Assignment 1 Report

Question1:

Observations on data by splitting data into 80:20 train validation set.

Simple Perceptron Without Margin:
Accuracy: 0.999210422424
Precision: 0.999237804878
Recall: 0.999237804878

Simple Perceptron With Margin:
Accuracy: 0.999210422424
Precision: 0.999237804878
Recall: 0.999237804878

Batch Perceptron With Margin:
Accuracy: 0.999210422424
Precision: 1.0
Recall: 0.998475609756

Batch Perceptron Without Margin:
Accuracy: 0.999210422424
Precision: 1.0
Recall: 0.998475609756

Remarks: Data always converges. No changes observed on changing the value of margin. Max margin used is 1000000

Question2:
Observations on data by splitting data into 80:20 train validation set.

Simple Perceptron With Relaxation and Margin:
Accuracy: 0.805825242718
Precision: 0.714285714286
Recall: 0.789473684211

Modified Perceptron:
Accuracy: 0.883495145631

Precision: 0.933333333333
Recall: 0.736842105263

Implementation of Modified Perecptron:
Its just a modification of simple batch perceptron with margin which accounts for the number of errors a particular weight vector gives. Each weight vector at the end of an epoch is multiplied by a cost which is inversely proportional to the number of errors it commits. In this way weight vectors which results into huge errors are highly penalised. Apart from that, I checked if the number of errors a particular weight vector is giving is greater than some 2-3 % of total samples, then we don't need to update the weight vector.

Remarks: Attained high accuracy compared to Simple Relaxed Perceptron

Question3:

Observations on data by splitting data into 80:20 train validation set.

Decision Tree:

Accuracy: 0.911555555556
Precision: 0.845238095238
Recall: 0.778793418647
Got 8/9 correct out of sample test data

Implementation of Decision tree:
Implemented a binary tree for the given data by making the discrete values to continous using the most appropriate point (according to entropy function) as the split point for any attribute. Once we have got the split point, we use it to split the data on the basis of this split value of that attribute and continue it recursively for left and right branches of the tree. If it reaches the max_depth or minimum number of data points in a particular branch comes out to be less then we make it as the terminal node and return the maximum occuring class.

Remarks: Tried out using different measures for finding best split points like gini-index, information gain etc, but entropy gave the best result.

Question4:

Accuracy: Got 8/10 right out of the sample test data for k=10

Implementation of BoW representation :
Tokenized all the sentences from all the files in all the folders. Took only unique words into my vocabulary and made them as features. Using these features, I generated feature vectors for all the files in different folders by making the feature values equal to the frequency of the that feature in that file, giving out the Bag of words representation.

Implementation of KNN Classfier:
Given train and test feature vectors,
1) Take one instance of test data and for this find the k nearest neighbors in the training data.

2) Distance metric used is manhattan distance (tried using euclidean distance too)
3)Out of this k nearest neighbors, predict the class as the output which has maximum occurrence in these k nearest neighbors set and then repeat steps 1-3 for each test instance.