PROJECT 2

CS691

Due Date : October 19, 2019

Sharif Kamran

Adam Lychuk

October 19, 2019

Contents

0.1 Task 1

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

The function takes the training and test samples along with the labels and the K as input to build a k-nearest neighbor. For each of the test samples we iterate and find the k nearest sample with respect to that test case. As a result, we calculate the euclidean distance using **Calculate_distance** function from the each of the training cases from that particular test case and append it into an array. After that we sort the array with **sorted_neighbor** function and keep a tracker for the indexes. This will help us to track the labels (signs) of each of the training case later on and find the majority sign (positive or negative) for that particular test case. For finding the sign of the majority case we use the **sign_pred** function and append the prediction to an array. After that, we call the **Accuracy** function send the prediction values and Y_test as input to get the overall accuracy.

0.1.2 Results: Predictions for K=1, K=3 and K=5

True label	K=1	K=3	K=5
1	-1	1	1
-1	1	1	-1
1	1	1	-1
-1	-1	-1	1
1	-1	-1	-1
-1	1	-1	-1
1	1	1	1
-1	1	1	1
1	-1	-1	1
-1	1	1	-1
Accuracy	30%	40%	60%

0.1.3 choose_K(X_train,Y_train,X_val,Y_val)

The function takes the training and validation samples as input and tries to find the best K for the give validation samples (test). For this it iterates from $K = 1$ to $K = number of training samples$ and then calls the **KNN_test** function as finds the accuracy for each of the sample. It also keeps the best accuracy for the given K so that it can return it as the best K for the test samples.

0.1.4 Best K

The best K for given test sample is 9. With accuracy of 70% this is the best performing K among all the values from 1 to 13.

0.2 Task 2

0.2.1 K_Means(X,K)

This function performs the K-Means algorithm. The function takes the samples, which can be any set of n-dimensional feature vectors, and assigns each to a random class. The number of classes is decided by the number of centers which is decided by input K. After this initialization the algorithm runs for an arbitrary 100 iterations in the hopes of convergence. Each iteration the function finds the distance between each cluster center and each sample point. It then assigns points to the initialized classes based on the center they're closest too. At the end of this process we check to see if any points have been reassigned to classes. If no points have been reassigned, k-means converged. Otherwise we compute new centers, by taking the mean of points in each class, and assigning that mean as the new center.

0.2.2 K_Means_better(X,K)

This function calls K_Means(X,K) until center values are found that are generally the same. Once it identifies those centers it returns them.

0.3 Task 3

0.3.1 perceptron_train(X,Y)

The function takes training sample sand labels as input and then iterates over them and optimizes the weights and the bias. First **weight_initialize** function is called where random weights are initialize with values of $[-1,1]$ with four decimal places. After that we iterate for 10 epochs to take one sample each time and do Dot product operation between the weight and the samples. After that we calculate if the activation is bigger than zero or not. If not then we update both the weights and the bias. The activation is calculated using the **activation** function and the weights are updated by calling **update_weights** function. We return all the weight value along with the bias at the end of the function.

0.3.2 perceptron_test(X_test, Y_test, w, b)

The function uses testing samples and labels along with weights and biases as input and finds the overall accuracy. First it iterates through the test samples and using the **activation** function finds the prediction and appends it into an array. Then it matches the prediction to the true label and finds how many of them are correct. Lastly, it finds the the overall accuracy in terms of percentage and returns it.

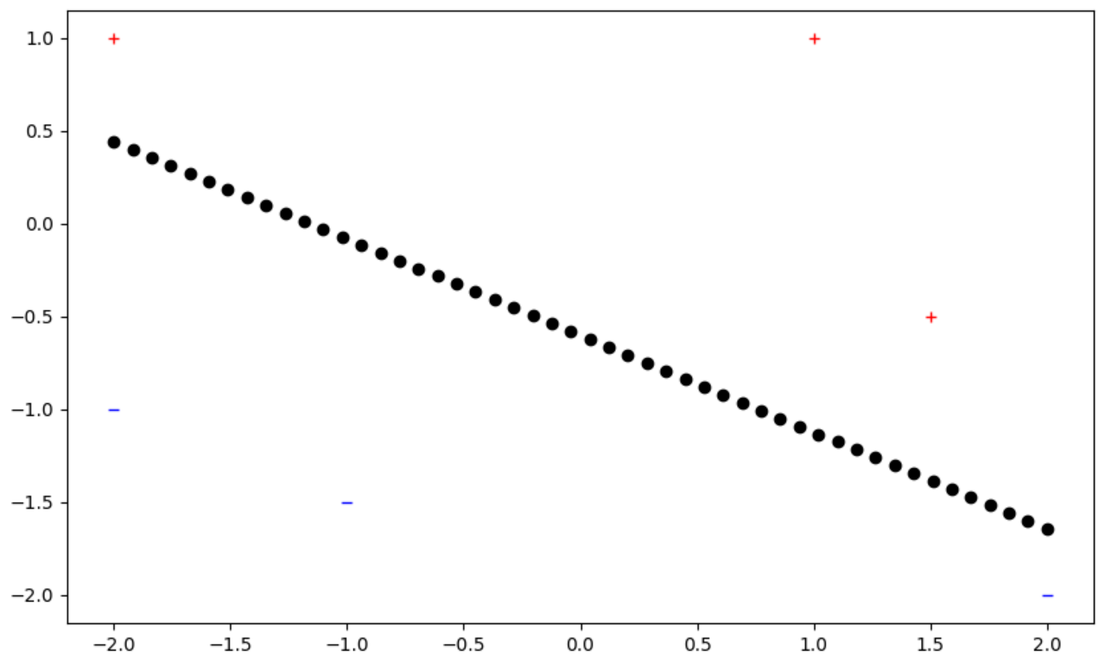

Figure 1: Decision Boundary