Project 4

Directions

- Expected time commitment: 3-6 hrs
- Download this ZIP file with the data and source code: iris.zip.
- Make sure your python environment has numpy installed
- Submit: the `classifiers.py` file
- Due: Friday, Nov. 16, 5:00PM

Dataset description

This dataset contains two species of iris: *Iris Versicolor* and *Iris Virginica*. Your task is to classify these two species using the Perceptron and K-nearest neighbors algorithms.

The observations are of four types:

- Sepal Length (in centimeters)
- Sepal Width (in centimeters)
- Petal Length (in centimeters)
- Petal Width (in centimeters)

The provided code will load the data and randomly split the data into test and train sets, using a random seed to ensure that results are the same each time you run the code. The parameters (N and K) that are used on the test data are chosen based on a 4-fold cross-validation of the training data.

Task 1 — Perceptron (4 pts)

As discussed in class, the perceptron algorithm finds a linear boundary between two classes (the general term for a linear boundary in multidimensional space is **hyperplane**). Your job is to code the `train()` method of the `Perceptron` class.

Your code should follow the algorithm listed in the comment at the top of the train method. It may additionally be helpful to scan the rest of the methods in the `Perceptron` class.

When your code is complete, you should be able to run `python main.py`, which will run cross-validation with different values for the parameter N, representing the number of times through the data to train. It will then use the best-performing value of N to train on all the training data, to achieve an accuracy of 92% on the test data.

Task 2 — K-nearest neighbors (4 pts)

Perceptron is limited in that it can only produce a boundary that is a hyperplane. We may be able to improve on this by using the k-nearest neighbors algorithm to classify the data. However, this algorithm doesn't require any training, so you will be coding the `predict()` method of the `KNN` class, which takes one input and generates a prediction for that input.

Again, you will find an algorithm listed in the comments for `predict()`. Note that it includes reference to the `Maxheap` class. To complete this algorithm, you will need to maintain a list of the k closest points to the current point, which you can do by checking if each point in the data is closer than the item in the list that is currently the farthest from the current point. If you use the `Maxheap` class correctly, you don't have to check against the entire list for each new data point, which is a much more efficient algorithm.

Again, when the code is complete, you should run `python main.py` to do cross-validation to come up with a good value for k. The program will also use the full train set to classify the test set, and should come up with a final accuracy of 92%.

Task 3 — Exploration (2 pts)

Explore the dataset and the algorithms a little by doing the following two tweaks:

1. Try at least two other values for the random seed that controls the initial weights for the perceptron. Does the random seed have an effect on the accuracy of your algorithms? Record the results of your experiments in the header at the top of the `classifiers.py` file.
2. Try at least two other distance functions for the kNN algorithm. Do they have any effect on the accuracy of your algorithm? Report the distance functions used and the effect on accuracy at the top of the `classifiers.py` file.