COL774: MACHINE LEARNING
ASSIGNMENT 2

SUBMITTED BY: SAMIDHA VERMA (2020CSY7575)

Note :

- **Please run 'chmod 777 run.sh' in case running run.sh gives error**
- **Please cd into the main folder where run.sh is kept and run all execution commands from there.**
- **In time consuming codes like Multiclass classification, I have saved the pickle files of model parameters and results of all 45 classifiers. If the evaluators want to run the training without the saved files, all they must do is replace the saved parameter with False instead of true in the main function of the codes for part (a). Since other parts use results from part (a), it would be redundant to make saved = False and recompute parameters in other cases. I have not saved the preprocessed data because it was taking up huge space and would not be able to fit in moodle limit of 20 MB. It took approximately 14 minutes to preprocess the data, and it happens once only when part a is run.**
- **Similarly in Naïve Bayes I have saved required files of vocabulary, model parameters etc. If saved parameter is made to False, training would be done from scratch. This is done for parts a, d, and g. Since part (e) vocabulary was larger and parameter space was larger, the files corresponding to it were not saved in the submission since they would not fit in moodle limit of 20 MB and training would be done from scratch in this part.**

1. NAIVE BAYES

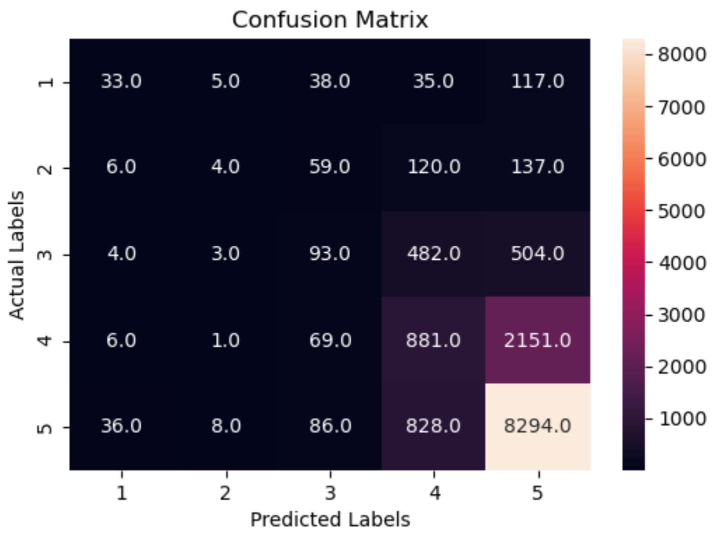
(a) Train accuracy : 69.934%
Test accuracy : 66.46%

(b) Since there are 5 classes in the dataset, if we uniformly distributed the labels at random, the accuracy that could be theoretically achieved is 20%. Practically, the following results were observed, and one can see that that the practical results approximate the theoretical ones**.**

Random prediction : 19.6%
Majority prediction : 66.085%

Improvement over random baseline : 46.86%
Improvement over majority baseline : 0.375%

(c) Confusion Matrix for part (a) on test results:

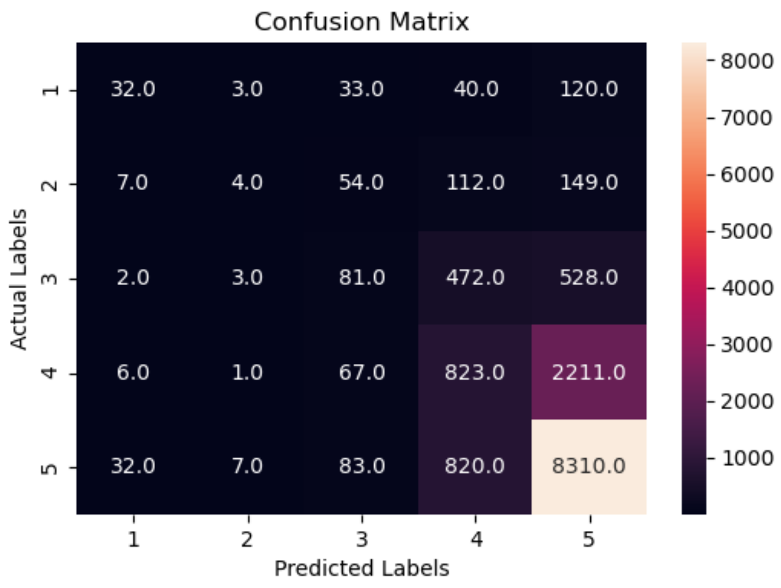


- The category that has the highest diagonal entry is 5. This means that the model trained in part (a) is correctly predicting most of the test instances of class 5, as opposed to the other classes.
- From further exploration of training data it was observed that the training data is highly skewed in terms of number of examples of class 5, therefore as we can see from the column corresponding to prediction class 5, most examples are being classified to class 5 only. This is why as seen in part (b), the multinomial naïve bayes model is only slightly better than majority prediction baseline.
- We can also see from index 2,2 that class 2 examples are mostly being misclassified.
- Most instances are being classified as class 4 or class 5, since both of them are second highest and highest classes being represented in the training data respectively.

(d) Results after cleaning review text were as follows:

- Accuracy on test set using removal of stop-words and stemming : 65.98%
- Since lemmatization of words is based on linguistics and words are more meaningful, therefore accuracy on test set using removal of stop-words and lemmatization : 66.07%
- Although it was expected that the results would improve after removal of stop words and lemmatization, however the accuracy slightly dropped. This maybe because the data is imbalanced, and probabilities may have become even more biased towards class 4 and class 5 that have higher representation in the training data. This can also be said based on the confusion matrix given below.

Confusion matrix for model with stop words removal and lemmatization was as follows:



(e) Feature engineering on data using n-grams :

Length of vocabulary on unigram (part a) : 90061
Length of vocabulary on unigram (part b) : 83276
Length of vocabulary only bigram: 2119560
Length of vocabulary on unigram + bigram: 2202836
Length of vocabulary on unigram + bigram + trigrams: 6534584

Features	Test Accuracy
Unigrams (part a)	66.46% (best)
Unigrams (part d)	66.07%
Bigrams only	66.436%
Unigrams + Bigrams	66.236%
Unigrams + Bigrams + Trigrams	66.15%

As expected, we can see that using bigrams/trigram features on top of model in part(d) improved the accuracy of the model, however as opposed to the computational overhead, the accuracy improved only slightly, and was still less than the accuracy of the original model (does not do stop word removal, lemmatization or ngram features). I think that if the training data was not heavily imbalanced then an improvement might occur.

*** As a further exercise, I would take equal samples from each class as training examples and compute the results.*

(f) The best performing model as per the observations was the Naïve Bayes model with original document (*part a*).

- F1 scores for each class in test dataset using above model are as follows:

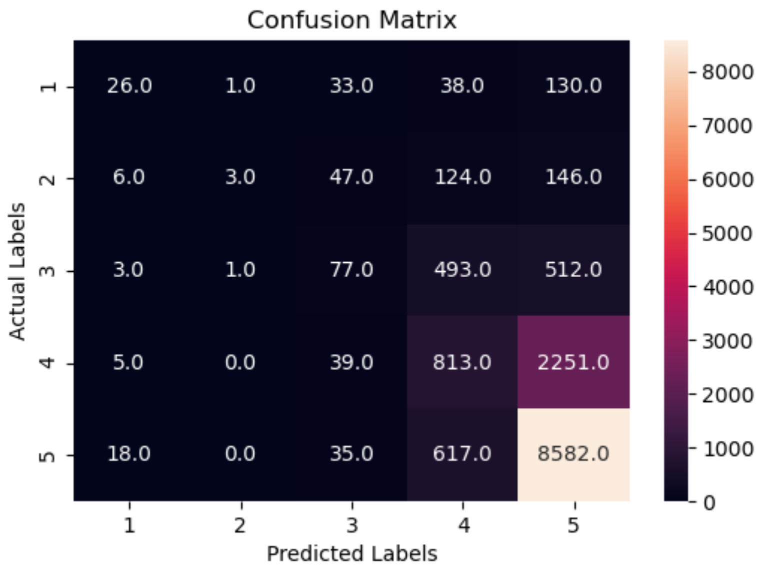
Class	F1-Score
1	0.208
2	0.023
3	0.115
4	0.306
5	0.808

- Macro averaged F1-score: 0.292
- The number of examples in the training data for classes 1, 2, 3, 4 and 5 were 2529, 2638, 5634, 13267, and 25932 respectively. This clearly shows us there is a class imbalance in the data. When there is high amount of imbalance in the data, accuracy is not a good metric. In this case, as we can see even though the test accuracy was 66.46%, the macro-f1 score is only 0.292, this is because the model is not correctly predicting correctly for examples that have less representation in the training data. There macro-averaged F1-score seems to be a better metric for this scenario.

(g) For this part, I trained a separate model based on the summary field of the training data. For the prediction, for each test instance, I computed the average log probability of each class using the predicted log probability of that class by original model based on review text, and the new model trained on summary. Argmax of the average log probabilities was the label of that instance. This gave slightly better performance on the test data in terms of test accuracy. Results are summarized in the table below:

Model	Test Accuracy
Multinomaial Naïve Bayes trained on review text only (part a)	66.46%
Multinomaial Naïve Bayes trained on summary text only.	66.31%
Ensemble model (Predicted class = argmax (avg log probability of classes)	67.86%

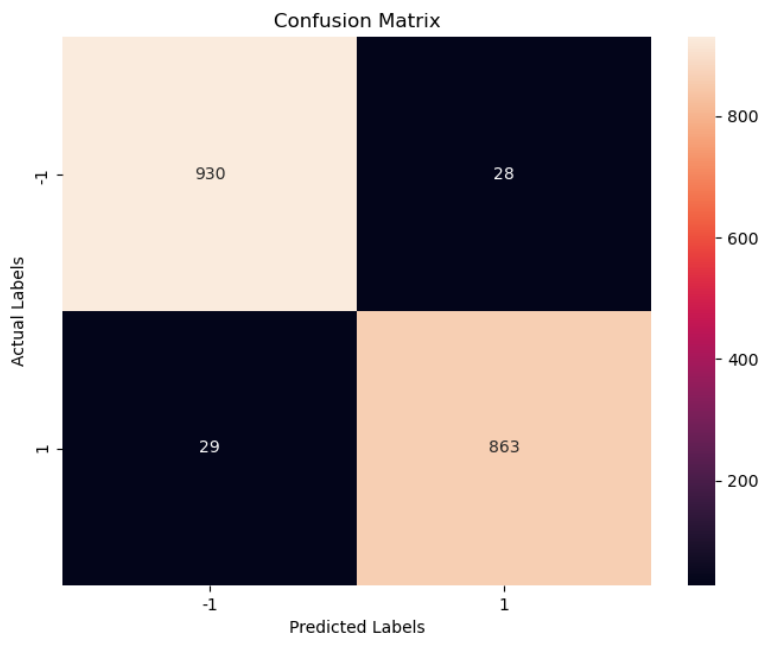
Confusion matrix of the ensemble model results is as follow:



2. (a) BINARY CLASSIFICATION : SVM

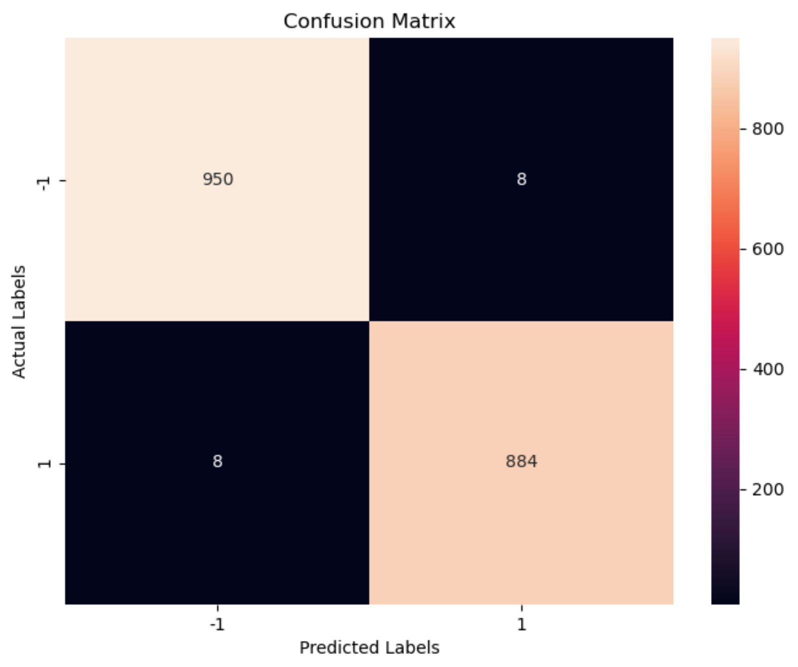
- For this question, I worked with classes 5 and 6. Since validation set was not given, I shuffled the training data which comprised of all training examples of classes 5 and 6, and took 80% for training and 20% for validation.
- C=1.0, gamma=0.05 (given)
- *Confusion matrix on test predictions of each model are given below.*
- *Results for all parts are summarized in the table given below.*
- In general we can see that SVM with gaussian kernel outperforms the one with linear kernel for both validation and test set.
- LIBSVM took less training time than CVXOPT for equivalent performance.
- Bias values are different in case of LIBSVM as compared to CVXOPT, maybe because support vectors chosen by LIBSVM are different than the CVXOPT one.
- Number of support vectors of CVXOPT were computed with lower threshold as 10^(-5) for alphas since for lower threshold it was choosing almost all training instances as support vectors and increasing computation time without significant improvement in test accuracy.
- We also observe that in general, at same threshold, gaussian kernel requires more support vectors than linear kernel.

(i) CVXOPT with Linear Kernel



Formulae used for combuting W and b:
W = np.dot(X, (alphas*Y).T)
b = 1/len(S_idx) * np.sum(Y[:,S_idx] – np.dot(W.T, X[:,S_idx]))
where S_idx is a list containing indices of support vectors in the training data, and '*' means element-wise product.

(ii) CVXOPT with Gaussian Kernel



In case of Gaussian Kernel, the transformed feature mapping becomes potentially infinite dimensional, therefore we cannot directly compute W and b values.

Method used for prediction at test time:

To compute b (here X and Y are training instances and labels):

K = kernel_matrix(X[:, S_idx], X[:, S_idx], 'gaussian')

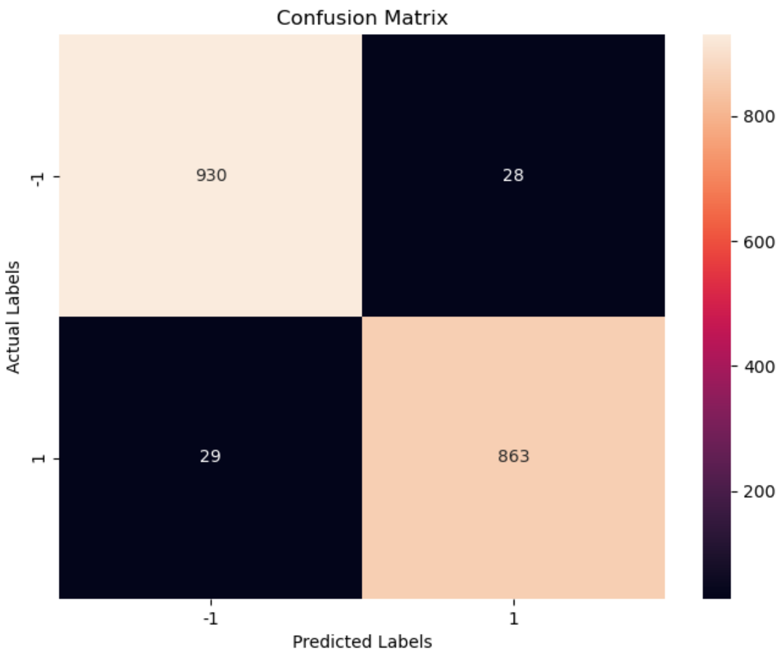
b = 1/len(S_idx) * np.sum(Y[:, S_idx] - np.dot(alphas[S_idx, :].T*Y[:, S_idx], K))

To compute result at test time:

K = kernel_matrix(X_train[:, S_idx], X_test, 'gaussian')

Y_hat = np.dot(alphas[S_idx, :].T*Y_train[:, S_idx], K) + b

(iii) LIBSVM with Linear Kernel



LIBSVM with Gaussian Kernel

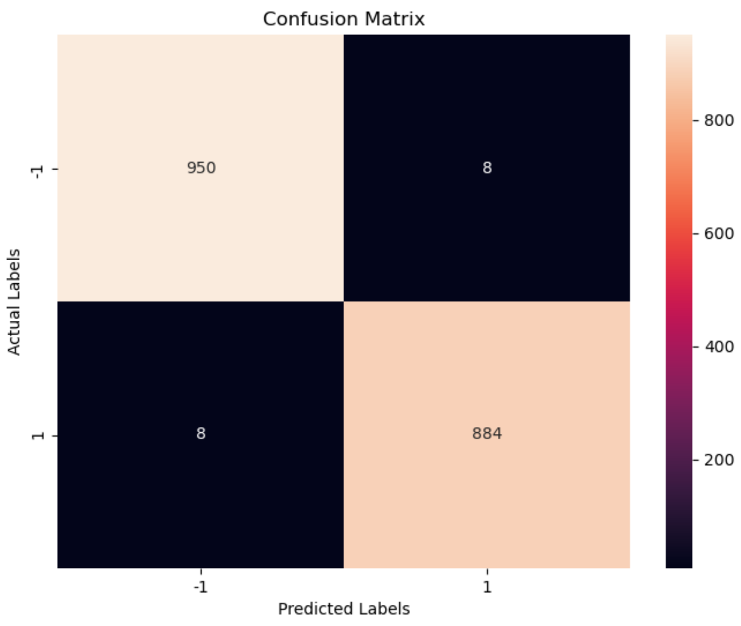


TABLE OF RESULTS

Model	Number of Support Vectors	b-value	Training time (in sec)	Validation accuracy (%)	Test accuracy (%)
CVXOPT (Linear Kernel)	206	1.61711	15.133	96.75	96.92
CVXOPT (Gaussian Kernel)	1338	0.131437	100.370	99.5	99.14
LIBSVM (Linear Kernel)	206	-0.89676	1.076	96.75	96.92
LIBSVM (Gaussian Kernel)	1303	-1.0	4.336	99.5	99.14

(b) MULTICLASS CLASSIFICATION : SVM

- Since validation set was not given, I shuffled the training data which comprised of all training examples and took 80% for training and 20% for validation.
- C=1.0, gamma=0.05 (given)

(i) CVXOPT (Gaussian Kernel)

F1 - scores of each class on test data:

```
Class:  0  -> 0.9832572298325724
Class:  1  -> 0.9872190392243279
Class:  2  -> 0.9601153291686689
Class:  3  -> 0.9663366336633663
Class:  4  -> 0.9730827831386492
Class:  5  -> 0.9730639730639731
Class:  6  -> 0.9765502866076081
Class:  7  -> 0.965041851304776
Class:  8  -> 0.9547993905535805
Class:  9  -> 0.9572649572649573

Macro averaged F1-Score:  0.969673147382248
```

(ii) LIBSVM (Gaussian Kernel)

F1 – scores of each class on test data:

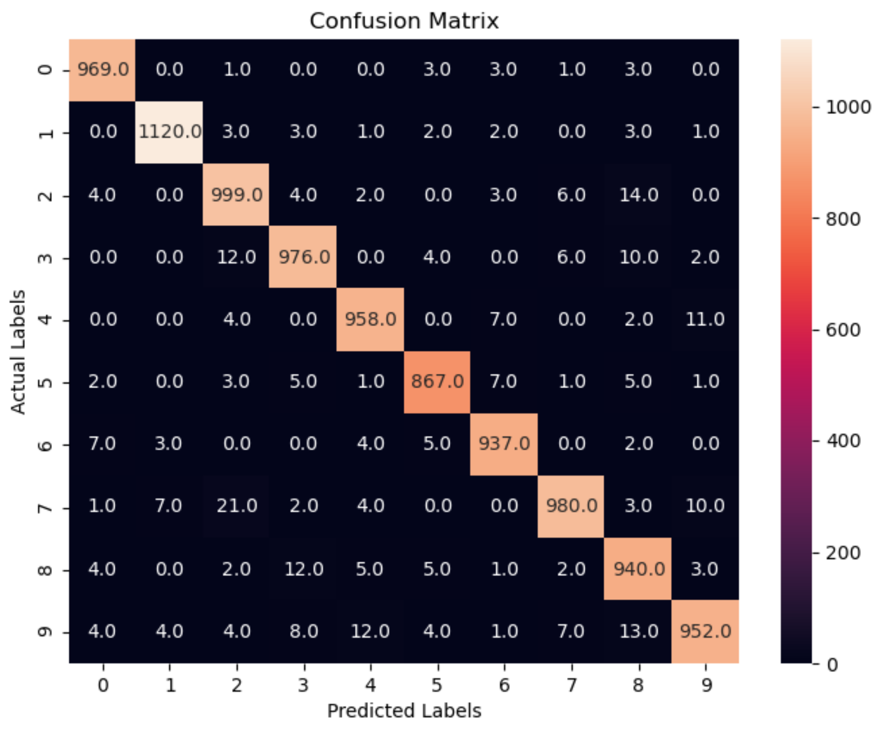
```
Class:  0  -> 0.9827411167512691
Class:  1  -> 0.9881004847950638
Class:  2  -> 0.9615384615384615
Class:  3  -> 0.9692765113974231
Class:  4  -> 0.9750889679715302
Class:  5  -> 0.9713965227145261
Class:  6  -> 0.9765747006767308
Class:  7  -> 0.9660933660933662
Class:  8  -> 0.9587366276107998
Class:  9  -> 0.9578736208625878
Macro averaged F1-Score:  0.9707420380411758
```

SUMMARY TABLE OF RESULTS (part (i) and (ii))

Model	**Validation accuracy (%)**	**Test accuracy (%)**	**Training time**
CVXOPT (Gaussian Kernel)	99.425	96.98	1.6 hours (approx)
LIBSVM (Gaussian Kernel)	97.7	97.09	195.21 sec

LIBSVM is significantly fast as compared to CVXOPT
**** Test time for CVXOPT – 1.8 hours (approx)*

(iii) Confusion Matrix of SVM using CVXOPT



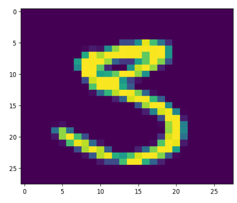
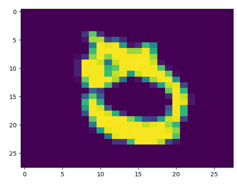
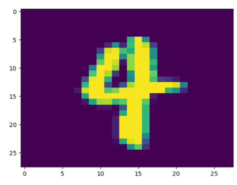
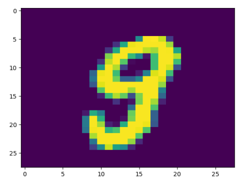
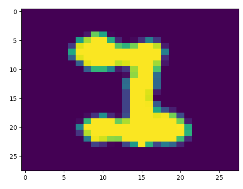
Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	969.0	0.0	1.0	0.0	0.0	3.0	3.0	1.0	3.0	0.0
1	0.0	1120.0	3.0	3.0	1.0	2.0	2.0	0.0	3.0	1.0
2	4.0	0.0	999.0	4.0	2.0	0.0	3.0	6.0	14.0	0.0
3	0.0	0.0	12.0	976.0	0.0	4.0	0.0	6.0	10.0	2.0
4	0.0	0.0	4.0	0.0	958.0	0.0	7.0	0.0	2.0	11.0
5	2.0	0.0	3.0	5.0	1.0	867.0	7.0	1.0	5.0	1.0
6	7.0	3.0	0.0	0.0	4.0	5.0	937.0	0.0	2.0	0.0
7	1.0	7.0	21.0	2.0	4.0	0.0	0.0	980.0	3.0	10.0
8	4.0	0.0	2.0	12.0	5.0	5.0	1.0	2.0	940.0	3.0
9	4.0	4.0	4.0	8.0	12.0	4.0	1.0	7.0	13.0	952.0

Observations:

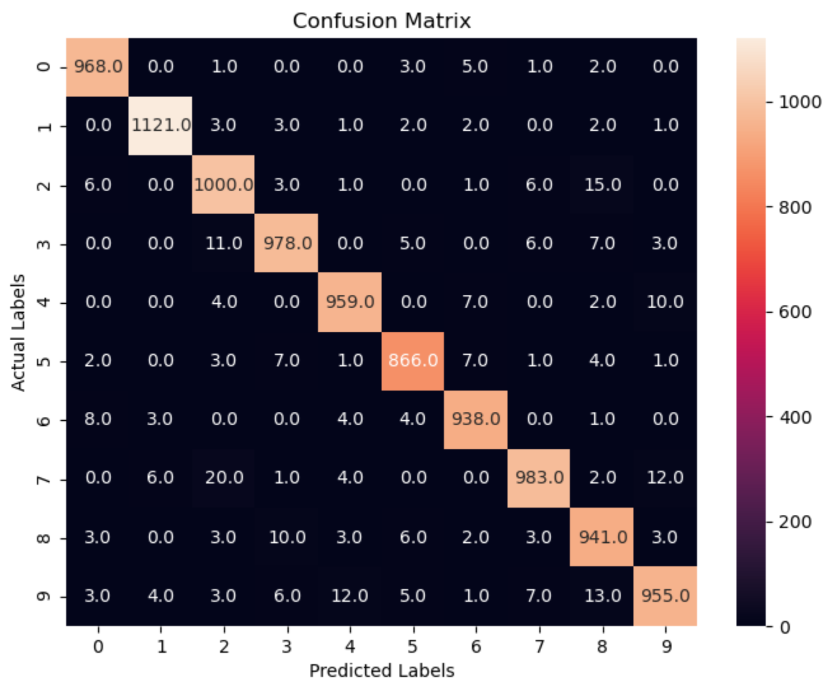
- **9 is often incorrectly predicted as 4 or 8.**
- **8 is often incorrectly predicted as 3.**
- **7 is often incorrectly predicted as 2.**
- **6 is often incorrectly predicted as 0, 5.**
- **5 is often incorrectly predicted as 3, 6.**
- **4 is often incorrectly predicted as 7,9.**

- **3 is often incorrectly predicted as 2, 8.**
- **2 is often incorrectly predicted as 8.**
- **1 is often incorrectly predicted as 2, 3 or 8. Although the support is low, so we can say that this is not significant.**
- **0 is often incorrectly predicted as 5, 6 and 8 although with low support.**

Some examples of misclassified digits using CVXOPT.
(Intuitive sense of above observations can be made through these examples)

Image	**Actual Label**	**Predicted Label**
	**3**	**5**
	**5**	**0**
	**4**	**9**
	**9**	**8**
	**1**	**2**

Confusion Matrix of SVM using LIBSVM

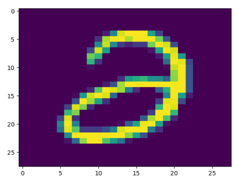
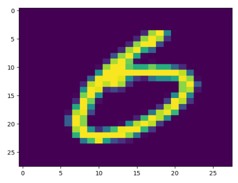
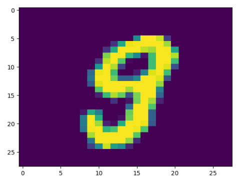
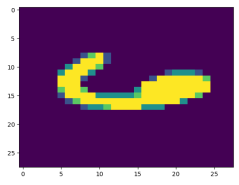
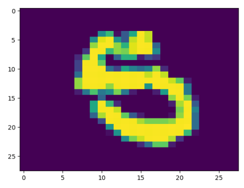


	0	1	2	3	4	5	6	7	8	9
0	968.0	0.0	1.0	0.0	0.0	3.0	5.0	1.0	2.0	0.0
1	0.0	1121.0	3.0	3.0	1.0	2.0	2.0	0.0	2.0	1.0
2	6.0	0.0	1000.0	3.0	1.0	0.0	1.0	6.0	15.0	0.0
3	0.0	0.0	11.0	978.0	0.0	5.0	0.0	6.0	7.0	3.0
4	0.0	0.0	4.0	0.0	959.0	0.0	7.0	0.0	2.0	10.0
5	2.0	0.0	3.0	7.0	1.0	866.0	7.0	1.0	4.0	1.0
6	8.0	3.0	0.0	0.0	4.0	4.0	938.0	0.0	1.0	0.0
7	0.0	6.0	20.0	1.0	4.0	0.0	0.0	983.0	2.0	12.0
8	3.0	0.0	3.0	10.0	3.0	6.0	2.0	3.0	941.0	3.0
9	3.0	4.0	3.0	6.0	12.0	5.0	1.0	7.0	13.0	955.0

Observations:

- **9 is often incorrectly predicted as 4 or 8.**
- **8 is often incorrectly predicted as 3 or 5.**
- **7 is often incorrectly predicted as 2 or 9.**
- **6 is often incorrectly predicted as 0.**
- **5 is often incorrectly predicted as 3, 6.**
- **4 is often incorrectly predicted as 9.**
- **3 is often incorrectly predicted as 2, 8.**
- **2 is often incorrectly predicted as 8.**
- **1 is often incorrectly predicted as 2, 3. Although the support is very low, so we can say that this is not significant.**
- **0 is often incorrectly predicted as 6.**

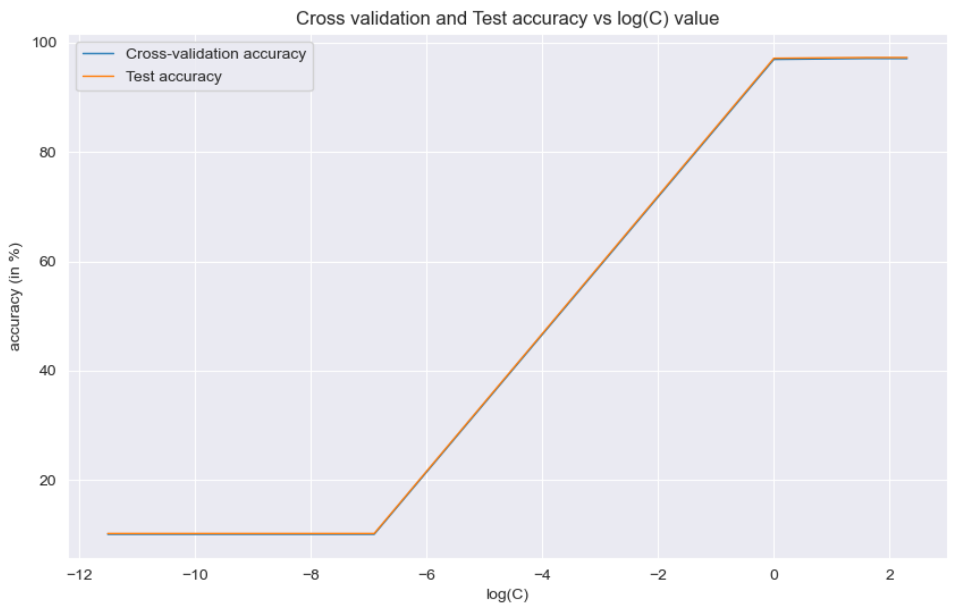
Some examples of misclassified digits using LIBSVM
(Intuitive sense of above observations can be made through these examples)

Image	**Actual Label**	**Predicted Label**
	**2**	**3**
	**6**	**0**
	**9**	**8**
	**6**	**4**
	**5**	**6**

(iv) Results of 5-fold cross validation using LIBSVM
***** Time taken for K-fold Cross Validation and testing simultaneously -** 8065.72 sec (approx. 2.24 hours)*

C	Cross-validation accuracy	Test Accuracy
10^{-5}	10.1438%	10.28%
10^{-3}	10.1438%	10.28%
1	97.0375%	97.09%
5	97.0625%	97.22%
10	97.08125%	97.22%

Plot of cross-validation accuracy and test accuracy vs log(C)



Observations:

1. C=10 gives the best cross validation and test set accuracy.
2. At C=5 and C=10, the test accuracy remains same, and possibly upon increasing C more, the test accuracy would start decreasing because it would force the slack variables to be smaller and move towards being like hard-margin SVMs.
3. Also we can see at very low values of C, slack variables can take higher values, therefore too much slack worsens the performance of the model on test and validation data significantly, maybe due to overfitting.