2EL1730: MACHINE LEARNING

CENTRALESUPÉLEC

Lab 9: k-Means and Spectral Clustering Algorithms

Instructors: Fragkiskos Malliaros and Maria Vakalopoulou
TAs: Enzo Battistella, Yoann Pradat, Jun Zhu

January 19, 2021

1 Description

In this lab, we will study unsupervised learning techniques, focusing on two well-known clustering algorithms: (i) *k-Means* and (ii) *Spectral Clustering*. Initially, we discuss the basic characteristics of each algorithm and then we examine how they can be applied to identify the underlying clustering structure of a dataset.

2 k-Means Algorithm

k-means is one of the simplest unsupervised learning algorithms that solves the well known clustering problem. The goal of a clustering algorithm is to split the instances of the dataset into k clusters, where each instance is assigned to the closest cluster as defined by a distance function. The main idea is to define k centers (centroids), one for each cluster. The algorithm defines an iterative process, where the following two steps take part at each iteration: (i) take each instance belonging to the dataset and assign it to the nearest centroid, and (ii) re-calculate the centroids of each of the k clusters. Thus, the k centroids change their location step by step until no more changes are done.

More formally, suppose that we are given a dataset $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$, where each $\mathbf{x}_i \in \mathbb{R}^n$. The goal of the k-means algorithm is to group the data into k cohesive clusters, where k in as input parameter of the algorithm. Algorithm 1 gives the pseudocode of k-means.

Algorithm 1 k-Means Clustering Algorithm

Input: Dataset $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$, where each $x_i \in \mathbb{R}^n$ and parameter k
Output: Clusters $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k$ (i.e., cluster assignments of each instance $C = \{c_1, c_2, \ldots, c_m\}$)

1: Initialize cluster centroids $\mu_1, \mu_2, \ldots, \mu_k$ by choosing k instances of $\mathbf{X}$ randomly
2: **repeat**
3: Assign each instance $\mathbf{x}_i \in \mathbf{X}$ to the closest centroid, i.e., $c_i = \arg\min_j \|\mathbf{x}_i - \mu_j\|$
4: Re-compute the centroids $\mu_1, \mu_2, \ldots, \mu_k$ of each cluster based on $\mu_j = \frac{1}{n_j} \sum_{\mathbf{x} \in \mathbf{C}_j} \mathbf{x}$, where $\mathbf{C}_j, j = 1, \ldots, k$ the j-th cluster and n_j the size of the j-th cluster
5: **until** Centroids do not change (convergence)

In the algorithm above, k is a parameter of the algorithm and corresponds to the number of clusters we want to find; the cluster centroids μ_j represent our current guesses for the positions of the centers of the clusters. To initialize the cluster centroids (in step 1 of the algorithm), we could choose k training examples randomly, and set the cluster centroids to be equal to the values of these k examples. Of course, other initialization methods are also possible, such as the kmeans++ technique[1]. To find the closest centroid, a distance (or similarity) function should be defined, and typically the Euclidean distance is used.

Based on this notion of similarity, the problem of clustering can be reduced to the problem of finding appropriate centroids. This, in turn, can be expressed as the task of minimizing the following objective function:

$$E(k) = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|. \tag{1}$$

Thus, minimizing Eq. (1) is to determine suitable centroids μ_j such that, if the data is partitioned into corresponding clusters C_j, distances between data points and their closest cluster centroid become as small as possible.

The convergence of k-means algorithm is highly dependent on the initialization of the centroids. Although the algorithm can converge, this may be to a local minimum of the objective function of Eq. (1). One way to overcome this problem is by executing the algorithm several times, with different initializations of the centroids.

Another issue is how to set parameter k, i.e., how to determine the number of clusters of the dataset. Intuitively, increasing k without penalty, will always reduce the amount of error in the resulting clustering, to the extreme case of zero error if each data point is considered its own cluster (i.e., when k equals the number of data points, m). One such method is known as the *elbow rule*[2]. The idea is to examine and compare the sum of squared error (SSE) given in Eq. (1) for a number of cluster solutions. In general, as the number of clusters increases, the SSE should decrease because clusters are, by definition, smaller. A plot of the SSE against a series of sequential cluster levels (i.e., different values) can be helpful here. That is, an appropriate cluster solution could be defined as the one where the reduction in SSE slows dramatically. This produces an "elbow" in the plot of SSE against the different values of k.

2.1 Pipeline of the task

Next we describe the pipeline of task contained in the `kmeans/main.py` Python script. The goal here is to apply k-means on two datasets. The first one is an artificial dataset where the data points form four distinct clusters, similar to the one shown in Fig. 1. The second one is the MNIST handwritten digits dataset that has been also used in the supervised learning labs. The basic difference here is that we do not take into account the class labels. We use a modified version of the dataset (similar to the one used in Lab 5). We have applied PCA on the data and we keep the first 8 principal components. We also keep a sample of the data consisting of 1000 instances. Thus, the size of dataset $\mathbf{X}$ is 1000×8. Our goal is to apply k-means clustering on $\mathbf{X}$.

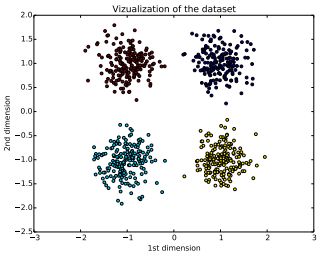


Figure 1: Example of artificial dataset.

For both datasets, initially we load the data and for illustration purposes, we visualize them. For

[1] Wikipedias lemma for *k-means++*: `http://en.wikipedia.org/wiki/K-means++`.

[2] Description of the *elbow rule* can be found in `http://www.mattpeeples.net/kmeans.html`.

example, in the case of the MNIST dataset we perform the steps shown below.

```
# Number of instaces and number of principal components (features)
n_instances = 1000
pca_features = 8

# Get the labels of each digit
images, labels_mnist = read_dataset(n_instances, pca_features);

# Create the dataset (data_mnist) that will be used in clustering
# load the PCA features of the test data set
data_mnist = array(list(csv.reader(open("test_data.csv","rb"),delimiter=',',))).astype('float')
data_mnist = data_mnist[:n_instances,:pca_features] #only 8 first features are kept
```

Then, we run k-means algorithm for different values of k. We also plot the data (two dimensions) based on clustering results produced by the algorithm.

```
# Run k-means algorithm for different values of k

k = 10
labels_pred_mnist = kmeans(data_mnist,k)
```

2.2 Tasks to be done

- Fill in the code of the `kmeans()` function in the `kmeans.py` file, based on Algorithm 1.
- Run the k-means algorithm for different values of k for the two datasets and examine the quality of the produced clusters.
- Use the *elbow rule* described above to determine the number of clusters k by examining the sum of squared error (SSE) for the clusters produced for different values of k.

3 Spectral Clustering

Spectral clustering techniques make use of the *spectrum* (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset.

Given a set of data points $\mathbf{x}_1, \ldots, \mathbf{x}_m, \forall \mathbf{x}_i \in \mathbb{R}^n$ and some notion of similarity s_{ij} between all pairs of data points $\mathbf{x}_i$ and $\mathbf{x}_j$, the intuitive goal of clustering is to divide the data points into several groups such that points in the same group are similar and points in different groups are dissimilar to each other. If we do not have more information than similarities between data points, a nice way of representing the data is in form of the similarity graph $G = (V, E)$. Each vertex v_i in this graph represents a data point $\mathbf{x}_i$. Two vertices are connected if the similarity s_{ij} between the corresponding data points $\mathbf{x}_i$ and $\mathbf{x}_j$ is positive or larger than a certain threshold, and the edge is weighted by s_{ij} . The problem of clustering can now be reformulated using the similarity graph: we want to find a partition of the graph such that the edges between different groups have very low weights (which means that points in different clusters are dissimilar from each other) and the edges within a group have high weights (which means that points within the same cluster are similar to each other).

How to create a similarity graph

There are several popular constructions to transform a given set $\mathbf{x}_1, \ldots, \mathbf{x}_m, \forall \mathbf{x}_i \in \mathbb{R}^n$ of data points with pairwise similarities s_{ij} or pairwise distances d_{ij} into a graph. When constructing similarity graphs the goal is to model the local neighborhood relationships between the data points.

- **k-Nearest Neighbors graph.** Here the goal is to connect vertex v_i with vertex v_j if v_j is among the k-nearest neighbors of v_i. However, this definition leads to a directed graph, as the neighborhood relationship is not symmetric. The most common way to deal with this, is to simply ignore the directions of the edges; that is, we connect v_i and v_j with an undirected edge if v_i is among the k-nearest neighbors of v_j or if v_j is among the k-nearest neighbors of v_i. The resulting graph is what is usually called the k-nearest neighbors graph.

- **The fully connected graph.** Here we simply connect all points with positive similarity with each other, and we weight all edges by s_{ij} . As the graph should represent the local neighborhood relationships, this construction is only useful if the similarity function itself models local neighborhoods. An example for such a similarity function is the Gaussian similarity function $s(\mathbf{x}_i, \mathbf{x}_j) = \exp(\|\mathbf{x}_i - \mathbf{x}_j\|^2)/(2\sigma^2)$, where the parameter σ controls the width of the neighborhoods.

The algorithm

Next we describe the pseudocode of the spectral clustering algorithm.

Algorithm 2 Spectral Clustering

Input: Dataset $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$, where each $\mathbf{x}_i \in \mathbb{R}^n$ and parameter k
Output: Clusters $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k$ (i.e., cluster assignments of each instance $C = \{c_1, c_2, \ldots, c_m\}$)

1: Construct the similarity graph G using one of the ways described above. Let $\mathbf{W}$ be the adjacency matrix of this graph.
2: Compute the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$. Matrix $\mathbf{D}$ corresponds to the diagonal degree matrix of graph G (i.e., degree of each node v_i (= number of neighbors) in the main diagonal).
3: Apply eigenvalue decomposition to the Laplacian matrix $\mathbf{L}$ and compute the eigenvectors that correspond to k smallest eigenvalues. Let $\mathbf{U} = [\mathbf{u}_1|\mathbf{u}_2|\ldots|\mathbf{u}_k] \in \mathbb{R}^{m \times k}$ be the matrix containing these eigenvectors as columns.
4: For $i = 1, \ldots, m$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of $\mathbf{U}$. Apply k-means to the points $(y_i)_{i=1,\ldots,m}$ (i.e., the rows of $\mathbf{U}$) and find clusters $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_k$.

In spectral clustering, the data is projected into a lower-dimensional space (the spectral/eigenvector domain) where they are easily separable, say using k-means. So, what is the reason to apply spectral clustering (in the similarity data matrix) and not applying directly k-means to the initial data? Typically, k-means algorithm is interesting in finding compact clusters of convex shape, while on the other hand spectral clustering methods are trying to identify connectivity patterns in the similarity graph. In many cases, we are interested in finding clusters that are non-convex and in this case the k-means algorithm does not behave well. Figure 2 shows an example of a dataset where the "natural" clusters in $\mathbb{R}^2$ do not correspond to convex

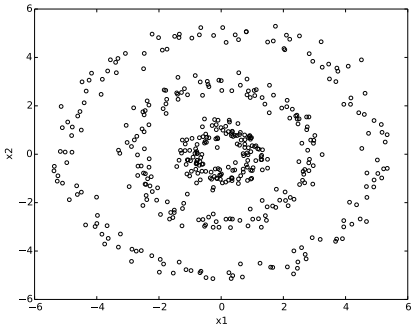
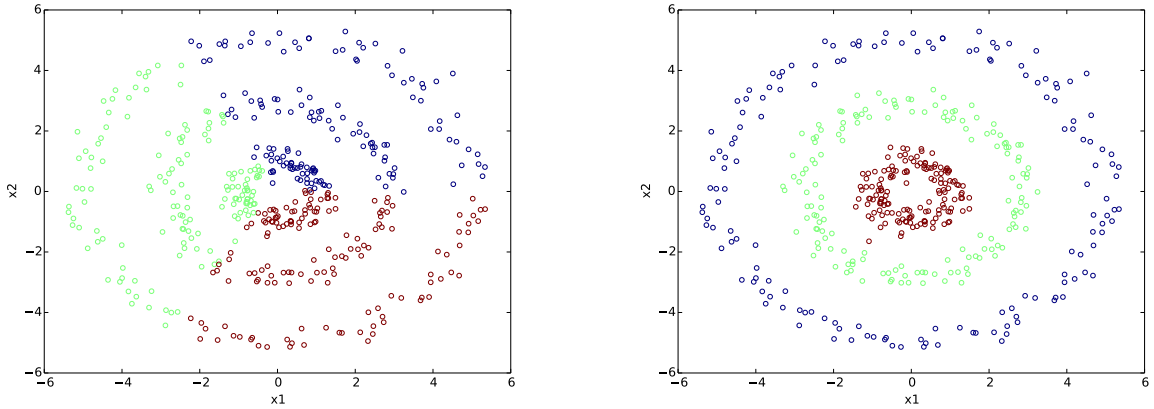


Figure 2: Example of a dataset.

compact regions. Applying k-means to this dataset will extract the clusters shown in Fig. 3 (a). On the other hand, as shown in Fig. 3 (b), applying spectral clustering, we are able to find non-convex clusters with good connectivity properties.



(a) k-means (b) Spectral clustering

Figure 3: Results using k-means and spectral clustering algorithms.

3.1 Pipeline of the task

Next we describe the pipeline of the task contained in the `spectral_clustering/main.py` file. The goal is to apply spectral clustering on the artificial data shown in Fig. 2. Initially, we call the function `generateData()` contained in the `generateData.py` file, to create the artificial data. Then, we use the build-in implementation of Python for the k-means algorithm to cluster this dataset (you can also use your own implementation) and we plot the results.

```
# Number of clusters
k = 3
# Cluster using kmeans
centroids, labels = kmeans2(data, k)
```

Then, we create the similarity graph, finding the N closest neighbors of each data instance, using the Euclidean distance. Notice that, after finding the closest neighbors using the `findClosestNeighbours()` function, we can directly form the adjacency matrix $\mathbf{W}$ of the graph. Here we create a binary (0 or 1) adjacency matrix (if two points are neighbors, we add the corresponding edge with weight 1).

```
N = 10
closestNeighbours = findClosestNeighbours(data, N)

# Create adjacency matrix
W = zeros((data.shape[0], data.shape[0]))
for i in range(data.shape[0]):
for j in range(n):
W[i,closestNeighbours[i,j]] = 1
W[closestNeighbours[i,j],i] = 1
```

Having the similarity graph (described by the adjacency matrix $\mathbf{W}$), we can apply the spectral clustering algorithm, finding the underlying clustering structure.

```
# Perform spectral clustering
```

```
labels = spectralClustering(W, k)
```

3.2 Tasks to be done

- Fill in the code of the `spectralClustering()` function in the `spectralClustering.py` file to implement the spectral clustering algorithm as described in Algorithm 2. Note that, the adjacency matrix $\mathbf{W}$ (step 1 of the algorithm) has been already created.
- Run the algorithm and reproduce the clustering results shown in Fig. 3 (b).

References

[1] Christopher M. Bishop. "Pattern Recognition and Machine Learning". Springer-Verlag New York, Inc., 2006.

[2] Tom M. Mitchell. "Machine learning". Burr Ridge, IL: McGraw Hill 45, 1997.

[3] Ulrike Von Luxburg. "A tutorial on spectral clustering". Statistics and computing, Springer, 2007.