CS 422/622 Project 1

Due 9/24 11:59pm

Logistics: You must implement everything stated in this project description that is marked with an **implement** tag. Whenever you see the **write-up** tag, that is something that must be addressed in the project README.txt. Graduate students are required to implement everything including items tagged with **622**. Students in 422 do not need to complete these extra elements. **You cannot import any packages unless specifically noted by the instructor.** In this project, you may import the numpy, math (for log), scipy.spatial (for euclidean distance) and random.randint (for random integer generation) packages. You are welcome to ask about particular packages as they come up. The idea is to implement everything from scratch.

Deliverables: Each student should submit a single ZIP file, containing their project code (∗.py files) and your writeup (README.txt). You should create a folder called Project1 (exactly like this with the capital P) and put all your code and your writeup in this folder. Then zip this folder up. That way when I extract your folder I can run my tests from outside the directory without having to change anything. Your zip file should be named lastname_firstname_project1.zip. Your code should run without errors in a linux environment. If your code does not run for a particular problem, you will lose 50% on that problem. You should submit only one py file, named accordingly (e.g. Decision Trees → decision_trees.py). If your file does not match the filename provided exactly, you will lose 50% on that problem.

Grading: I have provided you with a test script (test_script.py) you can use to test out your functions. I will be using a similar test script, so if your code works with this script it should work with mine! The output of the test script should look something like this if you have implemented everything correctly. Each student must work independently. You are to submit your own original work.

```
DT: 1.0
KNN: 1.0
KMeans: [[ 1],[10]]
KMeans: [[2.28571429, 1.28571429], [6.69230769, 5.23076923]]
```

The final KMeans output may be different since you'll use a random number generator to initialize your cluster centers. Feel free to test out your code with other data, but you are provided with 6 data files (specified in each section).

1 Decision Trees (60 Points)

File name: decision_trees.py

Implement a function in python:

`DT_train_binary(X,Y,max_depth)`

that takes training data as input. The labels and the features are binary, but the feature vectors can be of any finite dimension. The training feature data (`X`) should be structured as a 2D numpy array, with each row corresponding to a single sample. The training labels (`Y`) should be structured as a 1D numpy array, with each element corresponding to a single label. `Y` should have the same number of elements as `X` has rows. `max_depth` is an integer that indicates the maximum depth for the resulting decision tree. `DT_train_binary(X,Y,max_depth)` should return the decision tree generated using information gain, limited by some given maximum depth. If `max_depth` is set to `-1` then learning only stops when we run out of features or our information gain is 0. You may store a decision tree however you would like, i.e. a list of lists, a class, a dictionary, etc. **Write-Up**: describe your implementation concisely. Bulleted lists are okay! Binary data for testing can be found in in data_1.txt and data_2.txt.

Implement a function in python:

`DT_test_binary(X,Y,DT)`

that takes test data `X` and test labels `Y` and a learned decision tree model `DT`, and returns the accuracy (from 0 to 1) on the test data using the decision tree for predictions. **Write-Up**: describe your implementation concisely.

Implement the following function in python:

`DT_make_prediction(x,DT)`

This function should take a single sample and a trained decision tree and return a single classification. The output should be a scalar value. **Write-Up**: describe your implementation concisely.

622 Implement the following functions in python:

```
DT_train_real(X,Y,max_depth)
DT_test_real(X,Y,DT)
```

These functions are defined similarly to those above except that the features are now real values. The labels are still binary. Your decision tree will need to use questions with inequalities: $>$, $\geq$, $<$, $\leq$. **Write-Up**: describe your implementation concisely. Real-valued data for testing is provided in data_3.txt

2 Nearest Neighbors (20 points)

File name: nearest_neighbors.py

Implement a function in python:

`KNN_test(X_train,Y_train,X_test,Y_test,K)`

that takes training data, test data, and K as inputs. `KNN_test(X_train,Y_train,X_test,Y_test,K)` should return the accuracy on the test data. The training data and test data should have the same format as described earlier for the decision tree problems. Your function should be able to handle any dimension feature vector, with real-valued features. Remember your labels are binary, and they should be -1 for the negative class and 1 for the positive class. data_4.txt has some test data for your KNN algorithm. Try $K = 1$, $K = 3$, and $K = 5$. **Write-Up**: describe your implementation concisely.

622 Implement the following function in python:

`choose_K(X_train,Y_train,X_val,Y_val)`

that takes training data and validation data as inputs and returns a K value. This function must iterate through all possible K values and choose the best K for the given training data and validation data. K must be chosen in order to achieve the best accuracy on the validation data. **Write-Up**: describe your implementation concisely.

3 Clustering (20 points)

File name: clustering.py

Implement a function in python:

`K_Means(X,K,mu)`

that takes feature vectors `X` and a `K` value as input and returns a numpy array of cluster centers `C`. Your function should be able to handle any dimension of feature vectors and any $K > 0$. `mu` is an array of initial cluster centers, with either `K` or 0 rows. If `mu` is empty, then you must initialize the cluster centers randomly. Otherwise, start with the given cluster centers. **Write-Up**: describe your implementation concisely. data_5.txt and data_6.txt have some sample data for you to test out your implementation.

622 Implement the following function in python:

`K_Means_better(X,K)`

that takes feature vectors `X` and a `K` value as input and returns a numpy array of cluster centers `C`. Your function should be able to handle any dimension of feature vectors and any $K > 0$. Your function will run the above-implemented `K_Means(X,K,[])` function **many** times until the same set of cluster centers are returned a majority of the time. At this point, you will know that those cluster centers are likely the best ones. `K_Means_better(X,K)` will return those cluster centers. **Write-Up**: describe your implementation concisely.