COL774 – Machine Learning (Assignment-2)

Prakhar Aggarwal
2019CS50441

Part 1 (Text Classification)

(a) Implementing Naive Bayes Algorithm

For this part, I have simply computed the θ_{jl} values where "l" ϵ {1, 2, ..., |V|}, V being the vocabulary. I have also performed parameter typing i.e.
$\theta_{jl} = \theta_{j'l} \; \forall \; j,j'$ where j denotes the j^{th} feature.

Using θ_{jl}, we can compute the conditional probability $P(y|x; \Theta)$ for all classes and take the argmax to predict the label class.

Train Set Accuracy = 70.18 %
Test Set Accuracy = 66.49 %

(b) Random/Majority Prediction

Random Prediction Accuracy over Test Set = 20.79%
Majority Prediction Accuracy over Test Set = 66.09%
(High value of majority prediction is because of large number of examples for class 5)

On comparing the accuracy for random prediction,majority prediction with naive bayes prediction; naive bayes perform slightly better than the majority baseline.
Improvement over Random Prediction: 68.73%
Improvement over Majority Baseline: 0.60%

(c) Confusion Matrix

	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5
Actual 1	2	0	7	36	183
Actual 2	0	0	7	102	217
Actual 3	1	0	4	349	732
Actual 4	2	0	1	462	2643
Actual 5	7	1	7	397	8840

The highest value of the Diagonal Entry is corresponding to the 5 star rating i.e. 5 star review predicted as a 5 star review. This can be attributed to the fact that class 5 has the largest number of examples. Also, the accuracy of a 5 star review to be predicted correctly is ~ 95%, so, we can say that, given a 5 star example, probability of it being predicted correctly is quite high.

Other observations:

- Sum over k^{th} row in the confusion matrix denotes number of examples belonging to that class.
- The model performs badly over other classes {1, 2, 3, 4} majorly because of very high number of 5 star examples.
- For obtaining $P(y' = k | x, y = k)$ = (diagonal entry in k^{th} row)/(sum of k^{th} row) i.e. given a k star review, probability of it being predicted correctly by the model is given by the diagonal entry in k^{th} row by sum of k^{th} row.

(d) Stopword removal and Stemming

To remove the stopwords and for performing stemming, I iterated over the reviews, for the i^{ith} review, I split it and clean the words obtained.

Cleaning is done as follows:

- Special characters (characters other than A-Za-z0-9) are removed
- Question mark at the end i.e. \r?\n is replaced with ‘ ‘

After cleaning the data, I have checked for the stopwords and removed them from the data. In last, stemming is performed to obtain the root word.

Train Set Accuracy (stemmed data) = 69.704 %
Test Set Accuracy (stemmed data) = 66.200 %

Thus, here I find that the accuracy for the training set as well for the test set is reduced (~0.67%, 0.43%).

Reason for this can be explained by the removal of stopwords like “not”, “neither”, “did” which captures negative sentiment. Removing these from the dataset affect the accuracy.

(e) Feature Engineering

Two features included in the model:

1.) Bigrams:

Bigrams can be explained as follow:
Input: [“Today”, “is”, “raining”]
Output: [“Today_is”, “is_raining”] i.e. combines alternate words
Bigram helps to capture adjacent dependencies.

Thus, after performing stopword removal and stemming, the reviews were replaced by their bigrams in both training as well as test set i.e. this feature was added on top of the model obtained in part d.

Training set Accuracy (bigrams): 96.236%
Test set Accuracy (bigrams: 66.614%

We can see that the accuracy over training set is improved drastically, also, accuracy over the test set has also increased marginally.

2.) Skip-grams:

Skip-grams can be explained as follows:
Input: [‘a’, ‘b’, ‘c’, ‘d’] with a skip distance of 1
Output: [‘a_b’, ‘a_c’, ‘b_c’, ‘b_d’, ‘c_d’] i.e. combine words upto skip distance
Skip-gram helps to capture longe range dependencies between words.

Thus, after performing stopword removal and stemming, the reviews were replaced by their skipgrams (skip_distance = 3) in both training as well as test set i.e. this feature was added on top of the model obtained in part d.

Training set Accuracy (bigrams): 95.592%
Test set Accuracy (bigrams: 66.328%

We can see that the accuracy over training set is improved drastically, also, accuracy over the test set has also increased marginally (as compared to part d)

Model Summary

	Training set Accuracy(%)	Test set Accuracy(%)
Normal Naive Bayes	70.182	66.492
Random Guessing	-	20.791
Majority Prediction	-	66.092
Naive Bayes (stopwords, stemming)	69.704	66.200
Naive Bayes (stemming, bigrams)	**96.236**	**66.614**
Naive Bayes (stemming, skip-grams)	95.592	66.328

Out of all the models, **bigram model** performs best with increase in training as well as test accuracy.

(f) F1-score

F1-score is a measure of model's accuracy on a dataset.

F1-score over Naive Bayes (stopwords, stemming, bigrams) model:

	F1-score
Class 1	0.008733
Class 2	0.0
Class 3	0.009107
Class 4	0.127807
Class 5	0.803379

Macro F1-score: 0.189805

F1-score focuses more on False negatives and False positives. Also, F1-score is a better metric in case of imbalanced classes.

Our dataset dominates in a specific class (5 star), so, in such a case, F1-score becomes a obvious choice for metric.

(f) “Summary” field

In this part, instead of using ‘reviewText’ field, I used the summary field.

Training set Accuracy: 66.81%
Test set Accuracy: 67.27%

We can see there is an improvement in the test accuracy and this outperforms the best model that we have obtained (Naive Bayes (stemming, bigrams)) i.e. prediction accuracy is improved on using the **‘summary’** field

On combining the Bigram model obtained in part e with the ‘summary’ field:
Training set Accuracy: 84.664%
Test set Accuracy: 66.80%

Part 2 (MNIST Digit Classification)

Part a (Binary Classification)

For this part, I used the CVXOPT Package to solve the convex constrained optimization problem (SVM problem).

SVM's dual objective is given by:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \dots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

$W(\alpha)$ is represented in the form $(1/2)\alpha^T P\alpha + q^T\alpha$.

SVM's optimization problem changes to: (replace x with α)

$$\begin{aligned} \text{minimize} \quad & (1/2)x^T Px + q^T x \\ \text{subject to} \quad & Gx \le h \\ & Ax = b \end{aligned}$$

with parameters:

P: M * M array, where $P[i,j] = y^{\{i)} y^{(j)} <x^{(i)}, x^{(j)}>$
q: M * 1 array of -1

G: 2M * M array (the condition, $0 \le \alpha_i \le C$ is visualized as $(-\alpha_i) <= 0$ and $\alpha_i <= C$)
First half is a diagonal matrix with all entries -1
Second half is a diagonal matrix with all entries 1

h: 2M * 1 array, first M entries are 0, remaining M entries are C.
A: 1*M array, y^T
b: 0

Since my entry number ends in 1, I performed the binary classification problem over class 1 vs class 2.

After formulating the SVM problem, we can solve it using CVXOPT package and obtain the α_i values. Using a certain threshold value (1e-5), we can obtain the support vectors (values that determine the hyperplane).

For the Linear Kernel, we can easily compute w as feature vector is finite. ‘b’ can also be computed using the expression derived in class.

$w = \sum_{i=1}^{m} \alpha_i * y^{(i)} * x^{(i)}$ $\quad$ $b = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}$

Formula for prediction: $w^T x + b$

For the Gaussian kernel, we cannot compute w because of feature vector being infinte dimensional. Instead, we will use $w^T \varphi(x)$ directly to predict y as well as to compute b.

$w = \Sigma\, \alpha_i y^{(i)} \varphi(x^{(i)})$
So, $w^T \varphi(x) = \Sigma\, \alpha_i y^{(i)} \varphi(x^{(i)})^T \varphi(x)$

=> $w^T \varphi(x)$ can be computed using inner product over $<\varphi(x), \varphi(x^{(i)})>$ which further can be expressed as $K(x,x^{(i)})$

For the gaussian case, kernel value is given by:

$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right)$$

(i) SVM problem using a Linear Kernel (CVXOPT Package)

No. of Support Vectors: 159
weight matrix (w) -> (784,1) matrix
intercept term (b) -> -0.118919

Test set accuracy: 98.477%
Validation set accuracy: 97.975%

(ii) SVM problem using a Gaussian Kernel (CVXOPT Package)

No. of Support Vectors: 867
intercept term (b) -> 0.744976

Test set accuracy: 99.723%
Validation set accuracy: 100%

(ii) SVM problem using Linear/Gaussian Kernel (LIBSVM Package)

Linear:

	CVXOPT	LIBSVM
Test set accuracy	98.48%	99.031%
Validation set accuracy	97.98%	100%
Number of Support Vectors	159	158
Intercept Term	-0.118919	0.008999
Running Time	57 sec	3.5 sec

Gaussian:

	CVXOPT	LIBSVM
Test set accuracy	99.723%	99.585%
Validation set accuracy	100%	99.975%
Number of Support Vectors	867	848
Intercept Term	0.744976	0.040537
Running Time	104 sec	8.3 sec

So, we can see that the accuracy of LIBSVM and our CVXOPT implementation is similar, but the training + prediction time is significantly better in LIBSVM.
This is due to a highly optimised and parallelised code running in LIBSVM

Part b (Multi Class Classification)

For this part, we willl extend the SVM formulation for a binary classification problem. We will train a model on each pair of classes.
This method is also said to be one vs one classifier.

(i) Multi-Class SVM using Gaussian kernel (CVXOPT package)

Test set accuracy: 94.95%
Training set accuracy: (Tried running model 5 times for more than 4 hrs but laptop hanged)

Running time for test set prediciton: 9266.0056 seconds
Running time for validation set prediciton: 5+ hrs but laptop hanged

Confusion Matrix for the test set is given by:

	Predicted 0	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7	Predicted 8	Predicted 9
Actual 0	967	4	0	0	0	3	3	1	1	1
Actual 1	0	1135	0	0	0	0	0	0	0	0
Actual 2	7	75	908	10	4	0	5	15	6	2
Actual 3	0	41	2	944	0	4	0	12	3	4
Actual 4	0	11	1	0	951	0	5	1	1	12
Actual 5	2	25	0	5	3	841	10	2	2	2
Actual 6	4	12	0	0	5	2	933	0	0	0
Actual 7	0	45	2	0	0	0	0	971	0	10
Actual 8	5	41	0	9	10	6	3	8	878	14
Actual 9	3	19	1	5	7	1	1	5	0	967

(ii) Multi-Class SVM using Gaussian kernel (LIBSVM package)

In this part, one vs one classifier is implemented using LIBSVM package

Test set accuracy : 97.23%
Validation set accuracy : 99.92%
Running time (test set + training set prediction) : 760 seconds

Confusion matrix for the test set is:

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

Confusion matrix for the validation set is:

	Predicted 0	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7	Predicted 8	Predicted 9
Actual 0	2000	0	0	0	0	0	0	0	0	0
Actual 1	0	1997	1	0	1	0	0	1	0	0
Actual 2	0	0	2000	0	0	0	0	0	0	0
Actual 3	0	0	0	1999	0	0	0	1	0	0
Actual 4	0	0	0	0	1999	0	0	0	0	1
Actual 5	0	0	0	0	0	2000	0	0	0	0
Actual 6	0	0	0	0	1	0	1999	0	0	0
Actual 7	0	2	1	0	1	0	0	1995	0	1
Actual 8	0	1	0	0	0	0	0	0	1999	1
Actual 9	0	0	0	0	2	0	0	2	0	1996

(iii) Comparison between Mutli-Class CVXOPT vs Multi-Class LIBSVM

So, we can see, the accuracy of LIBSVM and our CVXOPT implementation is similar (3% difference), but the training + prediction time is significantly better in LIBSVM (150 mins compared to 13 mins)

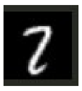
This is due to a highly optimised and parallelised code running in LIBSVM and SMO algorithm which works extremely faster than the CVXOPT matrix solver.

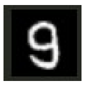
On observing the confusion matrix for the test set, we can see that examples for digit 9 (~5.6%) are the ones that are most missclassified into digit 4 and digit 8.

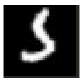
Whereas, digit 0 and digit 1 are the ones that are most correctly classified (~98.87%, ~98.76%).

Misclassified digits: 2 and 7, 2 and 8, 4 and 9, 8 and 9
Results makes sense as these digits look similar to each other and such errors are also very easy to be made by the human eye by just observing images.

Some examples of miss-classified digits:

1.)  Correct label: 2, Predicted label: 7 (Example: 322 in test set (indexing from 1)

2.)  Correct label: 9, Predicted label: 8 (Example: 242)

3.)  Correct label: 5, Predicted label: 3 (Example: 341)

(iv) K-fold cross validation

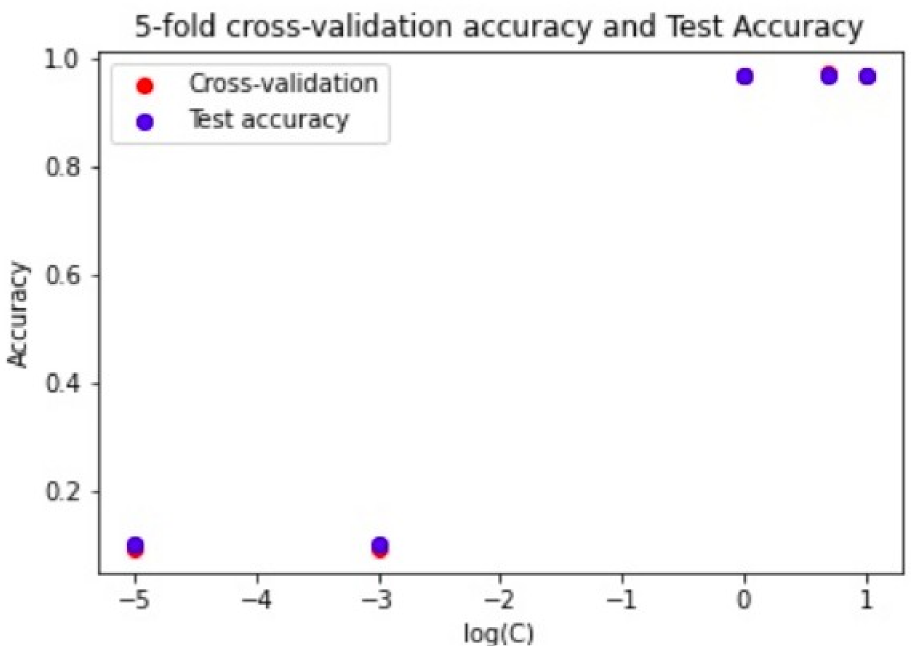
For this part, we had to divide the training data into 5 parts and then run the 5-fold cross validation.

Validation accuracies obtained for different values of C:
$C = 10^{-5}$ -> 0.09275
$C = 10^{-3}$ -> 0.0935
C = 1 -> 0.97175
C = 5 -> 0.97325
C = 10 -> 0.97125

Test set accuracies obtained for different values of C:
$C = 10^{-5}$ -> 0.1009
$C = 10^{-3}$ -> 0.1009
C = 1 -> 0.9705
C = 5 -> 0.9715
C = 10 -> 0.9706

We can see that for the lower values of C (10^{-5} and 10^{-3}), both, validation set accuracy and test set accuracy are quite low. This can be understood from the expression:
$(1/2)w^Tw + C * \sum_i \xi_i$

For lower values of C, we allow noise to be high thus affecting the accuracies. On the other hand, higher values of C ensures that we penalize the noise added.



We can see that for higher values of C (1, 5 and 10) the model works fine with high accuracies over validation as well as test set.

Among these, the maximum value for the 5-fold cross-validation accuracy comes for C = 5 and the same trend is observed over the test set.

It took approximately 1 hour for this part to run.