Machine Learning - Project 2

Hoang-Dung Bui, Akram Reshad

This is the write-up for Machine Learning - Project 2. The code should run on python 3.

1 Nearest Neighbors

There are three functions in the file nearest_neighbors.py: *KNN_test(X_train, Y_train, X_test, Y_test, K)*, *choose_K(X_train, Y_train, X_val, Y_val)*, and *Distance(X,Y)*.

def Distance(X, Y): This function is to return the distance between two point.

def KNN_test(X_train, Y_train, X_test, Y_test, K): This function takes four arrays *X_train, Y_train, X_test, Y_test* and K - number of nearest neighbors as the inputs. For each data *xtest* in *X_test*, the algorithm will run through *X_train* dataset, select K nearest neighbors to *xtest* by comparing the distance from the neighbor to *xtest*. After finding K nearest neighbors, the algorithm would make prediction based on the label of the neighbors *Y_train*. The *num_Correct* variable is used to measure the accuracy of the algorithm. Each time the prediction is right, the variable *num_Correct* will increase 1. After testing all the test sample, the accuracy is derived by division of variable *num_Correct* over the total sample.

def choose_K(X_train, Y_train, X_val, Y_val): this function will return the number of nearest neighbors K with highest accuracy. The training data contains 14 samples, so the K can be the odd numbers in the range of *1,3, ..., 13*. K is running by a *while* loop.
For the data above, $\mathbf{K = 9}$ will bring the highest accuracy 70%

2 Clustering

There are two functions in clustering task: *K_Means(X,K)* and *K_Mean_better(X,K)*.

def K_Means(X,K): take dataset X and number of clusters K as inputs and returns K cluster center positions. The initial center will be chosen randomly from dataset X, so the initial center for each cluster will be different for each running.
The idea is calculating all distances from the sample of dataset to the center points. The distance will be

compared to get the shortest one, then the data sample will be assigned to the cluster of the corresponding center points. After clustering all the data samples, the center point of each cluster will be recalculated. A *while* loop will be used to run algorithm, and will stop as the center positions are not changed (the update are zero). The new cluster center will be updated based on the average of all member's positions. A points belongs to a cluster if its distance to a centers is shorter.

The result of *K_Mean(X,K)* function with K = 2 for one time running. The center positions will be changed if the initial center are different.

Figure 1: Two centers of the clusters.

The result of K_Mean(X,K) function with K = 3 for one time running. The center positions with K = 3 are usually converge to the three positions: (2.29,1.29), (3.83,7.0), (9.14,3.71).

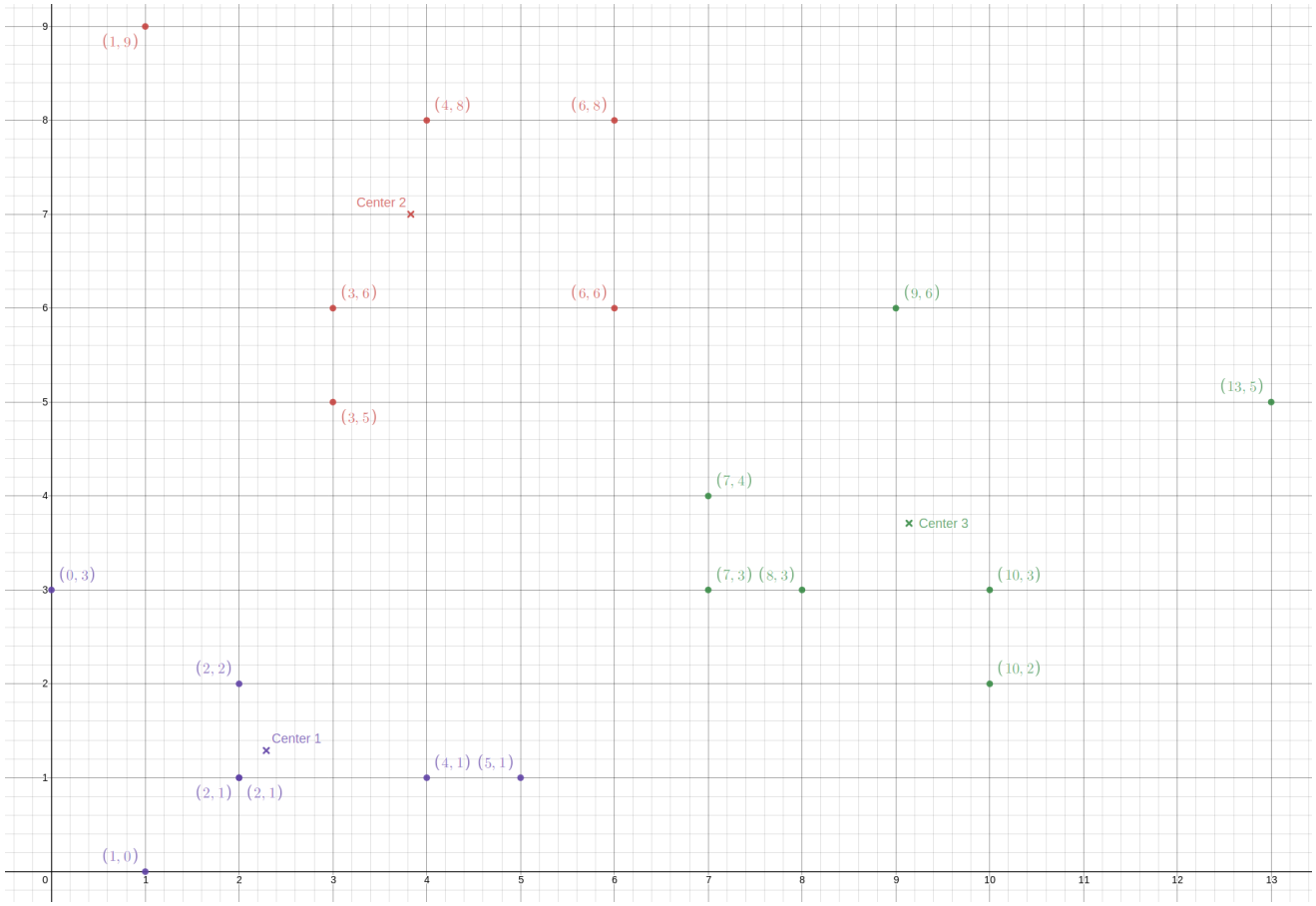

Figure 2: Three centers of the clusters from K_Mean(X,3).

K_Means_better(X,K)

This function will call function *K_Means* multiple times (here is 100 times), then calculate and return the best center as clustering. If the invoking of function *K_Means* returns a new clustering center, it will be added to the center vector. If not, the algorithm will find the existed clustering center with the same position and increase the center's index. There is a array *center_index* which stores the index value. The *while* loop will be terminated if it runs 100 rounds or a center get the majority of the appearance (higher than 80%).

The result of K_Mean_better(X,K) function with K = 2 is shown following. Due to the unstable of the initial centers, the convergence of this case are weak. The percentage of center with highest occurs are around 30%, and the *while* loop stops as reaching 100 iterations.

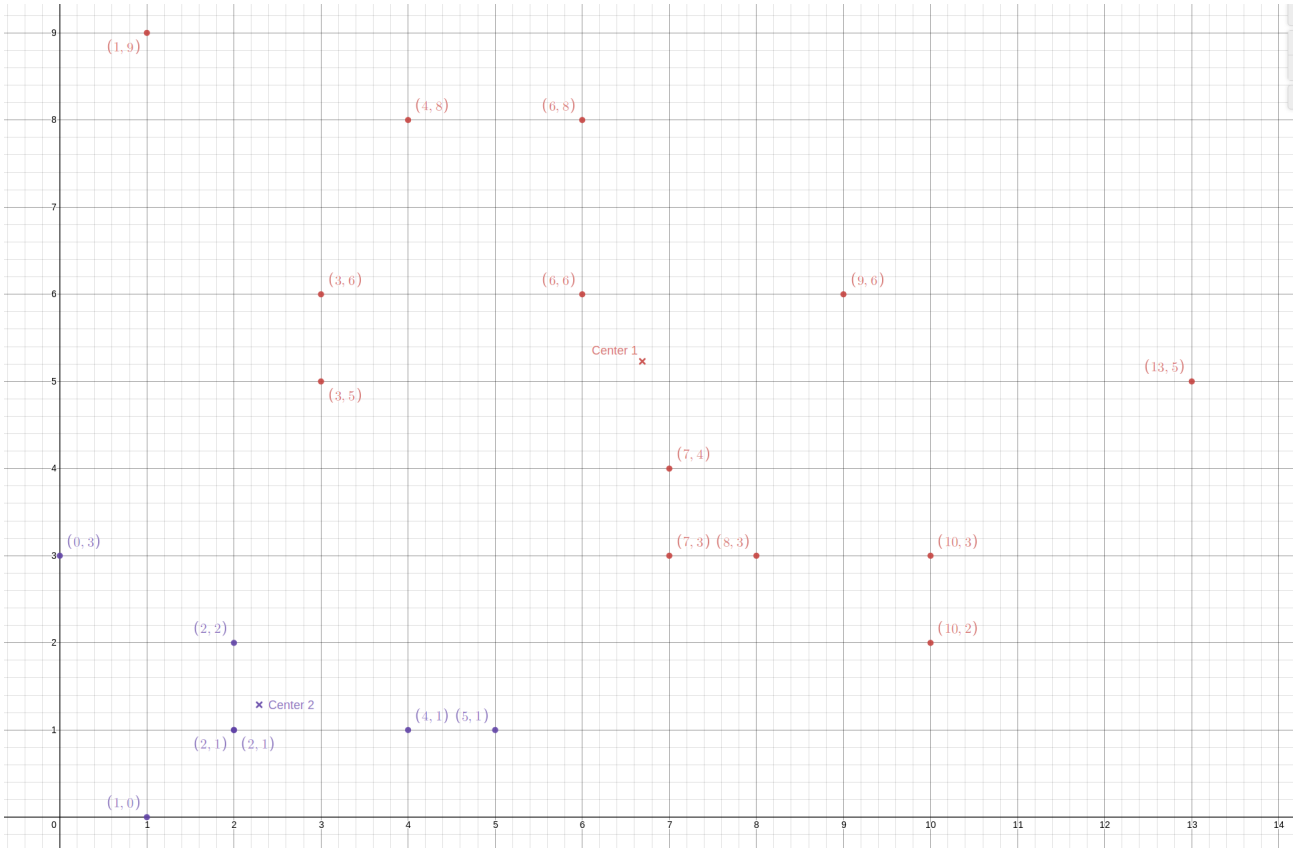

Figure 3: Two best centers of the clusters from K_Mean_better(X,2).

K_Mean_better(X,K) with K = 3 provides a better convergence. For all the time running the algorithm, the convergence of the clustering center reached 86% and never reach 100 iterations. The best clustering centers are shown on Figure 2.

3 Perceptron

In *perceptron.py* file, there are two functions: *perceptron_train(X, Y)* and *perceptron_test(X_test, Y_test, w, b)*.

def perceptron_train(X,Y) consists of a *while* loop to update the weight vector and bias for each data sample. If there is no update to the weights and bias with all samples, the *while* loop will stop and return the weights and bias.

From the dataset of project 2 with initial weights and bias are [0, 0] and 1, respectively, *perceptron_train* function will return weight vector = [1.0, 3.0] and the bias = 2. The decision boundary is shown as following.

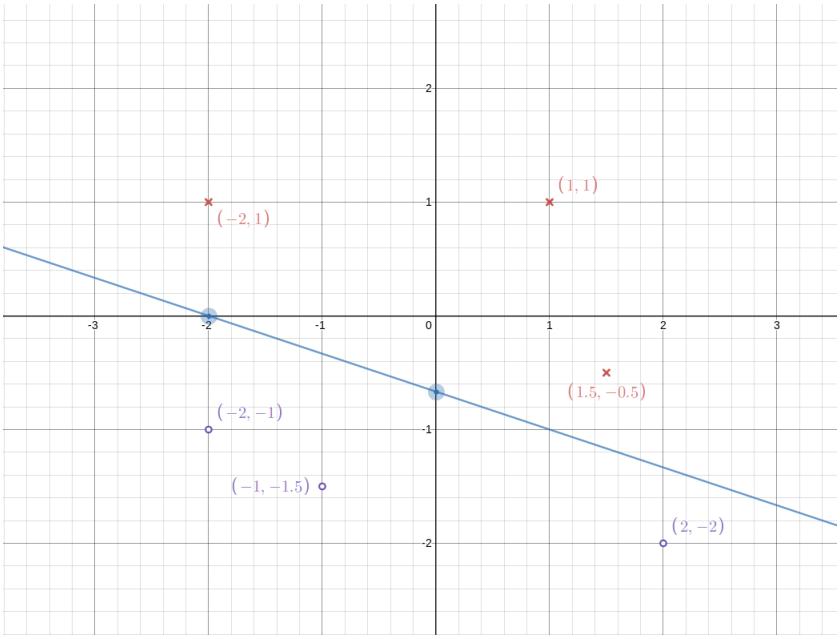

Figure 4: Boundary line of perception function.

def perceptron_test(X_test, Y_test, w, b) will return the accuracy of the model. The is a *for* loop will run through the *X_test*, calculate activation value for each sample, and make a prediction. If product of the prediction and the corresponding label is positive, the *correct* variable will increase by 1. The accuracy is calculated by dividing the *correct* variable over the total sample. With the weights and bias above, the accuracy returns from *perceptron_test* on the train data is 100%.