CS 386: Lab Assignment 5

(TA in charge: Pragy Agarwal)

The focus of this lab is **k-means clustering**. We will look at the vanilla algorithm, its performance, and some better variants. Finally, we will use clustering for classifying the MNIST data set.

Code and data sets

Some starter code and data sets have been provided for you here: rollno_lab5.tar.gz. Uncompress this directory to find the following files.

`kmeans.py`

This file implements the k-means clustering algorithm. This is the only file in which you need to write code. You need to implement the following 9 functions in the file, under different tasks (described below).

1. `distance_euclidean(p1, p2)`
2. `distance_manhattan(p1, p2)`
3. `initialization_forgy(data, k)`
4. `initialization_randompartition(data, distance, k)`
5. `initialization_kmeansplusplus(data, distance, k)`
6. `iteration_one(data, means, distance)`
7. `hasconverged(old_means, new_means, epsilon=1e-1)`
8. `iteration_many(data, means, distance, maxiter, epsilon=1e-1)`
9. `performance_SSE(data, means, distance)`

Tip: Each function is labeled with the task number. If you are working on Task 1, search the file for `task1`.

After implementing the functions, you can execute your code as follows; the command-line options are explained below.

```
python kmeans.py --input datasets/flower.csv --iter 100 --epsilon 1e-3 --init forgy --dist euclidean --k 8
```

Argument		Description
`--seed`	int	The RNG seed to be used.
`--input`	str	Data set filename
`--output`	str	Output filename
`--iter`	int	Maximum number of iterations of the k-means algorithm to perform (may stop earlier if convergence is achieved)
`--epsilon`	float	Minimum distance the cluster centroids move between two consecutive iterations for the algorithm to continue.
`--init`	str	The initialisation algorithm to be used, from among {forgy, randompartition, kmeans++}
`--dist`	str	The distance metric to be used, from among {euclidean, manhattan}
`--k`	int	The number of clusters to use

`autograder.py`

Execute this script as `python autograder.py [task-number]`. Running `python autograder.py all` will give you an estimate of your final grade. We will use our own test cases and judgement before finalising the grade.

`mnistplot.py`

This is a tool to visualise your MNIST cluster means, that you will need in Task 3. Execute it as `python mnistplot.py mnist.txt`.

`datasets/*`

This directory contains the data sets you will be using during this lab. Each line in the files is a comma separated list of numbers, representing a point in $\mathbb{R}^d$.

`100.csv`, `1000.csv`, and `10000.csv` were sampled from multivariate Gaussians with diagonal covariance. They have data in $\mathbb{R}^{10}$, $\mathbb{R}^{50}$, and $\mathbb{R}^{100}$, respectively.

Submission details and useful resources are at the bottom.

Section 1: Understanding k-means clustering

k-means clustering is an unsupervised learning algorithm, which aims to cluster the given data set into k clusters.

Formally, the clustering problem it tries to solve can be stated as:

Given a set of observations $(x^1, x^2, \ldots, x^n), x^i \in \mathbb{R}^d$, partition the n observations 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 distance functions of each point in the cluster to its cluster centre). In other words, its objective is to find:

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

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 is a heuristic approach 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 the ***Forgy initialisation***, is to use the values of k different entities as being 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 entities 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 can no longer change cluster.

Cluster Initialisation: Forgy

This is one of the simplest ways to initialise the k-means algorithm. The Forgy method randomly chooses k points from the data set and uses these as the initial means. This ensures that the initial clusters are uniformly spread out across the data.

Measuring Performance: Sum Squared Error (SSE)

k-means clustering tries to locally minimise the Sum Squared Error, where the error associated with each data point is taken as its Euclidean distance from the cluster centre.

Task 1: Implementing k-means clustering (6 marks)

Implement all of the following 7 functions in `kmeans.py`.

1. `distance_euclidean(p1, p2)`
2. `distance_manhattan(p1, p2)`
3. `initialization_forgy(data, k)`
4. `iteration_one(data, means, distance)`
5. `hasconverged(old_means, new_means, epsilon)`
6. `iteration_many(data, means, distance, numiter, epsilon)`
7. `performance_SSE(data, means, distance)`

Test your code by running this command.

```
python kmeans.py --input datasets/flower.csv --iter 100 --epsilon 1e-3 --init forgy --dist euclidean --k 8 --seed $RANDOM
```

Try different values of k, and try both Euclidean and Manhattan distances.

Evaluation: Each correctly implemented function will fetch you 0.5 mark. If all 7 functions are correctly implemented, you get another 2.5 marks.

Test with `python autograder.py 1`.

Task 2: Testing and Performance (3 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$= 10^{-2}$ and the Euclidean distance metric for every experiment. Here is an example.

```
python kmeans.py --epsilon 1e-2 --init forgy --dist euclidean --input datasets/100.csv --k 2 -numexperiments 100
```

Answer the following 4 questions in the file `solutions.txt`.

1. For each data set, report "average SSE" and "average iterations". **[1.5 marks]**
2. 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. Is your answer the same for both Euclidean and Manhattan distances?**[0.5 mark]**
3. Regarding the answer to question 2 , why do you think this happens? What does it imply? **[0.5 mark]**
4. 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? **[0.5 mark]**

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

It is worth noting that k-means can be made to perform arbitrarily poorly! Test your algorithm on the `datasets/rectangle.csv` data several times, using $k = 2$. Depending on the initialisation, k-means will converge to the worst possible clustering!

Section 2: Initialisation Methods

Random Partition

The Random Partition initialisation method splits the data into k randomly chosen partitions.

- For each data point $x_i \in \mathbb{R}^d$, assign it to exactly one random partition $s_i \in S, i \in \{1, 2, \ldots, k\}$.
- Compute the initial centroid of each partition s_i as the centroid of all the points assigned to that partition.

Random partition thus places the initial cluster means close to the centre of the data set.

k-means++

The clustering performance of the k-means algorithm is sensitive to the initial cluster means. A poor initialisation can lead to arbitrarily poor clustering. The k-means++ initialisation algorithm addresses this problem; standard k-means clustering is performed after the initialisation step. With the k-means++ initialisation, the algorithm is guaranteed to find a solution that is $\mathcal{O}(logk)$ competitive to the optimal k-means solution in expectation.

The intuition behind this approach is that spreading out the k initial cluster centres is a good idea: the first cluster centre is chosen uniformly at random from the data points, after which each subsequent cluster centre is chosen from the remaining data points with a probability proportional to the point's squared distance to the point's closest existing cluster centre.

The algorithm is as follows.

1. Choose one centre uniformly at random from among the data points.
2. For each data point x^i, compute $D(x^i)$, the distance between x^i and the nearest centre that has already been chosen.
3. Choose one new data point at random as a new cluster centre, using a probability distribution in which point x^i is chosen with probability proportional to $D(x_i)^2$.
   In other words, the probability that x^i is made a cluster centre is $\frac{D(x^i)^2}{\sum_{j \in \{1,2,\ldots,n\}, x^j \text{ is not a cluster centre}} D(x^j)^2}$.
4. Repeat Steps 2 and 3 until k centres have been chosen.
5. Now that the initial centres have been chosen, proceed using standard k-means clustering.

This seeding method yields considerable improvement in both the speed of convergence, and the final error of k-means.

Task 3: Implementing Random Partition and k-means++ (4 marks)

Implement all of the following functions in `kmeans.py`.

1. `initialization_randompartition(data, distance, k)`
2. `initialization_kmeansplusplus(data, distance, k)`

Note: You are expected to **provide elaborate comments along with your code** (for these 2 functions). Your marks depend on whether the TAs are able to understand your code and establish its correctness.

Test with `python autograder.py 3`.

Use your code by running this command.

```
python kmeans.py --input datasets/flower.csv --epsilon 1e-2 --init kmeans++ --dist euclidean --k 8
```

Try both random partition and kmeans++.

After implementing your code, test it on these 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$= 10^{-2}$ and the Euclidean distance metric for every experiment.

Run your experiments for both random partition and kmeans++ initialisation algorithms. Here is an example command.

```
python kmeans.py --epsilon 1e-2 --init randompartition --dist euclidean --input datasets/100.csv --k 2 -numexperiments 100
```

Answer the following question in the file `solutions.txt`.

1. For each data set and initialisation algorithm, report “average SSE” and “average iterations”.

Evaluation: Correct implementation of the randompartition function will fetch you 1 mark. The kmeansplusplus function is worth 2 marks. The text question is worth 1 mark.

Notice how:

- Randompartition tends to initialise all cluster means near the centre of the data.
- kmeans++ initialisation leads to considerably less cluster movement compared to Forgy initialisation.
- Despite using kmeans++, the algorithm will sometimes converge to poor solutions.

Section 3: Clustering for classification

Supervised machine learning techniques require a data set with a large number of training examples and labels. However, getting labels can be extremely expensive in practice. Unsupervised learning algorithms handle this issue by not requiring any labels at all.

Since k-means clustering is an unsupervised algorithm, we will now use it to classify the MNIST data set.

Procedure:

- Cluster the MNIST data set into 10 clusters.
- Visualise each cluster mean as an image.
- Hand label each cluster as a digit
- Use the cluster means for classification.

This procedure enables us to achieve a decent classification accuracy, by hand labeling only a few dozen images (as compared to thousands)!

Task 4: MNIST classification (2 marks)

Template File: `kmeans.py`
Data set: `datasets/mnist.csv`

Run your algorithm on the MNIST data set as follows.

```
python kmeans.py --input datasets/mnist.csv --iter 100 --epsilon 1e-2 --init kmeans++ --dist euclidean --k 10 --output
mnist.txt
```

Plot the so found cluster centres by executing this command.

```
python mnistplot.py mnist.txt
```

Look at the plots and find out a good mean for each of the 10 digits. Compile these means into the file `mnistmeans.txt`. You will have to run the clustering algorithm several times to get satisfactory means. Use the random seed to get different means.

File Format for `mnistmeans.txt`:
The i th line must contain the cluster mean for the $i-1$ th digit.
Each mean is represented by a comma separated list of 784 floats.

Evaluation: Your cluster means will be used to classify the MNIST data set.

Evaluation: Your cluster means will be used to classify the MNIST data set.

accuracy	marks
accuracy $\geq 50\%$	2
$50\% >$ accuracy $\geq 40\%$	1
otherwise	0

Test with `python autograder.py 4`.

In practice, you would want to choose multiple cluster means per class instead of just one, for increased accuracy.

Submission

Your submission directory must be named as `[rollnumber]_lab5`. It must contain the following 3 files.

- `kmeans.py`
- `solutions.txt`
- `mnistmeans.txt` (do not submit mnist.txt by mistake)

Do not include any other files.

For tasks 1 and 2, compress your directory into `[rollnumber]_lab5.tar.gz`, and upload it to Moodle under Lab Assignment 5.

For tasks 3 and 4, compress your directory (with solutions to **all** tasks (1, 2, 3, and 4)) into `[rollnumber]_lab5_outlab.tar.gz`, and upload it to Moodle under Lab Assignment 5 (Outlab).

Resources

https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/K-Means
https://en.wikipedia.org/wiki/K-means%2B%2B
https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set
http://theory.stanford.edu/~sergei/papers/kMeans-socg.pdf
https://en.wikipedia.org/wiki/K-medians_clustering
http://archive.ics.uci.edu/ml/datasets/Iris
http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/K-Means_Clustering_Overview.htm