BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

BITS F464 Machine Learning

Name	ID No.
Donkada Vishal Dheeraj	2018A7PS0239H
Thakur Shivank Singh	2018A7PS0439H
Pranav Reddy Pesaladinne	2018A7PS0238H

Linear Perceptron

- Thakur Shivank Singh - 2018A7PS0439H
- Pranav Reddy Pesaladinne - 2018A7PS0238H
- D Vishal Dheeraj - 2018A7PS0239H

Brief Description:

This is an implementation of the Linear Perceptron model in Python using only the NumPy library. The algorithm takes the following steps:

- Read the two datasets into numpy arrays

```
#reading the dataset

df1 = np.genfromtxt('dataset_LP_1.txt',delimiter=',')
df1[df1==0] = -1
df2 = np.genfromtxt('dataset_LP_2.csv',delimiter=',')
df2[df2==0] = -1
```

- Replace all occurences of class 0 with class -1
- Split the data into training and testing data in a 70:30 split

```
def train_test_split(dataset, frac):
  np.random.shuffle(dataset)
  indices = np.random.permutation(dataset.shape[0])
  training_idx, test_idx = indices[:int(frac*dataset.shape[0])], indices[int(frac*dataset.shape[0]):]

  X_train, y_train, X_test, y_test = dataset[training_idx,:-1], dataset[training_idx, -1:], dataset[test_idx,:-1], dataset[test_idx, -1:]
  X_train = X_train.T
  X_test = X_test.T
  return X_train, y_train, X_test, y_test
```

- The idea of this algorithm is to find a hyperplane of the form $\omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots = 0$. We do this by randomly initializing the weight vector ω and then iterating through the dataset and updating the weight vector according to the equation $\omega^{\tau+1} = \omega^{\tau} - \eta(-t_n x_n)$ where x_n is a misclassified sample and t_n is its corresponding class. We do this either until the cost function reaches zero or for 10^6 iterations, whichever comes first

```
def train_perceptron(X,y,w,n = 1000000,lr = 1):
  size = X.shape[1]
  count=0
  epochs=int(n/size)
  for epoch in range(epochs):
    if(epoch>0):
      if count==0:
        print(f"No Misclassified samples at this point, Iteration: {it}")
        break
      else:
        print(f"After {epoch} epochs, number of Misclassified samples are: {count}")
        count=0
    for it in range(size):
      cur_X = np.reshape(X[:,it],newshape=(X.shape[0],1))
      cur_Y = y[it]

      temp = np.dot(w.T, cur_X)
      if temp>=0:
        temp = 1.0
```

```
      else:
        temp = -1.0

      if temp!=cur_Y:
        w = np.add(w,lr*cur_Y*cur_X)
        count = count+1
    if count==0:
      break
  return w
```

- We then test the model on the test sets

```
def predict(dataset):
  misclass_count1 = 0
  misclass_count2 = 0

  if(dataset==1):
    for i in range(X_test1.shape[1]):
      X_testing = X_test1[:,i].T
      y_testing = y_test1[i]
      count = loss_function(X_testing,y_testing,w1)
      misclass_count1 += count
    print("Miscalculations in Test1 data : ",misclass_count1)
    acc1 = 100*(1 - (misclass_count1/X_test1.shape[1]))
    print("Accuracy of Test 1 model: ", acc1, "%")
  elif(dataset==2):
    for i in range(X_test2.shape[1]):
      X_testing = X_test2[:,i].T
      y_testing = y_test2[i]
      count = loss_function(X_testing,y_testing,w2)
      misclass_count2 = misclass_count2 + count
    print("Misclassifications in Test2 data : ",misclass_count2)
    acc2 = 100*(1 - (misclass_count2/X_test2.shape[1]))
    print("Accuracy of Test 2 model: ", acc2, "%")
```

- Accuracy on Test Set 1 is:

```
Final w is: [[-3.47931324]
 [-2.71740579]
 [-2.75108043]
 [-2.3571315 ]]
Miscalculations in Test1 data :  25
Accuracy of Test 1 model:  93.93203883495146 %
```

- Acucuracy on Test Set 2 is:

```
Final w is: [[-0.12267704]
 [ 0.15111662]
 [ 2.4724525 ]]
Misclassifications in Test2 data :  1
Accuracy of Test 2 model:  99.66666666666667 %
```


Due to random initialization of weights initially, the accuracies for both datasets vary but on average, Dataset 2 is much more linearly separable when compared to Dataset 1. Dataset 1 always hovers around 93% to 94% accuracy but Dataset 1 averages at 99%+ and sometimes even goes to 100% which leads to the conclusion that it is perfectly separable.

Limitations of Linear Perceptron:

- The biggest disadvantage is that it only provides a linear boundary, so if the dataset is not linearly separable, then the Perceptron can never have a perfect accuracy.
- Another issue is that the algorithm is similar to stochastic gradient in the sense that every update to the weight vector is done using only a single sample from the dataset each time. This means that the Loss function may not always decrease with every iteration, only the Loss with respect to that one sample will decrease. This means that the perceptron can go for a large number of iterations without getting much better than it already is