COL 703: Assignment 2

Machine Learning

Part 1: Text Classification

Command to run is: ./bash run.sh 1 <trainfile> <testfile> <a/b/c/d/e/f/g>

a)

Naive Bayes was implemented with Laplace smoothing. The log-likelihood was used so as to avoid underflow issues. The accuracies for the model are reported below:

Accuracy on training data: 0.6974557964637171
Accuracy on test data: 0.6714051003643118

b)

Accuracy on test data (random prediction): 0.20215729694978213
We can see that this is very less as compared to the 66% as obtained using the Naive Bayes algorithm.

Accuracy on test data (major prediction): 0.6608329166369027
One might think that this method is comparable to the one obtained using the Naive Bayes as both the accuracies are ~66%. But it is only because the test data had 66% reviews in the class 5 which happened to be the most occurring class in the training dataset. Had the training dataset has any other class as most frequent, the accuracy would drop by at least 30%. Hence, both of these methods are not preferable.

c)

The confusion matrix is as shown below. It can be observed that the reviews with class 5 are predicted most correctly. Also, a lot of reviews that belonged to other classes also got assigned the class 5, which shows that the model is a little biased towards the class 5.

Confusion Matrix		**Predicted class**				
		1	**2**	**3**	**4**	**5**
Actual class	**1**	10	1	7	34	176
	2	2	0	10	120	194
	3	4	0	8	451	623
	4	15	0	2	734	2357
	5	70	1	1	584	8586

d)

Stopword removal, tokenization and stemming is done using the nltk library in python.

Accuracy on training data: 0.6881750540043203
Accuracy on test data: 0.6702621615829703

As can be seen from the data above, it is evident that this transformation of data is not particularly helpful in these datasets. It does not mean that these procedures will always give comparatively bad results.

e)

Bigrams:

As suggested in the specification, each feature is now a combination of two consecutive words. The accuracies observed are reported below:

Accuracy on test data with stemming: 0.6621901564397457

It is degrading performance in the test dataset but the change is not very drastic.

Inverse Document Frequency

On top of the bigram model, we make a change in calculating the term frequency of a word. Extending the concept of idf in the tf-idf vectors, we use inverse *class* frequency to give less weight to the terms which are present in more classes.
Hence tf is changed to tf*5/(number of classes the term appears in).

Accuracy on test data with stemming(idf+bigram): 0.6621901564397457

It can be seen that the model has not changed from the bigram model. This is because 75% of the terms appear in each class. Hence the weight of a term remains the same for maximum terms. Even if the weight is changed, it is not changed by much. So, we don't see observable differences in accuracy.

Removing redundant data:

Extending the concept of removing stopwords, we try to observe the accuracy if only less frequent words are considered during the training of the model. The accuracies are reported below if only the x% least frequent terms are considered. Accuracies observed after doing the removing redundant check for the unigram model are reported below:

x	Test accuracy
10	0.6532609472105151
20	0.6446889063504536
30	0.6369026359025645

40	0.6339024216015435
50	0.6300450032145153
60	0.6262590185013215
70	0.6217586970497893
80	0.6116151153653833
90	0.6104721765840417

This is performing worse than the above two models for any value of x. Thus, we conclude it is not giving any improvement.

Hence, in the above three models tried, **bigram** works the best.

f)
The confusion matrix obtained for the bigram model is:

Confusion Matrix		**Predicted class**				
		1	**2**	**3**	**4**	**5**
Actual class	**1**	0	0	1	7	220
	2	0	0	1	25	300
	3	0	0	1	8	1005
	4	0	0	0	8	3028
	5	0	0	0	48	9203

F1 scores for each class are:

Class	F1 score
1	0.0
2	0.0
3	0.00183654729109274
4	0.04778972520908005

5	0.8000173860129526

The value of macro F1 score is: 0.16992873170262507

The ideal value of an F1 score is 1 but we are getting very less value. It is because our model is not able to predict the classes such as 1 and 2. Most of the reviews are allocated to classes of either 4 or 5. Since our test dataset itself has ~66% reviews in class 5, we are getting a better idea of our model's performance in the test dataset using the test error rather than the F1 score. This F1 score is implying that the model is performing very poorly on this dataset which is not the case.

g)
To include the 'summary' text in the training model, we add the words in summary to the vocabulary. Since summary is concise and contains the exact words that are relevant to the rating, we give more weight to the words that are in summary. So, if a term is in the 'summary' column, we increase the frequency of the term by 3 rather than just 1.

Accuracy on training data: 0.7085766861348908
Accuracy on test data: 0.6750482177298378

It can be seen that the accuracy has been improved from the first model for both the training and test datasets.

Part 2: MNIST Digit Classification

Binary Classification

Command to run is: ./bash run.sh 2 <trainfile> <testfile> 0 <a/b/c>
My entry number is '2018CS50420'. We classify the data in the two labels- '0' and '1'.

a) **Model for linear kernel**
   b: 0.7876914825918344 nSV: 53
   Accuracy on test data: 99.905%
   Time taken for training: 43.22s

b) **Model for gaussian kernel**
   b: -0.6622230741817642 nSV: 810
   Accuracy on test data: 99.858%
   Time taken for training: 23.89s

c) **LIBSVM for linear kernel**
   nu = 0.000579
   obj = -1.157460, rho = -0.787314, nSV = 53
   Accuracy = 99.9054%
   Time taken for training: 7.86s

   LIBSVM for gaussian kernel

   nu = 0.025473
   obj = -53.050160, rho = 0.752569, nSV = 745
   Accuracy = 99.8109%
   Time taken for training: 9.02s

Observations:

- It is observable from the above findings that our model gives comparable results to the libsvm model.
- The time taken for training is comparatively higher, which is expected.
- The accuracy achieved is very high in each part because '0' and '1' are visibly very different.

- Since we get 99.9% accuracy with the linear kernel, trying with the gaussian kernel is decreasing the accuracy. This might be because of increasing the kernel's dimension the model tends to overfit, and hence gives less accuracy with the test data.
- Number of support vectors is much higher in the gaussian kernel as compared to the linear kernel. This is because more points are needed to make a precise non-linear classifier than a linear classifier.

Multi-Class Classification

Command to run is: ./run.sh 2 <trainfile> <testfile> 1 <a/b/c/d>

a) As suggested in the specifications, the model was trained for 45 classifiers, and the values of alpha,b,x,y were stored for each combination of classes. The time taken for training is much higher, because the training is being done sequentially for each of the 45 class combinations. The accuracy observed is lower, but acceptable. Accuracy is lower because now the model has to distinguish between all the digits, so there is more chance of wrong prediction. Also, some of the combinations can be very confusing.

Accuracy on test data: 96.76%
Time taken for training: 1787.48s

a) The following observations show that libsvm is performing considerably better than our model in the case of multi-class classification. The accuracy is more and the training time is less.

Accuracy on test data: 97.23%
Time taken for training: 137.80s

b) The confusion matrix for our model is shown below:

Confusion Matrix			**Predicted class**								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual class	**0**	1124	3	1	0	1	2	0	4	0	0
	1	0	988	9	0	0	0	9	16	0	10
	2	0	3	994	1	3	0	2	5	2	0
	3	2	9	0	957	0	3	1	1	6	3
	4	0	6	7	1	855	6	1	9	2	5

	5	4	1	0	1	3	937	1	1	0	10
	6	5	20	12	5	0	0	968	2	13	3
	7	0	1	12	2	2	3	2	939	3	10
	8	6	3	8	24	2	0	7	6	940	13
	9	0	1	0	0	0	3	1	1	0	974

It can be seen that the diagonal entries have higher values which means the digits are being classified correctly to their respective classes. Most misclassified digits by our model are:

Digit	**Wrongly predicted as(>= 10 times)**
1	7,9
5	9
6	1,2,8
7	2,9
8	3,9

We can see that the misclassified digits actually look like the predicted digits. Example- 1,7 look the same, 8 and 3 have the exact same shape for the right halves of the digits.
Such similarities are leading the model to misclassify the digits.

Confusion Matrix			**Predicted class**								
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual class	**0**	1121	3	2	1	2	2	0	3	1	0
	1	0	1000	4	2	0	1	6	15	0	4
	2	0	8	985	0	4	0	6	5	2	0
	3	0	4	0	962	0	6	0	2	8	0
	4	0	3	6	1	866	7	1	5	1	2
	5	3	0	0	4	4	939	0	2	0	6
	6	4	19	2	4	0	0	987	2	9	1
	7	0	3	10	1	5	3	3	942	3	4

	8	4	3	8	13	4	0	9	12	952	4
	9	0	1	0	0	3	4	1	2	0	969

Most misclassified digits by LIBSVM model are:

Digit	**Wrongly predicted as(>= 8 times)**
1	7
2	1
6	1,8
7	2
8	3,7

It can be observed that LIBSVM is also giving some of the same wrong predictions as our model. (1->7, 6->1,8, 7->2, 8->3).