Clustering Project Report

[toc]

Report Requirement

The report will contain:

- Details about the implementation of your algorithms, including the decisions made during the implementation and the setup of the different parameters.
- The evaluation of the algorithms, including tables and/or graphs that show your results with comments about them.
- Justify your results and, in addition, reason each one of the questions defined above in your evaluation. Moreover, add any comment or observation that you consider important from your results.
- It is extremely important that you explain how to execute your code. Moreover, call files from the relative path to the project, not the global paths of your computer.**

Explain Data

- Which information can be obtained for each data set using each algorithm? Is it the same or not?
- Which clustering algorithm do you consider is the best one for datasets with categorical data, with numerical data and with mixed data?
- Did you find differences among algorithms? According to the data sets chosen, which algorithm gives you more advice for knowing the underlying information in the data set?
- Can you explain the setup that you have used for each algorithm?
- In the case of the K-Means and the other algorithms where you have to choose the K, which has been the best K value?

Introduction to Clustering

Cluster analysis is popular unsupervised learning method which divides data into groups (clusters) that are meaningful, useful, or both. If meaningful groups are the goal, then the clusters should capture the natural structure of the data. There are lots of specific clustering techniques in nowadays.

Hierarchical vs Partitional Clustering

The most commonly discussed distinction among different types of clustering is whether the set of clusters is nested or unnested, or in more traditional terminology, hierarchical or partitional.
A partitional clustering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. Whereas Hierarchical methods permit clusters to have sub-clusters, then we obtain a hierarchical clustering, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (sub-clusters), and the root of the tree is the cluster containing all the objects.
Basic approaches for generating a hierarchical clustering includes **bottom-up(agglomerative)** and **top-down(divisive)** methodologies. Agglomerative algorithm starts with each example in its own cluster and

iteratively combine them to form larger and larger clusters, while divisive algorithm starts with all the examples in a single cluster, and choose the best division by considering all the possible ways to divide the cluster into two.

Exclusive versus Overlapping versus Fuzzy

In the most general sense, an overlapping or non-exclusive clustering is used to reflect the fact that an object can simultaneously belong to more than one group (class). For example, in a fuzzy clustering, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn’t belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to any set with a weight that is between 0 and 1. In fuzzy clustering, we often impose the additional constraint that the sum of the weights for each object must equal 1.)

Different methods to group data

We could use different methods to group large sets of data into small sets of clusters of similar data, which including follows:

- Based on connectivity: Hierarchical clustering
- Based on centroids: K-means
- Distribution-based models: Mixture models, Expectation-Maximization
- Density models: DBScan, Optics
- Subspace models: Biclustering
- Group models
- Graph-based

Road Map to our algorithms

we use the following two simple, but important techniques to introduce many of the concepts involved in cluster analysis.

- **K-means:** This is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids.

- **Fuzzy_c_means:** As we have introduced in the earlier chapter, fuzzy clustering is a form of clustering in which each data point can belong to more than one cluster. Similarly, probabilistic clustering techniques compute the probability with which each point belongs to each cluster, and these probabilities must also sum to 1, since a fuzzy or probabilistic clustering does not address true multiclass situations, such as the case of a student employee, where an object belongs to multiple classes.

- **Comparison in theory:** Fuzzy c-means clustering can be considered a better algorithm compared to the k-Means algorithm. Unlike the k-Means algorithm where the data points exclusively belong to one cluster, in the case of the fuzzy c-means algorithm, the data point can belong to more than one cluster with a likelihood. **Fuzzy c-means clustering gives comparatively better results for overlapped data sets**.

Data pre-processing

Dataset Selection

Since k means and fuzzy c means algorithms only works on numerical data values, so we choose our datasets among the datasets of which all feature values is numerical. Based on the requirement, we need to analyse the behaviour of different clustering algorithms in well-known data sets from the UCI repository. But after we check dataset information one by one, we found that most of the datasets which have tens of features together with a limited number of samples. **For machine learning algorithms, we are actually producing meaningful conclusions based a meaningful sample distribution.** As a rough rule of thumb, your model should train on at least an order of magnitude more examples than trainable parameters. Simple models on large data sets generally beat fancy models on small data sets.

So we select two datasets with considerably larger number of samples compared to others - ***pen-based.arff*** and ***satimage.arff***.

Dataset Name	No. of features	No. of Sample
pen_base	16	10992
satimage	36	6435

Even the dataset size for these two datasets is closed to each other, but the satimage data set has more than twice the number of features in pen_base dataset.

Data pre-processing pipeline

Data pre-processing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, the greater is the reliability of the produced results. **Incomplete**, **noisy**, and **inconsistent** data are the inherent nature of real-world datasets. Data pre-processing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise, and resolving inconsistencies.

There are many stages involved in data pre-processing: **1)Data Cleaning**, **2)Data Integration**, **3)Data Transformation**, **4)Data Reduction**.

- **Data cleaning** attempts to impute missing values, smooth out noise, resolve inconsistencies, removing outliers in the data. We have implement **filling NA and dropping NA, dropping duplicates** in the **"*./dataset/data_preprocessing_numerical.py*"** file.
- **Data integration** integrates data from a multitude of sources into a single data warehouse. We haven't touched this part in our project, but it is critical procedure for merging data sets from different sources.
- **Data transformations**, such as normalization, may be applied in some cases. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values or losing information. There are several ways to perform normalization including **standardisation by Z scores**, **normalized into -1 to 1 range by mean value**, **scaling value between 0 and 1 with max and min values**, **Scaling values by considering the whole feature vector to be of unit length**.
- **Data reduction** can reduce the data size by dropping out redundant features. Feature selection and feature extraction techniques can be used. We do not cover this topic in work 1, but we will continue further steps for data pre-processing by using data reduction techniques in work 2.
- One extra step we do for data pre-processing is **shuffling data**.

- We must notice that the **true labels from raw data is either all text or binary format**. So here we utilize LabelEncoder from sklearn library to **transform target labels into value between 0 and n_classes-1**.

by running the main function in **"./dataset/data_preprocessing_numerical.py"**, we can successfully load and pre-process datasets into 2 separate pickle file, one with purely numerical features and the other one with labels with integer values.

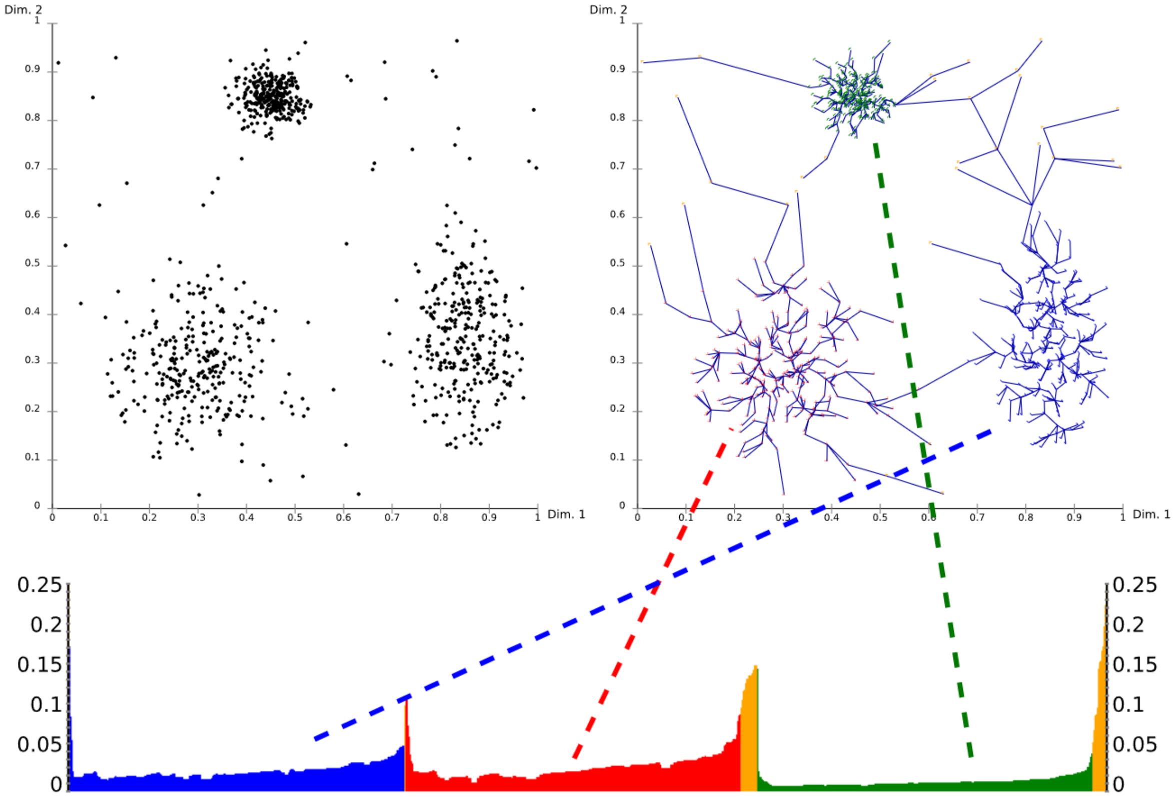
Algorithm implementation and evaluation

In this project, we are required to implement the following algorithms:

1. **OPTICS with sklearn library**
2. **K-Means implemented by our own code)**
3. **Fuzzy c means implemented by our own code)**

Optics implementation and evaluation

Ordering points to identify the clustering structure (OPTICS)[1] is an algorithm for finding density-based clusters in spatial data. Its basic idea is similar to DBSCAN,[3] but it addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clusters in data of varying density. To do so, the points of the database are (linearly) ordered such that spatially closest points become neighbors in the ordering. Additionally, a special distance is stored for each point that represents the density that must be accepted for a cluster so that both points belong to the same cluster. OPTICS



Reachability Plot

Using a ***reachability-plot*** (a special kind of dendrogram), the hierarchical structure of the clusters can be obtained easily. It is a 2D plot, with the ordering of the points as processed by OPTICS on the x-axis and the reachability distance on the y-axis. Since points belonging to a cluster have a low reachability distance to their nearest neighbor, the clusters show up as valleys in the reachability plot. The deeper the valley, the denser the cluster.

for optics algorithms we can call corresponding API as follow:
`class sklearn.cluster.OPTICS(*, min_samples=5, max_eps=inf, metric='minkowski', p=2, ...)`

There are two parameters, **max_eps** and **eps** which will be used to determine the maximum distance between two samples for one to be considered as in the neighborhood of the other. Based on our understanding, **the epsilon value in OPTICS is solely to limit the runtime complexity when using index structures..** So we choose to use default value which is np.inf to run our model. Another parameter **min_cluster_size** (to determine minimum number of samples in an OPTICS cluster) for which we do not have enough knowledge to tune, so we keep the default values. The only parameter we tune is **min_samples**, here we built a list $[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$ to investigate the performance of OPTICS.

- **Performance at different min_samples**

> dfd
> fds
> dfd 图图图 fds
> vdf

Kmeans implementation

The algorithm for kmeans is iterative which groups the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to a single specific group. The main objective is to make the intra-cluster vector points as equal as possible while also keeping the clusters as far different as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way our kmeans algorithm works is as follows:

1. **Training**

    1. Specify number of clusters K.
    2. Initialize kmeans centroids by random selecting each feature value between its minimum and maximum value.
    3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.
        - Compute the sum of the squared distance between data points and all centroids.
        - Assign each data point to the closest cluster (centroid).
        - Compute the centroids for the clusters by taking the average of all the data points in each cluster.

2. **Inference**

1. Compute the sum of the squared distance between data points and all centroids.
2. Assign each data point to the closest cluster (centroid)

Moreover, since there is **no right answer in terms of the number of clusters** that we should have in any problem, sometimes domain knowledge and intuition may help but usually that is not the case. In this methodology, we decide to evaluate how well the models are performing based on different K clusters by evaluation metrics built by us.
Our number of clusters setting for kmeans:

1. we use $[2, 3, 4, ..., 15]$ for pen_base dataset to evaluate model performance.
2. we use $[2, 3, 4, ..., 10]$ for satimage dataset to do the evaluation.

> **Few things to note here**: Given kmeans iterative nature and the random initialization of centroids at the start of the algorithm, different initializations may lead to different clusters since kmeans algorithm may stuck in a local optimum and may not converge to global optimum. Therefore, we **average 3 runs of kmeans algorithm on each dataset.**

- **Code Implementation** Our code implementation including the following components and logic in *models/kmeans.py*:
  - Compute distance between a sample and certain centriod

```
def distEclud(vecA, vecB):
 return np.sqrt(np.sum(np.power(vecA - vecB, 2)))
```

  - Initialize the cluster centroids
  - Main Logic of our algorithm. We set clusterChanged flag to check if any sample has changed membership, if $clusterChanged=False$, we will reach the end of training. If not, firstly we calculate minimum distance together with its index for all the samples with all centroids in the upper for loop. Later we calculate centroids positions according to the newly assigned indices (or cluster label) to samples. Finally we return the centroids and training cluster labels.

```
   while clusterChanged:
   clusterChanged = False
   for i in range(m):
       minDist = float(math.inf)

       minIndex = -1
       for j in range(k):
           distJI = distMeas(centroids[j,:],dataSet[i,:])
           if distJI < minDist:
               minDist = distJI
               minIndex = j

       if clusterAssment[i, 0] != minIndex: clusterChanged = True
       clusterAssment[i, :] = minIndex, minDist**2

   for cent in range(k):
       ptsInClust = dataSet[np.nonzero(clusterAssment[:,0].A == cent)[0]]
```

```
        centroids[cent, :] = np.mean(ptsInClust, axis=0)
    return centroids, clusterAssment.A[:, 0]
```

Fuzzy c means implementation

Fuzzy C-Means clustering is a soft clustering approach, where each data point is assigned a likelihood or probability score to belong to that cluster. The cost function of fuzzy c means is defined as below: $$ J_m = \sum_{i=1}^N \sum_{j=1}^C u_{ij}^m ||x_i - c_j||^2 , {\text{ 1 ≤ m < ∞}} \tag{1}$$ *where **m** is any real number greater than 1, $u_{ij}$ is the degree of membership of $x_i$ in the cluster $j$, $x_i$ is the ith of d-dimensional measured data, $c_j$ is the d-dimension center of the cluster, and* $||||$ is any norm expressing the similarity between any measured data and the center.*

The step-wise approaches of the Fuzzy c-means clustering algorithm which are wrapped in fuzzyCMean object in *models/fuzzy_c_means.py* are as follows:

- Firstly initialize the membership values to each cluster, these membership grades indicate the degree to which data points belong to each cluster.

  ```
  def initialize_membership_matrix(self):

      self.membership_matrix = self.random_generator.uniform(size=
  (self.n_samples, self.k_clusters))
      membership_normalizer = self.membership_matrix.sum(axis=1,
  keepdims=True)
      # normalize into
      self.membership_matrix = self.membership_matrix / membership_normalizer
  ```

- Secondly, Compute the distance between all samples and each centroid

  ```
  def _compute_distance(self):
      distances = np.sqrt(np.einsum("ijk->ij", (self.data[:, None, :] -
  self.centroids)**2))
      return distances
  ```

- Update membership matrix according to distances between samples and centroids: $$ u_{ij} = \frac {1}{\sum_{k=1}^C \left(\frac{||x_i - c_j||}{||x_i - c_k||} \right)^{\frac{2}{m-1}}} \tag{2}$$ *whereas k are the cluster index.*

  ```
  def update_membership_matrix(self):

      dist_samples_centroid = self._compute_distance()
      dist_samples_centroid = dist_samples_centroid ** float(2/(self.m-1))
      reshape_dist = dist_samples_centroid.reshape((self.data.shape[0], 1,
  -1))  # add a new dimension in the middle
      dist_on_all_centroids = reshape_dist.repeat(reshape_dist.shape[-1],
  ```

```
axis=1)
    denominator = dist_samples_centroid[:, :, np.newaxis] /
dist_on_all_centroids
    self.membership_matrix = 1 / denominator.sum(2)
```

- Update cluster centers according to obtained membership matrix in last step, With fuzzy c-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster, or, mathematically. $$c_j = \frac { \sum_{i=1}^N u_{ij} * x_i}{\sum_{i=1}^N u_{ij}^m}$$

```
def find_centroids(self):
    """Update cluster centers"""
    u_power_m = self.membership_matrix ** self.m
    numerator = np.dot(self.data.T, u_power_m)
    denominator = u_power_m.sum(axis=0)

    self.centroids = (numerator/denominator)
    self.centroids = self.centroids.T
```

- Training logic is organized in train function, and fuzzy partitioning is carried out through an iterative optimization of the objective function shown above by updating membership matrix and repeatly :

```
def train(self):
    self.initialize_membership_matrix()
    for _ in range(self.max_iter):
        old_membership_matrix = self.membership_matrix.copy()
        self.find_centroids()
        self.update_membership_matrix()

        if np.linalg.norm(self.membership_matrix - old_membership_matrix) <
self.epsilon:
            break
```

- This iteration will stop when, where **$\epsilon$** is a termination criterion between 0 and 1, or the maximum iteration steps are reached. This procedure converges to a local minimum or a saddle point of $J_m$.

Performance evaluation

The main goal of clustering approaches is to obtain **high intra-cluster similarity** and low **inter-cluster similarity** (objects in the same cluster are more similar than the objects in different clusters). Clustering validation has long been recognized as one of the vital issues essential to the success of clustering applications. In general, **clustering validation can be categorized into two classes, external clustering validation and internal clustering validation.**

Internal clustering validation

The internal measures evaluate the goodness of a clustering structure without respect to external information [4]. Internal validation measures can be used to choose the best clustering algorithm as well as the optimal cluster number without any additional information. In practice, **external information such as class labels is often not available in many application scenarios. Therefore, in the situation that there is no external information available, internal validation measures are the only option for cluster validation.** As the goal of clustering is to make objects within the same cluster similar and objects in different clusters distinct, internal validation measures are often based on the following two criteria:

- **Compactness.** It measures how closely related the objects in a cluster are. A group of measures evaluate cluster compactness based on variance. Lower variance indicates better compactness.
- **Separation.** It measures how distinct or well-separated a cluster is from other clusters. For example, the pairwise distances between cluster centers or the pairwise minimum distances between objects in different clusters are widely used as measures of separation. Also, measures based on density are used in some indices.

SSW & SSB

Minimizing Sum of Squares of the distance from the centroid of the cluster for cluster points within the cluster (SSW) and maximizing Sum of Square distance between the centroids of different clusters (SSB) are two generally used quality parameters of the clustering technique.

- **SSW** calculation $$SSW = \frac{1}{N} \sum_{i=1}^N ||x_i - C_{p_i}|| \tag{1}$$ where $ i = { 1, 2, \ldots, N}$is the set of clusters and C_{p_i} is corresponding cluster for sample i.

- **SSB** calculation $$SSW = \frac{1}{M(M-1)} \sum_{i=1}^M \sum_{j=1,j≠i}^M ||x_i - C_{p_i}|| \tag{2}$$ where $ i = { 1, 2, \ldots, N}$is the set of clusters and C_{p_i} is corresponding cluster for sample i, and M is number of samples.

Other concerns

> ***One thing to notice: Elbow Method*** Elbow method gives us an idea on what a good k number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters' centroids. We pick k at the spot where SSE starts to flatten out and forming an elbow. We'll use the geyser dataset and evaluate SSE for different values of k and see where the curve might form an elbow and flatten out.

Davies-Bouldin index

This is an internal evaluation scheme, where the validation of how well the clustering has been done is made using quantities and features inherent to the dataset. This has a drawback that a good value reported by this method does not imply the best information retrieval.

External clustering validation

Purity

An example of external validation measure is entropy, which evaluates the "purity" of clusters based on the given class labels [3]

Purity is a simple and transparent evaluation measure, and measures the extent to which a cluster contains objects of a single class. We assign a label to each cluster based on the most frequent class in it. Then the purity becomes the number of correctly matched class and cluster labels divided by the number of total data points.

$$P = \frac{1}{N} \sum_{k} \underset {j}{max} |w_k \bigcap c_j| \tag{1}$$
where $\Omega = { \omega_1, \omega_2, \ldots, \omega_K }$is the index of index of samples and set of classes. We interpret ω_k as the set of documents in ω_k and c_j as the set of documents in c_j in above equation.

Adjusted Rand Index

The adjusted Rand index is the corrected-for-chance version of the Rand index.Such a correction for chance establishes a baseline by using the expected similarity of all pair-wise comparisons between clusterings specified by a random model.

K means evaluation

We implement our K means algorithm exactly

Setting Hyperparameters

We

Fuzzy c means evaluation result

[1]Kriegel, Hans-Peter; Kröger, Peer; Sander, Jörg; Zimek, Arthur (May 2011). "Density-based clustering". Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 1 (3): 231–240. doi:10.1002/widm.30. [2]Mihael Ankerst; Markus M. Breunig; Hans-Peter Kriegel; Jörg Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure. ACM SIGMOD international conference on Management of data. ACM Press. pp. 49–60. CiteSeerX 10.1.1.129.6542. [3]Martin Ester; Hans-Peter Kriegel; Jörg Sander; Xiaowei Xu (1996). Evangelos Simoudis; Jiawei Han; Usama M. Fayyad (eds.). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231.

- Internal Evaluation [4] P.-N. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining. USA: Addison-Wesley Longman, Inc., 2005.

- Range Index W. M. Rand (1971). "Objective criteria for the evaluation of clustering methods". Journal of the American Statistical Association. American Statistical Association. 66 (336): 846–850. doi:10.2307/2284239. JSTOR 2284239. Lawrence Hubert and Phipps Arabie (1985). "Comparing partitions". Journal of Classification. 2 (1): 193–218. doi:10.1007/BF01908075. Nguyen Xuan Vinh, Julien Epps and James Bailey (2009). "Information Theoretic Measures for Clustering Comparison: Is a Correction for Chance Necessary?" (PDF). ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning. ACM. pp. 1073–1080.PDF. Alexander J Gates and Yong-Yeol Ahn (2017). "The Impact of Random Models on Clustering Similarity" (PDF). Journal of Machine Learning Research. 18: 1–28.PDF.

$a b$ \sum_{i=1}^n{x+y}}$$

$$y = x^2 + z^3 \tag{1}$$

$h[m,n] = \frac{\sum_{k,l}({(g[k,l]-\overline g)(f[m-k, n-l]-\overline f_{m,n})}}{\left(\sum_{k,l}(g[k,l]-\overline g)^2{\sum_{k,l}(f[m-k,n-l]-\overline f_{m,n})^2} \right)^{0.5}}$

$\sum_\limits{I=1}^{n}x_i-\bar{x}^2}\sum_\limits{I=1}^{n}$

$\sum_1^n$

$$ J_\alpha(x) = \sum_{m=0}^\infty \frac{(-1)^m}{m! \Gamma (m + \alpha + 1)} {\left({ \frac{x}{2} }\right)}^{2m + \alpha} \text {，独立公式示例} $$