Assignment 1

Carlos Ordonez

2/13/19

Perceptron Learning Algorithm

1 Solution to Problem 1.4

1.1 (a)

To generate a linearly separable data set of size 20 we set N in the following code to 20 and it produces the following graph

```
def main():
    itlst = []
    for x in range(1):
        N = 20
        # data
        X, y = make_blobs(n_samples=N, centers=2, n_features=2)
        y[y==0] = -1  # replace the zeros
        X = np.append(np.ones((N,1)), X, 1)    # add a column of ones

        # initialize the weigths to zeros
        w = np.zeros(3)
        it = 0
        pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
        while classification_error(w, X, y) != 0:
            it += 1
            # Pick random misclassified point
            x, s = choose_miscl_point(w, X, y)
            # Update weights
            w += s*x
        pltPer(X,y,w)
        print("Total iterations: " + str(it))
        itlst.append(it)
    print(itlst)
```

1.2 (b)

When running the above perceptron algorithm their were 3 iterations and the below chart shows the final solution. As you can see the solution line is very close to the top set meaning it is probably relatively far from the target function which would probably be more centrally located between the two sets.

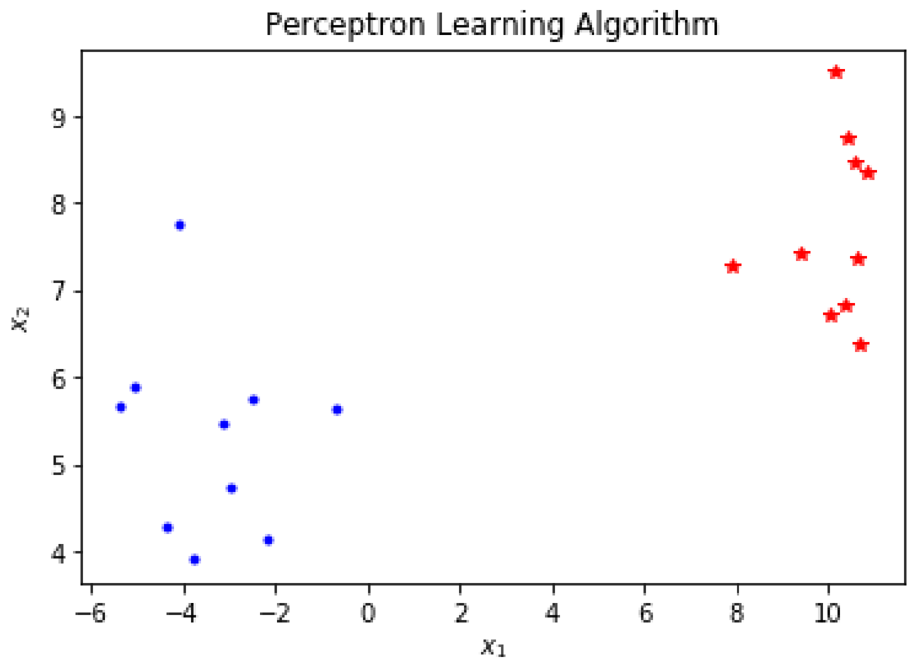

Figure 1: Plot of data set of with N=20 (1_4a)

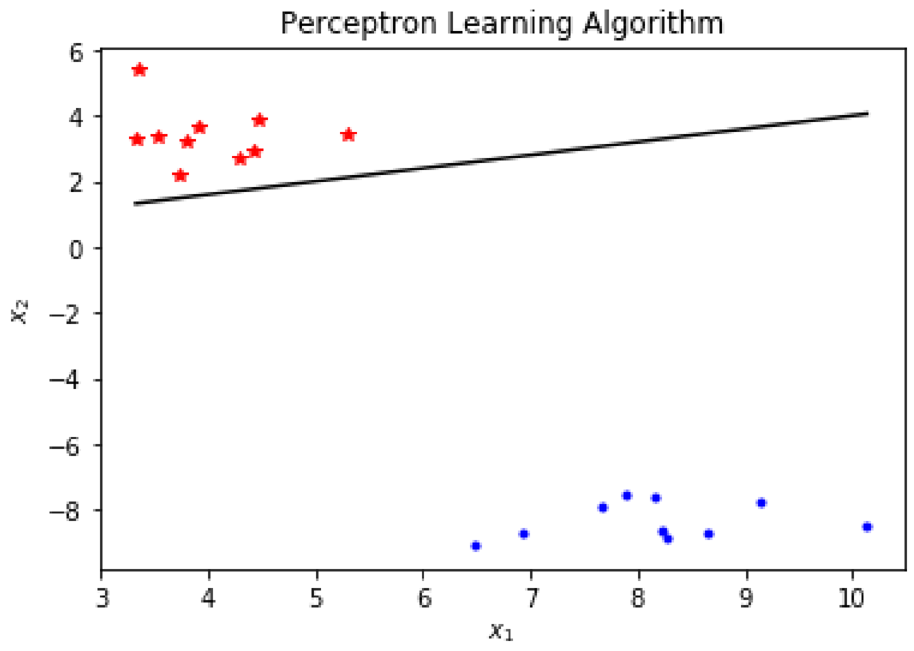

Figure 2: Plot of data set of with N=20 (1_4b)

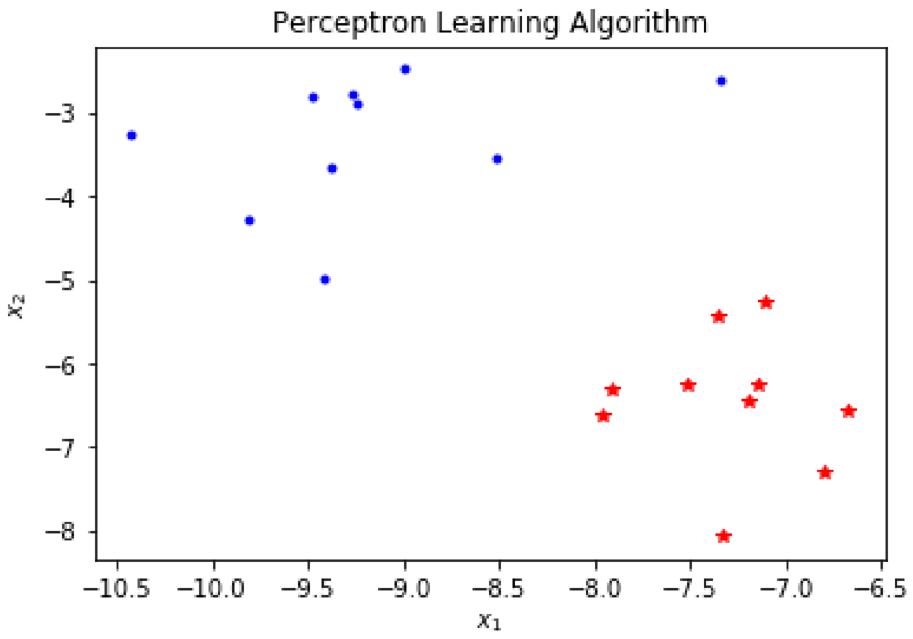

Figure 3: Plot of data set of with N=20(1_4c1)

1.3 (c)

When the results above are repeated we get another data set as above that is solved in 2 iterations with the following graph below. This chart seems to have a solution line which is closer to the lower group again meaning it is relativly far from the actual target function.

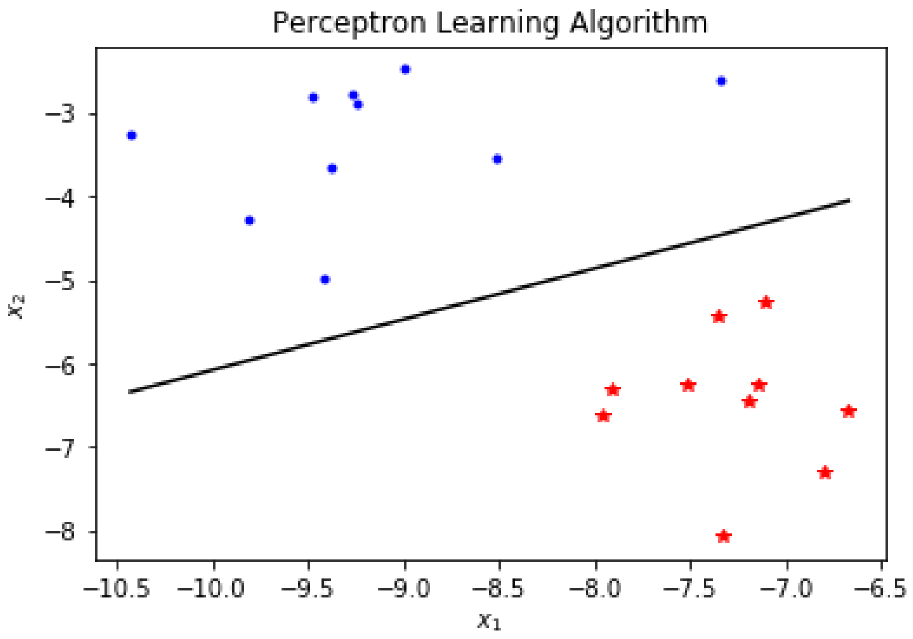

Figure 4: Solution of data set of with N=20(1_4c2)

1.4 (d)

If we repeat the above exercise again but with a data set with 100 instances instead of 20 we simply change N in the original algorithm to 100. By doing this we get a data set that is solved by the algorithm again in 3 iterations to get an initial data set and a final solution of the following chart. This solution line seems to be centrally located between the two sets and seems to be closer to the actual target function than the data sets with only 20 data points.

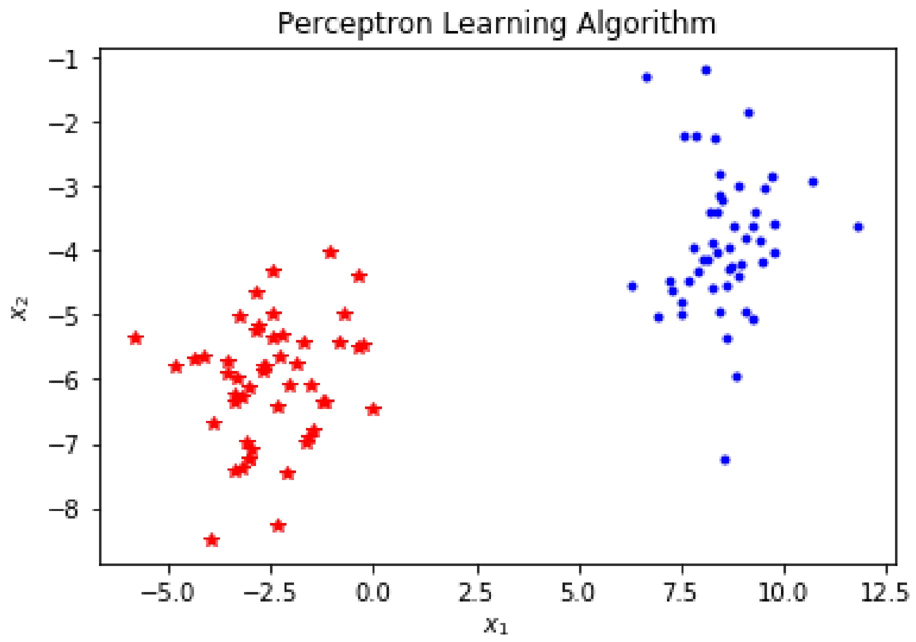

Figure 5: Plot of data set of with N=100 (1_4d1)

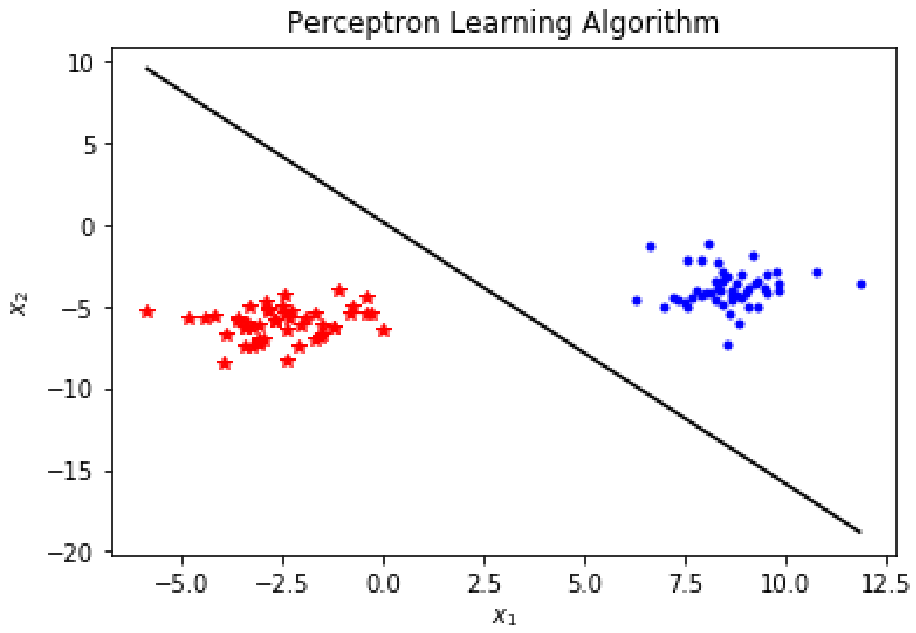

Figure 6: Solution of data set of with N=100(1_4d2)

1.5 (e)

Again when we change N = 1000 and repeated the algorithm we get the following charts. The solution function seems to be exactly in the middle of the two sets of points meaning it is probably much closer to the actual target function as compared to the two previous data sets

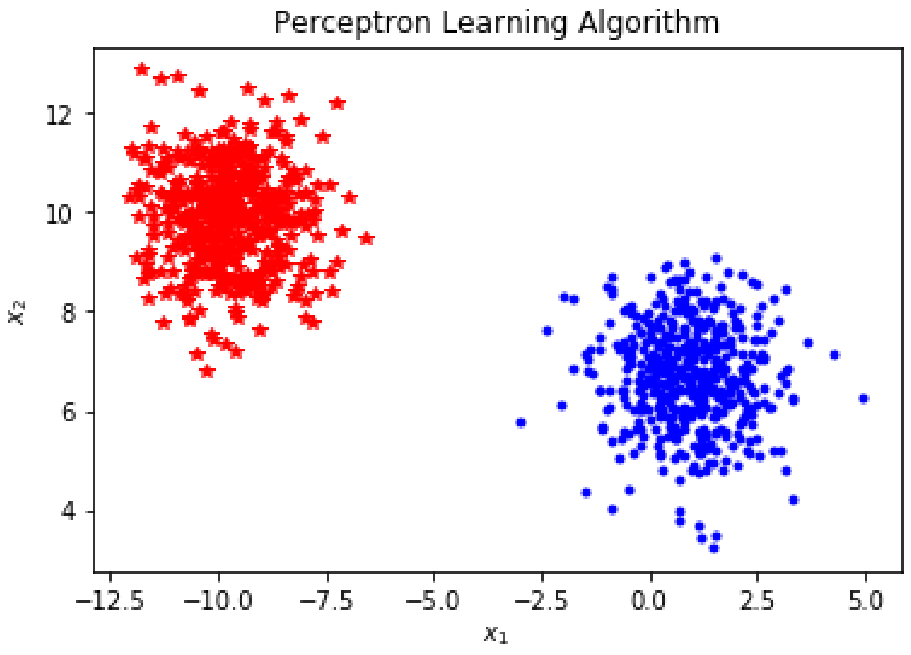

Figure 7: Plot of data set of with N=1000(1_4e1)

1.6 (f)

To change the data set to exist in 10 dimensions instead of 2 we must change the $n_features$ to equal 10 and initialize 11 weights at 0 (one for each dimension and an additional one for the bias) this will change the code to look as follows

```
def main():
    itlst = []
    for x in range(1):
        N = 20
        # data
        X, y = make_blobs(n_samples=N, centers=2, n_features=10)
        y[y==0] = -1  # replace the zeros
        X = np.append(np.ones((N,1)), X, 1)   # add a column of ones

        # initialize the weigths to zeros
        w = np.zeros(11)
        it = 0
        pltPer(X,y,w)  # initial solution (bad!)

        # Iterate until all points are correctly classified
        while classification_error(w, X, y) != 0:
            it += 1
            # Pick random misclassified point
            x, s = choose_miscl_point(w, X, y)
            # Update weights
            w += s*x
        pltPer(X,y,w)
        print("Total iterations: " + str(it))
        itlst.append(it)
    print(itlst)
```

1.7 (g)

2 Conclusions

As N increases so does the time it takes for the algorithm to finish but as N increases the final function gets closer and closer to the target function making the algorithm better with a higher N value. The dimension of your data set acts in a similar way where a higher D results in a shorter run time because it is easier to find a plane in higher dimensions which separate the points than in smaller dimensions. More points are needed though to provide a good result

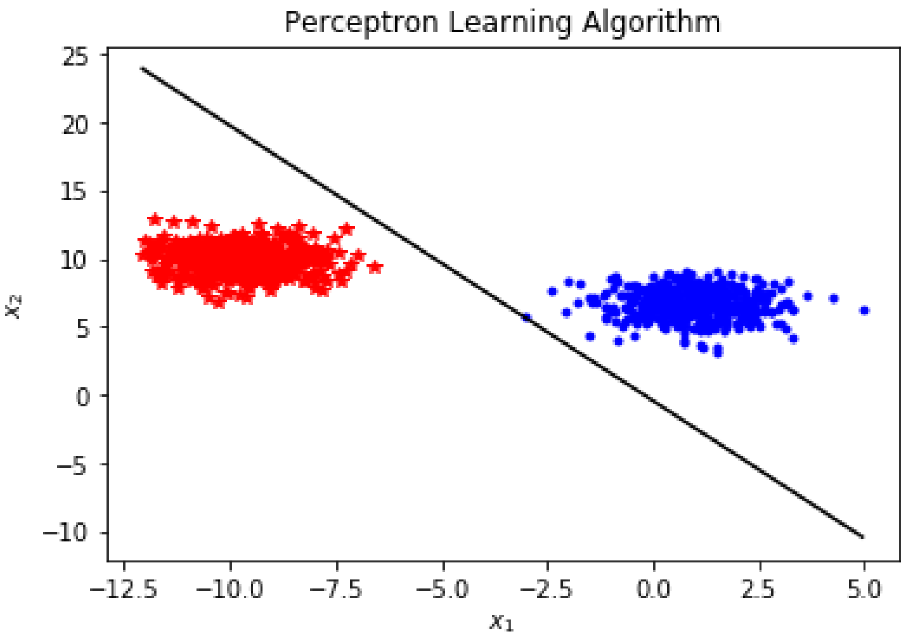

Figure 8: Solution of data set of with N=1000(1_4e2)

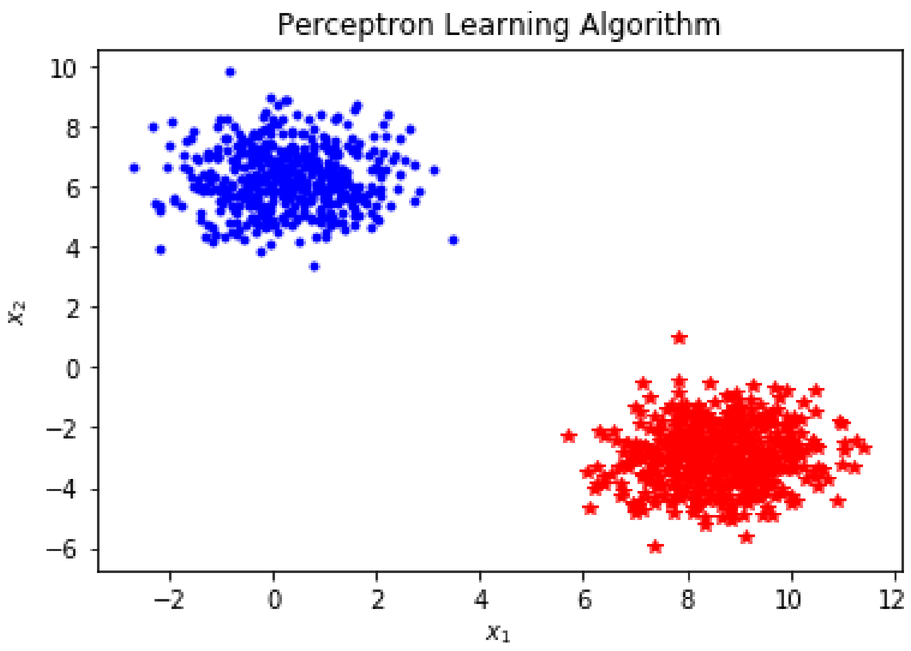

Figure 9: Solution of data set of with N=100(1_4f1)

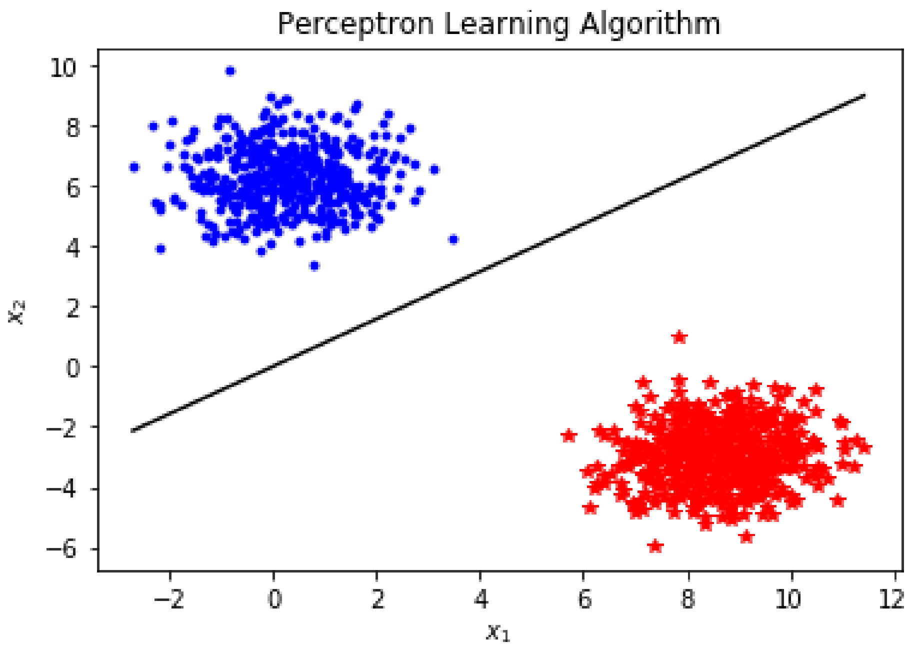

Figure 10: Solution of data set of with N=100(1_4f2)

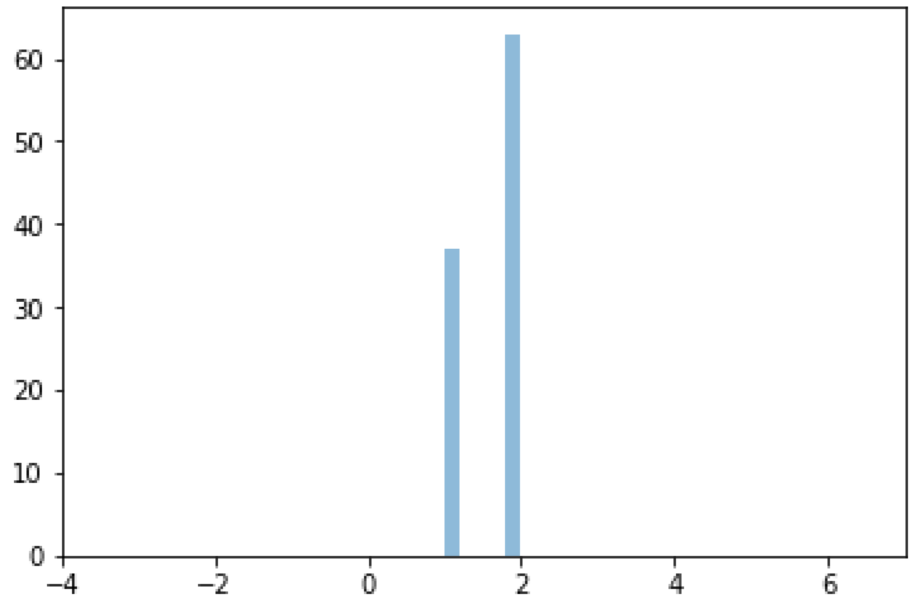

Figure 11: Solution of data set of with N=100(1_4g)