Rizwan Lubis, RAL 3232, Project 4

For accuracy on training data, KNN where K = 1 performs the best out of the given algorithms. In my implementation of KNN, I keep a mapping of files to their given categories that are given in the list of training examples. I then use an Inverted Index where I create an index of the training examples. The Inverted Index is already built to keep track of each indexed example's TFIDF and HashMapVector. To test, I retrieve from the Inverted Index, and the Inverted Index retrieval ranks the training examples that are most similar to the test example, and returns the ranked examples in an ordered list. I take the top K retrievals and classify the test example as the category that occurs the most in the top K retrievals.

In CVLearningCurve, the accuracy on the training data is calculated by training the classifier and then running tests using the training examples. This algorithm should correctly classify each training example that is used as a test, because a document's should be the most similar to itself after length is normalized and each training document's correct category has been saved during training. We can see this in the training curve for KNN where K = 1, as the curve is consistently close to perfect accuracy at all parts of the curve.

As K increases for KNN, we can see that the accuracy of the training curve performs worse over all points of the curve. This happens for multiple reasons. One reason is tie breaking. Since K is greater than 1, the possibility of a tie is now introduced. Ties are broken by randomly selecting a categories from the categories that tie for majority. Introducing a random selection also introduces randomly selecting the incorrect category. Another problem may be the size of precise results. By increasing K in KNN, we are increasing the number of retrievals that we are deeming as good recalls, but not increasing the actual precision of these results. For example, if only one test example is truly relevant to a test example with the same category but our K is greater than 1, then the test example will be categorized with either the wrong category or be forced to randomly choose between the correct category and multiple incorrect categories. This randomness leads to a higher variance when compared to Naive Bayes and Rocchio.

The Rocchio algorithms and Naive Bayes were generally more accurate on Training data than KNN for K > 1. I think that this is because both Naive Bayes and Rocchio develop a generalization while KNN does not.

At the beginning of the training curve, all of the algorithms start with high precision. KNN3 and KNN5 quickly and sharply fall in accuracy. Their accuracy increases until they stabilize at around .7 accuracy. KNN1's training curve stays close to 1 with a consistent accuracy. Both Rocchio curves have high accuracy, but their accuracies decrease as the number of examples increases. Naive Bayes starts with a lower accuracy then Rocchio but its generally accuracy increases with more examples.

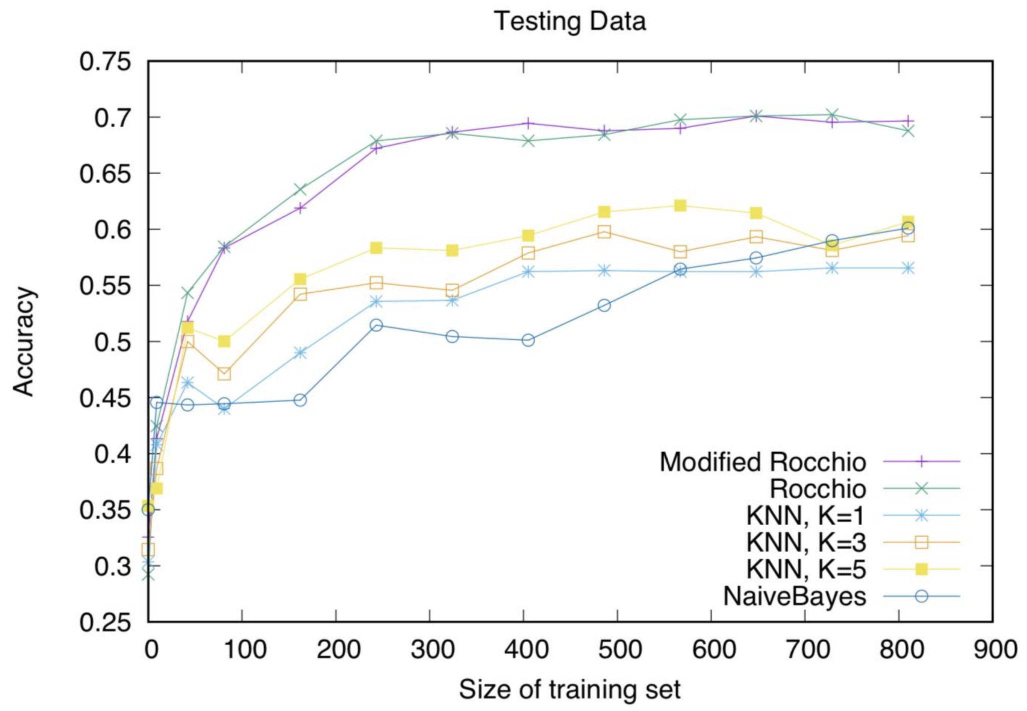
As for the testing curve, all of the algorithms begin with around .35 - .45 accuracy. As the size of the set increases, all of the curves generally have higher accuracy. The Rocchio algorithms exhibit the best accuracy for testing data, and I think that it is because the Rocchio algorithms created prototypes with generalized features that were seen in each category, as opposed to KNN that considered the training examples as a whole and not as a summation of their features. I also think that the type of documents in the corpus work in favor of the Rocchio algorithms.

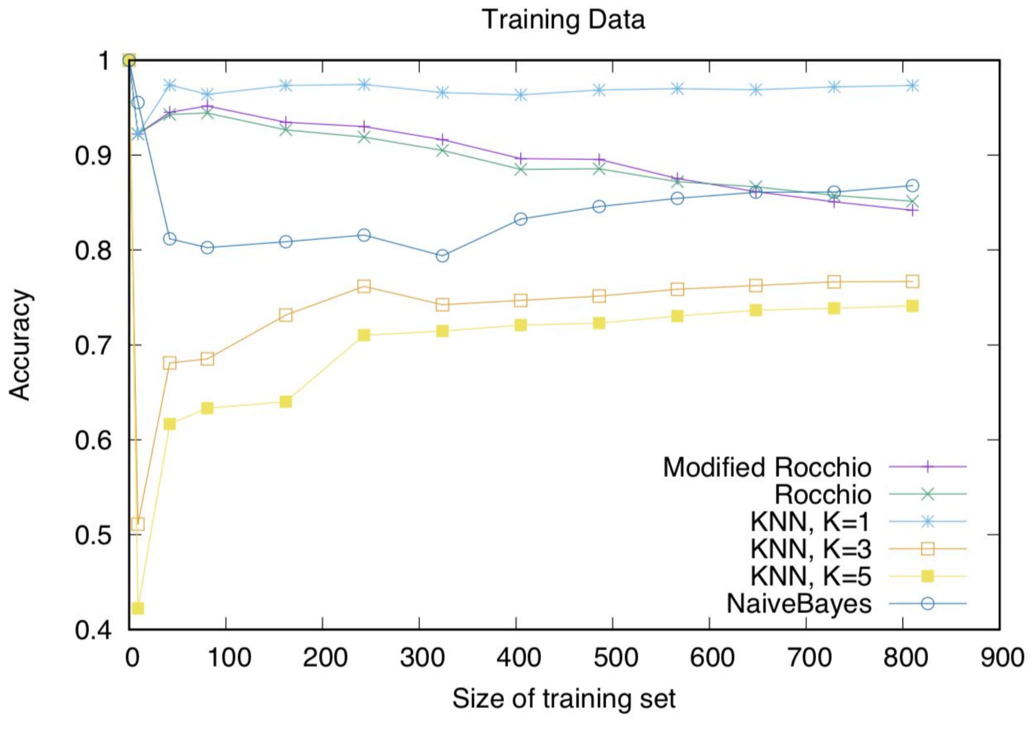
Biology, physics, and chemistry were the three categories in the corpus. The three subjects are quite different and involve differing technical vocabularies. Since we utilize IDF, uncommon words have higher weights. These weights are exacerbated when we can now create prototypes with summed amounts of these uncommon words. Now, when a testing example has an uncommon word that is used as a technical term in chemistry, it will be very similar to the chemistry category prototype when calculating cosine similarity because the weight of that term is heightened in the prototype.

Naive Bayes performed the worst in the testing learning curve, but it improved as the size of the set increased. When the set size was > 750, Naive Bayes was on par with KNN3 and KNN5. The rate of improvement for Naive Bayes and Rocchio were greater than KNN as the set size increased. This is because KNN is not a generative algorithm, and the documents outside of the K most similar do not impact classification of test data at all. The performance of KNN1 suggests that the algorithm is prone to overfitting, as KNN1 had stellar training data accuracy but poorer test data accuracy. KNN3 and KNN5 seemed to exhibit underfitting, as the learning curves for the training data were relatively lower while the testing accuracy was also relatively low when compared to Rocchio.

The runtime for training KNN was the best out of the three algorithms. Rocchio had the worst training run time. This is because both KNN and Rocchio implementation created an Inverted Index to calculate TFIDF of the terms in the documents. In addition to indexing, Rocchio had to go through every term in every training HashMapVector to generate the prototypes. However, Rocchio was faster than KNN when testing each example, because my implementation of Rocchio had calculated document lengths of prototypes for cosine similarity during the training phase. The runtime results for the testing an example are congruent with the time complexities discussed in class. Naive Bayes had the fastest test time and a relatively fast training time. This is because the training time complexity is O(nd) and the testing time complexity per example is O(cd), where n is the number of training examples, d is the number of unique tokens, and c is the number of categories.

Algorithm	**Training Time (seconds)**	**Testing Time per example (milliseconds)**
KNN, K=1	6.17	0.3
KNN, K=3	6.425	0.34
KNN, K=5	6.543	0.33
Rocchio	12.173	0.1
Modified Rocchio	17.566	0.13
Naive Bayes	7.317	0.08


Testing Data
Accuracy
Size of training set
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0
100
200
300
400
500
600
700
800
900
Modified Rocchio
Rocchio
KNN, K=1
KNN, K=3
KNN, K=5
NaiveBayes


Training Data
Accuracy
Size of training set
0.4
0.5
0.6
0.7
0.8
0.9
1
0
100
200
300
400
500
600
700
800
900
Modified Rocchio
Rocchio
KNN, K=1
KNN, K=3
KNN, K=5
NaiveBayes