DATAMINING USING NONLINEAR CLUSTERING ALGORITHM

INTRODUCTION ON DATAMINING

1.INTRODUCTION

1.1.DATAMINING

Data mining is a technique that discovers previously unknown relationships in data.Data mining is the practice of automatically searching large stores of data to discover patterns and trends that go beyond simple analysis. Data mining uses sophisticated mathematical algorithms to segment the data and to predict the likelihood of future events based on past events. Data mining is also known as Knowledge Discovery in Data (KDD).

The key properties of data mining are:

- Automatic discovery of patterns
- Prediction of likely outcomes
- Creation of actionable information
- Focus on large data sets and databases

Data mining can answer questions that cannot be addressed through simple query and reporting techniques.

Automatic Discovery

Data mining is performed by a model that uses an algorithm to act on a set of data. Data mining models can be used to mine the data on which they are built, but most types of models are generalizable to new data. The process of applying a model to new data is known as scoring.

Prediction

Many forms of data mining are predictive. For example, a model might predict income based on education and other demographic factors. Predictions have an associated probability (How likely is this prediction to be true?). Prediction probabilities are also known as confidence (How confident can I be of this prediction?).

Some forms of predictive data mining generate rules, which are conditions that imply a given outcome. For example, a rule might specify that a person who has a bachelor's degree and lives

in a certain neighborhood is likely to have an income greater than the regional average. Rules have an associated support (What percentage of the population satisfies the rule?).

Grouping

Other forms of data mining identify natural groupings in the data. For example, a model might identify the segment of the population that has an income within a specified range, that has a good driving record, and that leases a new car on a yearly basis.

Actionable Information

Data mining can derive actionable information from large volumes of data. For example, a town planner might use a model that predicts income based on demographics to develop a plan for low-income housing. A car leasing agency might use a model that identifies customer segments to design a promotion targeting high-value customers.

Data Mining and Data Warehousing

Data can be mined whether it is stored in flat files, spreadsheets, database tables, or some other storage format. The important criteria for the data is not the storage format, but its applicability to the problem to be solved.

Proper data cleansing and preparation are very important for data mining, and a data warehouse can facilitate these activities. However, a data warehouse will be of no use if it does not contain the data you need to solve your problem.

1.2 WHAT CAN DATAMINING DO?

Data mining is a powerful tool that can help you find patterns and relationship within your data. But data mining does not work by itself. It does not eliminate the need to know your business, to understand your data, or to understand analytical methods. Data mining discovers hidden information in your data, but it cannot tell you the value of the information to your organization.

You might already be aware of important patterns as a result of working with your data over time. Data mining can confirm or qualify such empirical observations in addition to finding new patterns that may not be immediately discernible through simple observation.

It is important to remember that the predictive relationships discovered through data mining are not causal relationships. For example, data mining might determine that males with incomes between $50,000 and $65,000 who subscribe to certain magazines are likely to buy a given product. You can use this information to help you develop a marketing strategy. However, you should not assume that the population identified through data mining will buy the product because they belong to this population.

Data mining yields probabilities, not exact answers. It is important to keep in mind that rare events can happen; they just do not happen very often.

Understanding Your Data

To ensure meaningful data mining results, you must understand your data. Data mining algorithms are often sensitive to specific characteristics of the data: outliers (data values that are very different from the typical values in your database), irrelevant columns, columns that vary together (such as age and date of birth), data coding, and data that you choose to include or exclude. Data Mining can automatically perform much of the data preparation required by the algorithm. But some of the data preparation is typically specific to the domain or the data mining problem. At any rate, you need to understand the data that was used to build the model in order to properly interpret the results when the model is applied.

1.3 DATAMINING PROCESS

This illustrates the phases, and the iterative nature, of a data mining project. The process flow shows that a data mining project does not stop when a particular solution is deployed. The results of data mining trigger new business questions, which in turn can be used to develop more focused models. The following figure shows datamining process:

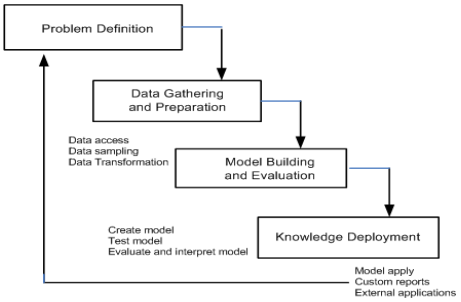


Fig 1: Illustration of phases and iterative nature of a data mining project

Problem Definition

This initial phase of a data mining project focuses on understanding the project objectives and requirements. Once you have specified the problem from a business perspective, you can formulate it as a data mining problem and develop a preliminary implementation plan**.**

For example, your business problem might be: "How can I sell more of my product to customers?" You might translate this into a data mining problem such as: "Which customers are most likely to purchase the product?" A model that predicts who is most likely to purchase the product must be built on data that describes the customers who have purchased the product in the past. Before building the model, you must assemble the data that is likely to contain relationships between customers who have purchased the product and customers who have not purchased the product. Customer attributes might include age, number of children, years of residence, owners/renters, and so on.

Data Gathering and Preparation

The data understanding phase involves data collection and exploration. As you take a closer look at the data, you can determine how well it addresses the business problem. You might decide to remove some of the data or add additional data. This is also the time to identify data quality problems and to scan for patterns in the data.

The data preparation phase covers all the tasks involved in creating the table or view that you will use to build the model. Data preparation tasks are likely to be performed multiple times, and not in any prescribed order. Tasks may include column selection and the creation of views, as well as data cleansing and transformation. For example, you might transform adate_of_birth column to age; you might insert the median income in cases where the income column is null.

Model Building and Evaluation

In this phase, you select and apply various modeling techniques and calibrate the parameters to optimal values. If the algorithm requires data transformations, you will need to step back to the previous phase to implement them.In preliminary model building, it often makes sense to work with a reduced set of data since the final data set might contain thousands or millions of rows.

Knowledge Deployment

Knowledge deployment is the use of data mining within a target environment. In the deployment phase, insight and actionable information can be derived from data.

Deployment can involve scoring (the application of models to new data), the extraction of model details (for example the rules of a decision tree), or the integration of data mining models within applications, data warehouse infrastructure, or query and reporting tools.

1.4 Data Mining Applications

Data mining is highly useful in the following domains −

- Market Analysis and Management
- Corporate Analysis & Risk Management
- Fraud Detection

Apart from these, data mining can also be used in the areas of production control, customer retention, science exploration, sports, astrology, and Internet Web Surf-Aid.

Market Analysis and Management

Listed below are the various fields of market where data mining is used −

- **Customer Profiling** − Data mining helps determine what kind of people buy what kind of products.
- **Identifying Customer Requirements** − Data mining helps in identifying the best products for different customers. It uses prediction to find the factors that may attract new customers.
- **Cross Market Analysis** − Data mining performs association/correlations between product sales.
- **Target Marketing** − Data mining helps to find clusters of model customers who share the same characteristics such as interests, spending habits, income, etc.
- **Determining Customer purchasing pattern** − Data mining helps in determining customer purchasing pattern.

- **Providing Summary Information** − Data mining provides us various multidimensional summary reports.

Corporate Analysis and Risk Management

Data mining is used in the following fields of the Corporate Sector −

- **Finance Planning and Asset Evaluation** − It involves cash flow analysis and prediction, contingent claim analysis to evaluate assets.
- **Resource Planning** − It involves summarizing and comparing the resources and spending.
- **Competition**− It involves monitoring competitors and market directions.

Fraud Detection

Data mining is also used in the fields of credit card services and telecommunication to detect frauds. In fraud telephone calls, it helps to find the destination of the call, duration of the call, time of the day or week, etc. It also analyzes the patterns that deviate from expected norm.

Clustering Analysis

2.1 **CLUSTERING**

Clustering analysis finds clusters of data objects that are similar in some sense to one another. The members of a cluster are more like each other than they are like members of other clusters. Different clusters can have members in common. The goal of clustering analysis is to find high-quality clusters such that the inter-cluster similarity is low and the intra-cluster similarity is high.

Clustering, like classification, is used to segment the data. Unlike classification, clustering models segment data into groups that were not previously defined. Classification models segment data by assigning it to previously-defined classes, which are specified in a target. Clustering models do not use a target.

Clustering is useful for exploring data. If there are many cases and no obvious groupings, clustering algorithms can be used to find natural groupings.

Clustering can serve as a useful data-preprocessing step to identify homogeneous groups on which to build supervised models.

Clustering can also be used for anomaly detection. Once the data has been segmented into clusters, you might find that some cases do not fit well into any clusters. These cases are anomalies or outliers.

2.1.1 How are Clusters Computed?

There are several different approaches to the computation of clusters. Oracle Data Mining supports the following methods:

- **Density-based** — This type of clustering finds the underlying distribution of the data and estimates how areas of high density in the data correspond to peaks in the distribution. High-density areas are interpreted as clusters. Density-based cluster estimation is probabilistic.
- **Distance-based** — This type of clustering uses a distance metric to determine similarity between data objects. The distance metric measures the distance between actual cases in the cluster and the prototypical case for the cluster. The prototypical case is known as the centroid.

- **Grid-based** — This type of clustering divides the input space into hyper-rectangular cells and identifies adjacent high-density cells to form clusters.

Scoring New Data

Although clustering is an unsupervised mining function, Oracle Data Mining supports the scoring operation for clustering. New data is scored probabilistically.

Support and Confidence

Support and confidence are metrics that describe the relationships between clustering rules and cases. Support is the percentage of cases for which the rule holds. Confidence is the probability that a case described by this rule will actually be assigned to the cluster.

Evaluating a Clustering Model

The choice of a suitable clustering algorithm and of a suitable measure for the evaluation depends on the clustering objects and the clustering task.

Since known classes are not used in clustering, the interpretation of clusters can present difficulties. How do you know if the clusters can reliably be used for business decision making.

Data Mining clustering models support a high degree of model transparency. You can evaluate the model by examining information generated by the clustering algorithm: for example, the centroid of a distance-based cluster.

Moreover, because the clustering process is hierarchical, you can evaluate the rules and other information related to each cluster's position in the hierarchy.

The goal of clustering analysis is to find high-quality clusters such that the inter-cluster similarity is low and the intra-cluster similarity is high.

Clustering evaluation is a complex task and many different approaches have been proposed over the years.

Most approaches incorporate measures of compactness and separation between the proposed clusters using clustering quality indexes

2.1.2 CLUSTERING APPROACHES AND ANALYSIS

For clustering algorithm to be advantageous and beneficial some of the conditions need to be satisfied.

1) **Scalability** - Data must be scalable otherwise we may get the wrong result.

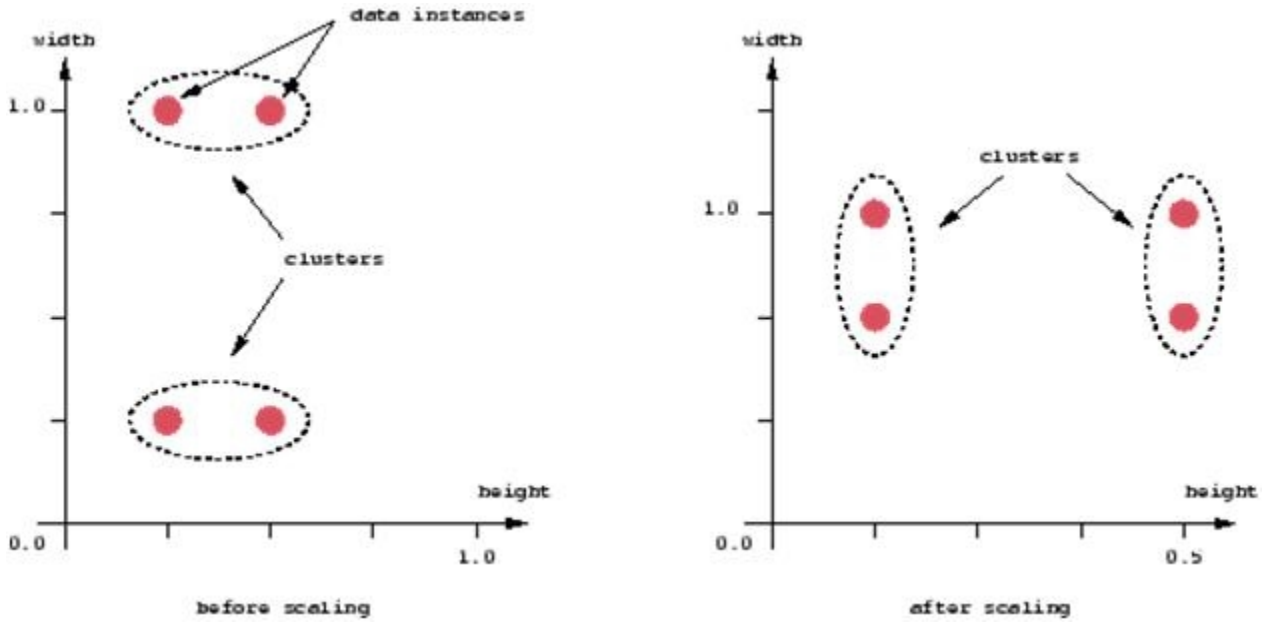


Fig 2 : Example showing where scalability may lead to wrong result

2) Clustering algorithm must be able to deal with different types of attributes.

3) Clustering algorithm must be able to find clustered data with the arbitrary shape.

4) Interpret-ability and Usability - Result obtained must be interpretable and usable.

5) Clustering algorithm must be able to deal with data set of high dimensionality.

Clustering Methods

Clustering methods can be classified into the following categories −

- Partitioning Method
- Hierarchical Method
- Density-based Method
- Grid-Based Method

- Model-Based Method

Partitioning Method

Suppose we are given a database of 'n' objects and the partitioning method constructs 'k' partition of data. Each partition will represent a cluster and $k \leq n$. It means that it will classify the data into k groups, which satisfy the following requirements −

- Each group contains at least one object.
- Each object must belong to exactly one group.

Points to remember −

- For a given number of partitions (say k), the partitioning method will create an initial partitioning.
- Then it uses the iterative relocation technique to improve the partitioning by moving objects from one group to other.

Hierarchical Methods

This method creates a hierarchical decomposition of the given set of data objects. We can classify hierarchical methods on the basis of how the hierarchical decomposition is formed. There are two approaches here −

- Agglomerative Approach
- Divisive Approach

Agglomerative Approach

This approach is also known as the bottom-up approach. In this, we start with each object forming a separate group. It keeps on merging the objects or groups that are close to one another. It keep on doing so until all of the groups are merged into one or until the termination condition holds.

Divisive Approach

This approach is also known as the top-down approach. In this, we start with all of the objects in the same cluster. In the continuous iteration, a cluster is split up into smaller clusters. It is down until each object in one cluster or the terminationnever be undone.

Approaches to Improve Quality of Hierarchical Clustering

Here are the two approaches that are used to improve the quality of hierarchical clustering −

- Perform careful analysis of object linkages at each hierarchical partitioning.
- Integrate hierarchical agglomeration by first using a hierarchical agglomerative algorithm to group objects into micro-clusters, and then performing macro-clustering on the micro-clusters.

Density-based Method

This method is based on the notion of density. The basic idea is to continue growing the given cluster as long as the density in the neighborhood exceeds some threshold, i.e., for each data point within a given cluster, the radius of a given cluster has to contain at least a minimum number of points.

Grid-based Method

In this, the objects together form a grid. The object space is quantized into finite number of cells that form a grid structure.

Advantage

- The major advantage of this method is fast processing time.
- It is dependent only on the number of cells in each dimension in the quantized space.

Model-based methods

In this method, a model is hypothesized for each cluster to find the best fit of data for a given model. This method locates the clusters by clustering the density function. It reflects spatial distribution of the data points.

This method also provides a way to automatically determine the number of clusters based on standard statistics, taking outlier or noise into account. It therefore yields robust clustering methods.

Constraint-based Method

In this method, the clustering is performed by the incorporation of user or application-oriented constraints. A constraint refers to the user expectation or the properties of desired clustering results. Constraints provide us with an interactive way of communication with the clustering process. Constraints can be specified by the user or the application requirement.

Clustering Analysis

WE ARE living in a world full of data. Every day, people encounter a large amount of information and store or represent it as data, for further analysis and management. One of the vital means in dealing with these data is to classify or group them into a set of categories or clusters. Actually, as one of the most primitive activities of human beings , classification plays an important and indispensable role in the long history of human development. In order to learn a new object or understand a new phenomenon, people always try to seek the features that can describe it, and further compare it with other known objects or phenomena, based on the similarity or dissimilarity, generalized as proximity, according to some certain standards or rules. “Basically, classification systems are either supervised or unsupervised, depending on whether they assign new inputs to one of a finite number of discrete supervised classes or unsupervised categories ,respectively.

In supervised classification ,the mapping from a set of input data vectors (where is the input space dimensionality), to a finite set of discrete class labels (where is the total number of class types), is modeled in terms of some mathematical function , where is a vector of adjustable parameters. The values of these parameters are determined (optimized) by an inductive learning algorithm (also termed inducer), whose aim is to minimize an empirical risk functional (related to an inductive principle) on a finite data set of input–output examples, where is the finite cardinality of the available representative dataset.

In unsupervised classification, called clustering or exploratory data analysis, no labeled data are available . The goal of clustering is to separate a finite unlabeled data set into a

finite and discrete set of "natural," hidden data structures, rather than provide an accurate characterization of unobserved samples generated from the same probability distribution . This can make the task of clustering fall outside of the framework of unsupervised predictive learning problems, such as vector quantization, probability density function estimation , and entropy maximization . It is noteworthy that clustering differs from multidimensional scaling (perceptual maps), whose goal is to depict all the evaluated objects in a way that minimizes the topographical distortion while using as few dimensions as possible. Also note that, in practice, many (predictive) vector quantizers are also used for (nonpredictive) clustering analysis .

Nonpredictive clustering is a subjective process in nature, which precludes an absolute judgment as to the relative efficacy of all clustering techniques . As pointed out by Backer and Jain , "in cluster analysis a group of objects is split up into a number of more or less homogeneous subgroups on the basis of an often subjectively chosen measure of similarity (i.e., chosen subjectively based on its ability to create "interesting" clusters), such that the similarity between objects within a subgroup is larger than the similarity between objects belonging to different subgroups". Clustering algorithms partition data into a certain number of clusters (groups, subsets, or categories). There is no universally agreed upon definition .

Most researchers describe a cluster by considering the internal homogeneity and the external separation, i.e., patterns in the same cluster should be similar to each other, while patterns in different clusters should not. Both the similarity and the dissimilarity should be examinable in a clear and meaningful way. Here, we give some simple mathematical descriptions of several types of clustering, based on the descriptions in . Given a set of input patterns , where and each measure is said to be a feature (attribute, dimension, or variable). • (Hard) partitional clustering attempts to seek a -partition .

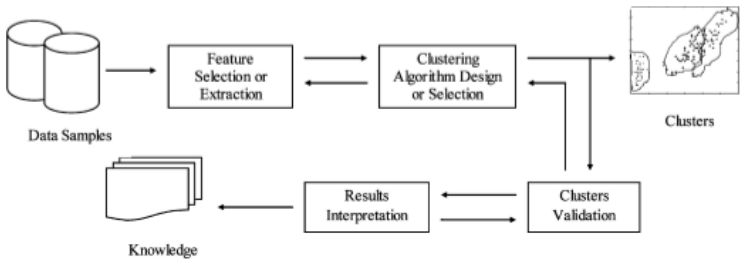


Fig.3 depicts the procedure of cluster analysis with four basic steps.

Feature selection or extraction

As pointed out by Jain et al. and Bishop , feature selection chooses distinguishing features from a set of candidates, while feature extraction utilizes some transformations to generate useful and novel features from the original ones. Both are very crucial to the effectiveness of clustering applications. Elegant selection of features can greatly decrease the workload and simplify the subsequent design process.Generally,ideal features should be of use in distinguishing patterns belonging to different clusters, immune to noise, easy to extract and interpret. We elaborate the discussion on feature extraction , in the context of data visualization and dimensionality reduction. More information on feature selection can be found.

Clustering algorithm design or selection

The step is usually combined with the selection of a corresponding proximity measure and the construction of a criterion function. Patterns are grouped according to whether they resemble each other. Obviously, the proximity measure directly affects the formation of the resulting clusters. Almost all clustering algorithms are explicitly or implicitly connected to some definition of proximity measure. Some algorithms even work directly on the proximity matrix, as defined . Once a proximity measure is chosen, the construction of a clustering criterion function makes the partition of clusters an optimization problem,which is welldefined mathematically, and has rich solutions in the literature.

Clustering is ubiquitous, and a wealth of clustering algorithms has been developed to solve different problems in specific fields. However, there is no clustering algorithm that can be universally used to solve all problems.“It has been very difficult to develop a unified framework for reasoning about it (clustering) at a technical level, and profoundly diverse approaches to clustering”, as proved through an impossibility theorem. Therefore,it is important to carefully investigate the characteristics of the problem at hand, in order to select or design an appropriate clustering strategy.

Cluster validation

Given a data set, each clustering algorithm can always generate a division, no matter whether the structure exists or not. Moreover, different approaches usually lead to different clusters; and even for the same algorithm, parameter identification or the presentation

order of input patterns may affect the final results. Therefore, effective evaluation standards and criteria are important to provide the users with a degree of confidence for the clustering results derived from the used algorithms.

These assessments should be objective and have no preferences to any algorithm. Also, they should be useful for answering questions like how many clusters are hidden in the data, whether the clusters obtained are meaningful or just an artifact of the algorithms, or why we choose some algorithm instead of another. Generally, there are three categories of testing criteria: external indices, internal indices, and relative indices. These are defined on three types of clustering structures, known as partitional clustering, hierarchical clustering, and individual clusters. Tests for the situation, where no clustering structure exists in the data, are also considered , but seldom used, since users are confident of the presence of clusters. External indices are based on some prespecified structure, which is the reflection of prior information on the data, and used as a standard to validate the clustering solutions. Internal tests are not dependent on external information (prior knowledge).

On the contrary, they examine the clustering structure directly from the original data. Relative criteria place the emphasis onthe comparison of different clustering structures, in order to provide a reference, to decide which one may best reveal the characteristics of the objects.We will not survey the topic in depth and refer interested readers to. However, we will cover more details on how to determine the number of clusters in Section II-M. Some more recent discussion can be found in. Approaches for fuzzy clustering validity are reported.

Results interpretation

The ultimate goal of clustering is to provide users with meaningful insights from the original data, so that they can effectively solve the problems encountered. Experts in there levant fields interpret the data partition .Further analyzes, even experiments, may be required to guarantee the reliability of extracted knowledge.

Clustering has been applied in a wide variety of fields, ranging from engineering (machine learning, artificial intelligence, pattern recognition, mechanical engineering, electrical engineering), computer sciences (web mining, spatial database analysis, textual document collection, image segmentation), life and medical sciences (genetics, biology, microbiology,

paleontology, psychiatry, clinic, pathology), to earth sciences (geography. geology, remote sensing), social sciences (sociology, psychology, archeology, education), and economics (marketing, business).

Accordingly, clustering is also known as numerical taxonomy, learning without a teacher (or unsupervised learning), typological analysis and partition. The diversity reflects the important position of clustering in scientific research. On the other hand, it causes confusion, due to the differing terminologies and goals. Clustering algorithms developed to solve a particular problem, in a specialized field, usually make assumptions in favor of the application of interest. These biases inevitably affect performance in other problems that do not satisfy these premises. For example, the -means algorithm is based on the Euclidean measure and, hence, tends to

COMPUTATIONAL COMPLEXITY OF CLUSTERING ALGORITHMS

Cluster algorithm	Complexity	Capability of tackling high dimensional data
K-means	$O(NKd)$ (time) $O(N+K)$ (space)	No
Fuzzy c-means	Near $O(N)$	No
Hierarchical clustering*	$O(N^2)$ (time) $O(N^2)$ (space)	No
CLARA	$O(K(40+K)^2+K(N-K))^*$ (time)	No
CLARANS	Quadratic in total performance	No
BIRCH	$O(N)$ (time)	No
DBSCAN	$O(N \log N)$ (time)	No
CURE	$O(N_{sample}^2 \log N_{sample})$ (time) $O(N_{sample})$ (space)	Yes
WaveCluster	$O(N)$ (time)	No
DENCLUE	$O(N \log N)$ (time)	Yes
FC	$O(N)$ (time)	Yes
CLIQUE	Linear with the number of objects, Quadratic with the number of dimensions	Yes
OptiGrid	Between $O(Nd)$ and $O(Nd \log N)$	Yes
ORCLUS	$O(K_0^3+K_0Nd+K_0^2d^3)$ (time) $O(K_0d^2)$ (space)	Yes

FIG 4. Computational complexity of clustering algorithms

2.2. CLUSTERING ALGORITHMS

Clustering algorithms can be broadly classified into two categories:

1. Unsupervised linear clustering algorithms

2. Unsupervised non-linear clustering algorithms

Unsupervised linear clustering algorithms:

1) K-means clustering algorithm
2) Fuzzy c-means clustering algorithm
3) Hierarchical clustering algorithm
4) Gaussian(EM) clustering algorithm
5) Quality threshold clustering algorithm

Unsupervised non-linear clustering algorithms:

1) MST based clustering algorithm
2) Kernel k-means clustering algorithm
3) Density based clustering algorithm

2.2.1 LINEAR CLUSTERING ALGORITHMS:

k-means clustering algorithm

k-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest

new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{C} \sum_{j=1}^{c} (||x - v||)$$

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.

'c' is the number of data points in i^{th} cluster.

'c' is the number of cluster centers.

Algorithmic steps for k-means clustering

Let $X = \{x_1, x_2, x_3, \ldots\ldots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \ldots\ldots, v_c\}$ be the set of centers.

1) Randomly select *'c'* cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers..

4) Recalculate the new cluster center using:

$$v = (1/c) \sum_{j=1}^{c} x$$

where, *'c_i'* represents the number of data points in i^{th} cluster.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3).

Advantages

1) Fast, robust and easier to understand.

2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.

3) Gives best result when data set are distinct or well separated from each other.

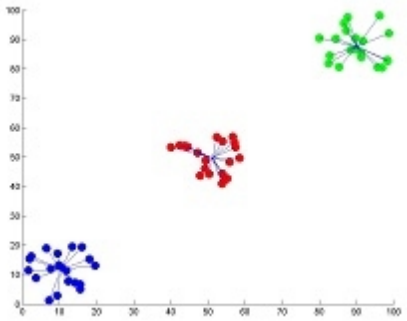


Fig 5: Showing the result of k-means for *'N'* = 60 and *'c'* = 3

Disadvantages

1) The learning algorithm requires apriori specification of the number of cluster centers.

2) The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.

3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).

4) Euclidean distance measures can unequally weight underlying factors.

5) The learning algorithm provides the local optima of the squared error function.

6) Randomly choosing of the cluster center cannot lead us to the fruitful result.

7) Applicable only when mean is defined i.e. fails for categorical data.

8) Unable to handle noisy data and outliers.

9) Algorithm fails for non-linear data set.

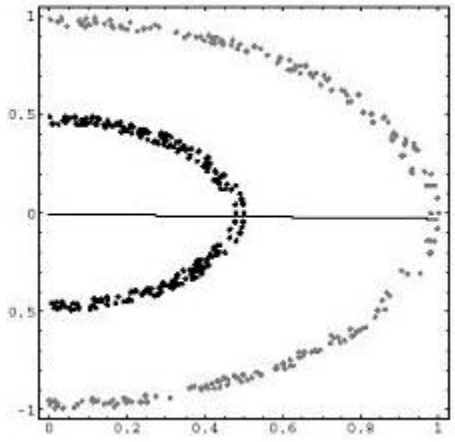


Fig 6: Showing the non-linear data set where k-means algorithm fails

A major drawback to k-means is that it cannot separate clusters that are non-linearly separable in input space. Two recent approaches have emerged for tackling such a problem. One is kernel k-means, where, before clustering, points are mapped to a higher-dimensional feature space using a nonlinear function, and then kernel k-means partitions the points by linear separators in the new space. The other approach is spectral clustering algorithms, which use the eigenvectors of an affinity matrix to obtain a clustering of the data. A popular objective function used in spectral clustering is to minimize the normalized cut .

On the surface, kernel k-means and spectral clustering appear to be completely different approaches. In this paper we first unite these two forms of clustering under a single framework. By generalizing the k-means objective function to use both weights and kernels, we show how the two approaches to clustering are related. Specifically, we can rewrite the weighted kernel k-means objective function as a trace maximization problem whose relaxation can be solved with eigenvectors. The result shows how a particular kernel and weight scheme is connected to the spectral algorithm of Ng, Jordan, and Weiss .

However, the advantage to our approach is that we can generalize the clustering algorithm to use arbitrary kernels and weights. Further, we show that by choosing the weights in

particular ways, the weighted kernel k-means objective function is identical to the normalized cut. Thus far, only eigenvectorbased algorithms have been employed to minimize normalized cuts in spectral clustering and image segmentation. However, software to compute eigenvectors of large sparse matrices (often based on the Lanczos algorithm) can have substantial computational overheads, especially when a large number of eigenvectors are to be computed. In such situations, our equivalence has an important implication: we can use k-means-like iterative algorithms for directly minimizing the normalized-cut of a graph.

We show the usefulness of our approach to the application of clustering gene expression data by applying a quadratic kernel (squared correlation) to obtain anti-correlated gene clusters and we illustrate the scalability of our algorithms in terms of computation time by applying it to a large handwriting recognition data set. A word about notation. Capital letters such as A,X,Y Polynomial Kernel κ(a,b) = (a·b + c)d Gaussian Kernel κ(a,b) = exp(−||a−b||2/2σ2) Sigmoid Kernel κ(a,b) = tanh(c(a·b) + θ) and Φ denote matrices; lower-case bold letters such as a,b denote column vectors; script letters such as A,B,V,E represent sets; ||a|| denotes the L2 norm of a vector; and ||X||F denotes the Frobenius norm of a matrix, and is given by ||X||F = (Pi,j X2) THE ESSENTIALS In this section, we summarize the seemingly different approaches of weighted kernel k-means and spectral clustering.

Weighted Kernel k-means The k-means clustering algorithm can be enhanced by the use of a kernel function; by using an appropriate nonlinear mapping from the original (input) space to a higher dimensional feature space, one can extract clusters that are non-linearly separable in input space. Furthermore, we can generalize the kernel k-means algorithm by introducing a weight for each point a, denoted by w(a). As we shall see later, this generalization is powerful and encompasses the normalized cut of a graph. Let us denote clusters by πj, and a partitioning of points as {πj}k j=1. Using the non-linear function φ, the objective function of weighted kernel k-means is defined as:

D({πj}k j=1) =k X j=1X a ∈ πj

w(a) kφ(a)−mjk where mj = Pb ∈πj w(b)φ(b) Pb ∈πj w(b) .Note that mj is the “best” cluster representative since mj = argminzX a∈πj w(a)kφ(a)−zk. The Euclidean distance from φ(a) to center mj is given by φ(a)−Pb ∈πj w(b)φ(b) Pb ∈πj w(b) = φ(a)·φ(a)− Pb ∈πj w(b)φ(a)·φ(b). The

dot products $\varphi(a)\cdot\varphi(b)$ are computed using kernel function κ , and are contained in the kernel matrix K. All computation is in the form of such inner products, hence we can replace all inner products by entries of the kernel matrix. The weighted kernel k-means algorithm shares many properties of standard k-means; for example, the objective function value defined in monotonically decreases with each iteration. Assuming we are able to store the whole anity matrix in main memory, we can analyze the time complexity of Algorithm 1. It is clear that the bottleneck is , i.e., the computation of distances. The first term in , $\varphi(a)\cdot\varphi(a)$,

Algorithm 1:

Weighted Kernel k-means.

Weighted Kernel kmeans(K, k, w, C1,...,Ck)

Input: K: kernel matrix,

k: number of clusters,

w: weights for each point

Output: C1,....,Ck: partitioning of the points

1. Initialize the k clusters: $C(0)_1$,...,$C(0)_k$.

2. Set t = 0.

3. For each point a, find its new cluster index as $j*(a) = \text{argmin}_{jk}\varphi(a)-mjk2$, using (2).

4. Compute the updated clusters as $Ct+1\ j = \{a : j*(a) = j\}$.

5. If not converged, set t = t + 1 and go to Step 3;

Otherwise, stop.

Fuzzy c-means clustering algorithm

This algorithm works by assigning membership to each data point corresponding to each cluster center on the basis of distance between the cluster center and the data point. More the data is near to the cluster center more is its membership towards the particular cluster center. Clearly, summation of membership of each data point should be equal to one. After each iteration

membership and cluster centers are updated according to the formula

$$\mu ij = {}^{1}\Big/_{\sum_{k=1}^{c}({}^{dij}\Big/_{dik})} \qquad vj = (\sum_{i=1}^{n}(\mu ij)^{m} xi))$$

Where

'n'is the number of data points.

'vj' represents the j^{th} cluster center.

'm' is the fuzziness index m € [1, ∞].

'c' represents the number of cluster center.

'µij' represents the membership of i^{th} data to j^{th} cluster center.

'dij' represents the Euclidean distance between i^{th} data andj^{th} cluster center.

Main objective of fuzzy c-means algorithm is to minimize:

$$j(u, v) = \sum_{i=1}^{n} (\mu ij)^{m} \, ||xi - vj||^{2}$$

where,

$'||x_i - v_j||'$ is the Euclidean distance betweeni^{th} data and j^{th} cluster center.

Algorithmic steps for Fuzzy c-means clustering

Let X = {x_1, x_2, x_3 ..., x_n} be the set of data points and V = {v_1, v_2, v_3 ..., v_c} be the set of centers.

1) Randomly select *'c'* cluster centers.

2) Calculate the fuzzy membership $'\mu_{ij}'$ using

$$\mu ij = {}^{1}\Big/_{\sum_{k=1}^{c}({}^{dij}\Big/_{dik})}$$

1) Compute the fuzzy centers $'v_j'$ using:

$$vj = (\sum_{i=1}^{n}(\mu ij)^m xi)) \ , \quad \text{vj} = 1,2,\ldots..\text{c}$$

Repeat step 2) and 3) until the minimum *'J'* value is achieved or $||U^{(k+1)} - U^{(k)}|| < \beta$.

'k' is the iteration step.

'J' is the objective function.

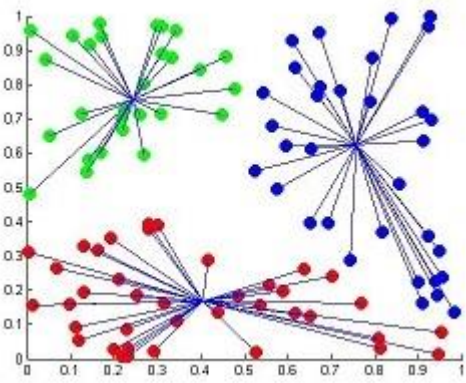


Fig 7: Result of Fuzzy c-means clustering

Advantages

1) Gives best result for overlapped data set and comparatively better then k-means algorithm.
2) Unlike k-means where data point must exclusively belong to one cluster center here data point is assigned membership to each cluster center as a result of which data point may belong to more.

Disadvantages

1) With lower value of β we get the better result but at the expense of more number of iteration.
2) Euclidean distance measures can unequally weight underlying factors.

Hierarchical clustering algorithm

Document clustering is widely applicable in areas such as search engines, web mining, information retrieval, and topological analysis. Most document clustering methods perform several preprocessing steps including stop words removal and stemming on the document set. Each document is represented by a vector of frequencies of remaining terms within the

document. Some document clustering algorithms employ an extra preprocessing step that divides the actual term frequency by the overall frequency of the term in the entire document set. The idea is that if a term is too common across different documents, it has little discriminating power (Rijsbergen, 1979). Although many clustering algorithms have been proposed in the literature, most of them do not satisfy the special requirements for clustering documents.

Hierarchical clustering algorithm is of two types:

i) Agglomerative Hierarchical clustering algorithm or AGNES (agglomerative nesting) and

ii) Divisive Hierarchical clustering algorithm or DIANA (divisive analysis).

Both this algorithm are exactly reverse of each other. So we will be covering Agglomerative Hierarchical clustering algorithm in detail.

Agglomerative Hierarchical clustering –

This algorithm works by grouping the data one by one on the basis of the nearest distance measure of all the pairwise distance between the data point. Again distance between the data point is recalculated but which distance to consider when the groups has been formed? For this there are many available methods. Some of them are:

1) single-nearest distance or single linkage.

2) complete-farthest distance or complete linkage.

3) average-average distance or average linkage.

4) centroid distance.

5) ward's method - sum of squared euclidean distance is minimized.

This way we go on grouping the data until one cluster is formed. Now on the basis of dendogram graph we can calculate how many number of clusters should be actually present.

Algorithmic steps for Agglomerative Hierarchical clustering

Let $X = \{x_1, x_2, x_3, ..., x_n\}$ be the set of data points.

1) Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.

2) Find the least distance pair of clusters in the current clustering, say pair (r), (s), according to d[(r),(s)] = min d[(i),(j)] where the minimum is over all pairs of clusters in the current clustering.

3) Increment the sequence number: m = m +1.Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to L(m) = d[(r),(s)].

4) Update the distance matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and column corresponding to the newly formed cluster. The distance between the new cluster, denoted (r,s) and old cluster(k) is defined in this way: d[(k), (r,s)] = min (d[(k),(r)], d[(k),(s)]).

5) If all the data points are in one cluster then stop, else repeat from step 2).

Divisive Hierarchical clustering - It is just the reverse of Agglomerative Hierarchical approach.

Advantages

1) No apriori information about the number of clusters required.

2) Easy to implement and gives best result in some cases.

Disadvantages

1) Algorithm can never undo what was done previously.

2) Time complexity of at least O($n^2 \log n$) is required, where *'n'* is the number of data points.

3) Based on the type of distance matrix chosen for merging different algorithms can suffer with one or more of the following:

i) Sensitivity to noise and outliers.

ii) Breaking large clusters.

iii) Difficulty handling different sized clusters and convex shapes.

4) No objective function is directly minimized.

5) Sometimes it is difficult to identify the correct number of clusters by the dendogram.

Gaussian(EM) clustering algorithm

This algorithm assumes apriori that there are *'n'* Gaussian and then algorithm try to fits the data into the *'n'* Gaussian by expecting the classes of all data point and then maximizing the maximum likelihood of Gaussian centers.

Algorithmic steps for Expectation Maximization(EM) clustering

Let $X = \{x_1, x_2, x_3, ..., x_n\}$ be the set of data points

$V = \{\mu_1, \mu_2, \mu_3, ..., \mu_c\}$ be the set of means of Gaussian

$P = \{p_1, p_2, p_3, \ldots, p_c\}$ be the set of probability of occurrence of each Gaussian

1)On the i^{th} iteration initialize.

2) Compute the "expected" classes of all data points for each class using.

3) Compute "maximum likelihood μ" given our data class membership distribution using:

$$\mu(t+1) = \frac{\sum_k p(wt|xk, \gamma t)xk}{\sum_k p(wt|xk, \gamma t)}$$

$$p(t+1) = \frac{\sum_k p(wt|xk,\gamma t)}{R}$$

where, *'R'* is the number of data points.

Advantage

1) Gives extremely great result for real world data set.

Disadvantage

1) Algorithm is highly complex in nature.

Quality Threshold (QT) clustering algorithm

This algorithm requires the apriori specification of the threshold distance within the cluster and the minimum number of elements in each cluster. Now from each data point we find all its candidate data points. Candidate data points are those which are within the range of the threshold distance from the given data point. This way we find the candidate data points for all data point and choose the one with large number of candidate data points to form cluster. Now data points which belongs to this cluster is removed and the same procedure is repeated with the

reduced set of data points until no more cluster can be formed satisfying the minimum size criteria.

Algorithmic steps for QT clustering

1) Initialize the threshold distance allowed for clusters and the minimum cluster size.

2) Build a candidate cluster for each data point by including the closest point, the next closest, and so on, until the distance of the cluster surpasses the threshold.

3) Save the candidate cluster with the most points as the first true cluster, and remove all points in the cluster from further consideration.

4) Repeat with the reduced set of points until no more cluster can be formed having the minimum cluster size.

Advantages

1) Quality Guaranteed - Only clusters that pass a user-defined quality threshold will be returned.

2) Number of clusters is not specified apriori.

3) All possible clusters are considered - Candidate cluster is generated with respect to every data points and tested in order of size against quality criteria.

Disadvantages

1) Computationally Intensive and Time Consuming - Increasing the minimum cluster size or increasing the number of data points can greatly increase the computational time.

2) Threshold distance and minimum number of element in the cluster has to be defined apriori

2.2.2 NON LINEAR CLUSTERING ALGORITHMS

MST based clustering algorithm

Many data clustering algorithms have been proposed in the literature. Clustering algorithms may be good candidates for microaggregation if they can be adapted to enforce the group size constraint. Among the most commonly used methods is one that is based on partitioning a minimum spanning tree (MST). The standard MST partitioning algorithm first constructs the MST of the complete graph G , where each node in V corresponds to a data point and the length

of each edge is the Euclidean distance between its two endpoints, and then removes the c1 longest edges to produce c trees, each of which corresponds to a cluster.Intuitively, the longest edges which are removed separate the natural clusters, and the shortest edges which are retained connect close data points within natural clusters.

The microaggregation problem differs from the classical clustering problem in two respects. In classical clustering, the number of desired clusters is given a priori whereas, in microaggregation, it is not. Second, in classical clustering, there is no constraint on the size of clusters whereas, in microaggregation, each cluster must contain no fewer than k data points where k is given.

The standard MST partitioning algorithm, which removes edges from the MST in order of decreasing length until the specified number of clusters results, does not solve the microaggregation problem since it does not accommodate the group size constraint. To adapt this algorithm for microaggregation, we modify the strategy for selecting edges for deletion from the MST: Edges are inspected in the order of decreasing length and removed only if each tree that results from the edge's removal has at least k nodes. The resulting trees then discern pronounced clusters while also satisfying the minimum group size constraint.

An efficient MST-based clustering technique that enforces the minimum group size constraint. Results of computational experiments indicate that our MST partitioning algorithm for microaggregation is sufficiently efficient to be practical for large data sets and yields results that are comparable to the best available heuristic methods. For data sets that contain pronounced clustering effects, our algorithm results in significantly lower information loss.

Density based clustering algorithm

The problem of detecting clusters of points in data is challenging when the clusters are of different size, density and shape. Many of these issues become even more significant when the data is of very high dimensionality and when it includes noise and outliers.

This document describes the two density-based clustering algorithms: DBSCAN [Ester1996] and SNN [Ertoz2003]. These algorithms were implemented within the context of the LOCAL project [Local2005] as part of a task that aims to create models of the geographic space (Space

Models) to be used in context-aware mobile systems. Here, the role of the clustering algorithms is to identify clusters of Points of Interest (POIs) and then use the clusters to automatically characterize geographic regions.

DBSCAN algorithm

The DBSCAN algorithm was first introduced by Ester, et al. [Ester1996], and relies on a density-based notion of clusters. Clusters are identified by looking at the density of points. Regions with a high density of points depict the existence of clusters whereas regions with a low density of points indicate clusters of noise or clusters of outliers. This algorithm is particularly suited to deal with large datasets, with noise, and is able to identify clusters with different sizes and shapes.

The algorithm

The key idea ofthe DBSCAN algorithm is that, for each point of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of points, that is, the density in the neighbourhood has to exceed some predefined threshold. This algorithm needs three input parameters: - k, the neighbor list size; - Eps, the radius that delimitate the neighbourhood area of a point (Epsneighbourhood); - MinPts, the minimum number of points that must exist in the Eps-neighbourhood.

The clustering process is based on the classification of the points in the dataset as core points, border points and noise points, and on the use of density relations between points (directly density-reachable, density-reachable, density-connected [Ester1996]) to form the clusters.

kernel k-means clustering algorithm

The basic idea of kernel-based clustering is to seek an assignment of each pointto one of some clusters in the kernel feature space such that the within-cluster similarity is highand the between-cluster one is low. However, exhaustive search for the optimal assignments ofthe data points in the projected space is computationally infeasible. Since the number ofall possible partitions of a dataset grows exponentially with the number of data points, there isan urgent need for an efficient approach to find satisfactory sub-optimal solutions.

The classical k-means is such an iterative method. Due to its significant influence in the

datamining community, k-means algorithm was identified as one of the top 10 algorithms in datamining. Despite great success, k-means has one serious drawback that its performance wouldeasily degenerate in the case of ill-initialization. For instance, in the randomly ill-initializedassignment, some cluster is assigned with a small number of remote and isolated points such that the prototype of this cluster is relatively far away from any points. As a result, in the later iterations, this prototype would never get a chance to be assigned with any point

The kernel k-means algorithm applies the same trick as k-means but with one difference that here in the calculation of distance, kernel method is used instead of the Euclidean distance.

Algorithmic steps for Kernel k-means clustering

Let $X = \{a_1, a_2, a_3, ..., a_n\}$ be the set of data points and *'c'* be the number of clusters.

1) Randomly initialize 'c' cluster center.

2) Compute the distance of each data point and the cluster center in the transformed space.

3) Assign data point to that cluster center whose distance is minimum.

4) Until data points are re-assigned repeat from step 2).

Advantages

1) Algorithm is able to identify the non-linear structures.

2) Algorithm is best suited for real life data set.

Disadvantages

1) Number of cluster centers need to be predefined.

2) Algorithm is complex in nature and time complexity is large.

In this paper, we propose a novel approach termed conscience online learning (COLL) to solve the optimization problem associated with kernel-based clustering in the online learning framework.

2.2.3 APPLICATIONS OF CLUSTERING ALGORITHMS:

CLUSTERING ALGORITHM IN IDENTIFYING CANCEREOUS DATA

Clustering algorithm can be used in identifying the cancerous data set. Initially we take known samples of cancerous and non cancerous data set. Label both the samples data set. We then randomly mix both samples and apply different clustering algorithms into the mixed samples data set (this is known as learning phase of clustering algorithm) and accordingly check the result for how many data set we are getting the correct results (since this is known samples we already know the results beforehand) and hence we can calculate the percentage of correct results obtained. Now, for some arbitrary sample data set if we apply the same algorithm we can expect the result to be the same percentage correct as we got during the learning phase of the particular algorithm. On this basis we can search for the best suitable clustering algorithm for our data samples.

CLUSTERING ALGORITHM IN SEARCH ENGINES:

Clustering algorithm is the backbone behind the search engines. Search engines try to group similar objects in one cluster and the dissimilar objects far from each other. It provides result for the searched data according to the nearest similar object which are clustered around the data to be searched. Better the clustering algorithm used, better are the chances of getting the required result on the front page

CLUSTERING ALGORITHM IN ACADEMICS

The ability to monitor the progress of students' academic performance has been the critical issue for the academic community of higher learning. Clustering algorithm can be used to monitor the students academic performance. Based on the students score they are grouped into different-different clusters (using k-means, fuzzy c-means etc), where each clusters denoting the different level of performance. By knowing the number of students in each cluster we can know the average performance of a class as a whole.

LITERATURE SURVEY

3. LITERATURE SURVEY

3.1. Paper1: Chang-Dong Wang, Jian-Huang Lai and Dong Huang. Kernel-Based Clustering with Automatic Cluster Number Selection.

In data clustering, it is an important task to discover nonlinearly separable clusters . Kernel-based clustering has been widely used for achieving this task, among which kernel k-means is one of the most well-known methods . Unfortunately, two inherent disadvantages of kernel k-means are that, it is easily trapped into degenerate local minima when the prototypes of clusters are ill-initialized, and the actual number of clusters has to be provided as input. Although some algorithms have been developed to handle the first problem (e.g., global kernel k-means , the COLL algorithm), there is still a lack of methods for automatically estimating the cluster number in kernel space. In this paper, inspired by the on-line learning framework developed and the rival penalization mechanism, we propose a novel approach termed KeCans (for Kernel-based Clustering method with automatic cluster number selection). In a wide range of applications, a priori knowledge of the actual cluster number is often unavailable. And it is one of the major problems to estimate the reasonable cluster number in data clustering . For instance, in large-scale document clustering, the most suitable number of clusters isnot known in advance, which should be estimated to achieve a precise clustering. A critical issue of pattern-based clustering is that mining and analyzing a huge number of pattern-based clusters is very time-consuming. Therefore, a maximal pattern-based clustering was developed to mine the maximal pattern-based clusters so as to eliminate the redundant clusters. In sequence clustering, it is also required to automatically estimate the number of clusters when training mixtures of Hidden Markov Models.One of the earliest work on estimating the cluster number is the Rival Penalized Competitive Learning (RPCL) algorithm proposed by Xu et al. In RPCL, more than the actual number of prototypes are randomly chosen in initialization, and in competition process, not only the winning prototype is learned to adapt to the input, but also its rival (i.e., the second winner) is delearned by a smaller delearning rate, such that the redundant prototypes can be automatically eliminated. Another milestone work is the Xmeansalgorithm . By efficiently searching the space of cluster locations and number of clusters to optimize some measure such as Bayesian Information Criterion (BIC), Xmeans extends k-means with the capability of cluster number estimation. Very recently, Muhlenbach and Lallich proposed a GBC method based on regions of

influence. A graph is constructed and the edges of the graph having higher values are cut according to a hierarchical divisive procedure. By calculating an index from the average size of the cut edges, an appropriate number of clusters can be detected. However, these efforts are limited only to the original data space and there is still a lack of methods for automatically estimating the cluster number in kernel space. In this paper, inspired by the online learning framework, and the rival penalization mechanism, we propose a novel kernel-based clustering approach termed KeCans (for Kernel-based Clustering method with automatic cluster number selection). To represent prototypes in kernel space without knowing the explicit mapping φ, the prototypes are represented by the prototype descriptor Wφ. The prototype descriptor is a real-valued matrix, whose rows represent prototypes as the inner products between a prototype and the feature space images of data points, as well as the squared length of the prototype. In initialization, more the actual number of prototypes are initialized by allocating more rows in the prototype descriptor. In competition process, rival penalization is used to driveaway the redundant rows of Wφ. That is, for each input data point, not only the winning row of Wφ is learned, but also its rival (i.e., the second winner) row of Wφ is delearned by a smaller delearning rate.

By using online framework we will overcome the disadvantages present in kernel k means.

3.2. Paper 2: Jian-Huang Lai and Chang-Dong Wang. Kernel and Graph: Two Approaches for Nonlinear Competitive Learning Clustering

Competitive learning has received a significant amount of attention in the past decades . Due to its adaptive on-line learning, it has been widely applied in the fields of data clustering , vector quantization, RBF net learning, shape detection, discretevalued source separation, Markov model identification , component analysis, scheduling, etc. Among them, clustering analysis is still one of the most active fields. Despite significant success of competitive learning in data clustering, most competitive learning clustering algorithms assume that data are linearly separable, which is unfortunately not always held in realworld applications. The square distance based winner selection only generates a list of convex regions, each for one cluster. These convex regions can separate linear clusters (i.e., clusters with convex boundaries) as shown in Fig. 1(a); however, they fail to identify nonlinear clusters (i.e., clusters with concave boundaries) as shown in Fig. 1(b). Note that real-world data do not always have convex boundaries. This motivates the development of nonlinear competitive learning to solve this problem. In this paper, two approaches are presented to realize nonlinear competitive learning clustering, one of which is from the viewpoint of kernel clustering and the other is based on the graph method. Kernel method is one of the most popular methodologies for nonlinear clustering. By mapping data points from the input data space into a linearly separable kernel space via a kernel mapping, the nonlinear clustering can be easily realized. Recently, a few kernel versions of competitive learning have been developed. However, these works are confined to some special cases. For instance, two methods in Refs. mapped the data into normalization-invariant kernel space (i.e., on a unit hypersphere $\varphi(x)$ 2= 1) before performing clustering. The constraint of normalizationinvariant kernel helps avoid the computational challengeof on-line learning without explicit mapping φ. Unfortunately, most kernels do not satisfy this constraint such as polynomial kernelall-subsets kernel,anova kernel , while normalization would greatly re-distribute the data. Thus, they are only allowed in a few kernels (e.g., Gaussian kernel). The other method suggested updating the distance between prototypes and data points. However, some of the crucial components of competitive learning can not be realized. In particular, the convergence criterion that is critical for measuring the convergence cannot be computed. This leads to some loss of the advantages of competitive learning, e.g., convergence guarantee. We provide a complete solution

for nonlinear competitive learning and propose kernel competitive learning (KCL). We first obtain a kernel model of competitive learning by mapping the data into kernel space before performing on-line learning, which is able to identify nonlinear clusters. However, it is practically infeasible to realize such a kernel model when the mapping function is unknown, since the winning prototype is updated in on-line mode rather than in batch like that in kernel k-means. To tackle this problem, we develop an efficient and elegant computational method by proposing a new prototype representation termed prototype descriptor, through which the computational challenge of the four crucial components including clustering initialization, winner selection, winner updating and convergence criterion is dexterously handled. Therefore, a complete nonlinear competitive learning clustering method termed kernel competitive learning (KCL) is proposed. Graph-based method is another popular approach for nonlinear clustering . In the graph-based method, a graph is first constructed, in which data points are represented by nodes, and the proximity between two data points is represented by the weight of the edge between the corresponding nodes. Then the notion of useful links between data points or the eigenstructure of the proximity matrix is used to generate clusters of an arbitrary shape. For instance, in the shared nearest neighbor (SNN) clustering method , the weight between two data points (nodes) is computed as the number of shared neighbors between nodes given that the nodes are connected (i.e., having useful link). The core-points are defined according to some prespecified parameters M inPts and Eps and used to obtain clusters of arbitrary shapes. In spectral clustering, the clustering problem is transferred to a graph partitioning problem where the goal is to obtain the prespecified number of components of the graph with minimizing some objective function such as normalized cut. To realize this, the eigenstructure of the proximity matrix is obtained, where the eigenvectors with the smallest eigenvalues are used to construct a partitioning. Each component obtained corresponds to a major structure of the graph, such that these components (i.e., clusters) can be of arbitrary shapes revealing the real structures. However, this type of method needs to prespecify either some parameters (e.g., M inPts and Eps in SNN) or the number of real clusters (e.g., in spectral clustering). Based on the graph method, we propose a novel nonlinear clustering algorithm named graph-based multiprototype competitive learning (GMPCL). This algorithm employs a graph-based method to generate an initial, coarse clustering. Multi-prototype competitive learning is then performed to refine the

clustering and identify clusters of an arbitrary shape. Therefore, the proposed approach consists of two phases, namely, graph-based initial clustering and multi-prototype competitive learning. In the multi-prototype competitive learning, to generate cluster boundaries of arbitrary shapes, each cluster is represented by multiple prototypes, whose subregions of the Voronoi diagram togetherapproximately characterize one cluster of an arbitrary shape.

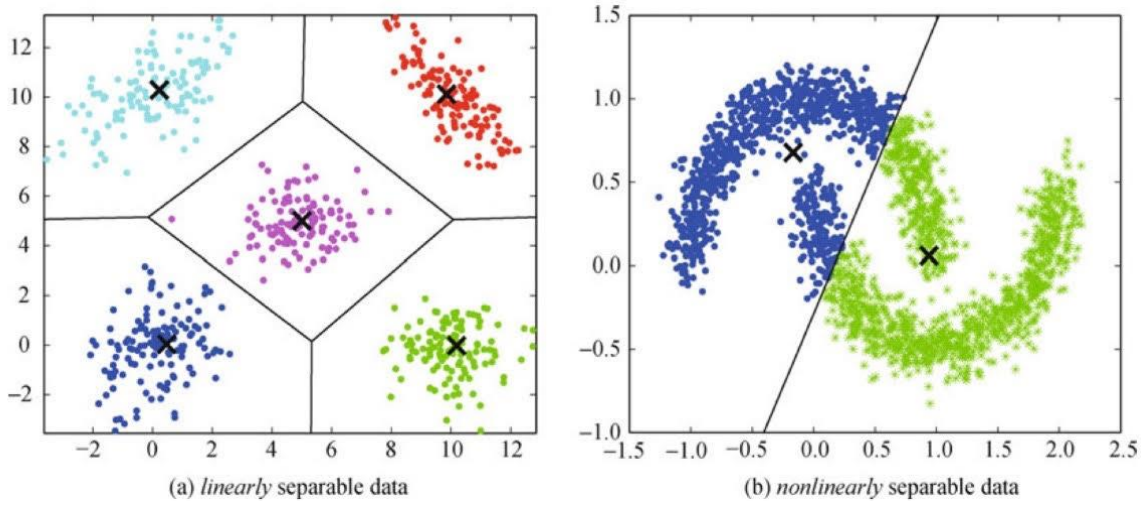


Fig 21: Diagram showing Linearly and Nonlinearly separable data.

3.3. Paper 3: KERNEL-BASED CLUSTERING OF BIG DATA, By Radha Chitta

There has been a rapid increase in the volume of digital data over the recent years. A study by IDC and EMC Corporation predicted the creation of 44 zettabytes (1021 bytes) of digital data by the year 2020. Analysis of this massive amounts of data, popularly known as big data, necessitates

highly scalable data analysis techniques. Clustering is an exploratory data analysis tool used to discover the underlying groups in the data. The state-of-the-art algorithms for clustering big data sets are linear clustering algorithms, which assume that the data is linearly separable in the input space, and use measures such as the Euclidean distance to define the inter-point similarities.

Though efficient, linear clustering algorithms do not achieve high cluster quality on real-world data

sets, which are not linearly separable. Kernel-based clustering algorithms employ non-linear similarity

measures to define the inter-point similarities. As a result, they are able to identify clusters of arbitrary shapes and densities. However, kernel-based clustering techniques suffer from two major

limitations:

(i) Their running time and memory complexity increase quadratically with the increase in the size of the data set. They cannot scale up to data sets containing billions of data points.

(ii) The performance of the kernel-based clustering algorithms is highly sensitive to the choice of the kernel similarity function. Ad hoc approaches, relying on prior domain knowledge, are currently employed to choose the kernel function, and it is difficult to determine the appropriate kernel similarity function for the given data set.

3.4. Paper 4:Chang-Dong Wang, Jian-Huang Lai and Jun-Yong Zhu. A Conscience On-line Learning Approach for Kernel-Based Clustering

Data clustering plays an indispensable role in various fields such as computer science, medical science, social science and economics. In the real-world applications, it is important to identify the nonlinear clusters and kernel-based clustering is a popular method for this purpose. The basic idea of kernel-based clustering is to seek an assignment of each point to one of some clusters in the kernel feature space such that, the within-cluster similarity is high and the between-cluster one is low. However, exhaustive search for the optimal assignments of the data points in the projected space is computationally infeasible. Since the number of all possible partitions of a dataset grows exponentially with the number of data points. There is an urgent need for an efficient approach to find satisfactory sub-optimal solutions. *k*-means is such an iterative method Despite great success, one serious drawback is that, its performance would easily degenerate in the case of ill-initialization. For instance, in the randomly ill-initialized assignment, some cluster is assigned with a small number of remote and isolated points such that the prototype of this cluster is relatively far away from any points. As a result, in the later iterations, this prototype would never get a chance to be assigned with any point. Several methods have been developed to overcome the ill-initialization problem. Bradley and Fayyad proposed to compute a refined initialization from a given one by estimating the modes of a distribution based on the subsampling technique. Another approach for refining the initial cluster prototypes is based on the observations that some patterns are very similar to each other such that they have the same cluster membership irrespective to the choice of initial cluster prototypes. Zhang et al computed a lower bound on the cost of the local optimum from the current prototype set and proposed a BoundIterate method. Evolutionary algorithms such as genetic algorithm have also been applied to *k*-means, aiming at avoiding the degenerate local minima. Very recently, Likas et al. proposed the global *k*-means that is deterministic and does not rely on any initial conditions. Although these methods have eliminated the sensitivity to the ill-initialization, however, most of them are computationally expensive. In this paper, we propose a novel approach termed conscience on-line learning (COLL) to solve the optimization problem associated with kernel-based clustering in the online learning framework. The proposed method starts with a random guess of the assignment the same as *k*-means. However, unlike *k*-means, in the procedure of each iteration,

for each randomly taken data point, the method selects the winning prototype based on the conscience mechanism, and updates the winner by the on-line learning rule. The procedure requires only one winning prototype to update slightly towards the new point rather than re-computing the mean of each cluster at each step, hence much faster convergence rate is achieved and other competitive mechanisms like conscience can be easily integrated. The advantage of the conscience mechanism is that, by reducing the winning rate of the frequent winners, all the prototypes are brought available into the solution quickly and the ill-initialized prototypes are biased so that each prototype can win the competition with almost the same probability . Consequently, two contributions have been made in this paper.

1) The proposed COLL method is insensitive to the illinitialization and can be generalized to tackle other degenerate problems associated with random initialization.

2) Compared with other techniques aiming at tackling illinitialization problem, such as global search strategy, our approach achieves faster convergence rate due to both on-line learning and conscience mechanism.

SYSTEM ANALYSIS

4. SYSTEM ANALYSIS

4.1 PROPOSED SYSTEM

This mainly used to separate similar type of data in the form of clusters by using clustering techniques.

The main objective is to select a non linear and unsupervised dataset and by using non linear clustering algorithms and clustering techniques in data mining ,we need to separate similar group of data in the form of clusters that is one cluster is of similar data and different when compared to other clusters.

4.2 ADVANTAGES OF PROPOSED SYSTEM

Now-a-days as there is immense increase in data flow So, this technique gives a scope to separate the information from data and thereby decreasing workload and redundancy..etc.,

4.3 SYSTEM REQUIREMENTS

- The first requirement is Ubuntu operating system with 8 Intel, 2.00GHz Processor, 4gb RAM.
- The second requirement is a system installed with Python with Numpy , Scipy , Panda modules.
- In order to implement the algorithm we need data sets. Here we are using various data sets such as quiz_A.txt,self_test.txt,test1_data.txt,test2_data.txt,xclara.txt.

TECHNOLOGIES USED

5 TECHNOLOGIES USED

5.1 PYTHON

PYTHON language is used to code a non linear clustering algorithm and implement it on a dataset to form clusters to get similar groups of data.

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:**

  Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive:**

  You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- **Python is Object-Oriented:**

  Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language:**

  Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

5.1.1 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

FEATURES AND PHILOSHYPHY

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution.

The design of Python offers some support for functional programming in the Lisp tradition. The language has map(), reduce() and filter() functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The core philosophy of the language is summarized by the document The Zen of Python *(PEP 20)*, which includes aphorisms such as:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex

- Complex is better than complicated
- Readability counts

5.1.2 Python Features

Python's features include

- **Easy-to-learn**

  Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

- **Easy-to-read**

  Python code is more clearly defined and visible to the eyes.

- **Easy-to-maintain**

  Python's source code is fairly easy-to-maintain.

- **A broad standard library**

  Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode**

  Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

- **Portable**

  Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

- **Extendable**

  You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

- **Databases**

  Python provides interfaces to all major commercial databases.

- **GUI Programming**

  Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

- **Scalable**

  Python provides a better structure and support for large programs than shell scripting.

5.1.3 DEVELOPMENT

Python's development is conducted largely through the **Python Enhancement Proposal** (PEP) process. The PEP process is the primary mechanism for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. Outstanding PEPs are reviewed and commented upon by the Python community and by Van Rossum, the Python project's benevolent dictator for life.

Enhancement of the language goes along with development of the CPython reference implementation. The mailing list python-dev is the primary forum for discussion about the language's development; specific issues are discussed in the Roundup bug tracker maintained at python.org. Development takes place on a self-hosted source code repository running Mercurial

CPython's public releases come in three types, distinguished by which part of the version number is incremented:

- Backwards-incompatible versions, where code is expected to break and must be manually ported. The first part of the version number is incremented. These releases happen infrequently—for example, version 3.0 was released 8 years after 2.0.
- Major or "feature" releases, which are largely compatible but introduce new features. The second part of the version number is incremented. These releases are scheduled to occur roughly every 18 months, and each major version is supported by bugfixes for several years after its release.

- Bugfix releases, which introduce no new features but fix bugs. The third and final part of the version number is incremented. These releases are made whenever a sufficient number of bugs have been fixed upstream since the last release, or roughly every 3 months. Security vulnerabilities are also patched in bugfix releases. Many alpha, beta, and release-candidates are also released as previews, and for testing before final releases. Although there is a rough schedule for each release, this is often pushed back if the code is not ready. The development team monitors the state of the code by running the large unit test suite during development, and using the BuildBot continuous integration system. .

5.1.4 USES

Since 2003, Python has consistently ranked in the top ten most popular programming languages as measured by the TIOBE Programming Community Index. As of August 2016, it is the fifth most popular language.[103] It was ranked as Programming Language of the Year for the year 2007 and 2010.[24] It is the third most popular language whose grammatical syntax is not predominantly based on C, e.g. C++, Objective-C (note, C# and Java only have partial syntactic similarity to C, such as the use of curly braces, and are closer in similarity to each other than C).

An empirical study found scripting languages (such as Python) more productive than conventional languages (such as C and Java) for a programming problem involving string manipulation and search in a dictionary. Memory consumption was often "better than Java and not much worse than C or C++".[104]

Large organizations that make use of Python include Wikipedia, Google Yahoo! CERN, NASA, and some smaller ones like ILM, and ITA The social news networking site, Reddit, is written entirely in Python.

Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of

complex applications. Pyjamas and IronPython can be used to develop the client-side of Ajax-based applications. SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.

Libraries like NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as BioPython and Astropy providing domain-specific functionality. SageMath is a mathematical software with a "notebook" programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus. The Python language re-implemented in Java platform is used for numeric and statistical calculations with 2D/3D visualization by the DMelt project.

Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like FreeCAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro and musical notation program or scorewriter capella.

GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS. It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go. Python is also used in algorithmic trading and quantitative finance. Python can also be implemented in APIs of online brokerages that run on other languages by using wrappers.

5.1.5 NAMING

Python's name is derived from the television series *Monty Python's Flying Circus*, and it is common to use Monty Python references in example code For example, the metasyntactic variables often used in Python literature are *spam* and *eggs*, instead of the traditional *foo* and *bar*. Also, the official Python documentation often contains various obscure Monty Python references.

The prefix *Py-* is used to show that something is related to Python. Examples of the use of this prefix in names of Python applications or libraries include Pygame, a binding of SDL to Python (commonly used to create games); PyS60, an implementation for the Symbian S60 operating system; PyQt and PyGTK, which bind Qt and GTK, respectively, to Python; and PyPy, a Python implementation originally written in Python.

5.1.6 LIBRARIES

Python has a large standard library, commonly cited as one of Python's greatest strengths providing tools suited to many tasks. This is deliberate and has been described as a "batteries included Python philosophy. For Internet-facing applications, many standard formats and protocols (such as MIME and HTTP) are supported. Modules for creating graphical user interfaces, connecting to relational databases, pseudorandom number generators, arithmetic with arbitrary precision decimals, manipulating regular expressions, and doing unit testing are also included.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP), but most modules are not. They are specified by their code, internal documentation, and test suite (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

The standard library is not needed to run Python or embed it in an application. For example, Blender 2.49 omits most of the standard library.

As of October, 2016, the Python Package Index, the official repository of third-party software for Python, contains over 91,000

packages offering a wide range of functionality, including:

- graphical user interfaces, web frameworks, multimedia, databases, networking and communications
- test frameworks, automation and web scraping, documentation tools, system administration
- scientific computing, text processing, image processing

5.1.7 DEVELOPMENT ENVIRONMENT

Most Python implementations (including CPython) can function as a command line interpreter, for which the user enters statements sequentially and receives the results immediately (read–eval–print loop (REPL)). In short, Python acts as a command-line interface or shell.

Other shells add abilities beyond those in the basic interpreter, including IDLE and IPython. While generally following the visual style of the Python shell, they implement features like auto-completion, session state retention, and syntax highlighting.

In addition to standard desktop integrated development environments (Python IDEs), there are also web browser-based IDEs, SageMath (intended for developing science and math-related Python programs), and a browser-based IDE and hosting environment, PythonAnywhere. Additionally, the Canopy IDE is also an option for creating programs written in Python.

5.2 LIBRARIES THAT ARE USED

5.2.1 NUMPY

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Operations on numPy

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

NumPy Ndarray object

The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.

Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**).

Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.

The basic ndarray is created using an array function in NumPy as follows –numpy.array()

NumPy Array Attributes

ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print a.shape
```

NumPy - Array Creation Routines

A new ndarray object can be constructed by any of the following array creation routines or using a low-level ndarray constructor.

numpy.empty

numpy.zeros

numpy.on

Numpy Indexing and slicing

Contents of ndarray object can be accessed and modified by indexing or slicing, just like Python's in-built container objects.

Basic slicing is an extension of Python's basic concept of slicing to n dimensions. A Python slice object is constructed by giving **start, stop**, and **step** parameters to the built-in **slice** function.

Example:

```
import numpy as np

a = np.arange(10)

s = slice(2,7,2)

print a[s]
```

NumPy - Advanced Indexing

It is possible to make a selection from ndarray that is a non-tuple sequence, ndarray object of integer or Boolean data type, or a tuple with at least one item being a sequence object. Advanced indexing always returns a copy of the data. As against this, the slicing only presents a view.

There are two types of advanced indexing − **Integer** and **Boolean**.

Integer Indexing

This mechanism helps in selecting any arbitrary item in an array based on its Ndimensional index. Each integer array represents the number of indexes into that dimension. When the index consists of as many integer arrays as the dimensions of the target ndarray, it becomes straightforward.

Example:

```
import numpy as np
```

```
x = np.array([[1, 2], [3, 4], [5, 6]])

y = x[[0,1,2], [0,1,0]]

print y
```

Boolean Array Indexing

This type of advanced indexing is used when the resultant object is meant to be the result of Boolean operations, such as comparison operators

Example:

```
import numpy as np

x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

print 'Our array is:'

print x

print '\n'

# Now we will print the items greater than 5

print 'The items greater than 5 are:'

print x[x > 5]
```

NumPy - Iterating Over Array

NumPy package contains an iterator object **numpy.nditer**. It is an efficient multidimensional iterator object using which it is possible to iterate over an array. Each element of an array is visited using Python's standard Iterator interface.

Example

```
import numpy as np

a = np.arange(0,60,5)

a = a.reshape(3,4)
```

```
print 'Original array is:'
print a
print '\n'
print 'Modified array is:'
for x in np.nditer(a):
   print x,
```

output:

```
Original array is:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]
Modified array is:
0 5 10 15 20 25 30 35 40 45 50 55
```

NumPy - Mathematical Functions

NumPy contains a large number of various mathematical operations. NumPy provides standard trigonometric functions, functions for arithmetic operations, handling complex numbers, etc.

NumPy - Arithmetic Operations

add()

substract()

multiply()

divide()

reciprocal()

pow()

mod()

NumPy - Statistical Functions

NumPy has quite a few useful statistical functions for finding minimum, maximum, percentile standard deviation and variance, etc. from the given elements in the array. The functions are explained as follows –

numpy.amin() and numpy.amax()

Example:

```
import numpy as np

a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print 'Our array is:'

print a

print '\n'

print 'Applying amin() function:'

print np.amin(a,1)

print '\n'

print 'Applying amax() function:'

print np.amax(a)

print '\n'

print 'Applying amax() function again:'

print np.amax(a, axis = 0)
```

Output:

```
Our array is:
[[3 7 5]
 [8 4 3]
 [2 4 9]]
Applying amin() function:
```

[3 3 2]

Applying amax() function:9

Applying amax() function again:

[8 7 9]

numpy.ptp()

The **numpy.ptp()** function returns the range (maximum-minimum) of values along an axis.

numpy.percentile()

Percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall. The function **numpy.percentile()** takes the following arguments.

```
numpy.percentile(a, q, axis)
```

numpy.median()

Median is defined as the value separating the higher half of a data sample from the lower half.

numpy.mean()

Arithmetic mean is the sum of elements along an axis divided by the number of elements. The numpy.mean() function returns the arithmetic mean of elements in the array. If the axis is mentioned, it is calculated along it.

numpy.average()

Weighted average is an average resulting from the multiplication of each component by a factor reflecting its importance. The **numpy.average()**function computes the weighted average of elements in an array according to their respective weight given in another array. The function can have an axis parameter. If the axis is not specified, the array is flattened.

Standard Deviation

Standard deviation is the square root of the average of squared deviations from mean. The formula for standard deviation is as follows −

```
std = sqrt(mean(abs(x - x.mean())**2))
```

If the array is [1, 2, 3, 4], then its mean is 2.5. Hence the squared deviations are [2.25, 0.25, 0.25, 2.25] and the square root of its mean divided by 4, i.e., sqrt (5/4) is 1.1180339887498949.

Variance

Variance is the average of squared deviations, i.e., **mean(abs(x - x.mean())**2)**. In other words, the standard deviation is the square root of variance.

5.2.2 SCIPY

Scipy is the core package for scientific routines in Python; it is meant to operate efficiently on numpy arrays, so that numpy and scipy work hand in hand. SciPy greatly extends the functionality of the NumPy routines. Many SciPy routines can be accessed by simply importing the module:

```
>>> import scipy
```

A number of sub-modules in SciPy require explicit import

```
>>> import scipy.interpolate
```

The functions in each module are well-documented and provide instant access to common numerical algorithms, and are very easy to implement. Thus, SciPy can save tremendous amounts of time in scientific computing applications since it offers a library of pre-written, pre-tested routines.

Scipy's different submodules correspond to different applications, such as interpolation, integration, optimization, image processing, statistics, special functions, etc.

Few examples to give a general idea of how to use scipy for scientific computing:

File input/output: scipy.io

Special functions: scipy.special

Linear algebra operations: scipy.linalg

Interpolation: scipy.interpolate

Optimization and fit: scipy.optimize

Statistics and random numbers: scipy.stats

Numerical integration: scipy.integrate

Fast Fourier transforms: scipy.fftpack

Signal processing: scipy.signal

Image manipulation: scipy.ndimage

The standard way of importing Numpy and these Scipy modules is:

```
>>> import numpy as np
>>> from scipy import io     #same for other sub-modules
```

Special functions: scipy.special

Special functions are transcendental functions. The docstring of the scipy.special module is well-written, so we won't list all functions here. Frequently used ones are:

Bessel function, such as scipy.special.jn() (nth integer order Bessel function)

Elliptic function (scipy.special.ellipj() for the Jacobian elliptic function, …)

Gamma function: scipy.special.gamma(), also note scipy.special.gammaln() which will give the log of Gamma to a higher numerical precision.

Erf, the area under a Gaussian curve: scipy.special.erf()

Optimization and fit: scipy.optimize

Optimization is the problem of finding a numerical solution to a minimization or equality.

The scipy.optimize module provides algorithms for function minimization (scalar or multi-dimensional), curve fitting and root finding.

```
>>> from scipy import optimize
```

Curve fitting

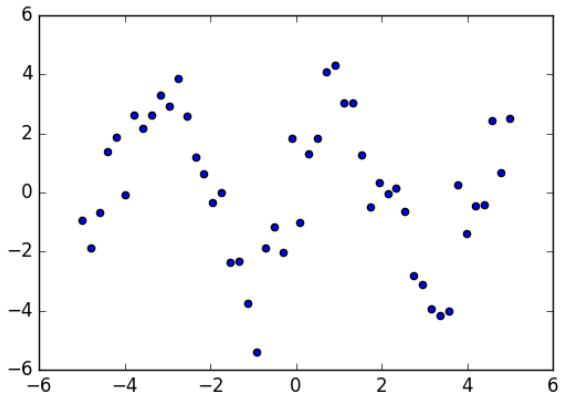


Fig 8: Curve Fitting

If we know that the data lies on a sine wave, but not the amplitudes or the period, we can find those by least squares curve fitting.

First generate some data

```
import numpy as np

# Seed the random number generator for reproducibility

np.random.seed(0)

x_data = np.linspace(-5, 5, num=50)

y_data = 2.9 * np.sin(1.5 * x_data) + np.random.normal(size=50)

# And plot it

import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))

plt.scatter(x_data, y_data)

#Now fit a simple sine function to the data

from scipy import optimize

def test_func(x, a, b):
```

```
    return a * np.sin(b * x)
params, params_covariance = optimize.curve_fit(test_func, x_data, y_data,p0=[2, 2])
print(params)
```

Out:

```
[ 3.05931973  1.45754553]
#And plot the resulting curve on the data
plt.figure(figsize=(6, 4))
plt.scatter(x_data, y_data, label='Data')
plt.plot(x_data, test_func(x_data, params[0], params[1]),label='Fitted function')
plt.legend(loc='best')
plt.show()
```

DESIGN AND IMPLEMENTATION

6. DESIGN AND IMPLEMENTATION

We have implemented k-means algorithm and kernel k-means algorithm over 5 datasets. K-means algorithm forms the clusters linearly and which creates an ill-initialization problem. So, We have implemented kernel k-means algorithm which forms the clusters non-linearly and avoid ill-initialization problem created by k-means.

This can be implemented by using python programming. The required libraries are

- NumPy
- Scipy
- Pandas

The algorithm can be applied over various data sets. The datasets used are

- test1_data.txt

  This dataset contains 2 attributes, 400 instances.
- test2_data.txt

  This dataset contains 2 attributes, 400 instances.
- quiz_A.data

  This dataset contains 2 attributes, 400 instances.
- self_test.data

  This dataset contains 2 attributes, 300 instances.
- xclara.txt

  This dataset contains 2 attributes, 3000 instances.

6.1 IMPLEMENTED CODE FOR K-MEANS

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.pyplot import cm
import time

# Below code has comments to add or remove input datasets to form different linearly seperated clusters for each run.
# The code can take only .txt because .csv or .tsv type data are not hashable.
# No.of.Iterations vary until final convergence based on the datasets inputted.

filePath1 = "test1_data.txt"
#filePath1 = "test2_data.txt"
#filePath1 =  "led7.tsv"
#filePath2 = "test2_data.txt"
#filePath2 = "led7.tsv"
#filePath1 = "xclara.txt"
#filePath2 = "xclara.csv"
#filePath1 = "quiz_A.data"
#filePath1 = "self_test.data"
dataTesting1 = np.loadtxt(filePath1, delimiter=" ")
#dataTesting2 = np.loadtxt(filePath2, delimiter=" ")
#dataTesting1 = pd.read_csv('xclara.csv')
#dataTesting2 = pd.read_csv('xclara.csv')
#dataTesting1 = pd.read_table('led7.tsv',sep='\t')
#dataTesting2 = pd.read_table('led7.tsv',sep='\t')

print("data testing: ", dataTesting1.shape)

# define params
k = 2  # numb of clusters
iterationCounter = 0  # clustering iteration counter
input = dataTesting1
Centroid_Initialization_Method = "badInitialization" # options: random, kmeans++, badInitialization, zeroInitialization


def initializeCentroid(dataIn, method, k):
    if (method == "random"):
        result = dataIn[np.random.choice(dataIn.shape[0], k, replace=False)]
    if (method == "kmeans++"):
        euclideanMatrix_ofAllCentroid = np.ndarray(shape=(dataIn.shape[0], 0))
        allCentroid = np.ndarray(shape=(0, dataIn.shape[1]))
```

```
    first = dataIn[np.random.choice(dataIn.shape[0], 1, replace=False)]
    allCentroid = np.concatenate((allCentroid, first), axis=0)
    repeatedCentroid = np.repeat(first, dataIn.shape[0], axis=0)
    deltaMatrix = abs(np.subtract(dataIn, repeatedCentroid))
    euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
    indexof_NextCentroid = (np.argmax(np.matrix(euclideanMatrix)))
    if (k > 1):
        for a in range(1, k):
            nextCentroid = np.matrix(dataIn[np.asscalar(indexof_NextCentroid), :])
            allCentroid = np.concatenate((allCentroid, nextCentroid), axis=0)
            for i in range(0, allCentroid.shape[0]):
                repeatedCentroid = np.repeat(allCentroid[i, :], dataIn.shape[0], axis=0)
                deltaMatrix = abs(np.subtract(dataIn, repeatedCentroid))
                euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
                euclideanMatrix_ofAllCentroid = \
                    np.concatenate((euclideanMatrix_ofAllCentroid, euclideanMatrix), axis=1)
            euclideanFinal = np.min(np.matrix(euclideanMatrix_ofAllCentroid), axis=1)
            indexof_NextCentroid = np.argmax(np.matrix(euclideanFinal))
    result = allCentroid
if (method == "badInitialization"):
    allCentroid = np.ndarray(shape=(0, dataIn.shape[1]))
    firstIndex = np.random.randint(0, dataIn.shape[0])
    first = np.matrix(dataIn[firstIndex, :])
    dataIn = np.delete(dataIn, firstIndex, 0)
    allCentroid = np.concatenate((allCentroid, first), axis=0)
    repeatedCentroid = np.repeat(first, dataIn.shape[0], axis=0)
    deltaMatrix = abs(np.subtract(dataIn, repeatedCentroid))
    euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
    indexof_NextCentroid = (np.argmin(np.matrix(euclideanMatrix)))
    if (k > 1):
        for a in range(1, k):
            nextCentroid = np.matrix(dataIn[np.asscalar(indexof_NextCentroid), :])
            dataIn = np.delete(dataIn, np.asscalar(indexof_NextCentroid), 0)
            euclideanMatrix_ofAllCentroid = np.ndarray(shape=(dataIn.shape[0], 0))
            allCentroid = np.concatenate((allCentroid, nextCentroid), axis=0)
            for i in range(0, allCentroid.shape[0]):
                repeatedCentroid = np.repeat(allCentroid[i, :], dataIn.shape[0], axis=0)
                deltaMatrix = abs(np.subtract(dataIn, repeatedCentroid))
                euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
                euclideanMatrix_ofAllCentroid = \
                    np.concatenate((euclideanMatrix_ofAllCentroid, euclideanMatrix), axis=1)
            euclideanFinal = np.min(np.matrix(euclideanMatrix_ofAllCentroid), axis=1)
            indexof_NextCentroid = np.argmin(np.matrix(euclideanFinal))
    result = allCentroid
if (method == "zeroInitialization"):
```

```
        result = np.matrix(np.full((k, dataIn.shape[1]), 0))

    color = iter(cm.rainbow(np.linspace(0, 1, k)))
    plt.figure("centroid initialization")
    plt.title("centroid initialization")
    plt.scatter(dataIn[:, 0], dataIn[:, 1], marker=".", s=100)
    for i in range(0, k):
        col = next(color)
        plt.scatter((result[i, 0]), (result[i, 1]), marker="*", s=400, c=col)
        plt.text((result[i, 0]), (result[i, 1]), str(i + 1), fontsize=20)
    return result


def plotClusterResult(listof_ClusterMembers, centroid, iteration, converged):
    n = listof_ClusterMembers.__len__()
    color = iter(cm.rainbow(np.linspace(0, 1, n)))
    plt.figure("result")
    plt.clf()
    plt.title("iteration-" + iteration)
    for i in range(n):
        col = next(color)
        memberof_Cluster = np.asmatrix(listof_ClusterMembers[i])
        plt.scatter(np.ravel(memberof_Cluster[:, 0]), np.ravel(memberof_Cluster[:, 1]), marker=".",
s=100, c=col)
        plt.scatter((centroid[i, 0]), (centroid[i, 1]), marker="*", s=400, c=col, edgecolors="black")
    if (converged == 0):
        plt.ion()
        plt.show()
        plt.pause(0.1)
    if (converged == 1):
        plt.show(block=True)


def kMeans(data, centroidInitialized):
    nCluster = centroidInitialized.shape[0]
    # looping until converged
    global iterationCounter
    centroidInitialized = np.matrix(centroidInitialized)
    while (True):
        iterationCounter += 1
        euclideanMatrixOf_AllCluster = np.ndarray(shape=(data.shape[0], 0))
        # assign data to cluster whose centroid is the closest one
        for i in range(0, nCluster):
            centroidRepeated = np.repeat(centroidInitialized[i, :], data.shape[0], axis=0)
            deltaMatrix = abs(np.subtract(data, centroidRepeated))
```

```
            euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
            euclideanMatrixOf_AllCluster = \
                np.concatenate((euclideanMatrixOf_AllCluster, euclideanMatrix), axis=1)
        clusterMatrix = np.ravel(np.argmin(np.matrix(euclideanMatrixOf_AllCluster), axis=1))
        listof_ClusterMember = [[] for i in range(k)]
        for i in range(0, data.shape[0]):  # assign data to cluster regarding cluster matrix
            listof_ClusterMember[np.asscalar(clusterMatrix[i])].append(data[i, :])
        # calculate new centroid
        newCentroid = np.ndarray(shape=(0, centroidInitialized.shape[1]))
        for i in range(0, nCluster):
            memberof_Cluster = np.asmatrix(listof_ClusterMember[i])
            centroidOf_Cluster = memberof_Cluster.mean(axis=0)
            newCentroid = np.concatenate((newCentroid, centroidOf_Cluster), axis=0)
        # break when converged
        print("iter: ", iterationCounter)
        print("centroid: ", newCentroid)
        if ((centroidInitialized == newCentroid).all()):
            break
        # update new centroid
        centroidInitialized = newCentroid
        plotClusterResult(listof_ClusterMember, centroidInitialized, str(iterationCounter), 0)
        time.sleep(1)
    return listof_ClusterMember, centroidInitialized


centroidInitialized = initializeCentroid(input, Centroid_Initialization_Method, k)
clusterResults, centroid = kMeans(input, centroidInitialized)
plotClusterResult(clusterResults, centroid, str(iterationCounter) + " (converged)", 1)
```

ALGORITHM OF KERNEL K MEANS

1: **Input**:
- $\mathcal{D} = \{x_1, \ldots, x_n\}, x_i \in \Re^d$: the set of n d-dimensional data points to be clustered
- $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$: the kernel function
- C: the number of clusters
- m: the number of randomly sampled data points ($C < m \ll n$)
- $MAXITER$: maximum number of iterations

2: **Output**: Cluster membership matrix $U \in \{0,1\}^{C \times n}$

3: Sample m data points from $\mathcal{D}$, denoted by $\hat{\mathcal{D}} = \{\hat{x}_1, \ldots, \hat{x}_m\}$.

4: Compute matrices $K_B = [\kappa(x_i, \hat{x}_j)]_{n \times m}$, $\hat{K} = [\kappa(\hat{x}_i, \hat{x}_j)]_{m \times m}$, and $T = K_B \hat{K}^{-1}$.

5: Randomly initialize the membership matrix U, ensuring that $U^\top 1 = 1$.

6: Set $t = 0$.

7: **repeat**

8: Set $t = t + 1$.

9: Compute the ℓ_1 normalized membership matrix $\hat{U}$ by $\hat{U} = [\text{diag}(U1)]^{-1} U$.

10: Calculate $\alpha = \hat{U} T$.

11: **for** $i = 1, \ldots, n$ **do**

12: Find the closest cluster center k_* for x_i by

$$k_* = \underset{k \in [C]}{\text{argmin}}\, \alpha_k^\top \hat{K} \alpha_k - 2\varphi_i^\top \alpha_k,$$

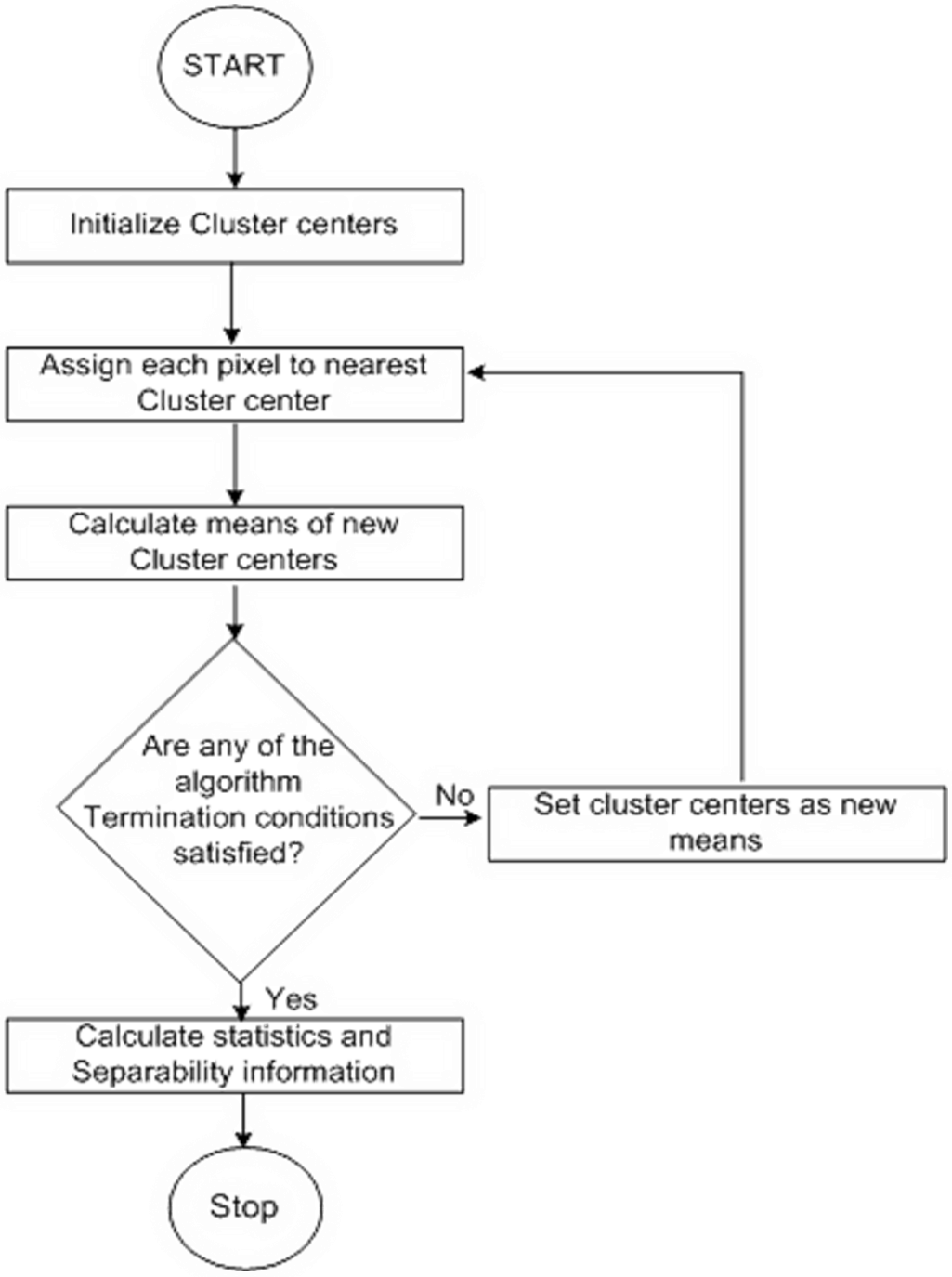
where α_k and φ_i are the k^{th} and i^{th} rows of matrices α and K_B, respectively.

13: Update the i^{th} column of U by $U_{k,i} = 1$ for $k = k_*$ and zero otherwise.

14: **end for**

15: **until** the membership matrix U does not change, or $t > MAXITER$

FLOWCHART OF KERNEL K MEANS



6.2 IMPLEMENTED CODE FOR KERNEL K-MEANS

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.pyplot import cm
import time

#Below code has comments to add or remove input dataset to form different non-linear clusters
for each run.
#The code can take only .txt dataset because .csv or .tsv type data are not hashable.
#No.of.Iterations vary until final convergence based on the datasets inputted.

#filePath1 = "test1_data.txt"
filePath2 = "test1_data.txt"
#filePath2 = "test2_data.txt"
#filePath2 = "quiz_A.data"
#filePath2 = "self_test.data"
#filePath2 = "xclara.txt"
#filePath2 = "xclara.csv"
#dataTesting1 = np.loadtxt(filePath1, delimiter=" ")
dataTesting2 = np.loadtxt(filePath2, delimiter=" ")
#dataTesting1 = pd.read_csv('xclara.csv')
#dataTesting2 = pd.read_csv('xclara.csv')

#params
k = 2 #number of cluster
var = 5 #var in RFB kernel
iterationCounter = 0
input = dataTesting2
Centroid_initializationMethod = "byOriginDistance" #options = random, byCenterDistance,
byOriginDistance

def initializeCluster(dataInput, nCluster, method):
    listOf_ClusterMember = [[] for i in range(nCluster)]
    if (method == "random"):
        shuffledDataIn = dataInput
        np.random.shuffle(shuffledDataIn)
        for i in range(0, dataInput.shape[0]):
            listOf_ClusterMember[i%nCluster].append(dataInput[i,:])
    if (method == "byCenterDistance"):
        center = np.matrix(np.mean(dataInput, axis=0))
        repeatedCentroid = np.repeat(center, dataInput.shape[0], axis=0)
        deltaMatrix = abs(np.subtract(dataInput, repeatedCentroid))
        euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
```

```
        dataNew = np.array(np.concatenate((euclideanMatrix, dataInput), axis=1))
        dataNew = dataNew[np.argsort(dataNew[:, 0])]
        dataNew = np.delete(dataNew, 0, 1)
        divider = dataInput.shape[0]/nCluster
        for i in range(0, dataInput.shape[0]):
            listOf_ClusterMember[np.int(np.floor(i/divider))].append(dataNew[i,:])
    if (method == "byOriginDistance"):
        origin = np.matrix([[0,0]])
        repeatedCentroid = np.repeat(origin, dataInput.shape[0], axis=0)
        deltaMatrix = abs(np.subtract(dataInput, repeatedCentroid))
        euclideanMatrix = np.sqrt(np.square(deltaMatrix).sum(axis=1))
        dataNew = np.array(np.concatenate((euclideanMatrix, dataInput), axis=1))
        dataNew = dataNew[np.argsort(dataNew[:, 0])]
        dataNew = np.delete(dataNew, 0, 1)
        divider = dataInput.shape[0]/nCluster
        for i in range(0, dataInput.shape[0]):
            listOf_ClusterMember[np.int(np.floor(i/divider))].append(dataNew[i,:])

    return listOf_ClusterMember

def RbfKernel(data1, data2, sigma):
    delta =abs(np.subtract(data1, data2))
    squaredEuclidean = (np.square(delta).sum(axis=1))
    result = np.exp(-(squaredEuclidean)/(2*sigma**2))
    return result

def thirdTerm(memberOf_Cluster):
    result = 0
    for i in range(0, memberOf_Cluster.shape[0]):
        for j in range(0, memberOf_Cluster.shape[0]):
            result = result + RbfKernel(memberOf_Cluster[i, :], memberOf_Cluster[j, :], var)
    result = result / (memberOf_Cluster.shape[0] ** 2)
    return result

def secondTerm(dataI, memberOf_Cluster):
    result = 0
    for i in range(0, memberOf_Cluster.shape[0]):
        result = result + RbfKernel(dataI, memberOf_Cluster[i,:], var)
    result = 2 * result / memberOf_Cluster.shape[0]
    return result

def plotResult(listOf_ClusterMembers, centroid, iteration, converged):
    n = listOf_ClusterMembers.__len__()
    color = iter(cm.rainbow(np.linspace(0, 1, n)))
    plt.figure("result")
```

```
    plt.clf()
    plt.title("iteration-" + iteration)
    for i in range(n):
        col = next(color)
        memberOf_Cluster = np.asmatrix(listOf_ClusterMembers[i])
        plt.scatter(np.ravel(memberOf_Cluster[:, 0]), np.ravel(memberOf_Cluster[:, 1]),
marker=".", s=100, c=col)
    color = iter(cm.rainbow(np.linspace(0, 1, n)))
    for i in range(n):
        col = next(color)
        plt.scatter(np.ravel(centroid[i, 0]), np.ravel(centroid[i, 1]), marker="*", s=400, c=col,
edgecolors="black")
    if (converged == 0):
        plt.ion()
        plt.show()
        plt.pause(0.1)
    if (converged == 1):
        plt.show(block=True)

def kMeansKernel(data, Centroid_initializationMethod):
    global iterationCounter
    memberInitilized_toCluster = initializeCluster(data, k, Centroid_initializationMethod)
    nCluster = memberInitilized_toCluster.__len__()
    #looping until converged
    while(True):
        # calculate centroid, only for visualization purpose
        centroid = np.ndarray(shape=(0, data.shape[1]))
        for i in range(0, nCluster):
            memberOf_Cluster = np.asmatrix(memberInitilized_toCluster[i])
            centroidof_Cluster = memberOf_Cluster.mean(axis=0)
            centroid = np.concatenate((centroid, centroidof_Cluster), axis=0)
        #plot result in every iteration
        plotResult(memberInitilized_toCluster, centroid, str(iterationCounter), 0)
        oldTime = np.around(time.time(), decimals=0)
        kernelResultClusterOf_AllCluster = np.ndarray(shape=(data.shape[0], 0))
        #assign data to cluster whose centroid is the closest one
        for i in range(0, nCluster):#repeat for all cluster
            term3 = thirdTerm(np.asmatrix(memberInitilized_toCluster[i]))
            matrixTerm3 = np.repeat(term3, data.shape[0], axis=0); matrixTerm3 =
np.asmatrix(matrixTerm3)
            matrixTerm2 = np.ndarray(shape=(0,1))
            for j in range(0, data.shape[0]): #repeat for all data
                term2 = secondTerm(data[j,:], np.asmatrix(memberInitilized_toCluster[i]))
                matrixTerm2 = np.concatenate((matrixTerm2, term2), axis=0)
            matrixTerm2 = np.asmatrix(matrixTerm2)
```

```
            kernelResultClusterI = np.add(-1*matrixTerm2, matrixTerm3)
            kernelResultClusterOf_AllCluster =\
                np.concatenate((kernelResultClusterOf_AllCluster, kernelResultClusterI), axis=1)
        clusterMatrix = np.ravel(np.argmin(np.matrix(kernelResultClusterOf_AllCluster), axis=1))
        listOf_ClusterMember = [[] for l in range(k)]
        for i in range(0, data.shape[0]):#assign data to cluster regarding cluster matrix
            listOf_ClusterMember[np.asscalar(clusterMatrix[i])].append(data[i,:])
        for i in range(0, nCluster):
            print("Cluster member numbers-", i, ": ", listOf_ClusterMember[0].__len__())
        #break when converged
        boolAcc = True
        for m in range(0, nCluster):
            prev = np.asmatrix(memberInitilized_toCluster[m])
            current = np.asmatrix(listOf_ClusterMember[m])
            if (prev.shape[0] != current.shape[0]):
                boolAcc = False
                break
            if (prev.shape[0] == current.shape[0]):
                boolPerCluster = (prev == current).all()
            boolAcc = boolAcc and boolPerCluster
            if(boolAcc==False):
                break
        if(boolAcc==True):
            break
        iterationCounter += 1
        #update new cluster member
        memberInitilized_toCluster = listOf_ClusterMember
        newTime = np.around(time.time(), decimals=0)
        print("iteration-", iterationCounter, ": ", newTime - oldTime, " seconds")
    return listOf_ClusterMember, centroid

clusterResult, centroid = kMeansKernel(input, Centroid_initializationMethod)
plotResult(clusterResult, centroid, str(iterationCounter) + ' (converged)', 1)
print("converged!")
```

RESULT ANALYSIS

7. RESULT ANALYSIS

7.1 RESULTS OF K-MEANS AND KERNEL K-MEANS OVER QUIZ_A DATASET

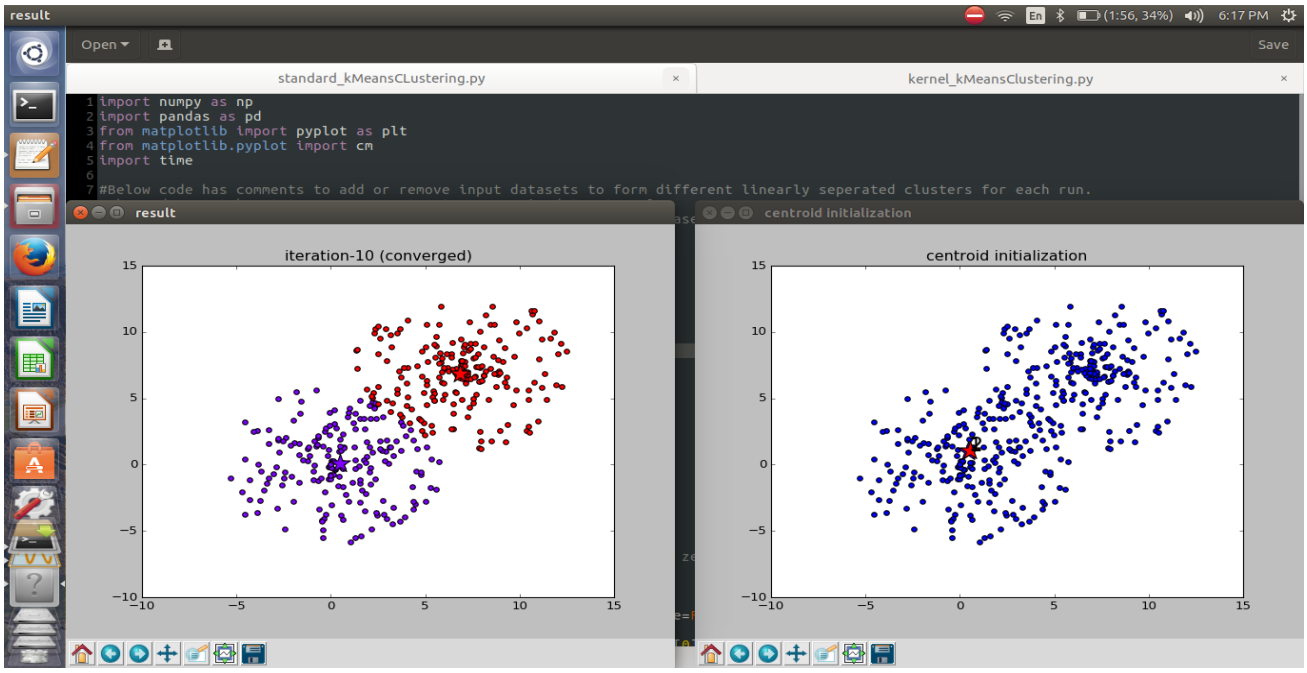


Fig 9: This figure shows the result of k-means algorithm implemented over quiz_A dataset.

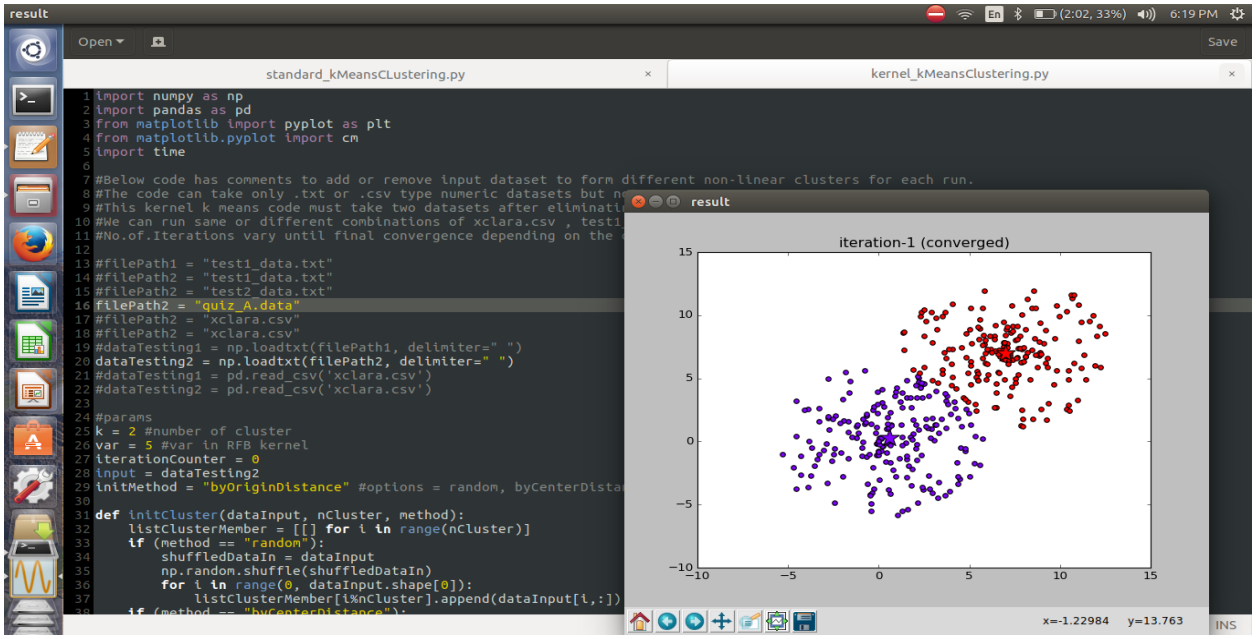


Fig 10: shows the result of kernel k-means algorithm implemented over quiz_A dataset.

7.2 RESULTS OF K-MEANS AND KERNEL K-MEANS OVER SELF_TEST DATASET

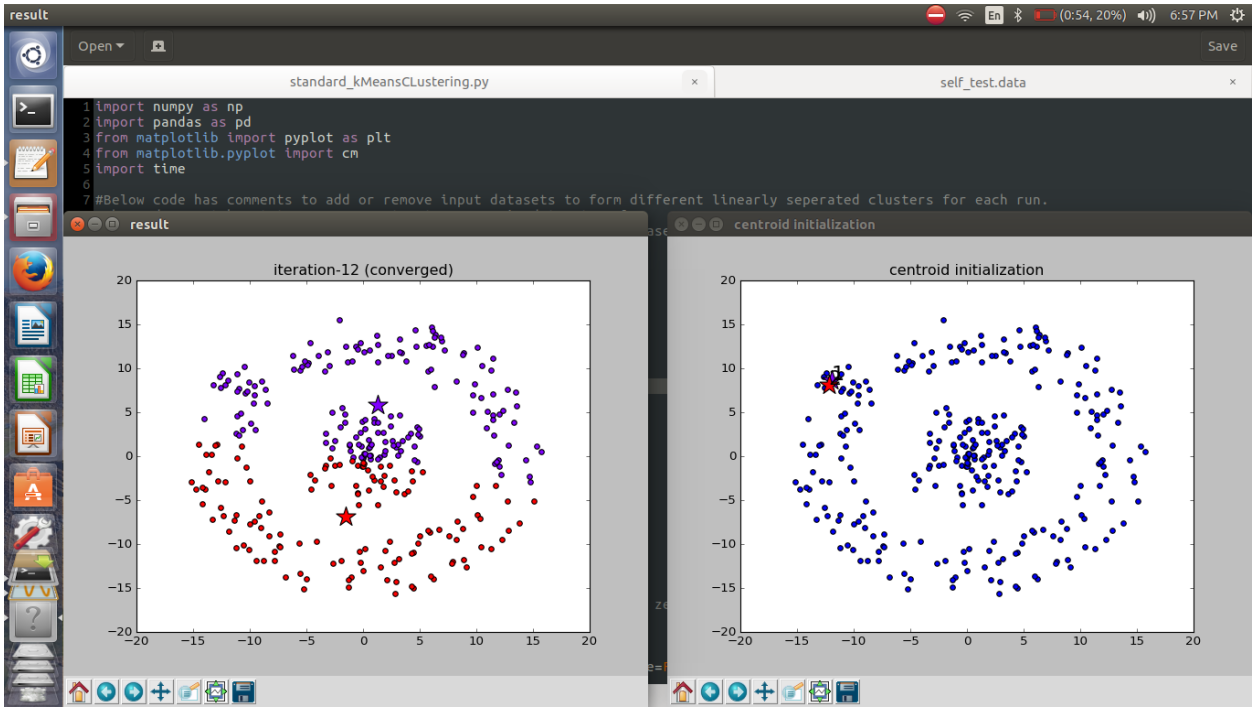


Fig 11: This figure shows the result of k-means algorithm implemented over self_test dataset.

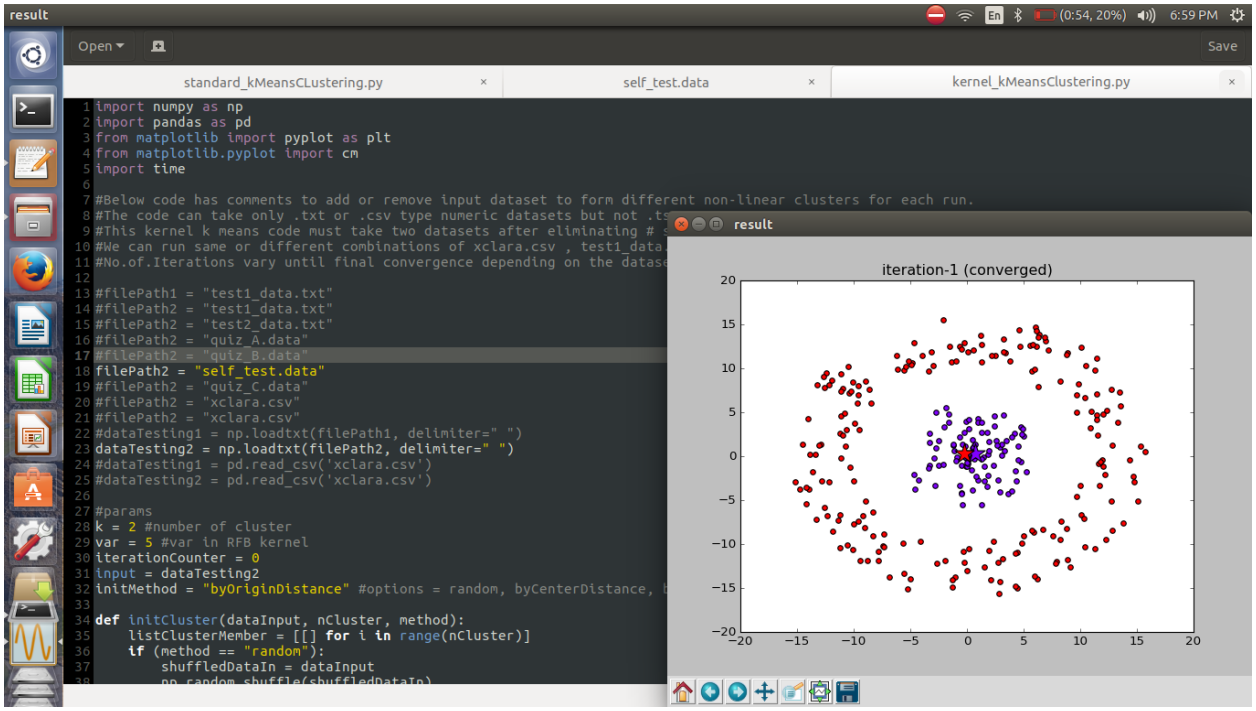


Fig 12: shows the result of kernel k-means algorithm implemented over self_test dataset

7.3 RESULTS OF K-MEANS AND KERNEL K-MEANS OVER TEST1 DATASET

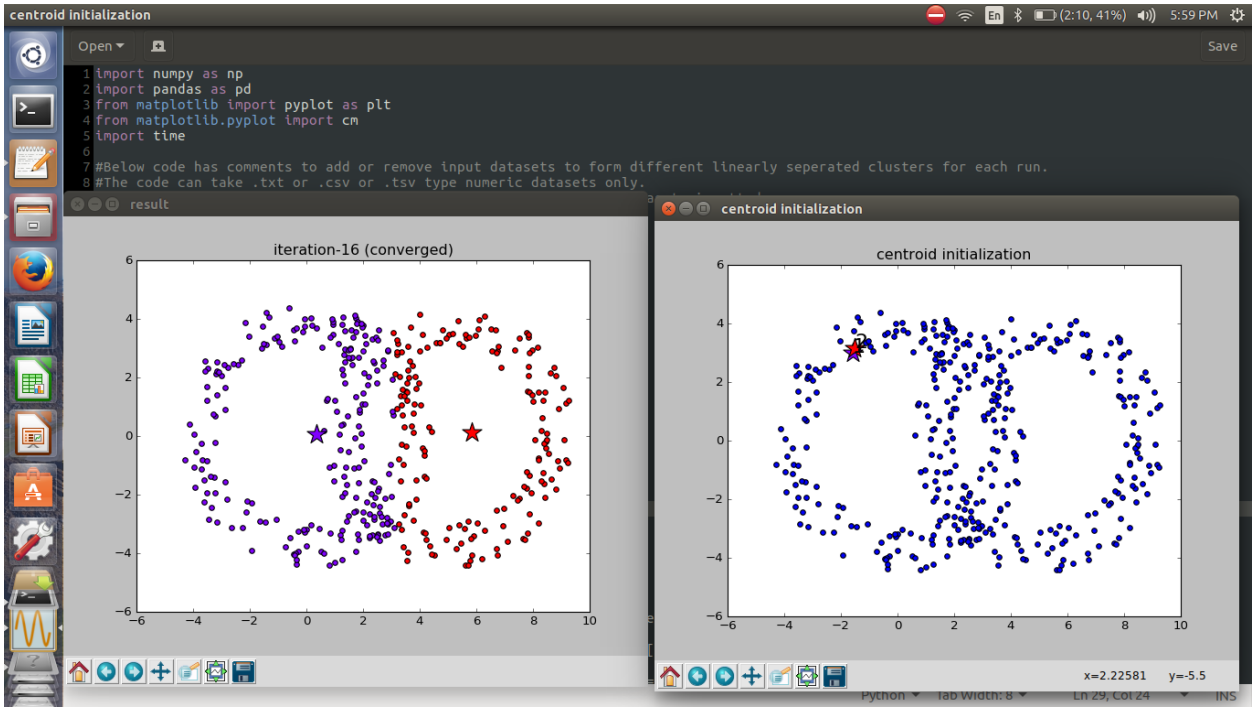


Fig 13: This figure shows the result of k-means algorithm implemented over test1_data dataset.

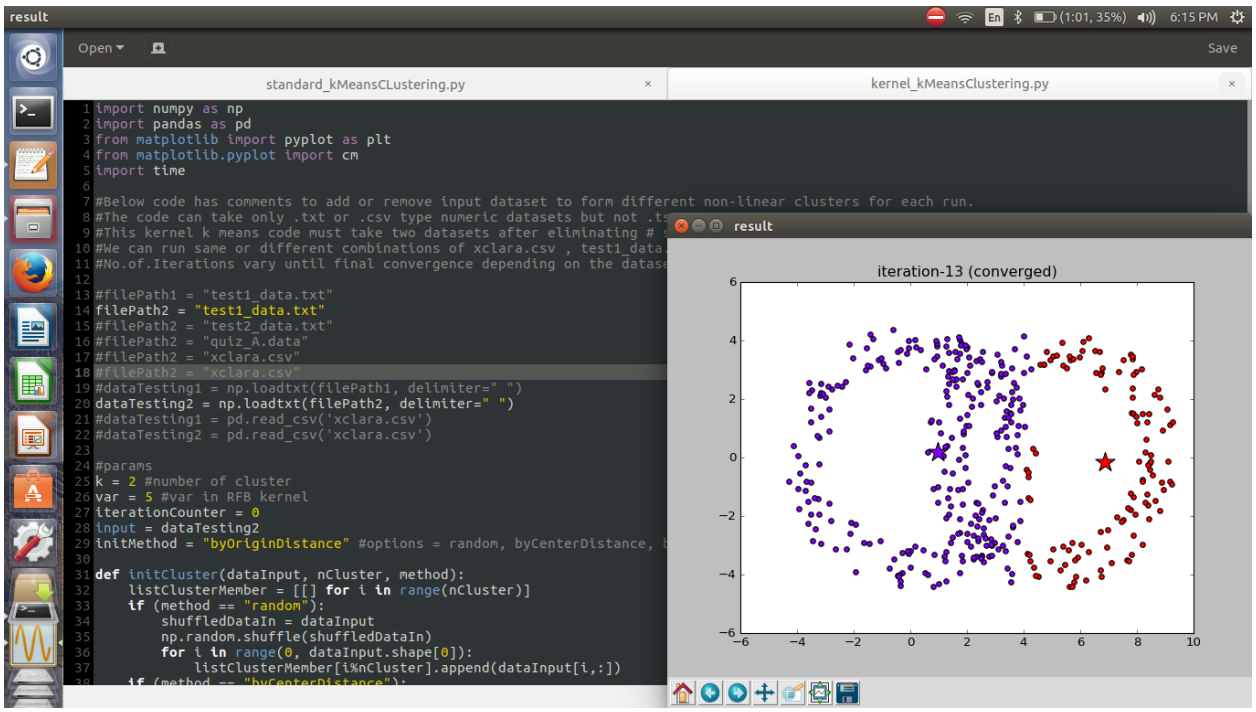


Fig 14: shows the result of kernel k-means algorithm implemented over test1_data dataset.

7.4 RESULTS OF K-MEANS AND KERNEL K-MEANS OVER TEST2 DATASET

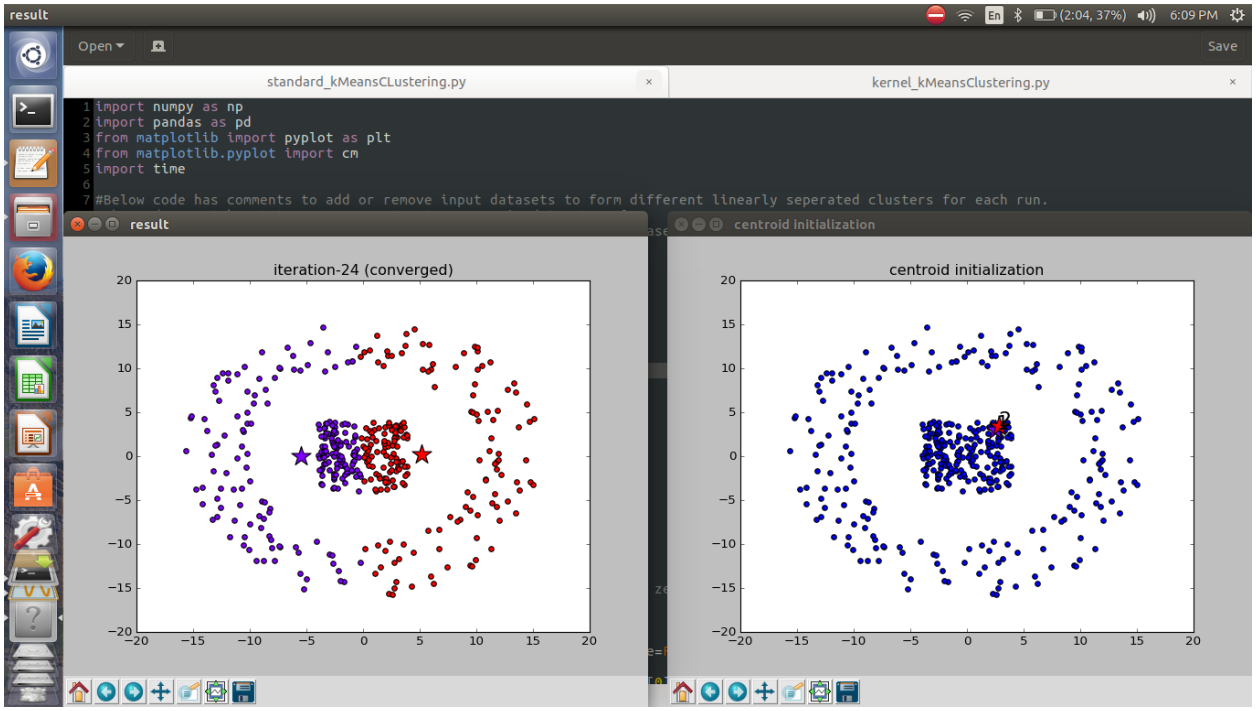


Fig 15: This figure shows the result of k-means algorithm implemented over test2_data dataset.

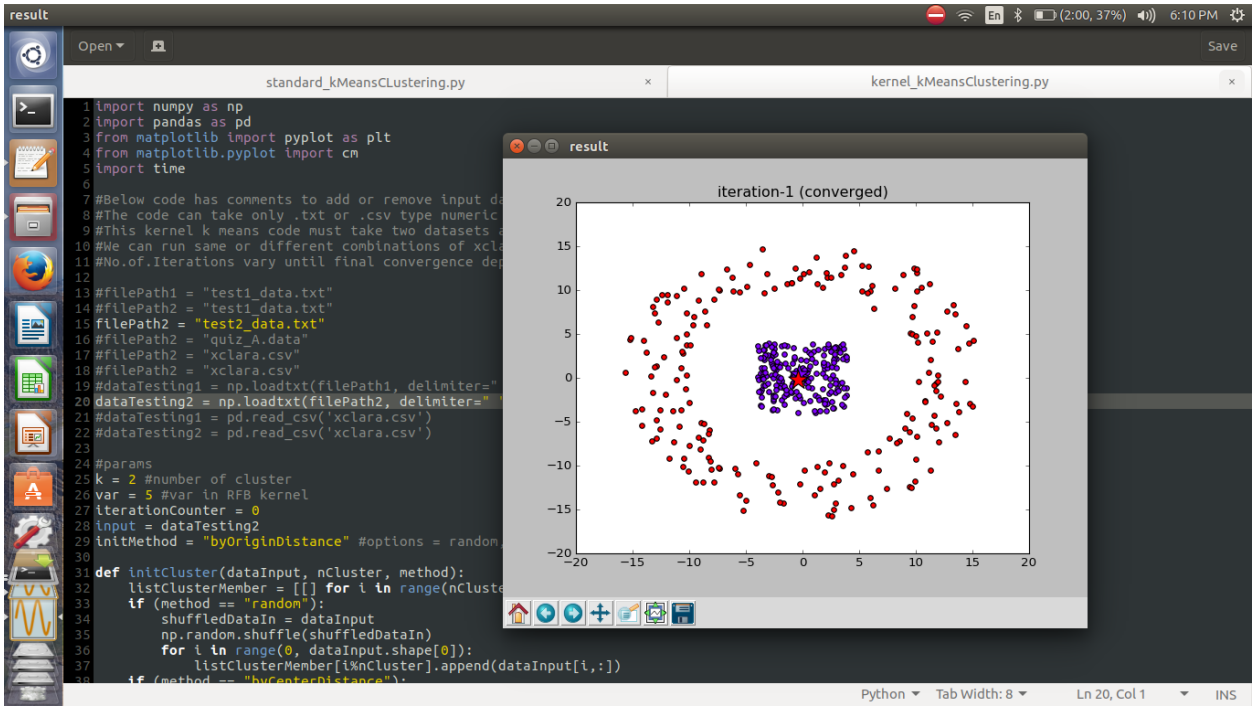


Fig 16: shows the result of kernel k-means algorithm implemented over test2_data dataset.

7.5 RESULTS OF K-MEANS AND KERNEL K-MEANS OVER XCLARA DATASET

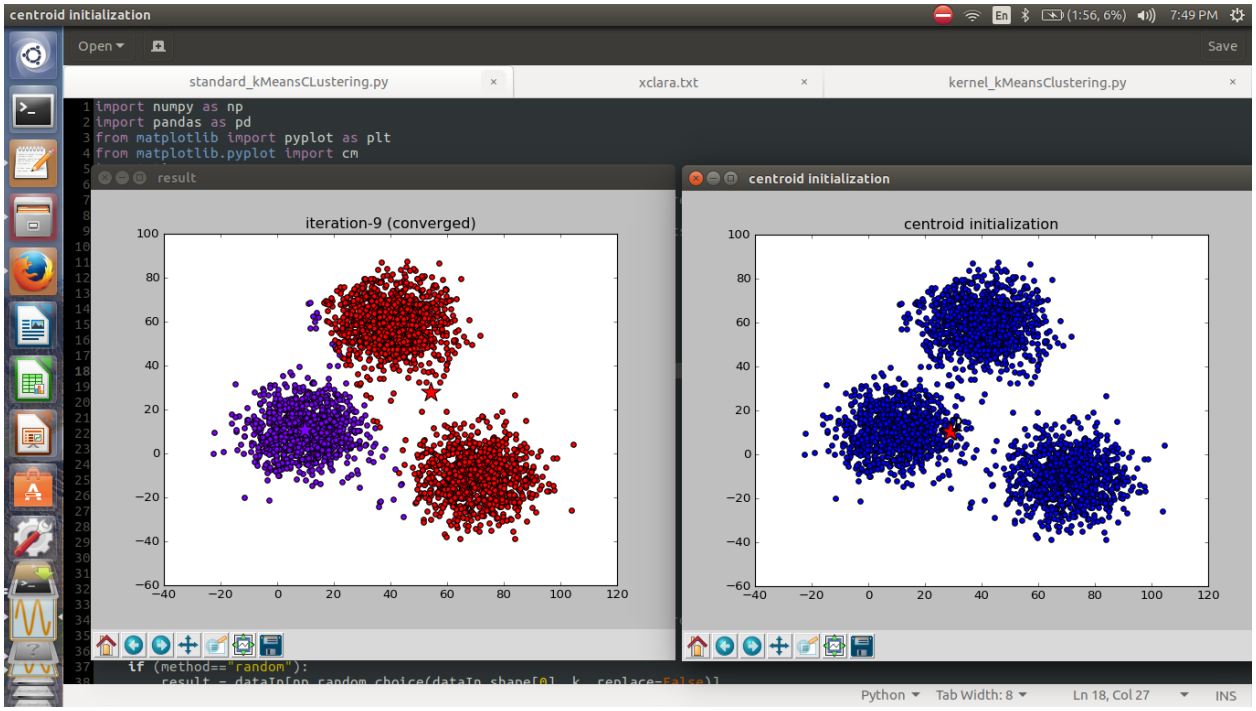


Fig 17: This figure shows the result of k-means algorithm implemented over xclara dataset.

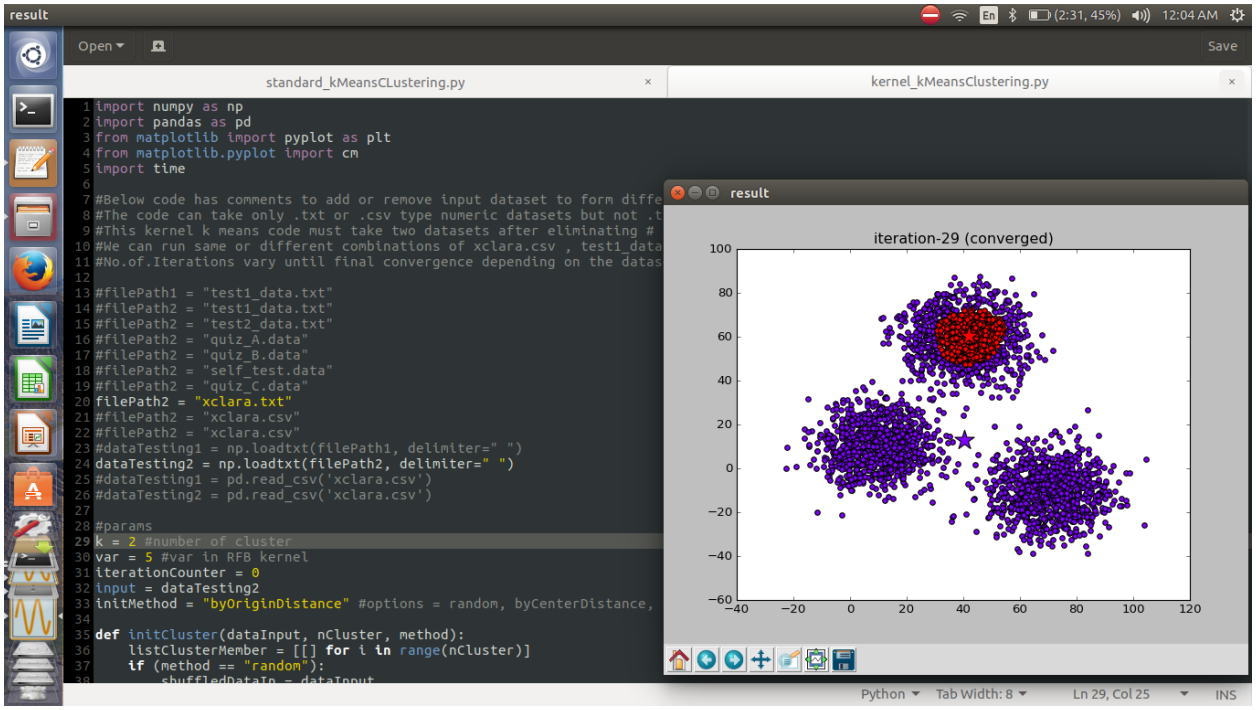


Fig 18: shows the result of kernel k-means algorithm implemented over xclara dataset.

Advantages of k-means

1) Fast, robust and easier to understand.

2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.

3) Gives best result when data set are distinct or well separated from each other.

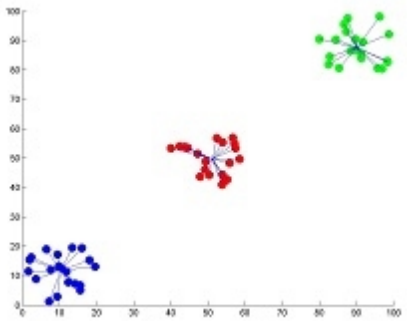


Fig 19: Showing the result of k-means for *'N'* = 60 and *'c'* = 3

Disadvantages of k-Means

1) Euclidean distance measures can unequally weight underlying factors.

2) The learning algorithm provides the local optima of the squared error function.

3) Randomly choosing of the cluster center cannot lead us to the fruitful result.

4) Applicable only when mean is defined i.e. fails for categorical data.

5) Unable to handle noisy data and outliers.

6) Algorithm fails for non-linear data set.

Advantages of kernel k-means

1) Algorithm is able to identify the non-linear structures.

2) Algorithm is best suited for real life data set.

Disadvantages of kernel k-means

1) Number of cluster centers need to be predefined.

2) Algorithm is complex in nature and time complexity is large.

TESTING AND VALIDATION

8. TESTING AND VALIDATION

8.1 INTRODUCTION

The completion of a system is achieved only after it has been thoroughly tested. Though this gives a feel the project is completed, there cannot be any project without going through this stage. Hence in this stage it is decided whether the project can undergo the real time environment execution without any break downs, therefore a package can be rejected even at this stage.

Testing is a set of activities that can be planned in advance and conducted systematically. The proposed system is tested in parallel with the software that consists of its own phases of analysis, implementation, testing and maintenance.

8.2 TESTING STRATEGIES

A Strategy for software testing integrates software test cases into a series of well planned steps that result in the successful construction of software. Software testing is a broader topic for what is referred to as Verification and Validation. Verification refers to the set of activities that ensure that the software correctly implements a specific function. Validation refers he set of

activities that ensure that the software that has been built is traceable to customer's requirements

UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design that is the module. Using procedural design description as a guide, important control paths are tested to uncover errors within the boundaries of the module. The unit test is normally white box testing oriented and the step can be conducted in parallel for multiple modules.

INTEGRATION TESTING

Integration testing is a systematic technique for constructing the program structure, while conducting test to uncover errors associated with the interface. The objective is to take unit tested methods and build a program structure that has been dictated by design.

TOP-DOWN INTEGRATION

Top down integrations is an incremental approach for construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control program. Modules subordinate to the main program are incorporated in the structure either in the breath-first or depth-first manner.

BOTTOM-UP INTEGRATION

This method as the name suggests, begins construction and testing with atomic modules i.e., modules at the lowest level. Because the modules are integrated in the bottom up manner the processing required for the modules subordinate to a given level is always available and the need for stubs is eliminated.

VALIDATION TESTING

At the end of integration testing software is completely assembled as a package. Validation testing is the next stage, which can be defined as successful when the software functions in the manner reasonably expected by the customer. Reasonable expectations are those defined in the software requirements specifications. Information contained in those sections form a basis for validation testing approach.

SYSTEM TESTING

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all system elements have been properly integrated to perform allocated functions.

PERFORMANCE TESTING

This method is designed to test runtime performance of software within the context of an integrated system.

We have observed various logical and syntactical errors and we have rectified them with all the above given strategies.

CONCLUSION

9. CONCLUSION

- Data clustering plays an indispensable role in various fields such as computer science, medical science, computational biology, mobile communication and economics.
- Its application in computer vision such as digital image clustering, video segmentation, and color image segmentation is wide.
- Due to its well found usage in various fields, it is clear that nonlinear clustering has been and will continue to be a hot research topic

REFERENCES AND BIBILIOGRAPHY

10. REFERENCES AND BIBILIOGRAPHY:

- DATAMINING WEBSITES, TUTORIALS
- CLUSTER ANALYSIS , CLUSTERING ALGORITHMS

IEEE REFERED PAPAERS

- Jian-Huang Lai and **Chang-Dong Wang**. Kernel and Graph: Two Approaches for Nonlinear Competitive Learning Clustering
- **Chang-Dong Wang**, Jian-Huang Lai and Dong Huang. Kernel-Based Clustering with Automatic Cluster Number Selection
- **Radha Chitta , Kernel Based Clustering BigData_PhD15.**
- **Chang-Dong Wang**, Jian-Huang Lai and Jun-Yong Zhu. Conscience Online Learning: An Efficient Approach for Robust Kernel-Based Clustering,

WEBSITES

- WWW.PYTHON.ORG
- WWW.PYTHONTUTORIALS.COM
- PYTHON WIKIPEDIA

BOOK REFERED

- HEAD FIRST BOOK OF PYTHON