COL774 - MACHINE LEARNING

ASSIGNMENT 2 REPORT

Submitted By:
Mehak
2018MCS2143

PART-A (Naive Bayes)

Q1. a. Naive Bayes algorithm implementation

With re (Removes Punctuation marks where re = regular expression)
Training set accuracy = 55.4940247386 %
Test set accuracy = 59.0803033249 %

With split
Training set accuracy = 51.0052872463 %
Test set accuracy = 61.113687013 %

With word_tokenizer (Doesn't remove punctuation, consider as new token)
Training set accuracy = 54.7471170673 %
Test set accuracy = 56.7702179213 %

The data used by the model is cleaned by removing any punctuation marks and all uppercase letters are made lowercase letters.
Computational Time = 481.495809078 seconds

b. Accuracy for random prediction on test set is = 19.9816030751 %
Accuracy for majority prediction on test set is = 43.9895900328 %
Test set accuracy = 59.0803033249 %

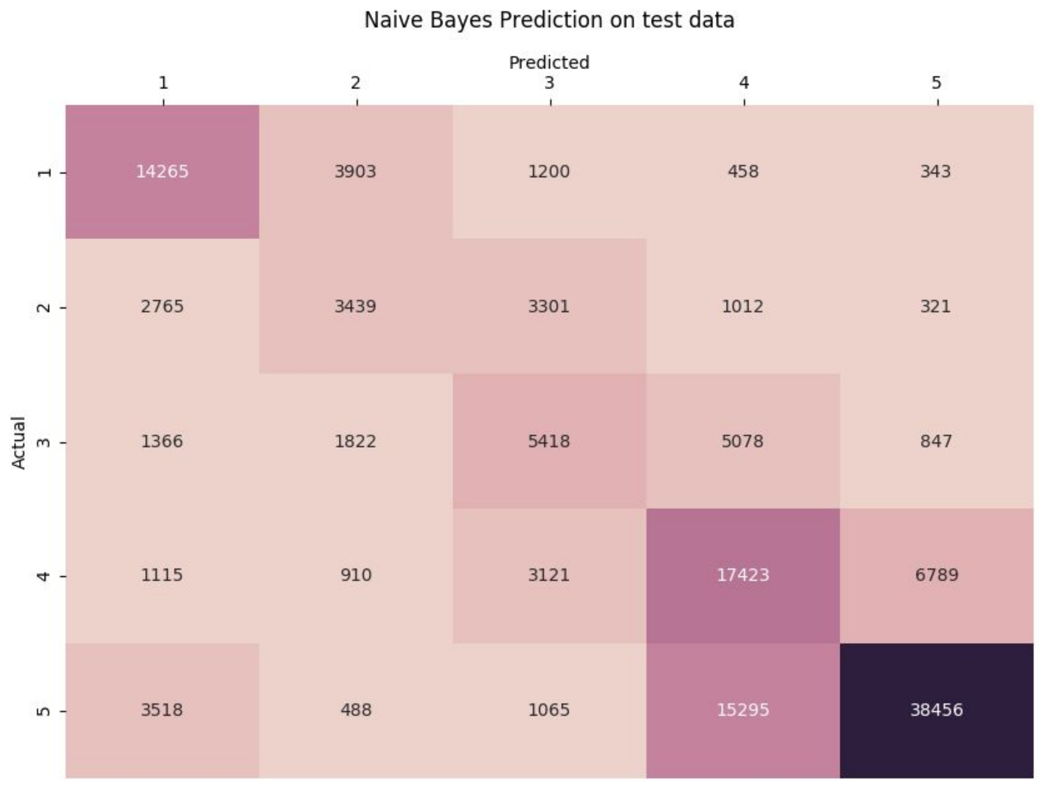
Improvement of algorithm
Over random = 39.0987003 %
Over majority = 15.0907133 %

c. Confusion Matrix

Naive Bayes (for testing data) - 59.0803033249 % accuracy

Confusion matrix on test set is:

```
[[ 14265  3903  1200   458   343]
 [  2765  3439  3301  1012   321]
 [  1366  1822  5418  5078   847]
 [  1115   910  3121 17423  6789]
 [  3518   488  1065 15295 38456]]
```



Observations:

Diagonal values indicate classes that are correctly classified. In this case, Class 5 has the highest value for diagonal entry which means the model is able to distinguish Class 5 from all other classes correctly.

Confusion matrix indicates number of examples classified correctly, number of examples classified incorrectly and their correlation.
As we can observe from diagonal, the classifier predicts Class 1, Class 4 and Class 5 correctly.

d. Naive Bayes implementation with stemming

Using split
Accuracy over train dataset = 55.1416039725 %
Accuracy over test dataset = 60.0614726514 %
Observation: Accuracy has dropped by 1% on test data.

Using tokenizer
Accuracy over train dataset = 55.1558129796 %
Accuracy over test dataset = 60.059229124 %
Observation: Accuracy is almost same as without stemming.

Using re
Accuracy over train dataset = 56.0455585635 %
Accuracy over test dataset = 59.0309457216 %
Observation: Accuracy is improved by 4% on test data.

Model using regular expression performs best with stemming. I have also removed digits along with stopwords and stemming in these case. However, without removing digits accuracy dropped by around 15% .
Computational Time = 1947.32240105 seconds

e. Feature Engineering

1st Feature: Bi-grams
I have tried using bi-grams as a feature to train a new model over the model used in last part. In this case,

Using re
Training set accuracy = 54.1293368631 %
Test set accuracy = 62.9930151513 %
Observations : Accuracy over test set is improved from both part a and d by ~3%

Using tokenizer
Training set accuracy = 66.4858134283 %
Testing set accuracy = 63.4454598483 %
Observations : Accuracy over test set is improved from part a by ~9% and from part d by ~3%

Using split
Training set accuracy = 66.5260099613 %
Testing set accuracy = 63.4357378962 %
Observations : Accuracy over test set is improved from part a by ~2% and from part d by ~3%
Computational time = 3012.39362693 seconds

2nd Feature: TF-IDF Score
In this, I have trained a new model over model in last part using tf-idf score as a feature. In this case,

Using split
Training set accuracy =50.3648536472 %
Test set accuracy = 49.4589359697 %
Observations : Accuracy over test set is decreased from part a by ~12% and from part d by ~11%

Using re
Training set accuracy = 59.7793490779 %
Test set accuracy = 59.8700249779 %
Observations : Accuracy over test set is almost similar to that in part a and part d

Using tokenize
Training set accuracy = 50.3703689855 %
Test set accuracy = 49.4679100794 %
Observations : Accuracy over test set is decreased from part a by ~7% and from part d by ~11%
Computational Time = 3093.87520409 seconds

Bi-grams feature help in improving the overall accuracy. It implies that the words taken in combinations are more effective for text classification rather than taking words independently.

f. F1-score

F1-score is a measure of test's accuracy that is the harmonic mean of precision and recall.
Since the best performing model is using bigrams with stemming. F1-score for the same is:

F1_score for test set is:
[0.75159642 0.07519432 0.13274387 0.46757133 0.79432009]

F1_macro_score for test set is:
0.44428520775285946

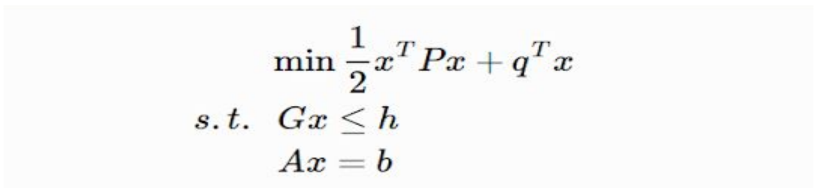
F1-macro score is more suited for this kind of dataset because it captures the average over all the classes i.e. takes into consideration the contribution of each class to the performance equally. However, test error just computes the accuracy over all the classes and will result in better result if there is class imbalance.

g. I have tried dividing data into chunks based on classes. But it was not able to compute due to memory issue.
But test accuracy is likely to improve with training over the entire data set as it will help to make prediction more accurately by making model more general.

PART - B (SVM)

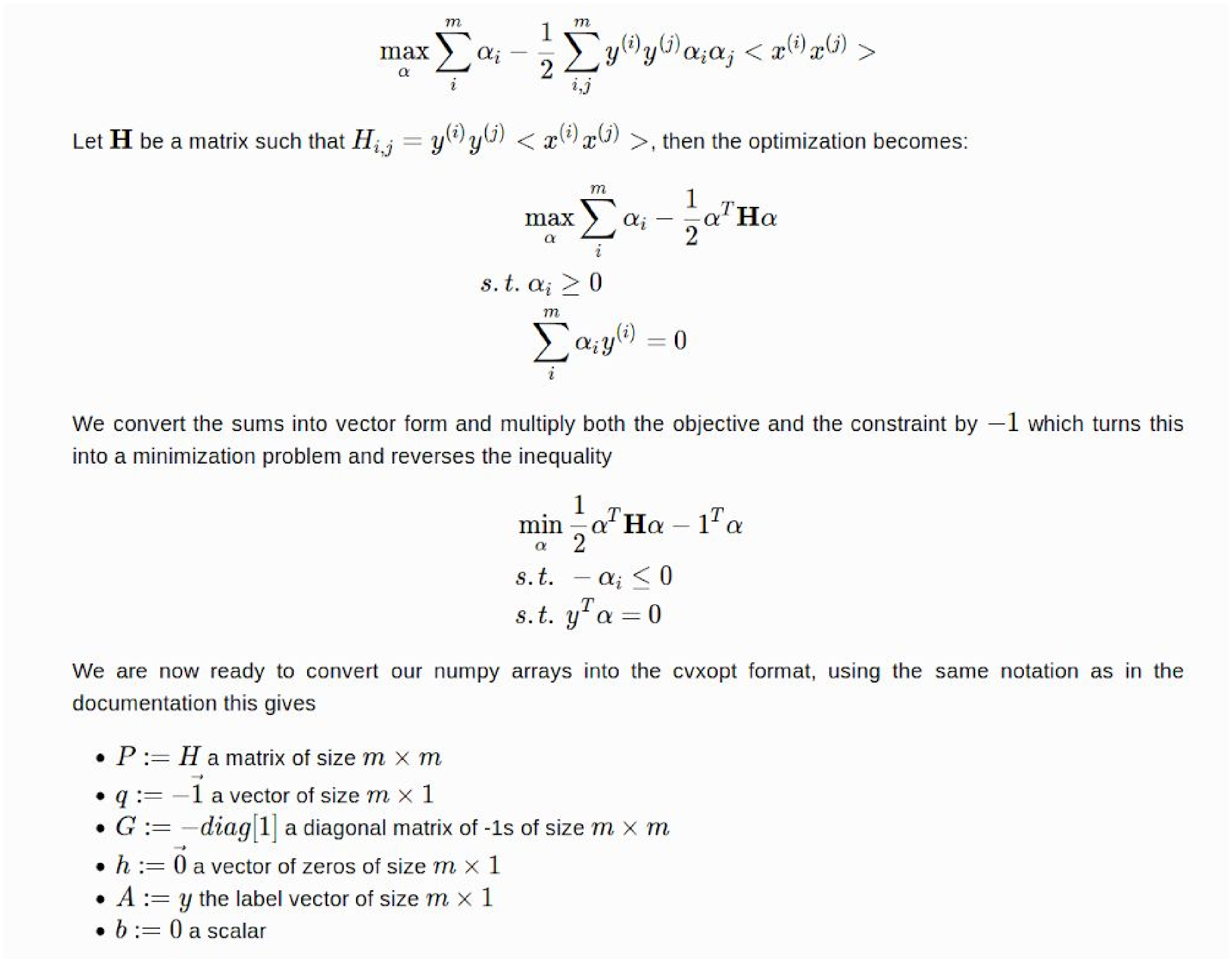
Q1. a. CVXOPT SVM dual objective

CVXOPT takes optimization problem in form



$$\begin{aligned} &\min \frac{1}{2}x^TPx + q^Tx \\ s.t.\quad &Gx \leq h \\ &Ax = b \end{aligned}$$

Soft margin SVM dual problem



$$\max_{\alpha} \sum_i^m \alpha_i - \frac{1}{2}\sum_{i,j}^m y^{(i)}y^{(j)}\alpha_i\alpha_j < x^{(i)}x^{(j)} >$$

Let $\mathbf{H}$ be a matrix such that $H_{i,j} = y^{(i)}y^{(j)} < x^{(i)}x^{(j)} >$, then the optimization becomes:

$$\begin{aligned} &\max_{\alpha} \sum_i^m \alpha_i - \frac{1}{2}\alpha^T\mathbf{H}\alpha \\ s.t.\ &\alpha_i \geq 0 \\ &\sum_i^m \alpha_i y^{(i)} = 0 \end{aligned}$$

We convert the sums into vector form and multiply both the objective and the constraint by -1 which turns this into a minimization problem and reverses the inequality

$$\begin{aligned} &\min_{\alpha} \frac{1}{2}\alpha^T\mathbf{H}\alpha - 1^T\alpha \\ &s.t.\ \ -\alpha_i \leq 0 \\ &s.t.\ y^T\alpha = 0 \end{aligned}$$

We are now ready to convert our numpy arrays into the cvxopt format, using the same notation as in the documentation this gives

- $P := H$ a matrix of size $m \times m$
- $q := -\vec{1}$ a vector of size $m \times 1$
- $G := -diag[1]$ a diagonal matrix of -1s of size $m \times m$
- $h := \vec{0}$ a vector of zeros of size $m \times 1$
- $A := y$ the label vector of size $m \times 1$
- $b := 0$ a scalar

Weight vector $w = \Sigma\ \alpha^{(i)} * y^{(i)} * x^{(i)}$

Bias $b = \Sigma\ y^{(i)} - w^T x^{(i)}$

Number of support vectors = 134 out of 4000 points

Accuracy on test set = 99.4979919679 %

Bias b = -0.02193316

Support vectors and weight vector are included in log.txt file.

Q1. b. Multi-class SVM with Gaussian Kernel (using CVXOPT library)

In case of Gaussian Kernel, in H matrix

$<x^{(i)},x^{(j)}> = \exp(-\gamma * ||x^{(i)} - x^{(j)}||^2)$

Number of support vectors = 295 out of 4000 points

Accuracy on test set = 99.6485943775 %

Bias b = -0.88183489

Support vectors are included in log.txt file.

Improvement of accuracy over linear kernel = 0.1506024

Q1. c. Multi-class SVM with Linear Kernel (using LIBSVM library)

Accuracy on test set = 99.4979919678 %

Bias b = -0.046822

Number of support vectors = 134 out of 4000

Training time with libsvm linear kernel 0.633661031723 seconds

Training time with cvxopt linear kernel 402.386872053 seconds

Comparison : Training time with libsvm is reduced by ~402 seconds

Weight vector is stored in log file.

Multi-class SVM with Gaussian Kernel (using LIBSVM library)

Accuracy on test set = 99.8995983935 %

Bias b = -0.194721

Number of support vectors = 1344 out of 4000

Training time with libsvm gaussian kernel 5.43338704109 seconds

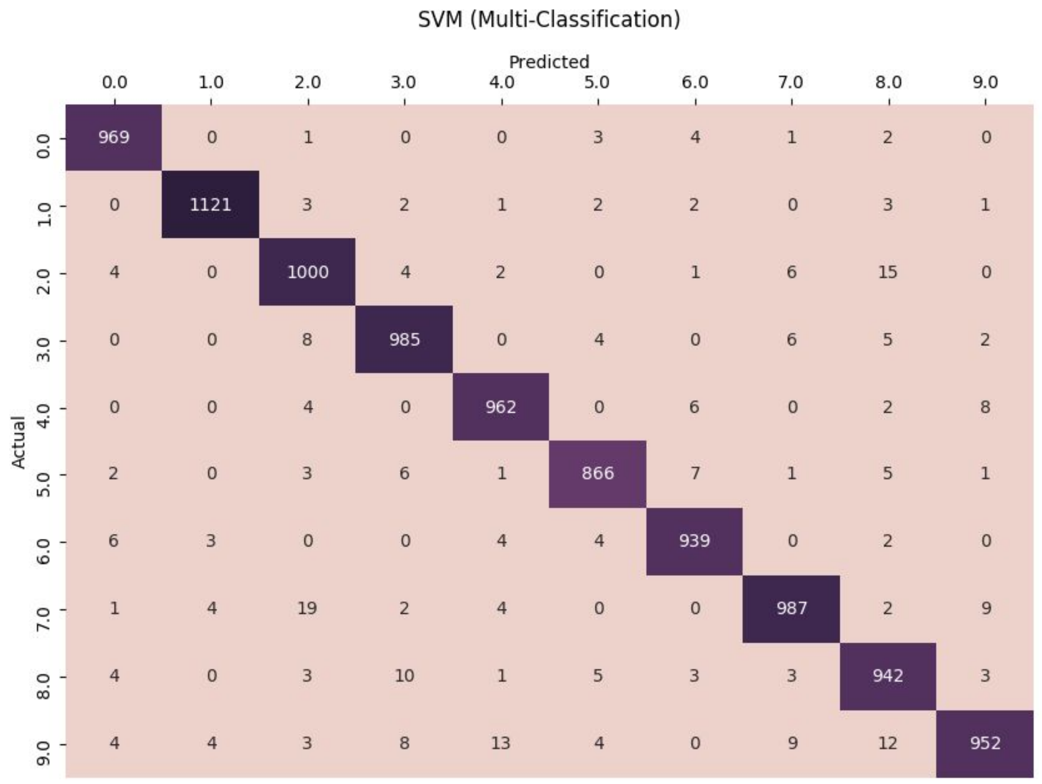
Training time with cvxopt gaussian kernel 487.347285032 seconds

Comparison : Training time with libsvm is reduced by ~482 seconds

Q2. a. Multi-class SVM using cvxopt
Accuracy over training set using libsvm = 99.45%
Accuracy over test set using libsvm = 95.31%

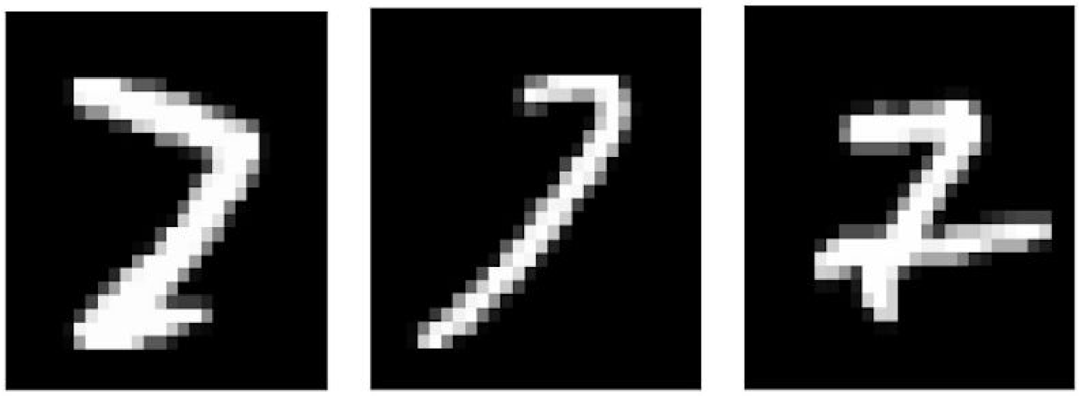
Q2. b. Multi-class SVM using libsvm
Accuracy over training set using libsvm = 99.92%
Accuracy over test set using libsvm = 97.23%
Training time with libsvm gaussian kernel = 268.355156183 seconds
Training time with cvxopt gaussian kernel = Around 3-4 hours
Comments : Accuracy over the test set is improved by ~2%. However the computational cost is improved a lot. The memory requirement and computational time using libsvm is improved a lot than cvxopt.

Q2. c.



	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	969	0	1	0	0	3	4	1	2	0
1.0	0	1121	3	2	1	2	2	0	3	1
2.0	4	0	1000	4	2	0	1	6	15	0
3.0	0	0	8	985	0	4	0	6	5	2
4.0	0	0	4	0	962	0	6	0	2	8
5.0	2	0	3	6	1	866	7	1	5	1
6.0	6	3	0	0	4	4	939	0	2	0
7.0	1	4	19	2	4	0	0	987	2	9
8.0	4	0	3	10	1	5	3	3	942	3
9.0	4	4	3	8	13	4	0	9	12	952

Observations : Digit 1 is classified most accurately. We can observe highest non-diagonal entry is between 7 and 2. So, digit 7 is mis-classified most often into digit 2.

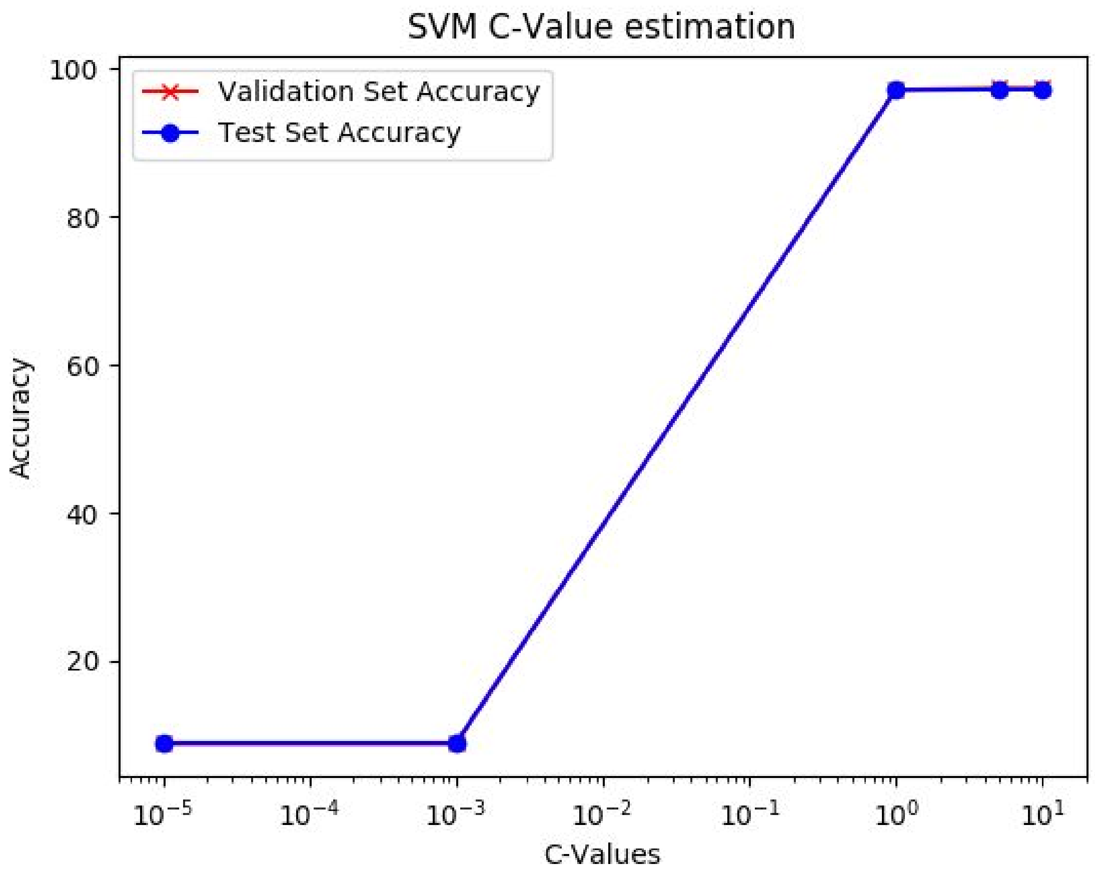


Here are the examples of digits that are likely to be misclassified. The results do make sense as some of these are even difficult for humans to classify.

However, digit 2 is mis-classified often as digit 8 where digit 9 is also oftenly mis-classified as 4 and 8.

Q2. d. C-parameter estimation using validation

C	10^(-5)	10^(-3)	1	5	10
V_Acc	8.75	8.75	97.05	97.35	97.35
T_Acc	10.28	10.28	97.1	97.17	97.17



C values = 5 and 10 gives best validation as well as test set accuracy.
Observation : As the value of C parameter increases, accuracy over validation and test set increases. Low values result in poor accuracy.