CSE 5243: Introduction to Data Mining

Assignment 2

Vaibhav Devekar (devekar.1)
Akshay Nikam (nikam.5)

Goal of the Assignment

This assignment aims at demonstrating various techniques of classifying the dataset based on feature vectors generated in Assignment 1.

Performance metrics:

For finding out how the algorithm performed, we used 3 different versions of accuracy:

1. Accuracy by at least once correct prediction per article
2. Accuracy by all correct predictions per article
3. Accuracy by 'correct predictions/total topics' 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)

Classification Techniques

We implemented the following classification algorithms:

1. Naïve Bayesian Classification:

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:

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

Where P(W|T) specifies the probability of appearance of W and T together in an article, P(T) is probability of appearance of topic in an article.

We need to compute this probability for every topic for a test instance. Since this is a multi-label classification, we take a look at number of actual topics, say t. We sort the probabilities in decreasing order and chose the first t topics to be our predicted topics. Based on predicted and actual topics, we update the metrics.

To compute P(W|T) , we maintain a count of each word and topic combination appearing together in the articles. The denominator is simply aggregate of all word and topic counts where the T is the topic. For optimization, we can ignore the $P(W_1 \dots W_n)$ term since they are common for each topic prediction.

Laplace correction: Some of the P(W|T) terms could be zero leading to skewed probabilities. To deal with this we perform the Laplace correction for all P(W|T) terms i.e. add 1 to numerator and size of vocabulary to denominator

$$P(W|T) = \frac{Count(W \text{ and } T)+1}{\sum_i Count(Wi \text{ and } T)+|V|}$$ *where, V is the vocabulary in training set*

The above approach is based on considering only the words present in a test instance i.e. multiplying the P(W|T) for the words present in article. An earlier approach we tried was calculating and multiplying probabilties of all words for a test instance(Taking 1 - P for the words not present).

Performance:

We noted that as the size of test dataset increases, online cost increases while the offline cost goes do"wn since we have lesser training data. The accuracy is better with larger training set as show in "Metric vs Test size" graph

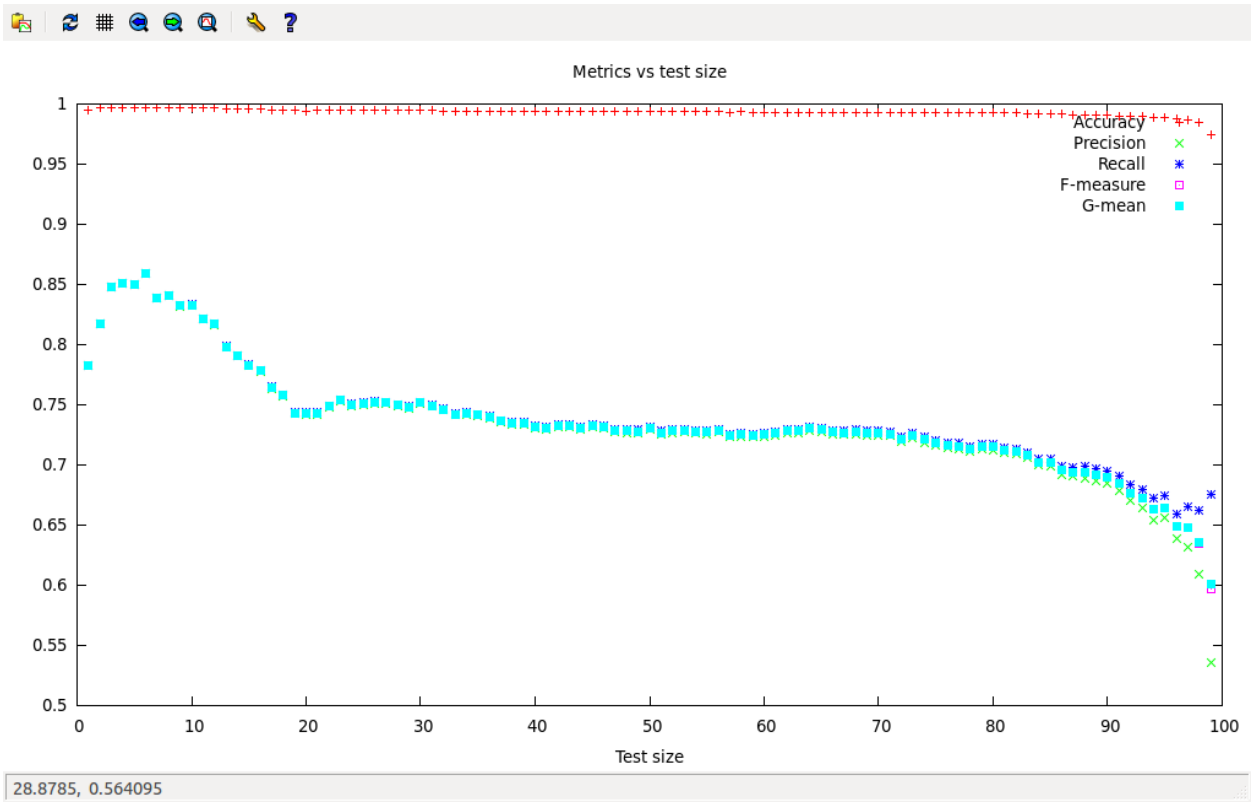
Following accuracy metrics were used and results were obtained for two different splits:

A0: Accuracy by at least one correct prediction per article
A1: Accuracy by all correct predictions per article
A2: Accuracy by 'correct predictions/total topics' per article

Splits	80/20	60/40
Offline cost(Train time)	0.70	0.53
Online cost(Test time)	9.29	16.78
Online cost per tuple	0.00411	0.00371
A0	83.41	81.13
A1	75.31	73.19
A2	79.35	77.24
Accuracy from Confusion Matrix	0.9942	0.9936
Precision	0.7417	0.7305
Recall	0.7432	0.7327
F-measure	0.7424	0.7316
G-mean	0.7424	0.7316

Scalability:

The Bayesian Classifier is fairly fast and runs in less than a minute even with test dataset as large as 80%. This is mostly since there are less number of computations involved as compared to other classifiers. Also we pre-compute all the probabilities required for classification into training phase, greatly reducing the execution time.



2. 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:

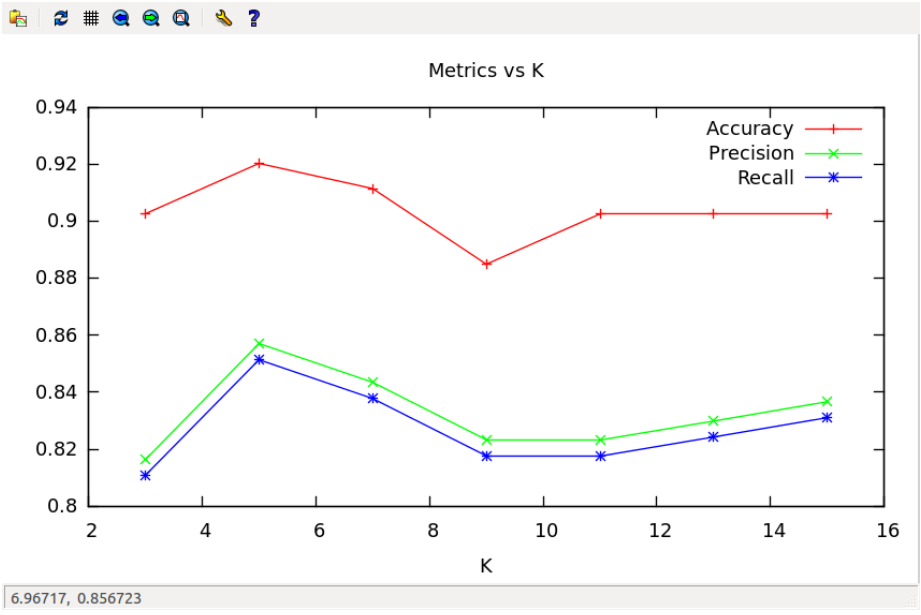
1. One of the ways we tried is to treat the topics individually and not compare them. Each topic becomes a class attribute and takes values “yes” and “no”. For each topic T_i, we check the values taken by it in the K neighbors and choose the majority value. If more neighbors report having a topic, the test instance should also have the 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. It is a slight advantage that the second method has.

Values of K:

We found out that lower values of K (for example, k=5, 7) perform better than (k=9,11,13,15). 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=5. The reason behind this could be that the extra neighbors contribute to the noisy topics that distort the frequency arrangement of topics. We are currently using k=5 as our optimal value. The following results are taken for 99:1 train-test split on the data matrix.

K	Accuracy	Precision	Recall
3	90.2655	0.8163	0.8108
5	92.0353	0.8571	0.8514
7	91.1504	0.8435	0.8378
9	88.4955	0.8231	0.8176
11	90.2655	0.8231	0.8176
13	90.2654	0.8299	0.8243
15	90.2654	0.8367	0.8311



Performance of different sizes of test sets:

We observed that performance slowly increased upto the split 15:85 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	92.0353	0.8571	0.8514	0.8542	0.8542	405
2	87.6106	0.7688	0.7410	0.7546	0.7547	805
3	89.3805	0.8076	0.7831	0.7952	0.7952	1197
4	89.3805	0.8130	0.7940	0.8034	0.8034	1575
5	89.3805	0.8247	0.8077	0.8161	0.8161	1947
10	89.9115	0.8350	0.8222	0.8285	0.8286	3618
15	89.9705	0.8302	0.8182	0.8241	0.8242	5203
20	88.4564	0.8194	0.8082	0.8138	0.8138	7314
40	87.8151	0.8161	0.8060	0.8110	0.8110	9120

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 decides the time to classify each instance. However, as the number of test instances the total time would increase, despite of the previous statement.

Individual Contributions

Akshay implemented the K-Nearest Neighbor classifier while Vaibhav implemented the Naïve Bayesian classifier. Both the classifiers were developed in Python.