CS 663 - Machine Learning Spring, 2024

Assignment 02 - KMeans

There are a few goals for this assignment, specifically:

1. To develop your version K-Means using the algorithm specified below. (This has been an interview question for jobs using machine learning.)
2. To compare the performance of different implementations.
3. To demonstrate your understanding of clustering algorithms like K-Means, DBSCAN and Hierarchical. (This will also start introducing you to skills needed for data challenges.)
4. To extend the functionality of the developed K-Means implementation through additional parameters. (This shows your ability to develop novel or custom algorithms.)

Background

We covered the algorithms and hyperparameter tuning for K-Means, DBSCAN and Hierarchical clustering. The algorithm (in psuedocode) for the K-Means algorithm is as follows:

```
place k centroids ($\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$) randomly
repeat to convergence:
    foreach x ∈ test_x:
        $c^{(i)}$ = index of closest centroid to x
    foreach k ∈ centroids:
        $\mu_k$ = mean $c^{(i)}$ | index($c^{(i)}$) == k
```

DBSCAN works by starting in a random place in a dataset and moving outwards, adding items to the cluster. Hierarchical clustering puts each instance in its own cluster and associates pairs of clusters by distance. Spectral clustering imposes a graph over each instance and introduces graph cuts to isolate instances into clusters.

Requirements (Process)

Accept the assignment by navigating to https://classroom.github.com/a/xLy4C3Pi. There are three required portions for this assignment, all detailed below:

1. Implement K-Means as described above.
2. Do a performance analysis among the expected labels, your implementation of K-Means (excluding the extended version), the version offered by the Scikit-learn library.

3. Choose and run clustering algorithms against some datasets and evaluate the results.

In addition to the required (above), there is one optional part:

4. Extend K-Means so that it balances the number of instances (rows) per cluster.

Implement K-Means

Create a python-based implementation of the K-Means algorithm.

This implementation may be a subclass of cluster.py, available in the github repository. As such, it must implement two member functions: `__init__(...)` and `fit(...)`, as described below.

- `__init__(...)` must allow the class' users to set the algorithm's hyperparameters: `k`, which is the target number of cluster centroids, and `max_iterations`, which is maximum number of times to execute the convergence attempt (`repeat` loop in the above Background section). The default values are required to be `k = 5` and `max_iterations = 100`.
- `fit(...)` must accept one parameter `X`, where `X` is a list (not columns of a Dataframe) of *n* instances in *d* dimensions (features) which describe the *n* instances. A successful call to the `fit(...)` function must return the following two items, in order:
    - A. A list (of length *n*) of the cluster hypotheses, one for each instance.
    - B. A list (of length at most *k*) containing lists (each of length *d*) of the cluster centroids' values.

For example, if the input (`X`) contains the following values in 2-dimensional space:

[[0, 0], [2, 2], [0, 2], [2, 0], [10, 10], [8, 8], [10, 8], [8, 10]]

… and k = 2, we expect the centroids should be [1, 1] and [9, 9]. The output of the `fit(...)` function should be as follows:

- A. [0, 0, 0, 0, 1, 1, 1, 1] — indicating that the first four instances belong to one cluster and the second four belong to a different cluster.
- B. [[1, 1], [9, 9]] — the values for the first and second centroid, respectively.

You may implement this algorithm in its own file — eg. `KMeans.py` — and import it as needed for performance comparison, etc.

Performance Comparison

In a Jupyter notebook, test your implementation using scikit-learn by generating clusters using the make_blobs function with the following commands:

```
from sklearn.datasets import make_blobs
```

```
X, cluster_assignments = make_blobs(n_samples=700, centers=4,
cluster_std=0.60, random_state=0)
```

This will generate 700 instances of data points in 2-dimensional space, with each of the instances belonging to one of 4 clusters. The coordinates for the instances are returned as `X`. The cluster assignments are returned as `cluster_assignments`. Do the following:

1. Use `X` as the parameter as input to your `fit(...)` function above, and use `cluster_assignments` to determine whether your implementation's hypotheses are correct.
2. Use `X` as the parameter as input to KMeans as implemented in scikit-learn, and use `cluster_assignments` to determine whether that implementation's hypotheses are correct.
3. Report two comparisons: one for your implementation's hypotheses against the expected values in `cluster_assignments`; another for your implementation's hypotheses against the hypotheses generated by scikit-learn. You may also show this on a chart, as is shown in Figure 1.

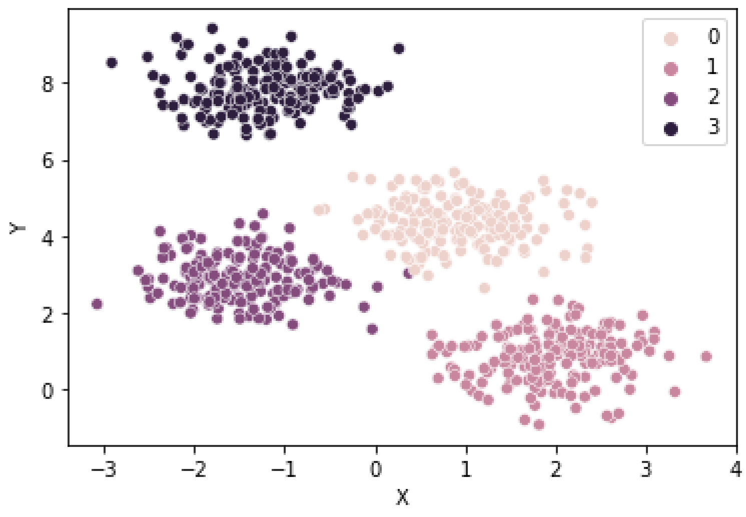

Figure 1: Instances generated from make_blobs

The values in `cluster_assignments` may not align to the values from your implementation's hypotheses or those from scikit-learn's implementation, so you should avoid using a metric like accuracy for this performance score.

Choose and run clustering algorithms

In the same Jupyter notebook as above, execute one or more clustering algorithms as implemented in scikit-learn (k-means, DBSCAN, Hierarchical) against the datasets below. Explain the following:

1. The reason why you chose the clustering algorithm(s)
2. Any pre-processing of the data or any hyperparameter settings
3. Output from the algorithm(s) -- show what clusters were generated
4. The metrics you used to evaluate the output. What kind of performance did you get from that algorithm? Is that what you expected?

Use the following datasets for this part:

- Chicago taxi data, an approximately week-long subset of the full dataset (which can be found here). Due to the size of the data (40MB), you may restrict clustering to using only the location data — eg. pickup or dropoff location coordinates.
- Mopsi data subset — taken from people tracking their routes around Finland. Routes consist of activities such as walking, cycling, skiing or vehicle transport, sometimes while conducting daily activities. (An insufficient description of Mopsi appears here.)

Extend K-Means (optional)

Sometimes it is useful to have the clusters of (roughly) the same size. For example, when sorting laundry to be placed in a washing machine, it is recommended that "dark" clothes be washed separately from "light" clothes. In this example, if a person has too many "light" clothes, the machine may be overloaded and will not do a good job at washing. However, in many wardrobes, some articles of clothing may fall in between "light" and "dark" and therefore may be used to balance the groups and get all the washing done efficiently.

Change your implementation of K-Means in the first part to include an additional optional Boolean (True/False) argument, `balanced`. The default value must be False. When `balanced` is set to True, the implementation changes so that each of the k clusters are (roughly) equal with respect to the number of instances in the cluster hypotheses — i.e. the implementation generates clusters of (roughly) the same size. When `balanced` is set to False, the logic is the canonical K-Means, described in the Background section.

Grading

Grades for this assignment will be evaluated as one of the following four outcomes:

- Exemplary: Submission correctly implements and tests all required parts defined above as well as the optional part. In addition, the implementation must meet implicit correctness criteria (eg. code is well-organised, uses a well-recognised standard for comments, is well-documented, is reusable, maintainable, etc.).
- Satisfactory: Submission correctly implements and tests all required parts defined above but does not provide an implementation of the required portion -OR- could improve implicit correctness criteria.
- Partial: Submission makes progress toward meeting all required parts.
- Unassessable: Not enough information to determine progress toward all required parts.

Avoid late submissions.

Submission

To submit your assignment for evaluation, follow all of the instructions:

1. Check into github your completed implementation, which must include a (single) Jupyter notebook with all cells executed and may also include a (single) python file, KMeans.py.
2. On Canvas, submit the URL for your completed assignment in the space provided. Add any comments, instructions for the grader, notes, etc. by changing the README.md file in the repository.