COL774 Assignment2

Aniket Gupta
Entry No. - 2019CS10327

6 October 2021

1 Text Classification

1.1 Simple Naive Bayes

In this part, I implemented Naive Bayes, using bag-of-words approach, to build a Text classification model. I kept alpha = 1 and used log probabilities, instead of probablities, to avoid underflow.

Time taken to train the model (in sec): 10.707
Accuracy Obtained:

- On Training Data : 69.918 %
- On Test Data : 66.707 %

F1-score:

Data-type	Micro-F1 (class 1)	Micro-F1 (class 2)	Micro-F1 (class 3)	Micro-F1 (class 4)	Micro-F1 (class 5)	Macro-F1 score
Training Data	0.355	0.123	0.418	0.636	0.817	0.470
Test Data	0.060	0.000	0.014	0.291	0.810	0.235

1.2 Random Prediction and Majority Prediction

Accuracy:

- Accuracy using Random Prediction on test-set = 19.15 %
- Accuracy using Majority Prediction on test-set: 66.08 %

F1-scores

Method	Micro-F1 (class 1)	Micro-F1 (class 2)	Micro-F1 (class 3)	Micro-F1 (class 4)	Micro-F1 (class 5)	Macro-F1 score
Random Prediction	0.030	0.041	0.106	0.208	0.292	0.135
Majority Prediction	0.000	0.000	0.000	0.000	0.796	0.159

Comparison with Naive Bayes Method:

- It is clear that our Naive Bayes implementation works much better than the Random Prediction approach. The Naive Bayes implementation gives 47.55 percent more accuracy on the test set than Random Prediction approach. The F1-score is also much better in our model.
- The Naive Bayes model is only slightly better than the Majority Prediction approach. The difference between there accuracy on test set is only 0.627 percent. The main reason is the non-uniform division of samples in different classes. The F1-score is much better in Naive Bayes model than Majority prediction approach.

1.3 Confusion Matrix

Confusion Matrix obtained for test set:

Actual/Predicted	**1**	**2**	**3**	**4**	**5**
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	10	584	8587

Observations:

- Category 5 has the highest no. of diagonal entries. This means that out of the correctly predicted samples, category 5 has the highest no. of samples.
- One important reason, in this case, is that the test set has disproportionately higher percent of category 5 samples as clear from the values in last row of the confusion matrix.
- The high number of samples of Class 5, visible from the last row of confusion matrix, explains why there is high accuracy even in case of majority prediction.

1.4 Stopword Removal and Stemming

- I removed the stopwords such as 'of', 'the', 'and' etc. from the training as well as test data. This step is important because stopwords are playing very little or no role in text classification (in this case of giving reviews) and thus worsen the model.
- Different forms of the same word such as 'eat' and 'eating' are treated differently in raw data. But in case of text classification, they play equivalent role. Thus, I merged such variations using stemming the words in training as well as test data.

Analysis after stopwords removal and stemming:

- Accuracy on Training Data: 68.788 %
- Accuracy on Test-Data: 67.021 %
- Macro F1-score on Training Data: 0.4670
- Macro F1-score on Test-Data: 0.2372

On stop-words removal and stemming, the accuracy decreases slightly (by 0.68 %) and Macro-F1 score increases by 0.0022 on the test set. In general, there is an improvement observed after stemming and stop-words removal, but in this case no improvement is observed. This may be due to a particular distribution of words in the given data set.

1.5 Feature Engineering

In this part, I used two other features to train the Text Classification model:

- **Using Bigrams:**
  Many times, the features of a text are not captured by unigrams. For example, part like "not good", if divided into unigrams, can lose their meaning depending on the context. Thus I used bigrams to capture such properties:
  Result obtained using Bigrams:
  - Time taken (in sec) : 176.47 (data preprocessing) + 6.07 (model training)
  - Accuracy on Training Data: 88.386 %
  - Macro F1-score on Training Data: 0.779
  - Accuracy on Test-Data: 66.614 %
  - Macro F1-score on Test-Data: 0.184

- **Using Weighted Unigram+Bigram+Trigram Features:**
  Initially I tried using trigrams to capture text features due to relative positions of words. But, using trigrams reduced the accuracy on both training as well as test sets. Using unigrams could not capture relative-positional features in the text. Thus, I decided to use all unigrams, bigrams and trigrams as features and giving higher weight to bigrams so that it could capture the relative features but also does not reduce the accuracy as in the case of trigrams. The model is similar to bag of words model with slight variation.
  Let us assume that the feature vector X for a sample can be represented by (X_u, X_b, X_t) where $X_u, X_b \text{and} X_t$ represents the unigram, bigram and trigram features of X. The below equation show the underlying generating probability assumption of the model.

$$P(X|Y;\theta) = P(X_u|Y;\theta)P(X_b|Y;\theta)^{1/w}P(X_t|Y;\theta) \quad \text{where } w \geq 1$$

  Result obtained using Weighted Unigram+Bigram+Trigram: I ran the algorithm with different values of weight parameter (w). Over the range of values of w that I tried, w=10 gave the best result. The below result is for w=10 in the above equation.
  - Time taken (in sec) : 148.581 (data preprocessing) + 19.425 (model training)
  - Accuracy on Training Data: 98.22 %
  - Macro F1-score on Training Data: 0.969
  - Accuracy on Test-Data: 66.99 %
  - Macro F1-score on Test-Data: 0.212

1.6 F1-score

- Out of the two methods that I have mentioned in feature engineering, Weighted Unigram+ Bigram+Trigram gave the best result. The F1-scores obtained on the test data in this case are:
  Micro F1-scores = Class 1: 0.049, Class 2: 0, Class 3: 0.021, Class 4: 0.187, Class 5: 0.808
  Macro F1-score = 0.2134

- Out of all the methods (including those in part a and c) that I implemented, Naive Bayes method, after stopwords removal and stemming, gave the best result on test data. The F1-scores obtained on the test data in this case are:
  Micro F1-scores = Class 1: 0.080, Class 2: 0.006, Class 3: 0.033, Class 4: 0.256, Class 5: 0.810
  Macro F1-score on test data = 0.2372

F1-score is a better evaluation metric for this kind of dataset.
Reason: The samples in the test set are not uniformly divided in different classes. Around 66 percent of samples belong to class 5. In this case, majority prediction itself gives test-accuracy of 66.08 percent. Thus, F1-score is a better evaluation metric for this test dataset as it can capture precision and recall values for different classes separately.

1.7 Using Summary Field

For the given dataset, the summary field seems to better describe the reviews in lesser no. of words. Thus, using summary can improve the prediction accuracy. I used a weighted prediction method in this part. For a given class k, the probability of a sample with reviewText X_R and summary X_S is given by:

$$P(X_R, X_S|Y = k; \Theta) = P(X_R|Y = k; \Theta)P(X_S|Y = k; \Theta)^{1/w}$$

In the above equation, we take w>=1. Since probability always lies between 0 and 1, using the above equation with w>1 gives higher weight to summary text. The result with different values of w is shown below:

w	Test Accuracy	Macro-F1 score
1	67.45	0.233
2	67.37	0.238
3	67.3	0.239
4	67.24	0.240
10	67.18	0.241

If we compare these results with one implemented in part 3 after stopwords removal and stemming, we can see that using summary field gives slightly better result.

2 MNIST Digit Classification

2.1 Binary Classification

My entry no. ends with 7. Thus, we will consider binary classification on classes 7 and 8.

2.1.1 Linear Kernel using CVXOPT package

Mathematical Formulation:
In SVM problem, we wanted to minimise

$$W(\alpha) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}x^{(i)T}x^{(j)} - \sum_{i=1}^{m}\alpha_i$$

subject to the constraints

$$0 \leq \alpha_i \leq C \ \forall i \in \{1, 2, ..., m\} \ \text{ and } \ \sum_{i=1}^{m}\alpha_i y^(i) = 0$$

Now, the standard form used by CVXOPT package is:

$$\min_{\alpha} \ \frac{1}{2}\alpha^T P\alpha + q^T\alpha$$

$$\text{subject to } \ G\alpha \leq h \ \text{ and } \ A\alpha = b.$$

Thus, by reformulating our equation for $W(\alpha)$ in the standard form required by CVXOPT, we get:

$$P = (ZZ^T)_{ij} \ \text{ where } Z \text{ is a matrix with } \ Z_{ij} = x_j{}^{(i)}y^{(i)}$$

$$q = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{bmatrix} \qquad h = \begin{bmatrix} C \\ C \\ \vdots \\ C \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$A = y^T \qquad\qquad b = 0$$

Thus the matrices (along with their dimensions) obtained are:

$$P_{(m\text{x}m)},\ q_{(m\text{x}1)},\ G_{(2*m\text{x}m)},\ h_{(2*m\text{x}1)},\ A_{(1\text{x}m)},\ b_{(1\text{x}1)}$$

The CVXOPT package returned the optimal solution to the above minimisation problem.

Results (with C=1):
No. of Support Vectors obtained (with threshold = 1e-6) = 159
Time taken to train the model (in sec): 20.34
Accuracy on test set = 98.45 %
Value of bias (b) obtained = 1.79642
The size of vector w is same as the no. of features in each sample.

2.1.2 Gaussian Kernel using CVXOPT package

For the case of Gaussian Kernel, $W(\alpha)$ changes to:

$$W(\alpha) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}\phi(x^{(i)})^T\phi(x^{(j)}) - \sum_{i=1}^{m}\alpha_i$$

$$\text{where, } \phi(x)^T\phi(z) = K(x,z) = exp^{-\gamma * ||x-z||^2}$$

All the matrices, except P, remains same as in the part 1 above. In this case, P is a m x m matrix such that

$$P_{ij} = y^{(i)}y^{(j)}exp^{-\gamma * ||x^{(i)}-x^{(j)}||^2}$$

In this case, w cannot be stored in form of a vector as value of $\phi(x)$ are not explicitly known. Instead, the support vectors are stored. We know that $w = \sum_{i=1}^{m}\alpha_i y^{(i)}\phi(x^{(i)})$. Thus, calculating during prediction, we get (by using the definition of Gaussian Kernel):

$$w^T x + b = \sum_{i=1}^{nsv}\alpha_i y^{(i)}exp^{-\gamma * ||x^{(i)}-x||^2} + b$$

The above summation is over the support vectors.

Results (with $C = 1$ and $\gamma = 0.05$):
No. of Support Vectors (with threshold = 1e-6) = 1442
Time taken to train the model (in sec) = 21.989
Accuracy on test set = 99.250 %
Value of bias (b): -0.3795

2.1.3 Linear and Gaussian Kernel using LIBSVM package

In this part, I used LIBSVM package to solve the problems in part (a) and (b) above.
Accuracy using Linear Kernel = 98.4515 %
Accuracy using Gaussian Kernel = 99.2507 %

Comparison with results obtained in part (a) and (b) above:

	Bias	nSV	Training Time	Accuracy
Linear Kernel (Part-a/CVXOPT)	1.79642	159	20.34 sec	98.45 %
Linear Kernel (LIBSVM)	1.79646	159	1.078 sec	98.4515 %
Gaussian Kernel (Part-b/CVXOPT)	-0.3795	1442	21.989 sec	9.250 %
Gaussian Kernel (LIBSVM)	-0.3814	1323	4.183 sec	99.2507 %

For Linear Kernel, the values of Bias (b), no. of Support Vectors and Accuracy are almost same for both the methods. But the training time is much lesser (takes around 19.33 sec lesser) when we are using LIBSVM package.
For Gaussian Kernel, the value of Bias (b) and Accuracy are almost same. The no. of support vectors are not exactly same as the no. of support vectors in part-b depends on the threshold value chosen. In this case as well, the LIBSVM package perform the training much faster (takes around 17.8 sec lesser) than in the part-b above.

2.2 Multi-Class Classification

2.2.1 One-vs-One Multi-Class Classification using CVXOPT package

In this part, I have implemented one-vs-one multiclass SVM using a Gaussian Kernel ($\gamma = 0.05$ and C=1). I trained 45 different binary classification models for each pair of classes (there are total 10 classes). During the testing, for a given sample, all the 45 models are used to predict the class. The class that is predicted by most no. of models is treated as the multi-class prediction for the sample. In case two classes 'a' and 'b' are both predicted by the maximum no. of models, then the output given be the binary classification model for ('a','b') is treated the multi-class prediction for the sample.

Result Obtained on test-set:

- Time taken to train Multi-Class one-vs-one SVM classifier (in sec) = 1134.725
- Accuracy on test-set = 97.17 %

2.2.2 Multi-Class Classification using LIBSVM package

In this part, I used LIBSVM package to train a Multi-class SVM classification model. The result obtained with $\gamma = 0.05$ and C=1 is:

- Time taken to train multiclass SVM using libsvm library = 180.335 sec
- Accuracy on test set = 97.23 %

Comparison with our One-vs-One Multi-Class Classification implementation:
The test accuracy in this case is almost same as that obtained in the part-a above but the training time in this case is significantly smaller as compared to that in part-a. There is a reduction of around 84 % in the training time using LIBSVM multi-class classifier.

2.2.3 Confusion Matrix

Confusion Matrix Obtained using one-vs-one Multi-Class SVM Classification ($\gamma = 0.05$ and C=1) in Part a:

Actual/Predicted	0	1	2	3	4	5	6	7	8	9
0	969	0	1	0	0	3	4	1	2	0
1	0	1120	4	3	0	2	2	0	3	1
2	4	0	1006	1	1	0	1	6	13	0
3	0	0	12	977	0	5	0	6	9	1
4	0	0	4	0	961	0	7	0	2	8
5	2	0	4	5	1	868	6	1	5	0
6	6	3	0	0	4	4	938	0	3	0
7	1	4	23	2	6	0	0	979	3	10
8	4	0	3	6	2	5	1	2	948	3
9	4	4	4	7	14	6	0	6	13	951

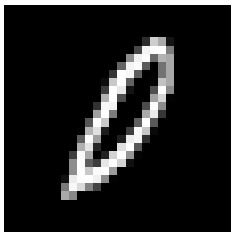
Confusion Matrix Obtained using Multi-Class SVM Classifier using LIBSVM package (with $\gamma = 0.05$ and C=1) in Part b:

Actual/Predicted	0	1	2	3	4	5	6	7	8	9
0	969	0	1	0	0	3	4	1	2	0
1	0	1121	3	2	1	2	2	0	3	1
2	4	0	1000	4	2	0	1	6	15	0
3	0	0	8	985	0	4	0	6	5	2
4	0	0	4	0	962	0	6	0	2	8
5	2	0	3	6	1	866	7	1	5	1
6	6	3	0	0	4	4	939	0	2	0
7	1	4	19	2	4	0	0	987	2	9
8	4	0	3	10	1	5	3	3	942	3
9	4	4	3	8	13	4	0	9	12	952

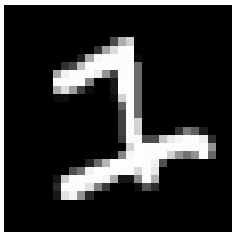
Observation:

- Most of the digits are correctly classified as clear from the diagonal values in the above Confusion matrices.
- Among the mis-classified images, some most common mis-classified digits are 2 mis-classififed into 8, 7 mis-classified into 2, 8 mis-classified into 3 and 9 mis-classified into 4 and 8.

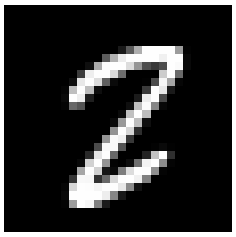
Visualization of some miss-classified digits:



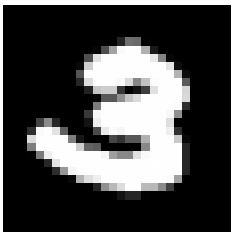
a). Predicted as 6



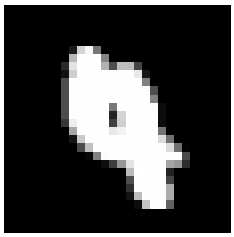
b). Predicted as 2



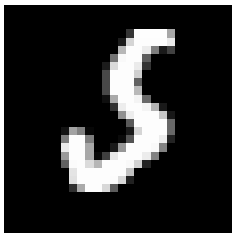
c). Predicted as 8



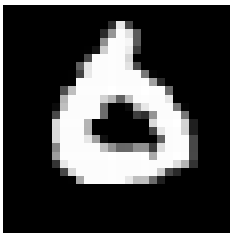
d). Predicted as 2



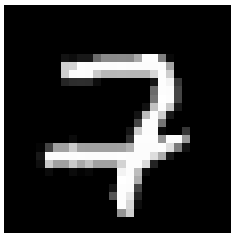
e). Predicted as 9



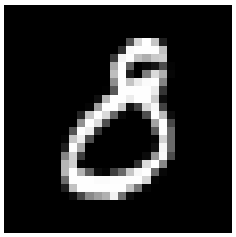
f). Predicted as 6



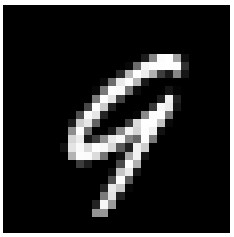
g). Predicted as 0



h). Predicted as 2



i). Predicted as 3



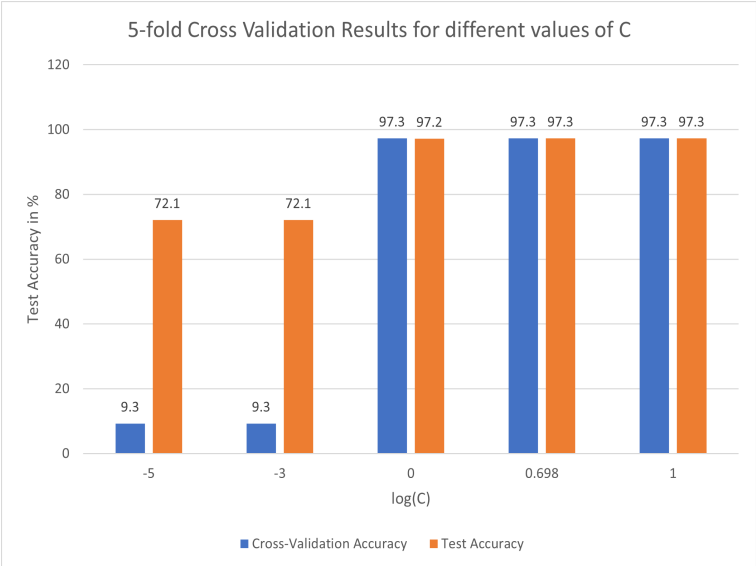
j). Predicted as 4

Based on the above images of mis-classified digits, the points mentioned in observation above do make sense in most cases. For example, in fig (b) above, 1 is mis-classified as 2 because the way in which 1 is written here is different from 2 only due to the middle vertical line. Similarly, 9 is predicted as 4 in fig (j) and 4 is predicted as 9 in fig (e). Also, both 4 and 9 shown above are very much similar. Similar argument applies for 0 classified into 6 in fig (a) and 6 classified into 0 in fig(g).

2.2.4 K-fold Validation

In this part, I calculated the 5-fold cross validation accuracy for different values of C in {1e-5, 1e-3, 1, 5, 10}. I also calculated the test accuracy for these values of C.
The result obtained is as below:

C	5-fold Cross-Validation Accuracy	Test Result
1e-5	9.375 %	72.12 %
1e-3	9.375 %	72.12 %
1	97.31 %	97.24 %
5	97.37 %	97.31 %
10	97.37 %	97.31 %



Observation: For the given dataset, the cross validation accuracy increases as we increase the value of C. The highest cross-validation accuracy is obtained for C = 5 and C = 10.
Also, the test accuracy also increases as the cross validation accuracy increases in this case.