Question 1

Normal Perceptron

Implementation

- I am random shuffling the training vectors and then dividing it into training and validation data by 80-20 ratio.
- After each epoch I am checking the accuracies, precision and recall with validation data.
- When perceptron stops updating, I am terminating.

For some runs of code, here are the accuracy, precision and recall after each epoch.

Run 1

```
Epoch : 0     Precision : 0.997004   Recall : 1.000000    Accuracy : 0.998421
Epoch : 1     Precision : 0.998498   Recall : 0.999249    Accuracy : 0.998816
Epoch : 2     Precision : 0.997751   Recall : 1.000000    Accuracy : 0.998816
Epoch : 3     Precision : 0.998497   Recall : 0.998497    Accuracy : 0.998421
Epoch : 4     Precision : 0.998498   Recall : 0.999249    Accuracy : 0.998816
Epoch : 5     Precision : 0.998497   Recall : 0.998497    Accuracy : 0.998421
Epoch : 6     Precision : 0.999248   Recall : 0.998497    Accuracy : 0.998816
Epoch : 7     Precision : 0.999248   Recall : 0.997746    Accuracy : 0.998421
Epoch : 8     Precision : 0.999248   Recall : 0.997746    Accuracy : 0.998421
Epoch : 9     Precision : 0.999248   Recall : 0.997746    Accuracy : 0.998421
Epoch : 10    Precision : 0.997751   Recall : 1.000000    Accuracy : 0.998816
Epoch : 11    Precision : 0.998498   Recall : 0.999249    Accuracy : 0.998816
Epoch : 12    Precision : 1.000000   Recall : 0.997746    Accuracy : 0.998816
Epoch : 13    Precision : 1.000000   Recall : 0.996995    Accuracy : 0.998421
Epoch : 14    Precision : 0.998497   Recall : 0.998497    Accuracy : 0.998421
Epoch : 15    Precision : 1.000000   Recall : 0.996995    Accuracy : 0.998421
Epoch : 16    Precision : 0.998498   Recall : 0.999249    Accuracy : 0.998816
Epoch : 17    Precision : 0.999248   Recall : 0.997746    Accuracy : 0.998421
Epoch : 18    Precision : 0.999248   Recall : 0.997746    Accuracy : 0.998421
```

Run 2

```
Epoch : 0     Precision : 0.997099   Recall : 1.000000    Accuracy : 0.998421
Epoch : 1     Precision : 0.997099   Recall : 1.000000    Accuracy : 0.998421
Epoch : 2     Precision : 0.997097   Recall : 0.999273    Accuracy : 0.998026
Epoch : 3     Precision : 0.994215   Recall : 1.000000    Accuracy : 0.996842
Epoch : 4     Precision : 0.997091   Recall : 0.997091    Accuracy : 0.996842
Epoch : 5     Precision : 0.994935   Recall : 1.000000    Accuracy : 0.997236
Epoch : 6     Precision : 0.997099   Recall : 1.000000    Accuracy : 0.998421
Epoch : 7     Precision : 0.997095   Recall : 0.998545    Accuracy : 0.997631
Epoch : 8     Precision : 0.997091   Recall : 0.997091    Accuracy : 0.996842
Epoch : 9     Precision : 0.997099   Recall : 1.000000    Accuracy : 0.998421
Epoch : 10    Precision : 0.998542   Recall : 0.996364    Accuracy : 0.997236
Epoch : 11    Precision : 0.998542   Recall : 0.996364    Accuracy : 0.997236
```

Run 3

```
Epoch : 0       Precision : 0.999254  Recall : 0.999254      Accuracy : 0.999210
Epoch : 1       Precision : 1.000000  Recall : 0.991797      Accuracy : 0.995657
Epoch : 2       Precision : 0.998511  Recall : 1.000000      Accuracy : 0.999210
Epoch : 3       Precision : 1.000000  Recall : 0.997017      Accuracy : 0.998421
Epoch : 4       Precision : 0.998510  Recall : 0.999254      Accuracy : 0.998816
Epoch : 5       Precision : 0.999251  Recall : 0.995526      Accuracy : 0.997236
Epoch : 6       Precision : 1.000000  Recall : 0.974646      Accuracy : 0.986577
Epoch : 7       Precision : 0.998511  Recall : 1.000000      Accuracy : 0.999210
Epoch : 8       Precision : 1.000000  Recall : 0.993289      Accuracy : 0.996447
Epoch : 9       Precision : 0.998511  Recall : 1.000000      Accuracy : 0.999210
Epoch : 10      Precision : 0.998511  Recall : 1.000000      Accuracy : 0.999210
```

Observation :

Since data is linearly separable, it is converging for the training data.
Also on validation data, it is giving fair results.

Perceptron with margin

Implementation

- I am random shuffling the training vectors and then dividing it into training and validation data by 80-20 ratio.
- After each epoch I am checking the accuracies, precision and recall with validation data.
- When perceptron stops updating, I am terminating.

For some runs of code, here are the accuracy, precision and recall after each epoch.

Run 1 (margin = 1000)

```
Epoch : 0       Precision : 0.997006  Recall : 0.998501      Accuracy : 0.997631
Epoch : 1       Precision : 0.999250  Recall : 0.998501      Accuracy : 0.998816
Epoch : 2       Precision : 0.999249  Recall : 0.997751      Accuracy : 0.998421
Epoch : 3       Precision : 0.999250  Recall : 0.998501      Accuracy : 0.998816
Epoch : 4       Precision : 0.998501  Recall : 0.998501      Accuracy : 0.998421
Epoch : 5       Precision : 1.000000  Recall : 0.998501      Accuracy : 0.999210
Epoch : 6       Precision : 0.999250  Recall : 0.998501      Accuracy : 0.998816
Epoch : 7       Precision : 1.000000  Recall : 0.998501      Accuracy : 0.999210
Epoch : 8       Precision : 1.000000  Recall : 0.997751      Accuracy : 0.998816
Epoch : 9       Precision : 1.000000  Recall : 0.998501      Accuracy : 0.999210
Epoch : 10      Precision : 1.000000  Recall : 0.998501      Accuracy : 0.999210
Epoch : 11      Precision : 0.999250  Recall : 0.998501      Accuracy : 0.998816
Epoch : 12      Precision : 0.999250  Recall : 0.998501      Accuracy : 0.998816
Epoch : 13      Precision : 1.000000  Recall : 0.997001      Accuracy : 0.998421
Epoch : 14      Precision : 0.999250  Recall : 0.998501      Accuracy : 0.998816
Epoch : 15    Precision : 0.999250  Recall : 0.998501    Accuracy : 0.998816
```

Run 2 (margin = 1000)

```
Epoch : 0     Precision : 0.997802  Recall : 0.998534    Accuracy : 0.998026
Epoch : 1     Precision : 1.000000  Recall : 0.996334    Accuracy : 0.998026
Epoch : 2     Precision : 1.000000  Recall : 0.995601    Accuracy : 0.997631
Epoch : 3     Precision : 0.999265  Recall : 0.997067    Accuracy : 0.998026
Epoch : 4     Precision : 1.000000  Recall : 0.996334    Accuracy : 0.998026
Epoch : 5     Precision : 1.000000  Recall : 0.996334    Accuracy : 0.998026
Epoch : 6     Precision : 1.000000  Recall : 0.997801    Accuracy : 0.998816
Epoch : 7     Precision : 1.000000  Recall : 0.996334    Accuracy : 0.998026
Epoch : 8     Precision : 0.999265  Recall : 0.997067    Accuracy : 0.998026
Epoch : 9     Precision : 0.999267  Recall : 0.999267    Accuracy : 0.999210
Epoch : 10    Precision : 1.000000  Recall : 0.996334    Accuracy : 0.998026
Epoch : 11    Precision : 0.999265  Recall : 0.997067    Accuracy : 0.998026
Epoch : 12    Precision : 0.998533  Recall : 0.997801    Accuracy : 0.998026
Epoch : 13    Precision : 0.999266  Recall : 0.997801    Accuracy : 0.998421
Epoch : 14    Precision : 1.000000  Recall : 0.997067    Accuracy : 0.998421
Epoch : 15    Precision : 0.999266  Recall : 0.998534    Accuracy : 0.998816
Epoch : 16    Precision : 0.999266  Recall : 0.998534    Accuracy : 0.998816
```

Run 3 (margin = 10000)

```
Epoch : 0     Precision : 1.000000  Recall : 0.994753    Accuracy : 0.997236
Epoch : 1     Precision : 1.000000  Recall : 0.997001    Accuracy : 0.998421
Epoch : 2     Precision : 1.000000  Recall : 0.997001    Accuracy : 0.998421
Epoch : 3     Precision : 1.000000  Recall : 0.994003    Accuracy : 0.996842
Epoch : 4     Precision : 1.000000  Recall : 0.997001    Accuracy : 0.998421
Epoch : 5     Precision : 0.999250  Recall : 0.998501    Accuracy : 0.998816
Epoch : 6     Precision : 0.999250  Recall : 0.998501    Accuracy : 0.998816
Epoch : 7     Precision : 1.000000  Recall : 0.997751    Accuracy : 0.998816
Epoch : 8     Precision : 0.999250  Recall : 0.998501    Accuracy : 0.998816
Epoch : 9     Precision : 0.999250  Recall : 0.998501    Accuracy : 0.998816
Epoch : 10    Precision : 1.000000  Recall : 0.998501    Accuracy : 0.999210
Epoch : 11    Precision : 0.999250  Recall : 0.998501    Accuracy : 0.998816
Epoch : 12    Precision : 1.000000  Recall : 0.998501    Accuracy : 0.999210
Epoch : 13    Precision : 1.000000  Recall : 0.997751    Accuracy : 0.998816
Epoch : 14    Precision : 1.000000  Recall : 0.997751    Accuracy : 0.998816
Epoch : 15    Precision : 1.000000  Recall : 0.997751    Accuracy : 0.998816
```

Run 4 (margin = 100000)

```
Epoch : 0     Precision : 1.000000  Recall : 0.997082    Accuracy : 0.998421
Epoch : 1     Precision : 0.999271  Recall : 0.999271    Accuracy : 0.999210
Epoch : 2     Precision : 1.000000  Recall : 0.989788    Accuracy : 0.994473
Epoch : 3     Precision : 1.000000  Recall : 0.998541    Accuracy : 0.999210
Epoch : 4     Precision : 1.000000  Recall : 0.985412    Accuracy : 0.992104
Epoch : 5     Precision : 1.000000  Recall : 0.997812    Accuracy : 0.998816
Epoch : 6       Precision : 1.000000   Recall : 0.991977      Accuracy : 0.995657
Epoch : 7       Precision : 1.000000   Recall : 0.993435      Accuracy : 0.996447
Epoch : 8       Precision : 1.000000   Recall : 0.997812      Accuracy : 0.998816
Epoch : 9       Precision : 1.000000   Recall : 0.997812      Accuracy : 0.998816
Epoch : 10      Precision : 1.000000   Recall : 0.997812      Accuracy : 0.998816
```

Observation :

It is giving better results than normal perceptron because of margin as we can see from above runs of code. Margin is handling the validation data very well to give better accuracy than normal perceptron.

Batch Percptron without margin

Implementation

- I am random shuffling the training vectors and then dividing it into training and validation data by 80-20 ratio.
- After each epoch I am checking the accuracies, precision and recall with validation data.
- It is running for max 200 epochs. It is converging very slowly because of batch update.

Run 1 (last column is the number of updates in each epoch)

```
Epoch : 0       Precision : 0.999226   Recall : 0.989272      Accuracy : 0.994078  10132
Epoch : 10      Precision : 0.998463   Recall : 0.995402      Accuracy : 0.996842  63
Epoch : 20      Precision : 0.998465   Recall : 0.996935      Accuracy : 0.997631  47
Epoch : 30      Precision : 0.998466   Recall : 0.997701      Accuracy : 0.998026  40
Epoch : 40      Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  37
Epoch : 50      Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  36
Epoch : 60      Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  33
Epoch : 70      Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  32
Epoch : 80      Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  31
Epoch : 90      Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  27
Epoch : 100     Precision : 0.998469   Recall : 0.999234      Accuracy : 0.998816  26
Epoch : 110     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  26
Epoch : 120     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  25
Epoch : 130     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  23
Epoch : 140     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  21
Epoch : 150     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  19
Epoch : 160     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  19
Epoch : 170     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  18
Epoch : 180     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  18
Epoch : 190     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  17
Epoch : 199     Precision : 0.999234   Recall : 0.999234      Accuracy : 0.999210  17
```

Observation :
Due to batch update, the convergance is slower than normal perceptron. But due to generalized update, accuracy over validation data is as better as previous.

Batch perceptron with margin

Run 1

```
Epoch : 0      Precision : 0.999258   Recall : 0.981063      Accuracy : 0.989341  10132
Epoch : 1      Precision : 0.999260   Recall : 0.983248      Accuracy : 0.990525  111
Epoch : 2      Precision : 0.999261   Recall : 0.984705      Accuracy : 0.991315  99
Epoch : 3      Precision : 0.999261   Recall : 0.985433      Accuracy : 0.991709  89
Epoch : 4      Precision : 0.999263   Recall : 0.986890      Accuracy : 0.992499  84
Epoch : 20     Precision : 0.997799   Recall : 0.990532      Accuracy : 0.993683  42
Epoch : 40     Precision : 0.997805   Recall : 0.993445      Accuracy : 0.995263  31
Epoch : 60     Precision : 0.997807   Recall : 0.994173      Accuracy : 0.995657  28
Epoch : 80     Precision : 0.997809   Recall : 0.994902      Accuracy : 0.996052  25
Epoch : 100    Precision : 0.999269   Recall : 0.995630      Accuracy : 0.997236  23
Epoch : 120    Precision : 0.999269   Recall : 0.995630      Accuracy : 0.997236  20
Epoch : 140    Precision : 0.999270   Recall : 0.996358      Accuracy : 0.997631  17
Epoch : 160    Precision : 0.999270   Recall : 0.996358      Accuracy : 0.997631  16
Epoch : 180    Precision : 0.999270   Recall : 0.996358      Accuracy : 0.997631  15
Epoch : 199    Precision : 0.999270   Recall : 0.996358      Accuracy : 0.997631  15
```

Run 2

```
Epoch : 199    Precision : 0.997734   Recall : 0.998488      Accuracy : 0.998026  18
```

Observation :
Margin is giving slightly better than previous

Question 2

Percpetron with relaxation

Implementation

- I am random shuffling the training vectors and then dividing it into training and validation data by 80-20 ratio.
- After each epoch I am checking the accuracies, precision and recall with validation data.
- It is running for max 200 epochs. I have tested for multiple learning rate also.

Note : Since the data is small, random shuffling is giving very different results each time.

Results of different runs

```
Epoch : 199    Precision : 0.933333   Recall : 0.965517      Accuracy : 0.970588
Epoch : 199    Precision : 1.000000   Recall : 0.971429      Accuracy : 0.990196
Epoch : 199    Precision : 1.000000   Recall : 0.970588      Accuracy : 0.990196
```

Observation

Relaxation is giving very fair approximation on the data which is not linearly separable.

Modified Perceptron

Implementation

- I am decreasing learning rate to x% of the previous one after each epoch.
- I also tested where the learning rate decreases on the basis of the number of wrongly classified vectors.

Results

learning rate is decreased 5% after each epoch

```
Epoch : 199    Precision : 1.000000   Recall : 0.971429       Accuracy : 0.990196
```

learning rate is decreased 7% after each epoch

```
Epoch : 199    Precision : 0.947368   Recall : 0.972973       Accuracy : 0.970588
```

learning rate is decreased 10% after each epoch

```
Epoch : 199    Precision : 0.951220   Recall : 0.975000       Accuracy : 0.970588
```

Observation

After testing on multiple decrease rate, I found 5% to work best for several runs.
If the decrease in learning is large then updates become small too soon.
If the decrease in learning is small then it’s not too much different from normal perceptron.

Question 3

Implementation

- On each run, I am randomly shuffling the vectors
- I am splitting training data into 80-20 split to validate for each run.

- Each node in tree is handling it’s own data. It is checking whether to split or not on some conditions. I check for the following conditions

1. If depth is too much then don’t split
2. If entropy is zero that means all vectors have same label. Thus don’t split.
3. If there are too less vectors remaining, say 10, then don’t split. Go with consensus.
4. If entropy is less than some threshold

- If a node has to split then it checks for each attribute and each possible split to find best split. I am checking for split at each unique value of each attribute. Implementation is compact so it is taking only 1 second for my implementation !!

Results for some runs and best terminating conditions

Terminating conditions :
- node has less than 10 vectors
- entropy is 0

```
Precision : 0.969314   Recall : 0.987132       Accuracy : 0.989333
Precision : 0.965517   Recall : 0.974855       Accuracy : 0.986222
```

Terminating conditions :
- depth is greater than 10
- entropy is 0

```
Precision : 0.969639   Recall : 0.975191       Accuracy : 0.987111
Precision : 0.986692   Recall : 0.966480       Accuracy : 0.988889
```

Terminating condition :
- 97% vectors have the same labels

```
Precision : 0.976699   Recall : 0.952652       Accuracy : 0.983556
Precision : 0.989059   Recall : 0.967880       Accuracy : 0.991111
```

Question 4

Implementation

- I am creating bag of words. Some trivial words are removed manually to make a good handcrafted feature. e.g. a, an, the
- For KNN, I have tried different distances like euclidean, cityblock, cosine.
- Different values of K is tested.
- For tie in last k values, I am taking the one which has the nearest to the query vector(1NN).

- I tested on small test set of 10 files. I also passed part of training data for validation.

Here are results, for different cases.
For each case, I have printed the confusion matrix.

K = 5
euclidean
(for small test)
Accuracy : 0.700000 (7 / 10)

K = 5
cityblock
(for small test)
Accuracy : 0.800000 (8 / 10)

```
K = 5
cosine
(for small test)
Accuracy : 0.900000 (9 / 10)

K = 7
cosine
confusion matrix :
[ 128.    0.    0.    0.    0.    3.    1.    0.    0.    0.]
[   0.  142.    0.    0.    0.   14.    4.    0.    0.    3.]
[  0.   0.  45.   0.   0.   1.   0.   0.   0.   0.]
[  0.   0.   1.  58.   0.   0.   4.   1.   2.   0.]
[   0.    0.    0.    0.  165.    0.    4.    1.    1.    0.]
[   0.    0.    0.    0.    0.  110.    0.    0.    0.    0.]
[   0.    0.    0.    0.    0.    0.  109.    0.    0.    0.]
[  0.   0.   0.   0.   0.   0.   3.  48.   0.   0.]
[  0.   0.   0.   0.   0.   0.   3.   0.  27.   0.]
[  0.   0.   0.   0.   0.   0.  17.   0.   0.  76.]
Accuracy : 0.935118 (908 / 971)


K = 7
euclidean
confusion matrix :
[ 126.    1.    0.    0.    0.    4.    1.    0.    0.    0.]
[   3.  150.    1.    0.    0.    6.    2.    0.    0.    1.]
[  2.   0.  31.   0.   0.  13.   0.   0.   0.   0.]
[  3.   0.   1.  42.   0.   2.  16.   1.   1.   0.]
[   0.    0.    0.    0.  162.    0.    5.    2.    1.    1.]
[   3.    0.    0.    0.    0.  107.    0.    0.    0.    0.]
[   0.    0.    0.    0.    0.    1.  107.    0.    0.    1.]
[  0.   0.   0.   0.   0.   0.   8.  40.   2.   1.]
[  0.   0.   0.   0.   0.   0.  10.   0.  20.   0.]
[  1.   0.   0.   0.   0.   4.  18.   0.   1.  69.]
Accuracy : 0.879506 (854 / 971)

K= 7
cityblock
confusion matrix :
[ 131.    1.    0.    0.    0.    0.    0.    0.    0.    0.]
[   4.  159.    0.    0.    0.    0.    0.    0.    0.    0.]
[  0.   0.  43.   0.   0.   3.   0.   0.   0.   0.]
[  4.   0.   0.  43.  13.   1.   0.   1.   4.   0.]
[   2.    0.    0.    0.  165.    0.    0.    4.    0.    0.]
[   2.    0.    0.    0.    0.  107.    1.    0.    0.    0.]
[   3.    0.    0.    0.    2.    1.  103.    0.    0.    0.]
[  1.   0.   0.   0.   1.   0.   0.  49.   0.   0.]
[  0.   0.   0.   0.   0.   0.   1.   0.  29.   0.]
[  6.   1.   0.   0.   0.   2.  17.   0.   0.  67.]
Accuracy : 0.922760 (896 / 971)
```

Observation :

- Removing stop words improves the accuracy greatly.

- It is observable from above results that cosine distance is more accurate because it doesn’t take account of vector length. It takes account of the vector direction. By that it compares the relative(probabilitistic) occurance of each word in the document. In case of cityblock or euclidean, the number of occurence matters if we don’t do normalization.

- Taking 1NN in the situation of tie, increases the accuracy which is intuitive.

- K values can only be found by trail and error and data specific information.