CSE, IITB

CS 335 - AI & ML
Lab 4 - Clustering (11 marks)

Please read the following important instructions before getting started on the assignment

1. The assignment should be completed individually. Do not look at solutions to this assignment or related ones on the Internet.
2. Solutions to the theory questions must be submitted in a single pdf file.
3. All the hyperparameters must be specified in pdf file under **Hyperparameter** section and resources consulted must be duly listed in the **References** section of the pdf file. Do not upload multiple pdf files.
4. **Upload Guidelines** Put all the assignment related files in folder with the convention **lab4-roll_no** and .zip the folder.
5. **Not following folder guidelines will attract penalty.**
6. All source code are checked with code plagiarism checker. Strict action will be taken against the defaulters.
7. Comment out all the print statements from the source before submission. Repetition of such mistakes will attract penalty.

Principal Component Analysis (2 marks)

Task 1 : Implementation (1 mark)

In this task, you will implement PCA using Numpy's `eig` function (which computes the eigenvalues/vectors of a given matrix) as a building block. Given a matrix $X \in \mathcal{R}^{n \times d}$ representing n points in $\mathcal{R}^d$ and $k \leq d$, your algorithm should return a list of length k whose i^{th} element is the normalized projection of zero-mean X (say $\hat{X}$) on the i^{th} principal component (i.e. $\frac{\hat{\mathbf{X}}\mathbf{u}_i}{\|\hat{\mathbf{X}}\mathbf{u}_i\|}$). Complete the function `pca_small` in `pca.py` and verify your implementation using `python3 pca.py small`. **You are not allowed to import any new function or library** (you may use ***** for dot product and **@** for matrix multiplication). Refer Problem 1 of Tutorial 6 for details.

Task 2 : Improving the algorithm (1 mark)

You may notice that your algorithm becomes quite slow when $n >> d$ (or $d >> n$). First, verify that this is true by finding and reporting execution times of `pca_small` for $d = 50,\ n \in \{50, 2000\}$ (or $d \in \{50, 2000\},\ n = 50$ respectively). Next, identify and report the bottleneck line in your code (i.e. the one that takes the maximum time). Now, suggest an alternative algorithm that is fast even when $n >> d$ or $d >> n$ by completing the function `pca_large`. You can verify your implementation using `python3 pca.py large`.

k-means clustering (9 marks)

k-means clustering is an unsupervised learning algorithm, which aims to cluster the given data set into k clusters. Formally, the algorithm tries to solve the following problem: Given a set of observations $(x_1,\ x_2,\ \ldots,\ x_n),\ x_i \in R^d$ partition them into k ($\leq n$) sets $S = \{S_1, S_2, \ldots, S_k\}$ so as to minimise the within-cluster sum of squares (WCSS) (sum of squared distance of each point in the cluster to its cluster centre). In other words, the objective is to find:

$$\text{argmin}_S \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||_2^2$$

where μ_i is the mean of points in S_i.

Finding the optimal solution to this problem is NP-Hard for general n, d, and k. k-means clustering presents an iterative solution to the above clustering problem, that provably converges to a local optimum.

k-means algorithm

1. Define the initial clusters' centroids. This step can be done using different strategies. A very common one is to assign random values for the centroids of all groups. Another approach, known as Forgy initialisation, is to use the positions of k different points in the data set as the centroids.

2. Assign each entity to the cluster that has the closest centroid. In order to find the cluster with the closest centroid, the algorithm must calculate the distance between all the points and each centroid.

3. Recalculate the centroids. Each component of the centroid is updated, and set to be the average of the corresponding components of the points that belong to the cluster.

4. Repeat steps 2 and 3 until points no longer change cluster.

Note

- In step 2, if more than one centroid is closest to a data point, you can break the ties arbitrarily.

- In step 3, if any cluster has no points in it, just pass on the value of the centroid from the previous step.

CS 335 - AI & ML
Lab 4 - Clustering (11 marks)

Task 1 : Implementing *k*-means clustering (3 marks)

Implement all of the following 6 functions in `cluster.py`.

1. `distance_euclidean(p1, p2)`
2. `initialization_forgy(data,k)`
3. `kmeans_iteration_one(data,centroids,distance)`
4. `hasconverged(old_centroids,new_centroids,epsilon)`
5. `iteration_many(data,centroids,distance,maxiter,algorithm,epsilon)`
6. `performance_SSE(data,centroids,distance)`

Test your code by running this command:

```
python cluster.py -s $RANDOM datasets/flower.csv
```

Try different values of k with different random seeds using command line options (see `python cluster.py -h`). You can also try changing epsilon and maximum number of iterations.

Evaluation: Each correctly implemented function will fetch you 0.5 marks. Test with `python autograder.py 1`. This shows your final grade for this task.

Task 2 : Testing and Performance (2 marks)

Test your code on the following data sets.

```
  datasets/100.csv :  Use k = 2, numexperiments = 100
  datasets/1000.csv :  Use k = 5, numexperiments = 25
datasets/10000.csv :  Use k = 20, numexperiments = 10
```

Use epsilon = 10^{-2}. Here is an example.

```
python cluster.py -e 1e-2 -k 2 -n 100 datasets/100.csv
```

Answer the following 2 questions in the pdf file.

1. Run your code on `datasets/garden.csv`, with different values of k. Look at the performance plots and answer whether the SSE of the k-means clustering algorithm ever increases as the iterations are performed. Why or why not? [**1 mark**]

2. Look at the files `3lines.png` and `mouse.png`. Manually draw cluster boundaries around the 3 clusters visible in each file (no need to submit the hand drawn clustering). Test the k-means algorithm with different random seeds on the data sets `datasets/3lines.csv` and `datasets/mouse.csv`. How does the algorithm's clustering compare with the clustering you did by hand? Why do you think this happens? **[1 mark]**

Evaluation: The text questions carry marks as specified. Make sure to write clean, succinct answers.

It is worth noting that k-means can sometimes perform poorly! Test your algorithm on the `datasets/rectangle.csv` data set several times, using $k = 2$. Depending on the initialisation, k-means can converge to poor clusterings.

Task 3: Kernel k-means clustering (4 marks)

$$D(\{\pi_j\}_{k=1}^{C}) = \sum_{k=1}^{C} \sum_{i \in \pi_k} ||c_k - \phi(x_i)||_H^2$$

where the symbols are set of points (π_k), centroid (c_k) and no. of elements (n_k) in cluster k.

$$c_k = \frac{\sum_{b \in \pi_k} \phi(b)}{n_k}$$

Here c_k is the best cluster representative in R^H (kernel space) of ϕ, obtained by minimizing the following quantity **(observe c_k cannot be explicitly computed)**

$$c_k = \arg\min_z \sum_{a \in \pi_k} ||\phi(a) - z||^2$$

To obtain good cluster representatives (c_k), for each data point **a**, retrieve the cluster which has the minimum distance from it (shortest distance from cluster centroids)

$$\arg\min_k D(a, \{\pi_j\}_{k=1}^{C}) = \arg\min_k \left|\left|\phi(a) - c_k\right|\right|^2 = \arg\min_k \left|\left|\phi(a) - \frac{\sum_{b \in \pi_k} \phi(b)}{n_k}\right|\right|^2$$

$$= \arg\min_k \left(\phi(a) \cdot \phi(a) - 2\frac{\sum_{b \in \pi_k} \phi(a) \cdot \phi(b)}{n_k} + \frac{\sum_{b,c \in \pi_k} \phi(b) \cdot \phi(c)}{n_k^2}\right)$$

the terms of the form $\phi(x) \cdot \phi(y)$ can be computed using a kernel

$$\kappa(x, y) = \phi(x) \cdot \phi(y)$$

CS 335 - AI & ML
Lab 4 - Clustering (11 marks)

Kernel k-means algorithm

1. For each entity (datapoint) assign it to a random cluster. Compute the centroids by averaging the coordinates of the datapoints in the respective clusters (in original feature space R^D)
2. Reassign each entity to the cluster that has the closest centroid. In order to find the cluster with the closest centroid, the algorithm must calculate the distance between all the points and each centroid in kernel space R^H.
3. Recalculate the centroids. Each component of the centroid is updated, and set to be the average of the corresponding components of the points that belong to the cluster (in feature space R^D).
4. Repeat steps 2 and 3 until points no longer change in the cluster.

Implement the following functions in `kernelcluster.py` (function definitions are given in the file) on `datasets/3lines.csv` and `datasets/mouse.csv`.

1. `RBFkernel(p1,p2,sigma)`
2. `initializationrandom(data,k)`
3. `firstTerm(p1)` - $\phi(a) \cdot \phi(a)$ (refer above equation for completeness)
4. `secondTerm(data,pi_k)` - $2\frac{\sum_{b \in \pi_k} \phi(a) \cdot \phi(b)}{n_k}$
5. `thirdTerm(pi_k)` - $\frac{\sum_{b,c \in \pi_k} \phi(b) \cdot \phi(c)}{n_k^2}$
6. `kernelkmeans(data,C)`
7. `hasconverged(clusterList,C)`

Report the final plots of the cluster in the folder `kernel_plots` and name them as `3lines.jpg` and `mouse.jpg`. Run the following command for program execution.

```
python3 kernelcluster.py
```