Assignment – 1
Deepanshu Garg(201501167)

Problem-1

1. The question is in four parts. The feature vectors are made in the start using np.genfromtxt() and made into np.ndarray(). The first column in the X matrix has been made all 1s. The common Perceptron algorithm is that the weight vector W has been initialised with 0s and then Xs whose prediction is wrong are added to W. I’ve multiplied X[i] with -1 for those cases where Y[i] == 0 so that while training, all predicitons should be the same.
   1. In the first part for every x in the training set, add it to the W vector if it is being wrongly classified.
   2. In the second part, I’m keeping a margin, therefore, for the checking condition while training, if dot(W, x)<=b (where b is the margin), x is added to W.
   3. In the third part, for every epoch, the wrongly classified xs are added and finally added to W at the end of the epoch.
   4. Forth part is the combination of the 2nd and the 3rd part, it keeps a margin and adds the wrongly classified xs in the end of the epoch.
2. Observations:
   1. In the first part, the algorithm converges really fast and on the provided dataset, it converges within 20 epochs.
   2. Second part takes longer to converge because it keeps a margin.
   3. 3rd part the batch perceptron was converging at 2000+ epochs on the given dataset.
   4. 4th part took even longer to converge.
3. Results:
   1. Basic perceptron:
      1. Accuracy: 99.905%
      2. Precision: 99.91%
      3. Recall: 99.91%
   2. Perceptron with Margin:
      1. Accuracy : 99.905%
      2. Precision: 99.91%
      3. Recall: 99.91%
   3. Batch Perceptron without Margin:
      1. Accuracy: 99.95%
      2. Precision: 100%
      3. Recall: 99.91%
   4. Batch perceptron with Margin:
      1. Accuracy: 99.95%
      2. Precision: 100%
      3. Recall: 99.91%

Problem-2

1. Part - 1
   1. The data was divided in two parts train and test (80:20 split).
   2. The 1st part is gradient descent, similar to the perceptron algorithm, the W vector is updated when an x is wrongly classified. W is updated with $(b-W_T X)*X*LearnRate/norm(X)$.
   3. Observation: The data isn’t linearly separable and hence the algorithm doesn’t converge.
   4. Results:
      1. Run 1
         1. Accuracy: 99.03%
         2. Precision: 97.61%
         3. Recall: 100%
      2. Run 2
         1. Accuracy: 96.11%
         2. Precision: 96.11%
         3. Recall: 92.68%
2. Part-2:
   1. The dataset is divided in 80:20 ratio in train and test data.
   2. The modification I did for handling outliers are that I’m taking eta=1 and updating it to eta*=(1-accuracy) at the end of every epoch. And the update to W is W+=eta*x for wrongly classified xs.
3. Observation:
   1. The data is not linearly separable and hence the algorithm never converges.
4. Results:
   1. Run 1:
      1. Accuracy: 98.05%
      2. Precision: 100%
      3. Recall: 93.93%
   2. Run 2:
      1. Accuracy: 94.05%
      2. Precision: 87.2%
      3. Recall: 100%
   3. Run 3:
      1. Accuracy: 98.03%
      2. Precision: 100%
      3. Recall: 96%

Problem-3

1. I’ve implemented a binary decision tree. It has been implemented as follows:
    1. The features which are strings have been mapped to integer values.
    2. The maximum and the minimum each feature can take are passed to every node.
    3. I’m iterating over features and then the values the features can take and finding the best quality which can be attained.
    4. For the continuous values, the step size at which I’m checking the quality is (max-min)/100 and for the discrete(integer) values, the step size is 1.
    5. The node is made into a leaf node if the number of data points passed to this are less than or equal to 5 or the dataset has all the data points belonging to the same class.
2. Observations:
    1. Number of nodes: ~2500
3. Results:
    1. Accuracy: ~97%

<u>Problem-4</u>

1. The knn algorithm is implemented as follows:
   1. A vocab has been made that maps words in the train dataset to integers.
   2. Feature vector for each file in the train data set is made where feature is the frequency of the words.
   3. Feature vector for each file in the test data set is made where feature is the frequency of the words.
   4. The data is normalised such that the feature vector x is divided by the total number of the words in the corresponding files. That is the feature vectors are now the probabilities of words in the files.
   5. I’m taking the manhattan distance of files in test with the ones in train and finding the most common one in 4 nearest neighbours (and the nearest farthest one of a class in case of ties).
2. Observation:
   1. K = 1
      1. Confusion Matrix:

```
[[27  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  1  0  0  0]
 [ 0  0 10  0  0  0  0  0  0  0]
 [ 0  0  0 19  0  0  3  0  0  0]
 [ 0  0  0  0 23  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  0 22  0  0  0]
 [ 0  0  0  0  0  0  0 33  0  0]
 [ 0  0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  0  0  0  0  6]]
```

      2. Accuracy: 97.53%
      3. Precision: 98.4%
      4. Recall: 98.04%
   2. K = 2
      1. Same as k = 1 because of how the algorithm is implemented.
   3. K = 3
      1. Confusion Matrix:

```
[[27  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  1  0  0  0]
 [ 0  0 10  0  0  0  0  0  0  0]
 [ 0  0  0 19  0  0  3  0  0  0]
 [ 0  0  0  0 23  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  0 22  0  0  0]
```

```
[ 0  0  1  0  0  0  0 32  0  0]
[ 0  0  0  0  0  0  0  0 15  1]
[ 0  0  0  0  0  0  0  0  0  6]]
```

   2. Accuracy: 96.92%
   3. Precision: 96.12%
   4. Recall 97.32%
4. K = 4
   1. Confusion Matrix:

```
[[27  0  0  0  0  0  0  0  0  0]
 [ 0 25  0  0  0  0  1  0  0  0]
 [ 0  0 10  0  0  0  0  0  0  0]
 [ 0  0  0 18  0  0  4  0  0  0]
 [ 0  0  0  0 23  0  0  0  0  0]
 [ 0  0  0  0  0 10  0  0  0  0]
 [ 0  0  0  0  0  0 22  0  0  0]
 [ 0  0  1  0  0  0  0 32  0  0]
 [ 0  0  0  0  0  0  0  0 16  0]
 [ 0  0  0  0  0  0  0  0  0  6]]
```

   2. Accuracy: 96.92%
   3. Precision: 97.23%
   4. Recall: 97.49%