CSE 4309 - Assignments - Assignment 6

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment6.zip, containing the following two files:

- answers.pdf, for the output that the programming tasks ask you to include. Only PDF files will be accepted. All text should be typed, and if any figures are present they should be computer-generated. Scans of handwriten answers will NOT be accepted.
- knn_classify.py, containing your Python code for Task 1.
- k_means.py, containing your Python code for Task 2.

These naming conventions are mandatory, non-adherence to these specifications can incur a penalty of up to 20 points. Your Python code should run on the versions specified in the syllabus, unless permission is obtained via e-mail from the instructor or the teaching assistant. Also submit any other files that are needed in order to document or run your code (for example, additional source code files).

Your name and UTA ID number should appear on the top line of both documents.

Task 1 (35 points, programming)

In this task you will implement k-nearest neighbor (k-nn) classification.

Function Name and Arguments

File knn_classify_main.py contains incomplete code that implements and evaluates a nearest neighbor classifier.

To complete that code, you must create a file called `knn_classify.py`, where you implement a Python function called `knn_classify`. Your function should be invoked as follows:

```
knn_classify(training_file, test_file, k)
```

The arguments provide the following information:

- The first argument, `training_file`, is the path name of the training file, where the training data is stored. The path name can specify any file stored on the local computer.
- The second argument, `test_file`, is the path name of the test file, where the test data is stored. The path name can specify any file stored on the local computer.
- The third argument, `k` specifies the value of k for the k-nearest neighbor classifier.

The training and test files will follow the same format as the text files in the UCI datasets directory. A description of the datasets and the file format can be found on this link. Your code should also work with ANY OTHER training and test files using the same format as the files in the UCI datasets directory.

Implementation Guidelines

- Each dimension should be normalized, separately from all other dimensions. Specifically, for both training and test objects, each dimension (i.e., each attribute) should be transformed using function F(v) = (v - mean) / std, using the mean and std of the values of THAT DIMENSION on the TRAINING DATA. To compute the std, use the formula that divides by (number of training objects minus 1). If you calculate an std equal to 0, have your code detect that case and set std to 1, so that you avoid dividing by 0.
- Use the L_2 distance (the Euclidean distance) for computing distances between two vectors.

Classification Stage

For each test object you should print a line containing the following info:

- object ID. This is the line number where that object occurs in the test file. Start with 1 in numbering the objects, not with 0.
- predicted class (the result of the classification). If your classification result is a tie among two or more classes, choose one of them randomly.
- true class (from the last column of the test file).
- accuracy. This is defined as follows:
    - If there were no ties in your classification result, and the predicted class is correct, the accuracy is 1.
    - If there were no ties in your classification result, and the predicted class is incorrect, the accuracy is 0.
    - If there were ties in your classification result, and the correct class was one of the classes that tied for best, the accuracy is 1 divided by the number of classes that tied for best.
    - If there were ties in your classification result, and the correct class was NOT one of the classes that tied for best, the accuracy is 0.

To produce this output in a uniform manner, in Python you should use:

```
print('ID=%5d, predicted=%10s, true=%10s, accuracy=%4.2f' % (object_id,
predicted_class, true_class, accuracy))
```

After you have printed the results for all test objects, you should print the overall classification accuracy, which is defined as the average of the classification accuracies you printed out for each test object. To print the classification accuracy in a uniform manner, use this Python line:

```
print('classification accuracy=%6.4f' % (classification_accuracy))
```

Output for answers.pdf

In your answers.pdf document, please provide ONLY THE LAST LINE (the line printing the classification accuracy) of the output by the test stage, for the following invocations of your program:

- Training and testing on the `pendigits` dataset, with k=1.
- Training and testing on the `pendigits` dataset, with k=3.

- Training and testing on the `pendigits` dataset, with k=5.

Expected Classification Accuracy

The results are deterministic, and consequently your results should match mine. These are the accuracy percentages I got with my implementation:

```
dataset     k = 1   k = 3   k = 5   runtime
pendigits   97.43   97.50   97.63   4 seconds
satellite   89.35   90.42   90.45   3 seconds
yeast       49.59   51.89   55.25   1 second
```

In this directory of results you can see the full output of my program for each of the cases above.

Task 1b (Extra Credit, maximum 10 points).

In this task, you are free to change any implementation options that you are not free to change in Task 1. Examples of such options include choice of distance function, choice of normalization, or any other relevant option for k-nn classifiers. You can submit a Python executable called knn_classify_opt.py, that implements your modifications. A maximum of 10 points will be given to the submission or submissions that, according to the instructor and GTA, achieve the best improvements (on any of the three datasets) compared to the specifications in Task 1. In your answers.pdf document, under a clear "Task 1b" heading, explain what modifications you made, what results you achieved, and what command line arguments to provide your program with in order to obtain those results.

Task 2 (35 points, programming)

In this task you will implement decision trees and decision forests. Your program will learn decision trees from training data and will apply decision trees and decision forests to classify test objects.

Function Name and Arguments

File k_means_main.py contains incomplete code that aims to implement the k-means clustering algorithm and apply that algorithm on a specific dataset.

To complete that code, you must create a file called `k_means.py`, where you implement a Python function called `k_means`. Your function should be invoked as follows:

```
k_means(data_file, K, initialization)
```

The arguments provide to the function the following information:

- The first argument, `data_file`, is a string that specifies the path name of a file where the data is stored. The path name can specify any file stored on the local computer.
- The second argument, `K`, is an integer that specifies the number of clusters.

- The third argument, `initialization`, is a string that specifies how the initial assignment of data points to clusters is done. This argument can have two possible values: `"random"`, or `"round_robin"`.

Example 1D and 2D datasets that you can test your code with can be found in the toy_data directory. In each file, each row corresponds to a data point, which can be a single number or a 2D vector. The two values of each 2D vector are separated by space.

Your code should work with any other files that follow the same format. You can assume that the input data will be either 1D or 2D, you do not need to worry about higher dimensions.

Implementation Guidelines

- To make the result deterministic, if the third argument is `"round_robin"`, the initial cluster assignments should be done in round-robin fashion. More specifically, the first object gets assigned to cluster 1, the second object gets assigned to 2, ..., the k-th object is assigned to cluster K, and so on. For example, for file set1a.txt, and K=3, this is how the initial assignments of points to clusters should be:
    - 7 --> cluster 1
    - 29 --> cluster 2
    - 11 --> cluster 3
    - 2 --> cluster 1
    - 16 --> cluster 2
    - 4 --> cluster 3
    - 37 --> cluster 1
    - 22 --> cluster 2

    Note that cluster IDs range from 1 to K, and there is NO cluster ID 0.
- Your algorithm should terminate when the assignment of objects to clusters stops changing. In other words, the algorithm should stop if the assignment of objects to clusters at the end of the current iteration is identical to the assignment at the end of the previous iteration.

Output

At the end, your program should print the final cluster assignments. If the dataset is one-dimensional (like file set1a.txt), the output should contain one line per data point, printed with the following formatting specification:

```
print('%10.4f --> cluster %d' % (data_point, cluster_id))
```

If the dataset is two-dimensional (like file set2a.txt), the output should contain again one line per data point, printed with the following formatting specification:

```
print('(%10.4f, %10.4f) --> cluster %d' % (data_point[0], data_point[1],
cluster_id))
```

Your can see examples of results printed using this format right below.

Example Results

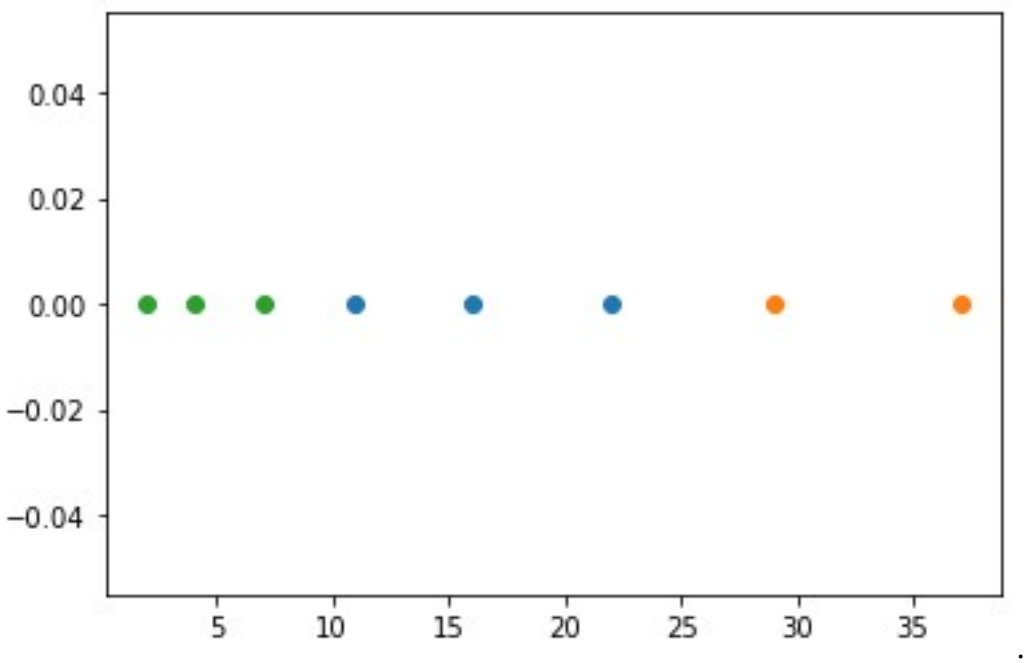
The following files show some results that my code gets. In all cases, the runtime is under one second.

- File set1a_3.txt shows the output for set1a.txt, with K=3 and round-robin initialization.
- File set2a_3.txt shows the output for set2a.txt, with K=3 and round-robin initialization.
- File set2b_3.txt shows the output for set2b.txt, with K=3 and round-robin initialization.
- File set2c_2.txt shows the output for set2c.txt, with K=2 and round-robin initialization.

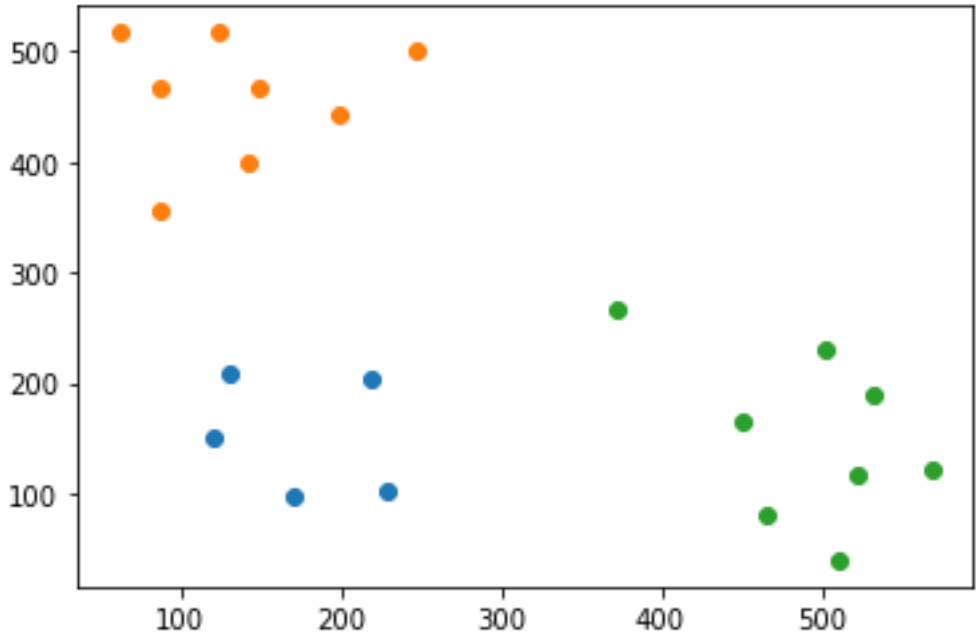
Plotting Datasets and Results

This is entirely optional, but if you are interested, Python file drawing.py contains code that you can use to visualize the datasets and results. Function `draw_points (data)` plots a 1D or 2D dataset. Function `draw_assignments(data, assignments)` plots each cluster with a different color. Here are some examples of such plots:

- Plot of the result for set1a.txt, with K=3 and round-robin initialization:

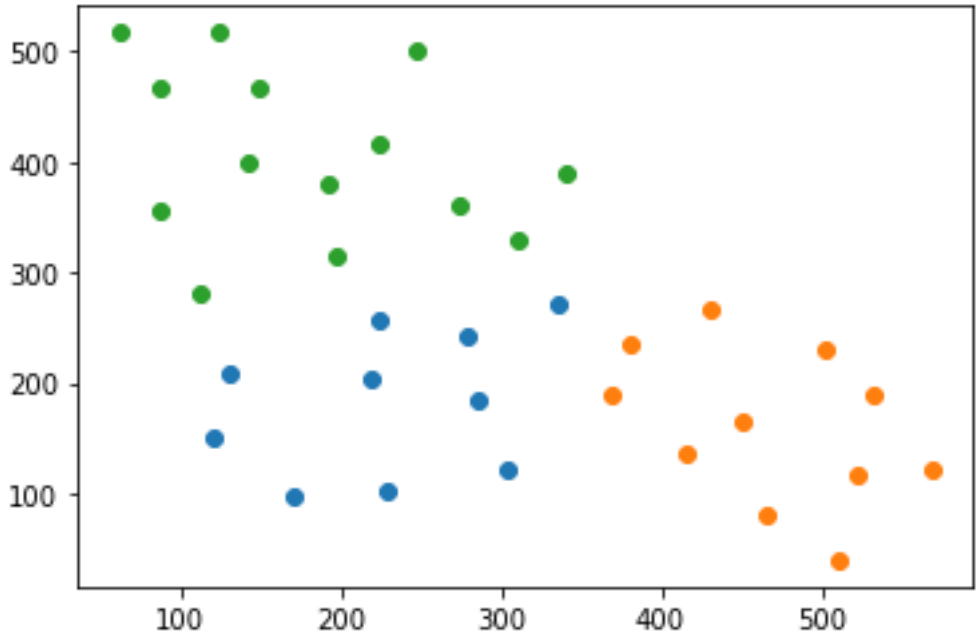


- Plot of the result for set2a.txt, with K=3 and round-robin initialization:



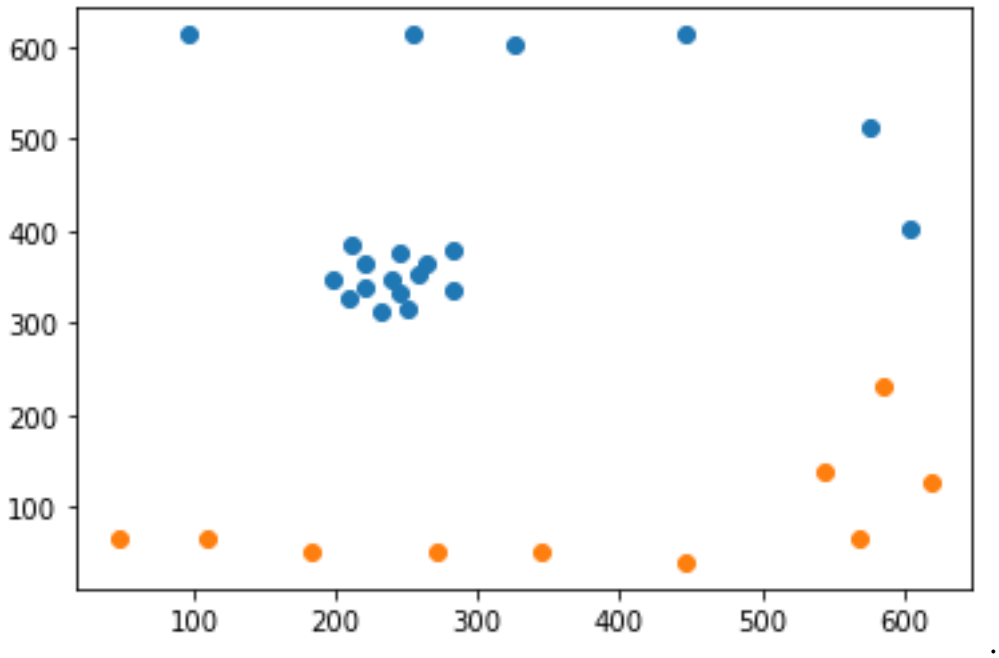
.

- Plot of the result for set2b.txt, with K=3 and round-robin initialization:

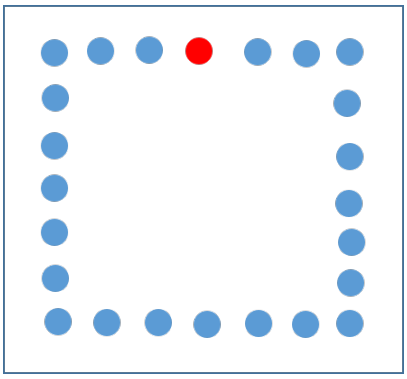


.

- Plot of the result for set2c.txt, with K=2 and round-robin initialization:



Task 3 (10 points)



Consider the set of points above. Each dot corresponds to a point. Consider a clustering consisting of two clusters, where the first cluster is the set of all the blue dots, and the second cluster has the red dot as its only element. Can this clustering be the final result of the k-means algorithm? Justify your answer.

Your answer should be based on your visual estimations of Euclidean distances between dots. For this particular question, no greater precision is needed.

Task 4 (10 points)

For both parts of this task, assume that the algorithm in question **never needs to break a tie** (i.e., it never needs to choose between two distances that are equal).

Part a: Will the EM algorithm always give the same results when applied to the same dataset with the same K? To phrase the question in an alternative way, is there any dataset where the algorithm can produce different results if run multiple times, with the value of K kept the same? If your answer is that EM will always give the same results, justify why. If your answer is that EM can produce different results if run multiple times, provide an example.

Part b: Same question as in part a, but for agglomerative clustering with the d_{min} distance. Here, as "result" we consider all intermediate clusterings, between the first step (with each object being its own cluster) and the last step (where all objects belong to a single cluster). Will this agglomerative algorithm always give the same results when applied to the same dataset? If your answer is "yes", justify why. If your answer is "no", provide an example.

Task 5 (10 points)

Consider a dataset consisting of these eight points: 2, 4, 7, 11, 16, 22, 29, 37.

Part a: Show the results (all intermediate clusterings) obtained by applying agglomerative clustering to this dataset, using the d_{min} distance.

Part b: Show the results (all intermediate clusterings) obtained by applying agglomerative clustering to this dataset, using the d_{max} distance.

Extra Credit Task (optional, 10 points extra credit)

Implement EM clustering. For more precise instructions, please refer to this page.

Optional Tasks, No Credit

Here some tasks that are optional, and for which you will receive NO credit. These tasks will give you some additional hands-on experience with clustering.

- **Optional Task 1, 0 points**: If you have implemented EM clustering, implement a way to identify when the EM algorithm has converged, so as to stop the iterations of the main loop.
- **Optional Task 2, 0 points**: Implement k-medoid clustering, and apply it on the time series dataset of the optional DTW assignment, with DTW as the underlying distance measure.
- **Optional Task 3, 0 points**: Implement agglomerative clustering, with d_{mean} as the distance measure between clusters. Apply your implementation on the time series dataset of the optional DTW assignment, with DTW as the underlying distance measure.
