Practical Assignment 2

COMP 451 - Fundamentals of Machine Learning

Prof. William L. Hamilton

Winter 2021

Preamble

- The assignment is due March 23rd at 11:59pm via MyCourses. Late work will be automatically subject to a 20% penalty, and can be submitted up to 5 days after the deadline.
- You may consult with other students in the class regarding solution strategies, but you must list all the students that you consulted with on the first page of your submitted assignment and you should not share code. You may also consult published papers, textbooks, and other resources, but you must cite any source that you use in a non-trivial way (except the course notes).
- You must write your code yourself and be able to explain the solution to the professor, if asked.
- Python is strongly recommended, but you might write your solution in Java, C, or MATLAB if you prefer. The solution and autograding script will be in Python. If using Python, please submit as .py files and not as .ipynb files.
- You may use array processing and matrix libraries (e.g., numpy), but you may not use machine learning libraries (e.g., sklearn or PyTorch) in your solution.
- You must submit a single zip file that contains:
  - A single code file with your K-means implementation, named in an obvious way (e.g., `kmeans.py`).
  - A single code file with your GMM implementation, named in an obvious way (e.g., `gmm.py`).
  - Two output tsv files named `kmeans_output.tsv` and `gmm_output.tsv`, structured in the manner described below.

Overview of the assignment

In this assignment, you will implement both K-means and a Gaussian mixture model (GMM) to cluster data. We will use the famous Iris flower dataset, which can be found at `https://archive.ics.uci.edu/ml/datasets/iris`. Each example in this dataset corresponds to a flower, with features corresponding to various measurements of the flower (e.g., the petal length and width).

The dataset contains three different types of flowers, each with 50 examples. We will not use the class labels as targets during training, since we will use a fully unsupervised clustering approach. However, we will evaluate how well the clustering algorithms can recover the underlying classes. In particular, we will ask you to run clustering algorithms with $K = 3$, and we will see if the discovered clusters reflect the underlying partition of the data into three classes.

To evaluate the performance of your clustering algorithms, we will use an unsupervised accuracy score. The basic idea is that we will test whether your cluster assignments match the underlying labels. However, since the cluster labels are arbitrary integers, we must take the maximum accuracy over different possible mappings between cluster labels and class labels. In particular, assume that the three classes are labelled by the integers 1, 2, and 3. let $f_\pi(\mathbf{x})$ denote the learned cluster assignment under a particular cluster label permutation π (e.g., we might have that π_i labels the clusters as $(1, 2, 3)$ while π_2 labels the clusters as $(2, 3, 1)$). We will compute the accuracy as

$$\text{Acc} = \min_{\pi} \frac{\sum_{(\mathbf{x},y)} I(f_\pi(\mathbf{x}) = y)}{|\mathcal{D}|}, \tag{1}$$

where

$$I(f_\pi(\mathbf{x}) = y) = \begin{cases} 1 & \text{if } f_\pi(\mathbf{x}) = y \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Getting started

1. Download the files at `https://cs.mcgill.ca/~wlh/comp451/files/practical_assn_2.zip`
2. Find the dataset file from the assignment folder. It is available as **Data.tsv**, i.e., in a tab separated format.
3. The data contains 150 sample points. Each sample has 4 features, i.e., sepal length, sepal width, petal length and petal width.
4. Load the dataset. In Python, using numpy, the data can be read as,

```
X = np.genfromtxt('Data.tsv', delimiter='\t')
```

Task 1: Implement K-means [12 points]

Your first task is to implement a (hard) K-means clustering model.

1. Implement a basic K-means model. This is described in Chapter 14 of the class notes. Note that you should **not** implement soft K-means. You should implement the K-means as described at the beginning of Section 14.2 in the notes
2. Run K-means on the Iris dataset with $k = 3$, i.e., 3 clusters.
3. Initialize the centroids as:

```
[[ 1.03800476  0.09821729  1.0469454   1.58046376]
[ 0.18982966 -1.97355361  0.70592084  0.3957741 ]
[ 1.2803405   0.09821729  0.76275827  1.44883158]]
```

4. Run the K-means clustering algorithm until the cluster assignments stop updating, i.e., the centroids do not change position in successive updates.

5. Your K-means must output a file named `kmeans_output.tsv` that contains the cluster assignments for each point in `Data.tsv`. In particular, you should use the same ordering for the data points as in `Data.tsv`; the file should have one line with tabs separating the cluster labels for each point.

6. The script **autograderClustering.py** can be used for testing the accuracy of your model. Run the script as:

```
python autograderClustering.py kmeans_output.tsv
```

Task 2: Implement Gaussian Mixture Model [12 points]

Your second task is to implement a Gaussian mixture model (GMM). This model is described in Chapter 19 of the notes.

1. Implement a Gaussian mixture model (GMM). For implementation details follow Section 15.2 in Chapter 15 of the notes.

2. Run the GMM on the Iris dataset with number of components, i.e., $k = 3$.

3. For initialization of the mean vector and co-variance matrices, you should follow these steps:

   (a) Split the dataset into 3 equal sub-parts, e.g., using the `numpy.array_split(X, 3)` function. You should split the data into 3 equal parts based on the ordering of the points provided in the `data.tsv` files.

   (b) Compute the means and covariances of these 3 sub-datasets and use these three distinct mean and covariance estimates to initialize the mean and covariance parameters for three distinct clusters, respectively.

   (c) For initialization of the π vector, assume that the 3 classes have equal probabilities, i.e., [0.3333, 0.3333, 0.3333].

4. The GMM can be run until the the absolute difference in the log likelihoods between successive iterations is less than 10^{-5}. (See the end of Section 15.2 in the notes for background details.) The GMM can usually be trained properly in around 20-30 iterations.

5. Your GMM must output a file named `gmm_output.tsv` that contains the cluster assignments for each point in `Data.tsv`. In particular, you should use the same ordering for the data points as in `Data.tsv`; the file should have one line with tabs separating the cluster label for each point. `example_output.tsv` for an example.

6. The script **autograderClustering.py** can be used for testing the accuracy of your model. Run the script as:

```
python autograderClustering.py gmm_output.tsv
```

Rubric

We will use the unsupervised accuracy score described at the beginning of the assignment to evaluate your models. Note that the GMM model should have higher accuracy than the K-means, and we will grade the performance of each model independently. The rubric for grading both the K-means and GMM implementations are as follows:

- 0-2 points are awarded if the solution does not run in the autograder. Up to 2 points can be obtained if the code contains a non-trivial attempt at the implementation.
- 2-6 points are awarded if the implementation runs but achieves an accuracy that is 20% lower than the reference solution. The TA will award discretionary points based on the quality of the code and the nature of the error in the implementation.
- 7-8 points will be awarded if the implementation achieves an accuracy that is lower than the accuracy of our reference implementation by an amount in the range [5%, 20%). The TA will award discretionary points based on the quality of the code and the nature of the error in the implementation.
- 9-10 points will be awarded if the implementation achieves a test accuracy that is lower than the accuracy of our reference implementation by an amount in the range (1.5%, 5%) or more. The TA will award discretionary points based on the quality of the code and the nature of the error in the implementation.
- 11-12 points will be awarded if the implementation achieves a test accuracy that is equivalent to our reference solution (with a tolerance of $\pm 1.5\%$). The TA will award discretionary points based on the quality of the code; full points will be awarded as long as there are no major issues with the code (e.g., obvious and extreme inefficiencies).

Important note: There is no test set, since this is an unsupervised learning task, and it is easy to obtain the true class labels for the dataset (e.g., by inspecting the autograding script). However, if you simply cheat and submit manually created tsv files that are not actually generated by your models, then you will receive 0 points. We will verify that your submitted tsv files are genuine.

Note: This is not a software engineering course, and we will not be harsh when grading code. You may lose points if your code is confusing to the point of being unreadable (e.g., no meaningful variable names, confusing or unnecessary use of data structures) or if your implementation is obviously orders of magnitude less efficient than it could be (e.g., contains unnecessary nested loops over the training data). We will not provide detailed feedback on the code quality (e.g., detailed suggestions of how to fix the code).