can be seen as n(i,j) +μ where n(i,j) is the noise. According to Hartigan’s algorithm, by splitting the original data matrix into a set of biclusters, variance is used to compute constant biclusters. So, a perfect bicluster is a matrix with variance zero. Also, in order to prevent the partitioning of the data matrix into biclusters with only one row and one column, Hartigan assumes that there are K biclusters within the data matrix. When the data matrix is partitioned into K biclusters, the algorithm ends.

2.Biclusters with constant values on rows or columns

This kind of biclusters can't be evaluated just by variance of its values. To finish the identification, the columns and the rows should be normalized at first. There are other algorithms, without normalization step, can find biclusters have rows and columns with different approaches.

3.Biclusters with coherent values

For biclusters with coherent values on rows and columns, an overall improvement over the algorithms for biclusters with constant values on rows or on columns should be considered. That means a sophisticated algorithm is needed. This algorithm may contain analysis of variance between groups, using co-variance between both rows and columns. In Cheng and Church's theorem, a bicluster is defined as a subset of rows and columns with almost the same score. The similarity score is used to measure the coherence of rows and columns.

3.EVALUATION OF BICLUSTER:-

There are two ways of evaluating a biclustering result: internal and external. Internal measures, such as cluster stability, rely only on the data and the result themselves. Currently there are no internal bicluster measures in scikit-learn. External measures refer to an external source of information, such as the true solution. When working with real data the true solution is usually unknown, but biclustering artificial data may be useful for evaluating algorithms precisely because the true solution is known.

To compare a set of found biclusters to the set of true biclusters, two similarity measures are needed: a similarity measure for individual biclusters, and a way to combine these individual similarities into an overall score.

To compare individual biclusters, several measures have been used. For now, only the Jaccard index is implemented:

$$J(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

where A and B are biclusters, $|A \cap B|$ is the number of elements in their intersection. The Jaccard index achieves its minimum of 0 when the biclusters to not overlap at all and its maximum of 1 when they are identical.

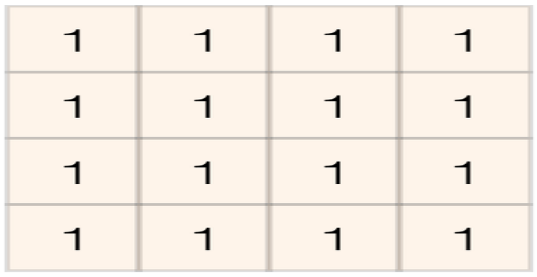
Several methods have been developed to compare two sets of biclusters. For now, only consensus score (Hochreiter et. al., 2010) is available:

1. Compute bicluster similarities for pairs of biclusters, one in each set, using the Jaccard index or a similar measure.

2. Assign biclusters from one set to another in a one-to-one fashion to maximize the sum of their similarities. This step is performed using the Hungarian algorithm.

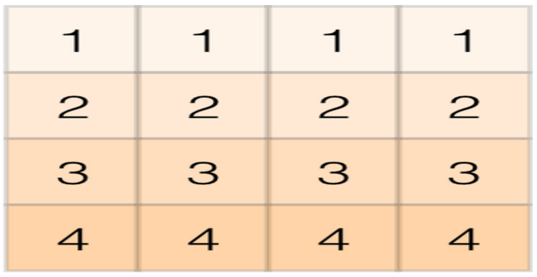
3. The final sum of similarities is divided by the size of the larger set.

The minimum consensus score, 0, occurs when all pairs of biclusters are totally dissimilar. The maximum score, 1, occurs when both sets are identical.



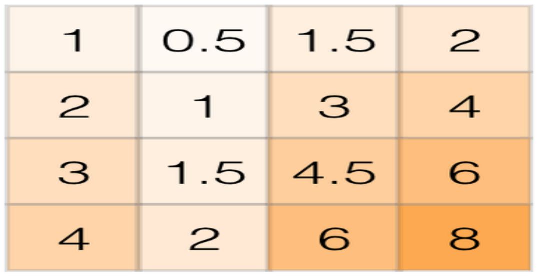
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

(A)



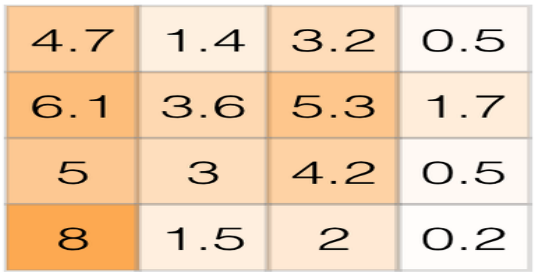
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

(B)



1	0.5	1.5	2
2	1	3	4
3	1.5	4.5	6
4	2	6	8

(C)



4.7	1.4	3.2	0.5
6.1	3.6	5.3	1.7
5	3	4.2	0.5
8	1.5	2	0.2

(D)

6. ALGORITHMS:-

There are several algorithm used to find the bi-clusters. they are as follows:-

- **Cheng & Church:-** A greedy search for low residue score submatrices
- **Getz et al. : -**Iterative clustering of genes (or conditions) across conditions (or genes)subsets
- **Ben-Dor et al. :-** Identification of order preserving submatrices
- **Segal et al. :-** Based on the Bayesian method
- **Ihmels et al. :-**"Stable" subsets of genes and conditions are built around known gene subsets
- **Lazzeroni & Owen:-** Introduces the Plain Model

6.1. CHENG & CHURCH:-

The algorithm of Cheng and Church presents a simple, greedy, approach towards finding maximal sized biclusters satisfying a certain condition [3]. The input is a matrix A = (ai,j), where rows represent genes and columns represent conditions. The algorithm attempts to find a submatrix B, representing a bicluster. The quality of B as a bicluster is measured using the Residue score.

6.1.1. RESIDUE SCORE:-

The basic assumption of the algorithm is that expression levels in a bicluster are constant up to row and column (additive) effects. Formally, let B be a bicluster, I the row indices of B and J the column indices in B. Then there exist functions $b : I \rightarrow R$ and $c : J \rightarrow R$ such that $a_{i,j} \approx b(i) + c(j) + const$ for all $i \in I$ and $j \in J$.

Lemma 1 If ai,j=bi+cj+const for all i ∈ I and j ∈ J then ai,j = ai,J +aI,j −aI,J. Where I and J are row and column subsets representing a sub-matrix, and:

$a_{I,j} = \sum_{i\in I}(a_{i,j})/|I|$ (i.e. sub-matrix column j average)

$a_{i,J} = \sum_{j\in J}(a_{i,j})/|J|$ (i.e. sub-matrix row i average)

$a_{I,J} = \sum_{i\in I, j\in J}(a_{i,j})/(|I||J|)$ (i.e. entire sub-matrix average)

Then: $a_{i,j} = a_{I,j} + a_{i,J} - a_{I,J}$

Proof:

$a_{I,j} = \sum_{i\in I}(b_i+c_j+const)/|I| = b_i + c_j + const$

$a_{i,J} = \sum_{j\in J}(b_i+c_j+const)/J| = b_i + c_j + const$

$a_{I,J} = \sum_{i\in I}\sum_{j\in J}(b_i+c_j+const)/(|I||J|) =$

$=\sum_{i\in I}(b_i+c_j+const)/(|I|) =$

$= (b_i + c_j + const)$

$\Rightarrow a_{I,j} + a_{i,J} - a_{I,J} =$

$= (b_I + c_j + const) + (b_i + c_J + const) - (b_I + c_J + const) =$

$= (b_i + c_j + const) = a_{i,j}$

We proceed to define the residue score of an element $a_{i,j}$ as: $a_{i,j}$-$a_{I,j}$-$a_{i,J}$+ $a_{I,J}$ .This indicates the distance of this element's expression data from our expectations. The mean squared residue score for the sub-matrix $A_{I,J}$ is then: $H(I, J) = \sum_{i\in I, j\in J}(a_{i,j}-a_{I,j}- a_{i,J}+ a_{I,J})^2/|I||J|$. This measures how deviant each element in the bicluster is (on average) from our expectations, and is a natural assessment criterion for the quality of our bicluster.

<u>6.1.2. δ−BICLUSTERING:-</u>

Perhaps the most natural goal of this algorithm would be to find a bicluster minimizing the mean squared residue score. It is easy to see that the mean squared residue score is 0 iff the submatrix satisfies our assumption. However, it is clear to see that such an approach would yield trivial (one gene and one condition) biclusters, and would, in general, prefer small biclusters. In order to overcome this, we define $A_{I,J}$ as a δ- bicluster if H(I, J) ≤ δ. Assuming this threshold indicates strong similarity, we may confine ourselves to finding large δ-biclusters, i.e. δ-biclusters with maximum area. We shall first focus on finding a single bicluster, and then we will discuss how the algorithm can be modified to find several biclusters.

<u>6.1.3. FINDING MORE THAN ONE BI-CLUSTER:-</u>

So far we explained how a single bicluster is found. Note that the algorithm is completely deterministic. Consecutive runs of this algorithm on the same matrix will always yield the same bicluster. In order to find other biclusters, the complete algorithm repeats the process described above, after masking the bicluster

found in the previous iteration. Masking is perfromed by filling the positions of the bicluster with random values. This method has the benefit of reducing significantly the overlap between detected biclusters. The new random values will probably not form any recognizable pattern, and therefore they will be the first candidates to be removed in the row/column deletion phase.

6.1.4. ALGORITHM:-

Following is the complete Cheng-Church algorithm :

```
• Cheng-Church(E,δ)
• while (bicluster found)
•         do I ← All genes in E
•         J ← All conditions in E
•         AdditionPhase(I,J)
•         DeletionPhase(I,J)
•         Report bicluster I,J
•          Mask(I,J)
```

6.2. STATISTICAL-ALGORITHMIC METHOD OF BI-CLUSTERING ANALYSIS(SAMBA):-

This SAMBA algorithm shifts the problem domain from finding sub-matrices with coherent behavior to the well-researched domain of graph theory. The algorithm first converts the input genes vs. conditions expression data into a bipartite graph G = (U,V,E) where U is the set of conditions, V is the set of genes and (u,v)∈ E iff gene v responds in condition u, that is, if the expression level of v changes significantly in u with respect to its normal level. This reduces the problem of discovering the most significant biclusters in the data to finding the densest subgraphs in a bipartite graph.

6.2.1. STATISTICAL METHOD:-

Because different genes/conditions have a typical noise characteristics, not all dense subgraphs are statistically significant (the more noisy genes/conditions have high probability of appearing in dense subgraphs at random). To distinguish between ”real” dense sub-graphs and statistically insignificant ones we will compare the graph to a random graph with similar characteristics. The result will be a weight function on the pairs (u,v). In order to do that a random graph model will be used to produce a likelihood ratio score.

Random graph model: The null hypothesis model, or random graph mode, assumes that each vertex pair (u,v) forms an edge with probability p(u,v), independently of all other edges. p(u,v) is defined to be the probability of observing an edge (u,v) in a random degree-preserving bipartite graph. p(u,v) is well approximated by $\frac{d_{ud_v}}{m}$, where d_u is the degree of U, d_v is the degree of V , and m is the total number of edges.

The alternative hypothesis model, or bicluster model, assuems that each edge of a bicluster appears with a constant high probability p_c.

Likelihood ratio score: The likelihood ratio score L of a sub-graph B = (U',V ',E') is

$$\textbf{L(B)=}\prod_{(u,v)\in E'}\frac{p_c}{p(u,v)}\prod_{u,v\ni E'}\log\frac{1-p}{1-p(u,v)}$$

$$\log L(B)=\sum_{(u,v)\in E'}\log\frac{p_c}{p(u,v)}\sum_{(u,v)\ni E'}\log\frac{1-p_c}{1-p(u,v)}$$

Setting the weights of the edges to $\log\frac{p_c}{p(u,v)}$ **and non edges to** $\log\frac{1-p_c}{1-p(u,v)}$ will result in the score of B being simply the sum of weights of its edges and non edges.

6.2.2. FINDING MAXIMUM BOUNDED BICLIQUE:-

To cope with the NP-hardness of the problem we examine a restriction of it, where one side of the bipartite graph is assumed to have a bounded degree. This is motivated by the observation that most genes have bounded degree and that high degree genes are less informative (will appear in biclusters at random with a high probability). Under this assumption a maximal bounded bipartite subgraph can be found in $O(n^{2d})$ time, where d is the upper bound for the degree, and n is the number of genes. As we shall later see, this assumption is not used in practice.

The Maximum Bounded Biclique problem calls for identifying a maximum weight complete subgraph of a given weighted bipartite graph G, such that the vertices on one side of G have degrees bounded by d. Given the weight function w and a bi-graph G = (U,V,E) a maximal bounded biclique is a biclique B = (U',V',E') such that the weight W(B) =$\sum_{u\in U',v\in V'}w(u,v)$ is maximal. Naturally, the amount of conditions in the biclique is bounded by d. We define N(v) as the neighborhood of v. Formally, N(v) ={u ∈ U |(v,u)∈ E}. The following algorithm can then be used to find the maximum biclique in $O(n^{2d})$:

6.2.3 ALGORITHM:-

```
1. MaxBoundBiClique(U,V,E,d):
2. Initialize a hash table weight;
3. weightbest ← 0 For all v ∈ V do For all S ⊆ N(v)
4.          do weights[S] ← weights[S]+ max{0,w(S,{v})}
5.          If (weight[S] > weightbest)
  6.                 Ubest ← S weightbest ← weight[S]
7. Compute Vbest =Tu∈Ubest N(u)
8. Output (Ubest,Vbest)
```

The iterative stage of this algorithm, attempts to find the best scoring condition group. This is done by scanning, for each vertex v ∈ V , all O(n^{2d}) subsets of its neighbors (Note that this means we do not scan all possible condition groups). We then update the score for the optimal biclique for this condition group by adding v to the biclique (if this improves the score). When the iterative stage is done, each tested condition group has the best biclique score associated with it, and we know the best overall biclique score. Finally, we compute the group of genes Vbest participating in the biclique with Ubest by taking only vertices neighboring to all conditions in the group Ubest. Note that the iteration over subsets of N(v) is done by repeatedly changing the current subset S by adding or removing a single element, updating w(S,{v}) in constant time.

6.2.4. FINDING MAXIMUM BOUNDED BIPARTITE SUB-GRAPH:-

Given a bipartite graph **G = (U,V,E)** and a weight function w, find a sub-graph **B =(U',V',E')**such that w(B) will be maximal. A weight function that assigned +1 for edges and−1 for non-edges will be used here, but, the same reasoning can be expanded for a general weight function. Formally, given a bipartite graph **G = (U,V,E)** assume that the weights are +1 for an edge **(u,v)∈ E** and -1 for an edge **(u,v)/∈ E.**

Lemma 3 For an optimal bicluster **B = (U',V',E')**each condition **u∈U'** is adjacent to at least **|V '|/2** genes.

Proof: If there is such condition u that is adjacent to less than half the genes, then it contributes more edges of weight -1 than +1 and therefore has a total negative contribution to the score. It can therefore be removed and heavier sub-graph will be formed, contradicting B's optimality.

Lemma 4 For an optimal bicluster **B = (U',V',E')** each gene **v ∈ V '**is adjacent to at least **|U'|/2** conditions.

Proof: The same argument as the proof of the previous Lemma.
Lemma 5 For an optimal bicluster **B = (U',V',E'), |U'|≤2d**
Proof: This is an immediate result of the previous Lemma: If each gene is adjacent to at least |U'|/2 conditions and each gene has a bounded neighborhood of d conditions then **|U'|≤2d.**
Lemma 6 For any subset **X⊆U'** there exist a gene **v∈V'** such that v is adjacent to at least|X|/2 members of X.
Corollary 1
For an optimal bicluster **B=(U',V',E'), U'** can be covered with at most **log(2d)** sets each contained in the neighborhood of some **v∈V'.** This allows an $O\left(\mathrm{n}2^{\mathrm{d}})^{\log(2\mathrm{d})}\right)$ algorithm that can be generalized to produce k heaviest biclusters by storing the k heaviest in a heap (and adding a multiplicative factor of log(k)).

6.2.5. ALGORITHMIC ANALYSIS:-

Although the algorithm described above is polynomial in n, it is still infeasible with today's computational power, due to the fact that d is too large. The SAMBA algorithm attempts to address this problem. SAMBA proceeds in three phases. First, the model bipartite graph is formed and the weights of vertex pairs are computed. Second, several heavy subgraphs are sought around each vertex of the graph. This is done by starting with good seeds around the vertex and expanding them using local search. The seeds are found using the hashing technique of the MaxBoundedBiClique algorithm, for all subsets of neighbours of size 4 -6 (effectively restricting the algorithm complexity). The local improvement procedure iteratively applies the best modification to the current bicluster (addition or deletion of a single vertex) until no score improvement is possible. To avoid similar biclusters whose vertex sets differ only slightly, a final step greedily filters similar biclusters with more than some threshold overlap (usually 20%). The following summarizes the second phase:
1. Find heavy bicliques using the algorithm, while restricting to 4-6 neighbors of each gene.
2. Greedy expansion of heaviest bicliques to contain each gene/condition.
3. filter overlapping biclusters, this is done to avoid similar biclusters whose vertex set differ only slightly.

6.3 SPECTRAL BI-CLUSTERING:-

The **SpectralBiclustering** algorithm assumes that the input data matrix has a hidden checkerboard structure. The rows and columns of a matrix with this structure may be partitioned so that the entries of any bicluster in the Cartesian product of row clusters and column clusters are approximately constant. For instance, if there are two row partitions and three column partitions, each row will belong to three biclusters, and each column will belong to two biclusters.
The algorithm partitions the rows and columns of a matrix so that a corresponding block wise-constant checkerboard matrix provides a good approximation to the original matrix.

6.3.1 MATHEMATICAL FORMULATION:-

The input matrix A is first normalized to make the checkerboard pattern more obvious. There are three possible methods:

1.*Independent row and column normalization*, as in Spectral Co-Clustering. This method makes the rows sum to a constant and the columns sum to a different constant.

2.Bistochastization: repeated row and column normalization until convergence. This method makes both rows and columns sum to the same constant.

3.Log normalization: the log of the data matrix is computed: $L = \log A$. Then the column mean $\overline{L_{i\cdot}}$, row mean $\overline{L_{\cdot j}}$, and overall mean $\overline{L_{\cdot\cdot}}$ of L are computed. The final matrix is computed according to the formula:

$$K_{ij} = L_{ij} - \overline{L_{i\cdot}} - \overline{L_{\cdot j}} + \overline{L_{\cdot\cdot}}$$

After normalizing, the first few singular vectors are computed, just as in the Spectral Co-Clustering algorithm.

If log normalization was used, all the singular vectors are meaningful. However, if independent normalization or bistochastization were used, the first singular vectors, u_1 and v_1. Arediscarded. From now on, the "first" singular vectors refer to $u_2 \ldots u_{p+1}$ and $v_2 \ldots v_{p+1}$ except in the case of log normalization.

Given these singular vectors, they are ranked according to which can be best approximated by a piecewise-constant vector. The approximations for each vector are found using one-dimensional k-means and scored using the Euclidean distance. Some subset of the best left and right singular vector is selected. Next, the data is projected to this best subset of singular vectors and clustered.

For instance, if ***p*** singular vectors were calculated, the ***q*** best are found as described, where ***q<p*** Let ***U***be the matrix with columns the best left singular vectors, and similarly ***V*** for the right. To partition the rows,the rows of ***A*** are projected to a ***q*** dimensional space:***A*V***Treating the ***m*** rows of this ***m×q***matrix as samples and clustering using k-means yields the row labels. Similarly, projecting the columns to $A^T * \mathbf{U}$ and clustering $\mathbf{n \times q}$this matrix yields the column labels.

6.3.2. APPLICATIONS:-

COCLUSTERING GENES AND COMDITIONS:-

Global analyses of RNA expression levels are useful for classifying genes and overall phenotypes. Often these classification problems are linked, and one wants to find "marker genes" that are differentially expressed in particular sets of "conditions." We have developed a method that simultaneously clusters genes and conditions, finding distinctive "checkerboard" patterns in matrices of gene expression data, if they exist. In a cancer context, these checkerboards correspond to genes that are markedly up- or downregulated in patients with particular types of tumours. Our method, spectral biclustering, is based on the observation that checkerboard structures in matrices of expression data can be found in eigenvectors corresponding to characteristic expression patterns across genes or conditions. In addition, these eigenvectors can be readily identified by commonly used linear algebra approaches, in particular the singular value decomposition (SVD), coupled with closely integrated normalization steps. We present a number of variants of the approach, depending on whether the normalization over genes and conditions is done independently or in a coupled fashion. We then apply spectral biclustering to a selection of publicly available cancer expression data sets, and examine the degree to which the approach is able to identify checkerboard structures. Furthermore, we compare the performance of our biclustering methods against a number of reasonable benchmarks.

6.4. BIMAX ALGORITHM:-

The focus of this post is BiMax, a method developed to serve as a baseline in a comparison of biclustering algorithms.BecauseBiMax was developed as a reference method, its objective is intentionally simple: it finds all biclusters consisting entirely of 1s in a binary matrix.

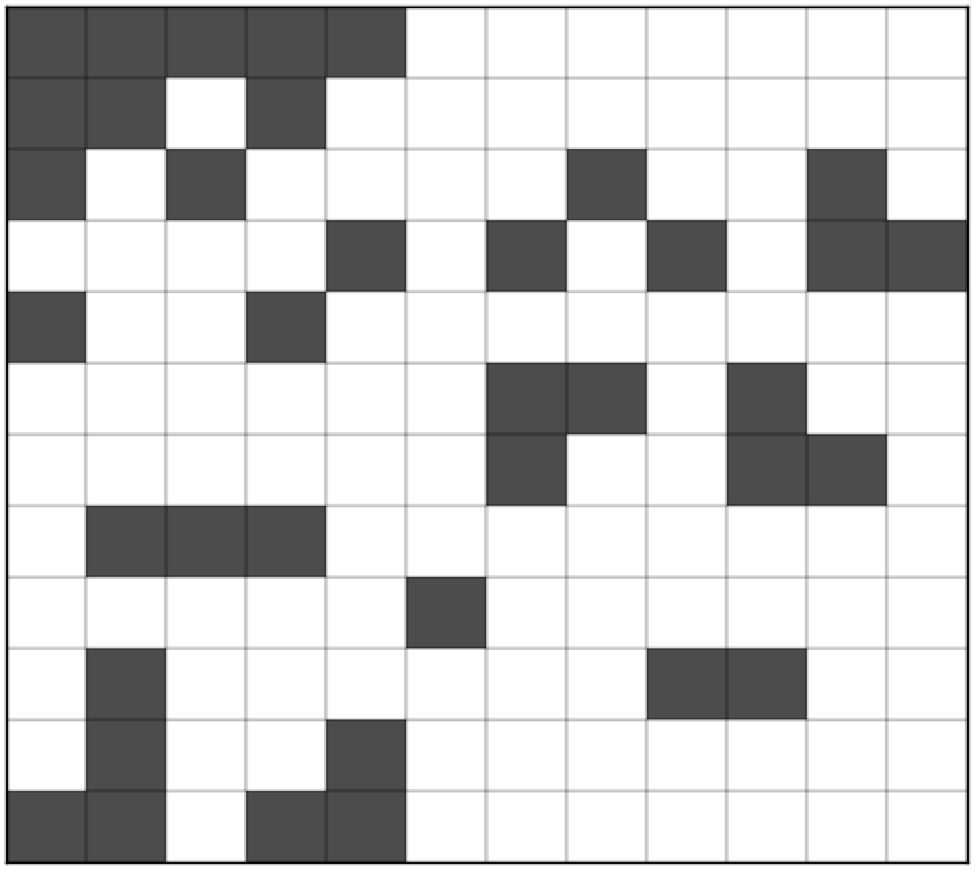
Specifically, BiMax enumerates all inclusion-maximal biclusters, which are biclusters of all 1s to which no row or column can be added without introducing 0s. More formally, an inclusion-maximal bicluster of m×n binary matrix M is a set of rows and set of columns (R,C)(R,C), with $R\in 2^{\{1,\dots,m\}}$R∈2{1,...,m} and $C\in 2^{\{1,\dots,n\}}$C∈2{1,...,n} such that:

- $\forall i\in R,\forall j\in C: M[i,j]=1$∀i∈R,∀j∈C:M[i,j]=1
- Forany other bicluster (R',C') that meets the first condition, $(R\subseteq R'\wedge C\subseteq C')\Longrightarrow(R=R'\wedge C=C')$(R⊆R'∧C⊆C')⟹(R=R'∧C=C')

BiMax only works with binary matrices, but of course non-binary data can be converted to binary data in a number of ways. The simplest method is to threshold it at some threshold t , where t could be the mean, median, some other quantile, or any arbitrary threshold. Of course, more sophisticated binarizing methods may also be used.Now let us see how the algorithm works.

6.4.1. METHODS:-

BiMax uses a recursive divide and conquer strategy to enumerate all the biclusters in a m×nmatrix M . To illustrate the process we will use the following data matrix, taken from the original paper:



The process begins by choosing any row containing a mixture of 0s and 1s. In this case, all the rows fits this criterion. (If no such row exists, then clearly either all entries of M are 1, in which case the entire matrix is a single bicluster, or all its entries are 0, in which case it has no biclusters.) We arbitrarily choose the first row. The chosen row **r∗** will be used to divide MM into two submatrices, each of which can be processed independently.

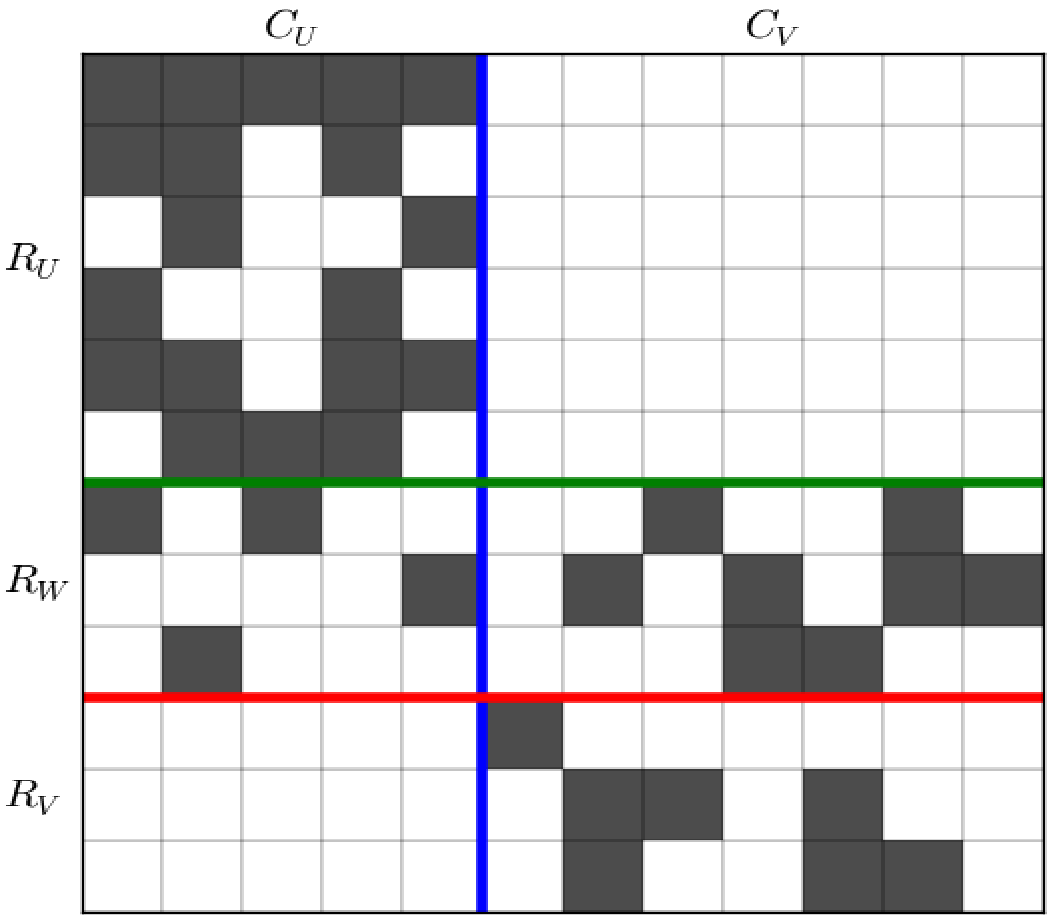
To find this submatrices, first the columns C={1,…,n}C={1,…,n} are divided into two sets: those for which row **r∗**is 1, and those for which it is 0.

- C_U={c:M[r∗,c]=1}
- C_V **= C -** C_U

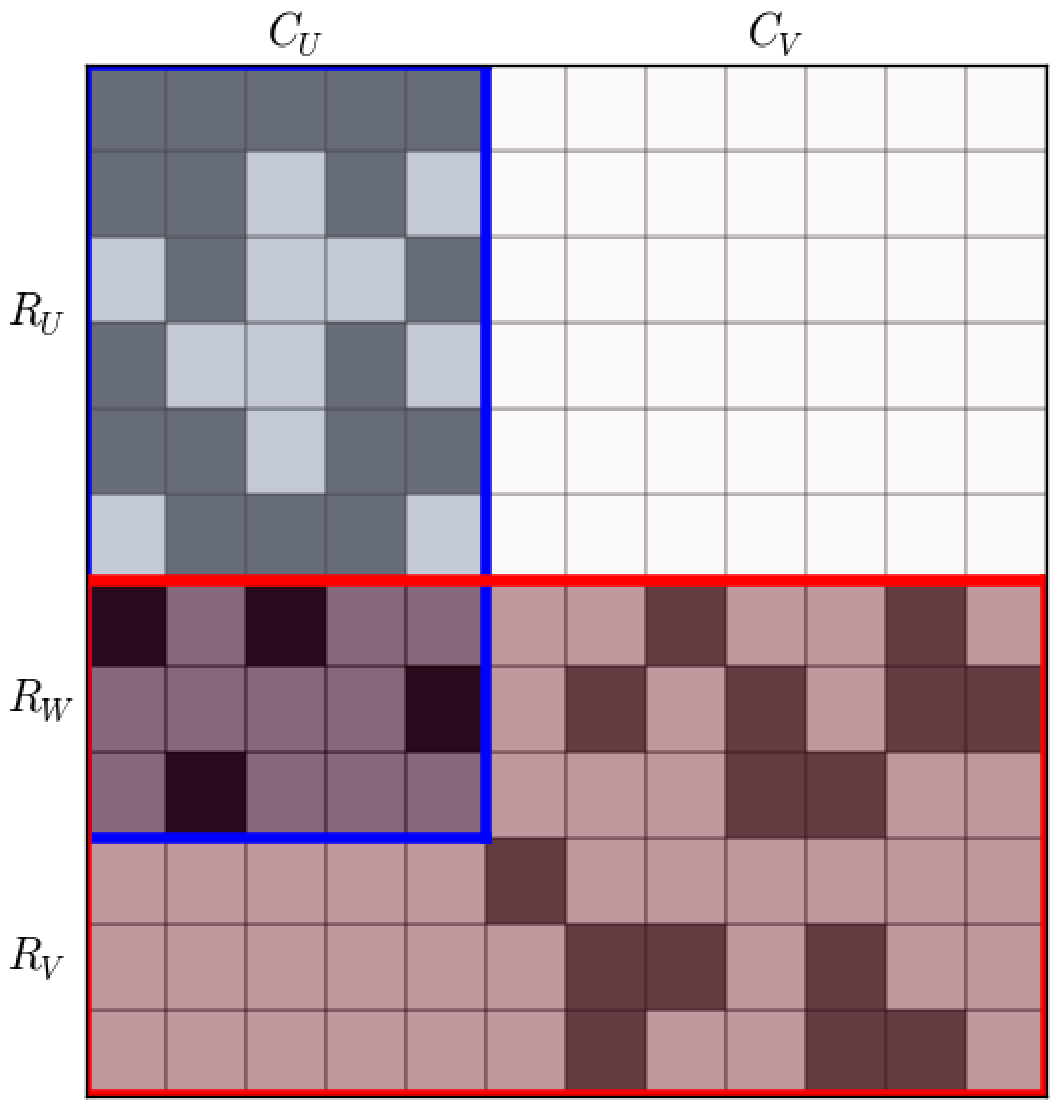
Next the m rows of M are divided into three sets:

- R_U : rows with 1s only in C_U
- R_W : rows with 1s in both $C_U$and C_V
- R_V: rows with 1s only in C_V

After rearranging the rows and columns of M to make each set contiguous, the matrix looks like this:



This image makes it clear how to divide the matrix for the recursive step. The submatrix formed by $(R_U,C_V)$is empty and cannot contain any biclusters. The submatrix U=($R_U \cup R_W$,C_U), shown in blue in the next figure, and the submatrix V=($R_W \cup R_V$,$C_U \cup C_V$), shown in red, contain all possible biclusters in M. The procedure continues by recursively processing U and then V.



There is one problem yet to address: because Uand V overlap, it is possible that a bicluster in U has some rows in V. Then that part of the bicluster would be a maximal bicluster in V but not in M. To avoid this error, BiMax requires that any bicluster found in Vmust contain at least one column from each column set in Z, where Zis the set of all C_V column sets in the current call stack.

6.4.2. INCREMENTAL ALGORITHM:-

The incremental procedure, see below, is based on work by Alexe et al., 2002, who propose a method to find all inclusion-maximal cliques in general graphs. Shortly summarized, each node in the input graph is visited, and all maximal cliques are found that contain that node. A visit-to-a-node operation comprises iteration through all other nodes of the graph as well, and each newly found bicluster is globally extended to its maximality. For the special class of bipartite graphs we are dealing with, it is important to notice that several steps of the above method are redundant: it suffices to iterate through only one partition of the graph nodes—in matrix terminology this means we will have to iterate either through the set of rows or columns, but not both. Moreover, extending new biclusters can be avoided with a guarantee that no bicluster will be missed this way.

```
1: procedure IncrementalAlgorithm(E)
2: M ← ∅
3: for i ← 1 to n do
4:     C* ←{ j |eij =1∧ 1 ≤ j ≤ m}
5:    for each (G,C) ∈ M do
6:       C’← C ∩C*
7:        if ∃(G”,C”) ∈ M with C”= C’ then
8:           M ← M \{ (G”,C”)}∪{ (G”∪{i},C”)}
9:        else
10:          M ← M ∪{ (G” ∪{ i},C’)}
11       end if
12:    end for
13:   if∃(G”,C”) ∈ M with C” = C* then
14:     M ← M ∪{ ({i},C*)}
15   : end if
16:   end for
```

```
17: return M
18: end procedure
```

6.5. X-MOFIT ALGORITHM:-

Gene expression plays an important role in cell differentiation, development, and pathological behaviour. DNA microarrays1,2 offer biologists the remarkable ability to monitor the expression levels of thousands of genes in a cell simultaneously. High-throughput gene expression analysis promises to produce new insights into cell function as well as stimulate the development of new therapies and diagnostics. When a gene's expression level is measured across a variety of samples, the expression values usually span a wide range. Biologically, these values correspond to a small number of distinct states that the gene is in, e.g., upregulated or down-regulated. Since the up-regulation of a gene is a temporal process, it is often difficult to determine a gene's state based on a set of noisy expression values. However, on average, it might be possible to differentiate the expression level of an up-regulated gene in one tissue type (e.g.,a cancer tissue) from its level in another type of tissue (e.g., a healthy tissue) where the gene is not expressed with the same abundance. Motivated by this observation, we say that a gene's expression level is conserved across a subset of samples if the gene is in the same state in each of the samples in this subset. A conserved gene expression motif or xmotif is a subset of genes whose expression is simultaneously conserved for a subset of samples; we say that each of these samples matches the motif. In this paper, we use a range of expression values to represent a gene's state. If we map each gene to a dimension, each sample to a point, and each expression value to a coordinate value, an xmotif is identical to a multi-dimensional hyper rectangle that is bounded in the dimensions corresponding to the conserved genes in the motif and unbounded in the other dimensions. Mathematically, the expression values of a sample that matches a motif satisfy a conjunction of inequalities, two for each gene in the motif.

In this work, we address the task of identifying xmotifs in gene expression data. We concentrate on data sets where each sample belongs to a particular class, e.g., different types of cancer,3 cancerous and healthy tissues,4 and patients with different survival rates.5 We believe that this work is the one of the first to suggest representing gene expression data concisely in the form of xmotifs. Such a representation has several potential biological advantages and applications. First, by comparing and contrasting the gene motifs for different classes, we can identify genes that are conserved in multiple classes but are in different states in different classes. If the classes correspond to different diseases or to diseased and normal tissues, such genes are possible drug targets. Second, if the genes in a motif are believed to

interact in a pathway, the information present in the motif about which genesare highly <
expressed and which are suppressed could be used to deduce and refine the structure of the <
pathway. Third, byrequiring that multiple genes be simultaneouslyconserved across the <
samples matchinga motif, we might be able tocharacterise sub-classes inthe data that no <
gene on its own provides high-quality evidence for. Before attempting to develop an algorithm<
for computing xmotifs, it isuseful to consider thepropertiesthat anxmotif should have. <
Computingone motif for eachsample makes the representationover-specific. Therefore, we <
desire that each motif for a class should be matched by a large fraction of the samples in that <
class, if not all the samples in that class. Each motif should contain as many conserved genes <
as possible. While a motif that contains one or two genes is biologically feasible, it may not be <
statisticallysignificant sincesuch a motif could appear with high probabilityin randomly-
generated data. On the other hand, a motif that contains too many genes may be too restrictive <
sinceno samplemaymatch themotif. Motivated bytheseobservations, weproposethe <
following formal definition of an xmotif:

6.5.1 DEFINITION:-

Given a set of genes whose expression levels are measure dacross a set <
of samples anduser-definedparameters 0< α, β< 1, aconservedgene expressionmotif or <
xmotif is a pair (C, G), where Cis a subset of the samples and Gis a subset of the genes, that <
satisfies the following conditions:

Size: the number of samples in C is at least an α-fraction of all the samples,

Conservation: every gene in G is conserved across all the samples in C, i.e., the gene is in the same state in all the samples in C, and

Maximality: for every gene not in G, the gene is conserved in at most a β-fraction of the samples in C.

The maximality condition enforces a balance between the number of genes in the motif and the number of samples matching the motif. If we add a gene to the motif, then the number of samples matching the new motif will decrease by a fraction of at least β, a cost we may not be willing to pay. Given this definition, the gene expression data may contain many xmotifs. Among all xmotifs, we are interested in the largest xmotif, the one that contains the maximum number of conserved genes. In order to cover all the classes completely using xmotifs, we adopt the following iterative algorithm: find the largest xmotif in the data, remove the samples that satisfy this motif from the data, find the largest motif in the remaining data, and continue in this manner until all samples satisfy some motif. Our approach has several desirable features. (i) We allow a gene to appear in more than one motif and in motifs representing different classes, modelling the possibility that the gene's expression level may be regulated in multiple conditions. (ii) By not deleting the samples that match a computed xmotif, we can allow samples to appear in different motifs. This property may be useful when a sample belongs to multiple classes or when we are interested in discovering new classifications of the samples. (iii) The system need not be told beforehand how many motifs to compute.a (iv) Using this approach, we can find xmotifs with vastly different numbers of genes; we are likely to discover motifs with many genes in earlier iterations and motifs with fewer genes in later iterations.

6.5.2. ALGORITHM:-

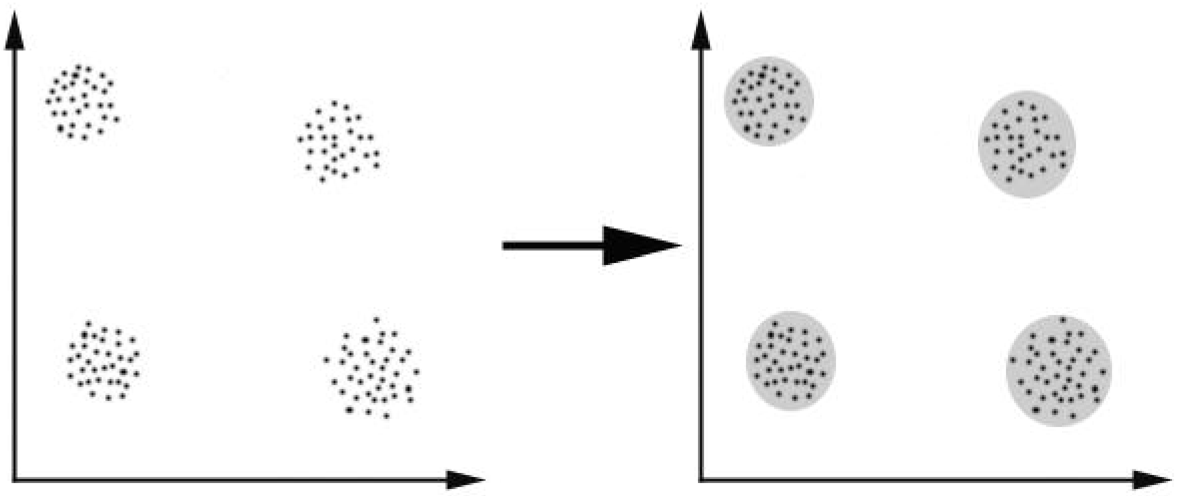
We are now ready to describe our algorithm for computing the largest xmotif. The input to the algorithm is a set of genes, a set of samples, an expression value for each gene -sample pair, and for each gene, a list of intervals representing the states in which the gene is expressed in the samples. To determine an xmotif, we have to compute the set G of conserved genes, the states that these genes are in, and the set C of samples that match the motif. We observe that if we are given (i) the set G of conserved genes, (ii) the states of the conserved genes, and (iii) one sample c that matches this motif, we can compute the remaining samples in C simply by checking for each sample c’ whether the genes in G are in the same state in c and c’. Informally, c is a “seed” from which we can compute the entire motif. This observation is the starting point of our algorithm. Suppose we know a sample c that matches the largest xmotif. Given such a sample c, how can we compute the genes in the largest motif and the states they are in? Suppose we are given a set D of samples with the following properties: (i) for every sample c’ in D and for every gene in the largest motif, there is exactly one state such that the gene is in that state in samples c and c’and (ii) for every gene g that is not in the largest motif, there exists a sample c’ in D such that gene g is not in the same state in samples c and c’. We call D a discriminating set. The key property of a discriminating set is that given the seed sample c and such a set D, we can compute the largest xmotif by including exactly those gene-states that satisfy these conditions and exactly those samples that agree with c on all these gene-states. Algorithm 1 describes the steps of our probabilistic algorithm. We assume that for each gene, the intervals corresponding to that gene’s states are disjoint.c It proceeds by selecting ns samples uniformly at random from the set of all samples. These samples act as seeds. For each random seed, we select n_d sets of samples uniformly at random from the set of all samples; each set has s_d elements. These sets serve as candidates for the discriminating set.

For each seed-discriminating set pair, we compute the corresponding xmotif as explained above. We discard the motif if less than an α-fraction of the samples match it. Finally, we return the motif that contains the largest number of genes. Suppose our data consists of n samples and m genes. We can extend the arguments made by Procopiuc et al. to prove that by choosing **n_s= O(1/α), s_d=O(logm/log(1/β)),** and **n_d=O(1/α^{s_d}),** we can compute the largest xmotif with probability greater than 1/2 in time **O(n$n_s n_d$)=**$nm^{O(\log(1/\alpha)/\log(1/\beta)}$**.** We can increase the probability of success by repeatedly executing this algorithm.

```
1: for i = 1 to $n_s$ do
2:   Choose a sample c uniformly at random.
3:   for j = 1 to $n_d$ do
4:     Choose a subset D of the samples of size $s_d$ uniformly at random.
5:     for each gene g, if g is in the state s in c and all the samples in D,
       include  the pair (g, s) in the set$G_{i,j}$.
6:       $C_{i,j}$. = set of samples that agree with c in all the gene-states in $G_{i,j}$.
7:       Discard ($C_{i,j}$ , $G_{i,j}$) if $C_{i,j}$ contains less than αn samples.
8:     return the motif (C*, G*) that maximizes |$G_{i,j}$|, 1 ≤ i ≤ $n_s$, 1 ≤ j ≤ $n_d$.
```

CLUSTERING:

Clustering involves the grouping of similar objects into a set known as cluster. Objects in one cluster are likely to be different when compared to objects grouped under another cluster. Clustering is one of the main tasks in exploratory data mining and is also a technique used in statistical data analysis. While clustering is not one specific algorithm, it is a general task that can be solved by means of several algorithms. Some of the popular clustering methods that are used include hierarchical, partitioning, density-based and model-based.



Clustering is the act of creating various clusters that have all objects under the data set. Further, clustering can be distinguished into hard and soft clustering. Under hard clustering, an object either belongs to a cluster or it does not. However, with soft clustering (fuzzy clustering) an object can belong to many clusters. The ultimate aim of clustering is to intrinsically group unlabeled data. It finds applications in market research, pattern recognition, data mining and analysis, data compression, image recognition and more.

The concept of a cluster cannot be easily defined, and this is largely why several algorithms are available for clustering. These algorithms differ in their properties, and therefore, researchers are known to apply different cluster models based on the data set in question and also what it is intended to be used for. For example, hierarchical clustering is based on distance connectivity, while distribution models are based on statistical distributions

APPLICATIONS:

1. News clustering
2. Gene expression clustering
3. Marketing
4. Clustering weblogs data to discover groups of similar pattern
5. Recognize communities in social network

Various clustering algorithm:

1. KMeans or partitinal clustering algorithm:-

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to *k*, *k*-means clustering gives a formal definition as an optimization problem: find the *k* cluster centers and assign the objects to the nearest cluster center, such that the squared distances from the cluster are minimized.

The optimization problem itself is known to be NP-hard, and thus the common approach is to search only for approximate solutions. A particularly well known approximate method is Lloyd's algorithm often just referred to as "*k-means algorithm*" (although another algorithm introduced this name). It does however only find a local optimum, and is commonly run multiple times with different random initializations. Variations of *k*-means often include such optimizations as choosing the best of multiple runs, but also restricting the centroids to members of the data set (*k*-medoids), choosing medians (*k*-medians clustering), choosing the initial centers less randomly (*k*-means++) or allowing a fuzzy cluster assignment (fuzzy c-means).

Most *k*-means-type algorithms require the number of clusters – *k* – to be specified in advance, which is considered to be one of the biggest drawbacks of these algorithms. Furthermore, the algorithms prefer clusters of approximately similar size, as they will always assign an object to the nearest centroid. This often leads to incorrectly cut borders of clusters (which is not surprising since the algorithm optimizes cluster centers, not cluster borders).

K-means has a number of interesting theoretical properties. First, it partitions the data space into a structure known as a Voronoi diagram.

TYPES OF CLUSTERING:-

1. PARTITIONING:constructs various partitions and evaluate them by some criterion.

2.HIERARCHICAL:create a hierarchical decomposition of the set ob object using some criterion.

3.MODEL BASED:hypothesize a model for each cluster and find best fit models to data.

4.DENSITY BASED:guided by connectivity and density function

5.GRAPH THEORATIC CLUSTERING

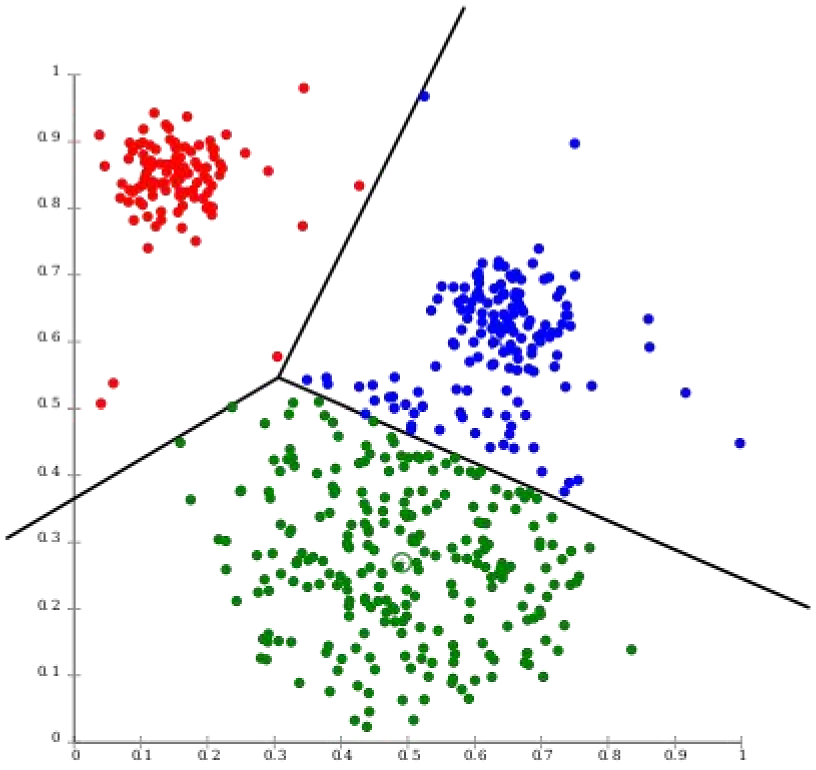
CHARACTERISTICS OF CLUSTERING

The quality of depends on the algorithm,the distance function,and the application.

A distance fucntion is a similarity function which measures the quality of how similar two objects are.eg:-Euclidian,Minkowski distance.

To define the quality of cluster we define two measures:
1. intra clsuter distance-The distance between points between the cluster,which should be maximized.
2. Inter cluster distance-The distance between points in the cluster which should be minimized.

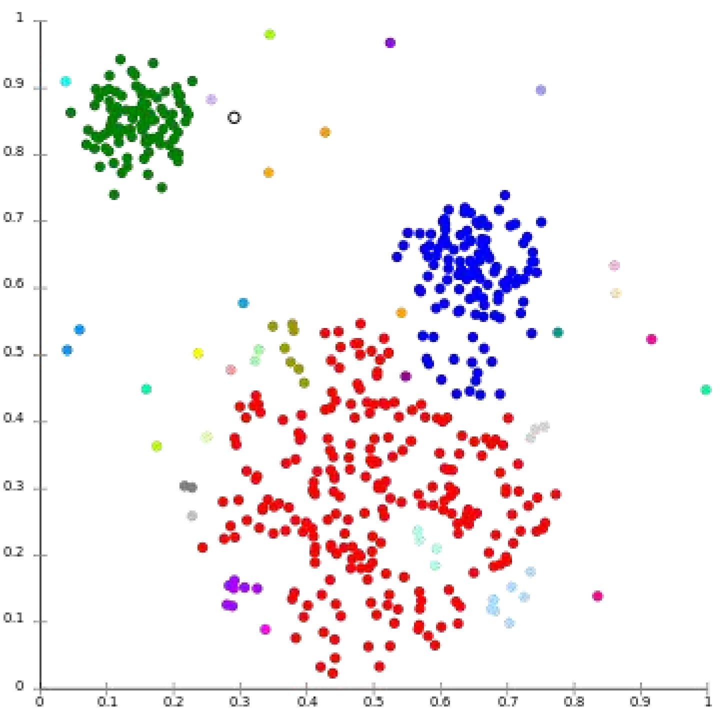
Second, it is conceptually close to nearest neighbor classification, and as such is popular in machine learning. Third, it can be seen as a variation of model based clustering.



2.Hierarchical clustering:-

Connectivity-based clustering, also known as *hierarchical clustering*, is based on the core idea of objects being more related to nearby objects than to objects farther away. These algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using a dendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix.

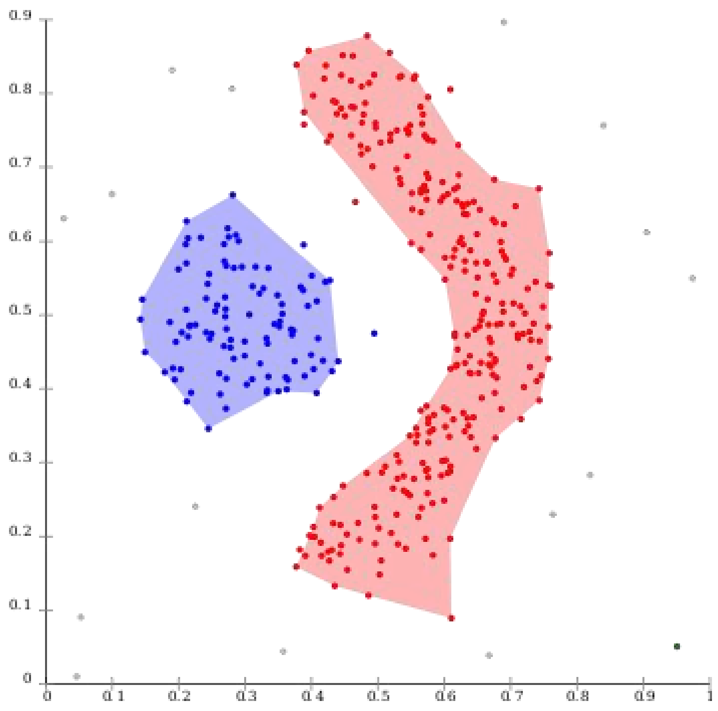
Connectivity-based clustering is a whole family of methods that differ by the way distances are computed. Apart from the usual choice of distance functions, the user also needs to decide on the linkage criterion (since a cluster consists of multiple objects, there are multiple candidates to compute the distance) to use. Popular choices are known as single-linkage clustering (the minimum of object distances), complete linkage clustering (the maximum of object distances) or UPGMA ("Unweighted Pair Group Method with Arithmetic Mean", also known as average linkage clustering). Furthermore, hierarchical clustering can be agglomerative (starting with single elements and aggregating them into clusters) or divisive (starting with the complete data set and dividing it into partitions).



2. Density based(DBSCAN):-

In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points.

The most popular density based clustering method is DBSCAN.In contrast to many newer methods, it features a well-defined cluster model called "density-reachability". Similar to linkage based clustering, it is based on connecting points within certain distance thresholds. However, it only connects points that satisfy a density criterion, in the original variant defined as a minimum number of other objects within this radius. A cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods) plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low – it requires a linear number of range queries on the database – and that it will discover essentially the same results (it is deterministic for core and noise points, but not for border points) in each run, therefore there is no need to run it multiple times.



WORK DONE:-

Clustering-

1.We took a gene expression dataset from UCL machine learning library.

2.Performed feature scaling.

3.As it is a large dataset,there are so many independent column we can't cluster the data using simple algorithm in 2D.

4.We use dimensionality reduction algorithm(PCA) to reduce the dimension into 55732*2.

5.After that we perform KMeans,Hierarchical algorithm on the dataset.

Biclustering-

1.We took a binary dataset.

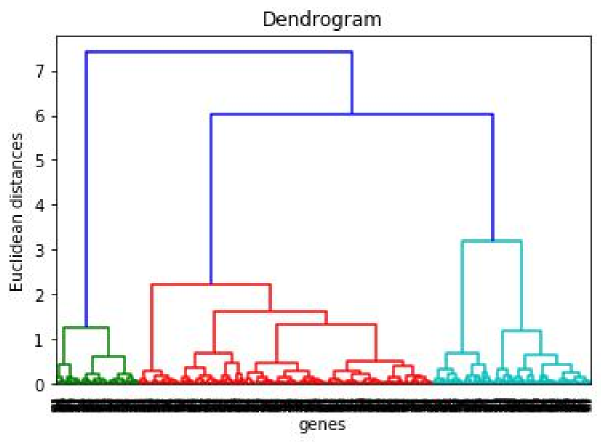
2.In that dataset we took two rows at a time as a tuple and the perform AND operation on it.

3.After that we store the result as a tuple and find the total number of bi-cluster using the result.

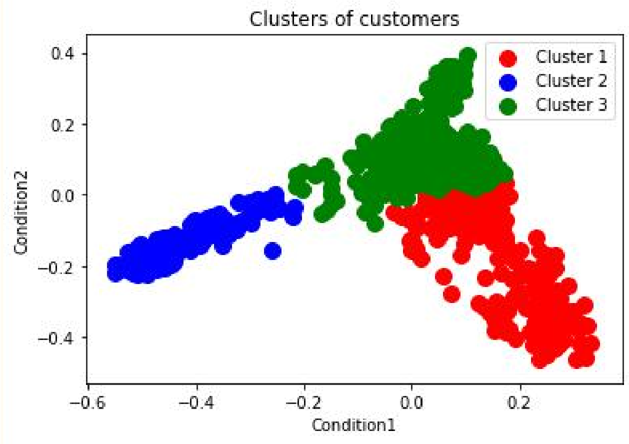
4.we search for the pattern same as the result throughout the dataset.

KMEANS ALGORITHM IMPLEMENTATION:-

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('data.csv')
X = dataset.iloc[:,1:].values

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 2, kernel = 'rbf')
X = kpca.fit_transform(X)

from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Condition1')
plt.ylabel('Condition2')
plt.legend()
plt.show()

from sklearn import metrics
metrics.silhouette_score(X,y_kmeans,metric='euclidean',random_state=0,sample_size=None)
```



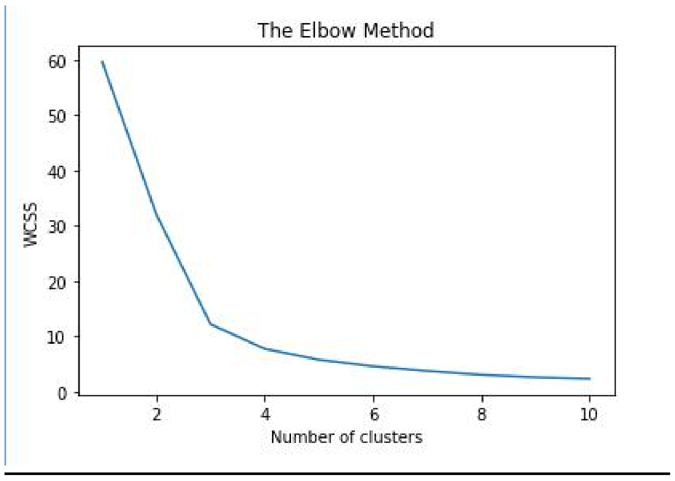
We found the optimal number of cluster using the dendogram.



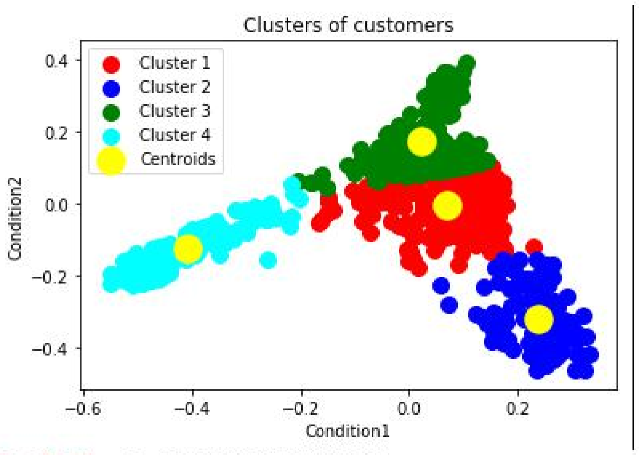
```
Out[15]: 0.535799135709756
```

We implemented the hierarchical clustering and its result is shown avbove along with the silhouette co-efficient.

OUTPUT:-



Using elbow method we found the optimal number of clusters in Kmeans clustering.



```
Out[12]: 0.4931073491607943
```

The result of the Kmeans clustering algorithm and its centroids.The silhouette co-efficient.

HIERARCHICAL CLUSETRING IMPLEMENTATION:-

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('data.csv')
X = dataset.iloc[:,1:].values

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 2, kernel = 'rbf')
X = kpca.fit_transform(X)

import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()

from sklearn.cluster import AgglomerativeClustering
hc=AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)

plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.title('Clusters of customers')
plt.xlabel("Condition1")
plt.ylabel('Condition2')
plt.legend()
plt.show()

from sklearn import metrics
metrics.silhouette_score(X,y_hc,metric='euclidean',random_state=0,sample_size=None)
```

BI-CLUSTERING ALGORITHM:

```
import numpy as np
import pandas as pd

dataset=pd.read_csv("matrix_match_score.csv")
x=dataset.iloc[:,1:].values

h={}
for i in range(x.shape[0]):
        for j in range(i+1,x.shape[0]):
                l=[]
                col=[]
                c1=np.array(x[i])
                c2=np.array(x[j])
                c3=c1 & c2
                for i in range(len(c3)):
                        if (c3[i]==1):
                                col.append(i)
                if col==[]:
                        continue
                else:
                        for k in range(x.shape[0]):
                                c4=np.array(x[k])
                                if ((c3==c4).all()):
                                        l.append(k)
                                        continue
                        if(len(l)!=0):
                                l=tuple(l)
                                h[l]=col
print("the bi-clusters and its corresponding columns are:")
for keys,values in sorted(h.items()):
        print("{} : {}".format(keys,tuple(values)))
        print("\n")
```

CONCLUSION:-

We have presented a comprehensive survey of the classification algorithm developed in the field of biclustering and clustering and its application. The list of applications presented is by no means exhaustive, and an all-inclusive list of potential applications would be prohibitively long.From our project we have learned that in field of gene expression finding biclustering produce better result than clustering because in case of biclustering it search through the data row and column simouttaneously .But in case of clustering it search only through the row or column.

.

So we can conclude that biclustering and clustering,both are useful for bioinformatics and many other application.

BIBLIOGRAPHY:-

- Tariq Rashid: “Clustering” http://www.cs.bris.ac.uk/home/tr1690/documentation/fuzzy_clustering_initial_report/node11.html

- Osmar R. Zaïane: “Principles of Knowledge Discovery in Databases - Chapter 8: Data Clustering” http://www.cs.ualberta.ca/~zaiane/courses/cmput690/slides/Chapter8/index.html

- Pier Luca Lanzi: “Ingegneria della Conoscenza e Sistemi Esperti – Lezione 2: Apprendimento non supervisionato” http://www.elet.polimi.it/upload/lanzi/corsi/icse/2002/Lezione%202%20-%20Apprendimento%20non%20supervisionato.pdf

- G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. PNAS, 97(22):12079–12084, 2000.

- A. Ben-Dor, Z. Yakhini, and R. Shamir. Clustering gene expression patterns. Journal of Computational Biology, 6:281–297, 1999.

- Y. Cheng and G.M. Church. Biclustering of expression data. In Proc, ISMB’00 AAAI Press:93–103, 2000.