Assignment 2

Jigar Chandra (chandra.67)
Gaurav Shah (shah.889)

Goal

To classify the dataset based on feature vectors generated in Assignment 1 using Naïve Bayes and KNN.

Classification Techniques

1. K-Nearest Neighbor:

This is a simple classification algorithm that defers the training step calculation to the testing of a test-instance. The feature space can be viewed as a multidimensional space with dimensionality equal to the number of attributes, in this case, the number of words.

Our implementation takes the value of K (number of neighbors) and T (the percentage of test-set out of the total available instances) as input. Rest (100 – T) percent of the records are considered as the training set.

In the first step, the algorithm finds cosine distances between a test instance and all the training instances.

$$cosine\ distance\ = 1 - cosine\ similarity$$

$$cosine\ similarity = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{(\sum_{i=1}^{n}(A_i)^2)} \times \sqrt{(\sum_{i=1}^{n}(B_i)^2)}}$$

Once the cosine distance is computed, we find the K instances from training set that are nearest to the test instance.

There are several approaches here onwards to predict the topics for test instance based on the K neighbors:

1. One of the ways we tried is to treat the topics individually and not compare them. Each topic becomes a class attribute and takes the values 1 or 0. For each topic T_i, we check the values taken by it in the K neighbors and choose the majority value. If more neighbors report having that topic, the test instance should also have the same topic.
2. Another way we tried is to collect all the topics present in all neighbors of the test instance to be classified. Sort the topic labels in the decreasing order of their frequency of occurrence in the neighbors. Pick first N topics where N is the number of topics this test instance actually has.

We noticed that the second method gives slightly more accuracy. This could be because in the first method some of the topics that should actually belong to the test instance were ignored because majority neighbors did not have them. Second method predicts more classes per test instance as we give the number of classes we expect and thus does better.

Feature Vector Transformation:

For comparing the predicted topic and actual topic of each feature vector, we converted our third Feature Vector – FV3.txt to a csv file – FV3.csv , appending 1 to all the Feature vectors if every corresponding topic is the actual topic for that Feature Vector and 0 otherwise.

Performance metrics:

For finding out how the algorithm performed, we used 2 different versions of accuracy:

1. Accuracy by at least once correct prediction per article
2. Accuracy by all correct predictions per article

For KNN, we report the best accuracy.

In addition, we also calculated confusion matrix and using it we computed the following metrics:

1. Precision
2. Recall
3. F-measure (Harmonic mean between Precision and Recall)
4. G-mean (Geometric mean between Precision and Recall)

Performance for Various Values of K:

We found out that lower values of K (for example, k=5, 7) perform better than (k=11,20,40). As the value of K increases, the performance of the algorithms gradually degrades till k=9, then increases slowly but doesn't catch up to the high performance given by k=7. The reason behind this could be that the extra neighbors contribute to the noisy topics that distort the frequency arrangement of topics. The following results are taken for 99:1 train-test split on the data matrix.

K	Accuracy	Precision	Recall
3	80.9816	0.5677	0.5737
5	82.8221	0.6135	0.5553
7	83.4356	0.6507	0.5839
9	82.2806	0.6380	0.5077
11	80.9816	0.6206	0.5265
20	82.8221	0.6653	0.5590
40	81.5951	0.6965	0.5253

Performance of different sizes of test sets:

We observed that performance slowly increased upto the split 80:20 and then started reducing as we went on increasing the size of test set. After this threshold, performance can deteriorate because of the size of training set getting smaller. Following results are taken for different split sizes and K= 5

Percentage of test set	Accuracy	Precision	Recall	F-measure	G-mean	Time (seconds)
1	82.8221	0.6135	0.5553	0.5831	0.5837	441
2	74.0413	0.5113	0.4940	0.4450	0.4488	1015
3	78.0761	0.5506	0.5030	0.4654	0.4711	1285
4	78.4560	0.4955	0.4704	0.4241	0.4286	1743
5	79.4286	0.4882	0.4826	0.4290	0.4322	2125
10	81.4734	0.5031	0.4814	0.4339	0.4390	4015

20	79.8964	0.5020	0.4523	0.4140	0.4206	7180
40	79.2990	0.4529	0.3164	0.3747	0.3812	10466

Scalability:

KNN algorithm does not require an explicit training step before starting the test phase. Hence the distinction between offline cost and online cost is vague.

Time to classify a new instance increases as the training set grows. This is because the algorithm has to find distance of the test instance with every training instance. The split percentage decides the time to classify each instance.

NAÏVE BAYES CLASSIFIER

The Naïve Bayesian classifier is based on applying the Bayes Theorem with independence assumptions. In this case we assume that the appearance of a word given topic is independent for the articles.

We apply the Bayes Theorem as:

$$P(T|W1 \dots Wn) = \frac{P(W1 \dots Wn|T)P(T)}{P(W1 \dots Wn)} = \frac{P(W1|T) \dots P(Wn|T)P(T)}{P(W1 \dots Wn)}$$

To accomplish this task we have used the **Orange Library**.

Challenges observed and actions taken

1) Multi-Labelling : As a single document could have multiple topics so we took 2 approaches
   a) Calculate the count of each topic across all documents so that while giving it the label we would use that topic from the list of topics for that document that has the highest count.
   b) Use all the topics as is.

   In approach a) the accuracy and precision were better as compared to scenario b) which is only obvious as we are reducing the number of False positives that would have otherwise been generated because of multiple labels.
2) Unique Words and Topics : There were certain number of words in the set of unique words across all documents which were exactly same as the name of the topic in set of topics across all documents. These words proved to be misleading because they were giving me very high values of precision and accuracy which were actually wrong as this would classify the document as that word. So I had to eliminate all the words which were same as the topics and eliminate corresponding entries from the feature vector as well

Feature Vector and Data organization:

Orange prefers to open data files in its native, tab-delimited format. First line contains variable(attribute) names, followed by their types(discrete, continuous, string) in the second line and optional flags(meta,class) in the third.

```
DocID   Topic   Places          japan   consider….. other words
d       d       d               d       d
meta    class
0       topic1  place1 place2.. 1       2       3….. other counts     ← record 1
                                                                :
                                                                :
                                                              ← record n
```

First line is the name of all the attributes for the data. Here the attributes I have used are DocID, Topic, Places, and list of all those unique words across all documents that are not same as any of the topics across all datasets. By specifying each word as an input we are capturing the independence relationship with the topic that we desire for Bayes rule to work properly.

Second row is the row of optional flag. I have mentioned each attribute as discrete attribute because values are fixed in a domain.
Third row is the row of optional flags. Doc Id is the meta attribute used to identify records and topic is marked class as that is what we are trying to predict.
Line 4 onwards will be record pertaining to each document.

Feature vector : The feature vector is the term frequency vector which gives count of each unique word for every document.

ALGORITHM

1) Generate the .tab file from the term frequency feature vector.
2) Load the data from the .tab file and assign learner as NaiveBayes learner from Orange Library
3) Build the classifier from the loaded data and also mention the training and testing split(80-20,60-40 etc).Also mention the type of sampling used as stratified sampling. (Also this is timed as later on this will be used to calculate online cost).
4) Generate the confusion matrix based on the result obtained after classification
5) Calculate **Accuracy**, **Precision**, **Recall** and **F1 score** from the confusion matrix.
6) Also separately provide the call **learners(data)** and time it. This will be the offline cost.
7) The difference between the time required for step 3) and step 6) is the online cost

Different Experiments and Analysis

1) Initially I started out by trying to provide every feature vector as a bag of words and labelling it as 'basket' in the second row of .tab file. But after running the script for a sufficient amount of times. I was getting a very low accuracy and precision. Actually what was happening was that it was considering the entire feature vector as a single attribute of type basket and was attaching the topic to this combination of words. So chances of finding this combination was very low and hence the drop in precision and accuracy.

Classifier	CA	Precision	Recall	F1-score
Bayes	0.770888302551	0.323232323232	0.0658436213992	0.109401709402

2) After the first experiment I used each word, from the set of unique words across all documents, as a unique attribute and the corresponding term frequency vector for each document. I was getting a high amount of accuracy and precision. However what actually was happening was that it was using those words which were same as topics eg : cocoa is present in set of unique words and unique topics, for prediction. So such words would be used for prediction and that was causing the accuracy and precision to go up.

Classifier	CA	Precision	Recall	F1-score
Bayes	0.975373790677	0.926380368098	0.957716701903	0.941787941788

3) So Now I changed my feature vector such that count of all the words that matched the topic was removed and now I did the classification but because I had multiple topics for the same document so the number of false positives was pretty high (because of multiple class labels).

Classifier	CA	Precision	Recall	F1-score
Bayes	0.898834396305	0.710320901995	0.866666666667	0.7807435653

4) Finally I switched to the approach wherein I had removed all those words, which matched topics exactly, from the attributes and if the class had multiple topics then I would attach just that topic which occurs the most out of all the topics for that document. The results are documented below.

Results:

Data Split	80-20	60-40
Accuracy	0.915127528584	0.91159005938
Precision	0.769797421731	0.756294964029
Recall	0.860082304527	0.865226337449
F1-Score	0.812439261419	0.807101727447
Offline Time	1.15299987793 sec	0.893000125885 sec
Online Time	4.38700008392 sec	8.54299974442 sec

References:

1] Orange Library Reference Material:
http://docs.orange.biolab.si/reference/rst/index.html