COL774 Assignment 2

Mustafa Chasmai | 2019CS10341

October 2021

1. **Text Classification**

(a) **Naive Bayes algorithm**

In similar terminology and notation used in class, the optimal parameters in the multi-class version of the Laplace smoothed Naive Bayes model are:

$$\phi_{k|y=c} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} 1\{x_j^{(i)} = k\}1\{y^{(i)} = c\} + \alpha}{\sum_{i=1}^{m} 1\{y^{(i)} = c\}n_i + \alpha|V|} \tag{1}$$

$$\phi_c = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = c\}}{m} \tag{2}$$

Intuitively, $\phi_{k|y=c}$ is simply the fraction of words in the reviews with label c, that are also the k^{th} word of the vocabulary. Similarly, ϕ_c is simply the fraction of reviews that have a label c. Thus, estimating these parameters from the input data is straight forward.

Accuracies

i. Training set: 52.616 %

ii. Test set: 50.971 %

(b) **Random and Majority classifiers**

i. **Random Guesses**

For each example in the training set, a value is chosen at random with a uniform probability (using numpy random choice) from the set of unique classes in the training dataset. For this classifier, each sample getting any label has a probability of 1/C, where C is the set of classes in the training set. Thus, the probability of the classifier predicting the correct label is also 1/C. Thus, the expected accuracy of this classifier is 100/C, which is 100/5 = 20%.

Accuracies

A. Training set: 20.214 %

B. Test set: 20.364 %

Thus, the Naive Bayes algorithm gives close to 30% improvement over the random classifier baseline.

ii. **Majority Guesses**

The labels of the training set are analysed and a single label with maximum samples in the training set is chosen. In the test set, this same majority label is predicted for each of the samples.

Accuracies

A. Training set: 51.864 %

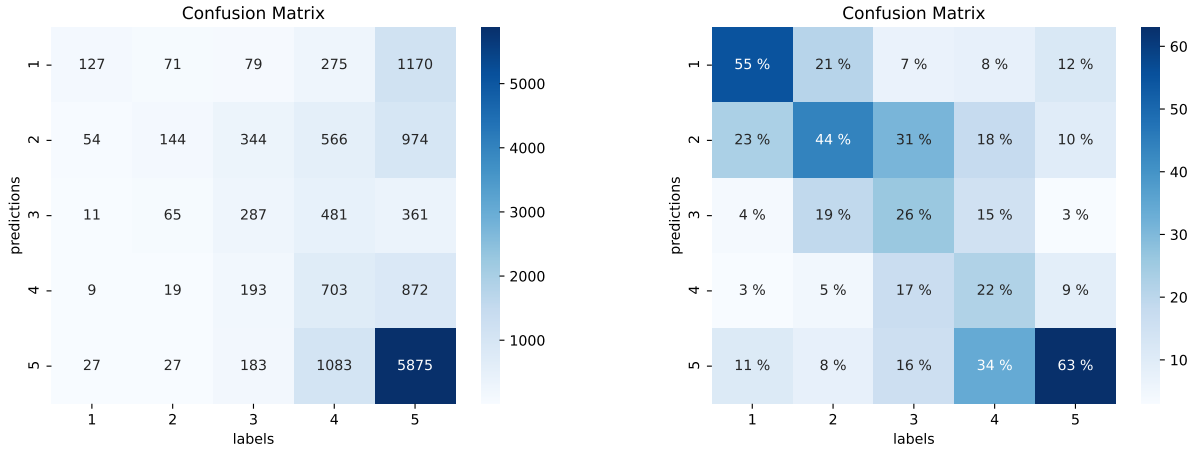
B. Test set: 66.085 %

Thus, the random classifier baseline gives close to 15% improvement on the test set, while the Naive Bayes algorithm gives close to 1% improvement on the training set.

Although the actual test accuracy here is better than that obtained from the Naive Bayes model, this may not be true for a general scenario. In this example, both the training set and test set have more than half of their examples belonging to one class only. If the data was not so highly imbalanced, the majority classifier would be giving lower accuracies.

(c) **Confusion Matrix**

The confusion matrix of the trained classifier on the test set can be seen in Fig 1



(a) True Confusion Matrix

(b) Normalised Confusion Matrix

Figure 1: Confusion matrices for the performance on the test set

Observations

i. The highest value of the diagonal entry corresponds to category 5. This means that this category has the maximum number of correct predictions. The non-diagonal elements correspond to wrong predictions, and as can be seen in Fig 1a, this category also has high non-diagonal elements (both in the row, and in the column). This means that many examples of this category are also misclassified, and that many other categories are actually misclassified as category 5. Thus, although it has the maximum correctly classified samples, it may not be the category with the best overall classification performance.

ii. From Fig 1a it may seem that the remaining classes have very poor performance. Fig 1b shows a different picture, with many other categories having high value as well. This plot shows the percentage of samples having a particular true label that have been predicted to have a particular category. Thus, in categories 1 through 4 as well, 22 to 55% of samples have been correctly predicted.

iii. From non diagonal entries, it can be seen that many samples with true label 4 get wrongly classified as category 5. This makes intuitive sense as both these samples would be highly positive reviews, and it may be difficult to differentiate between them. Many samples belonging to category 5 are also being wrongly classified as category 1 and category 2. This seems strange, since the model should have been able to differentiate them easily. This observation implies that the model could have learned better.

iv. Another observation is that the non diagonal values tend to be higher around the diagonal, rather than at the non-diagonal corners. This shows that there is confusion between consecutive categories. This makes intuitive sense since consecutive ratings would be expected to have similar reviews.

(d) **Stop-Word removal and Stemming**

Thess preprocessing steps was performed using the nltk package. A set of stop words commonly used in English was querried, and words in the input data corresponding to these were removed. On the filtered words, stemming was performed. Stop word removal reduced the vocabulary size from 219955 to 219808, stemming alone reduced it to 187980 and both together reduced the vocabulary size to 187935. The accuracies on the test set are:

Accuracies

i. Original: 50.971%

ii. Stop-Word removal: 49.178%

iii. Stemming: **54.11**%

iv. Stop-Word removal and Stemming: 53.021%

Thus, stemming on its own gives the best performance out of the four models. The confusion matrix obtained using both stemming and stop word removal is shown in Fig 2 below.

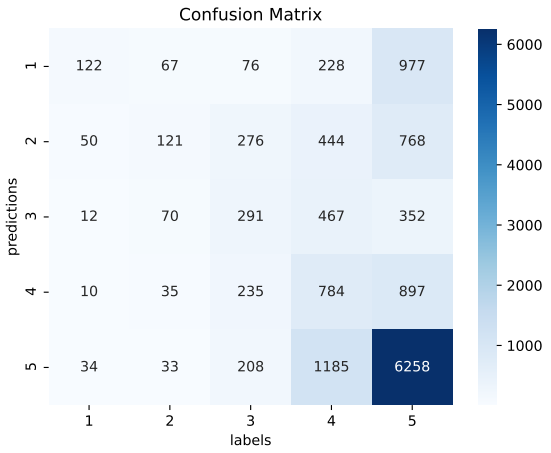


Figure 2: Confusion matrices with stemming

Comments

i. With only stop word removal, the performance decreases by approximately 1% over the original model. The fact that the performance changes implies that the stop words were also being used by the classifier. The fact that the performance decreased implies that the stop words actually helped the classifier differentiate between samples. This is probably because of the comparatively smaller size of the dataset, and the inability of the model to learn generalisable features (overfitting).

ii. With stemming, the performance increased by more than 3%. The change in performance implies that the model is sensitive to different forms of the same word. Increase in performance implies that the model tends to get confused in the presence of these forms of the same words. One possible reason is that the different forms of the same word are not independent conditionally over the class, i.e. they do not satisfy the Naive Bayes assumption. Since these samples are outside the modelling assumption, replacing them by the same word should increase the performance (as observed).

iii. Further using stemming after stop word removal led to a performance increase over the original, but a decrease over that of stemming alone. The drop in performance was very close to that in the case of applying stop word removal on the original model. This implies that the two methods are somewhat independent of each other, and further validates the previous two points.

(e) **Feature Engineering**

For feature engineering, two different methods were tried out, as described in what follows.

i. **Frequency thresholded bag of words**
There are some words in the text corpus that are not very common in english. These outlier words can affect the naive bayes model, even if they do not carry particular meaning in them.
Thus, this approach counts from the test sample only the words that have occurred atleast some threshold (tunable) number of times in the train data. It produces an accuracy of **63.307%** with a threshold of 5.
(Soft frequency thresholding)
Instead of a 0 if less than threshold, and 1 if more than, a different formulation is to use a smoother variation of 'weights' for each word in the vocabulary. However, the accuracy did not change significantly here.

ii. **Bigrams**
A bigram is simply a set of two consecutive words. Since two consecutive words more strongly confirm an idea compared to a single word, bigrams are expected to perform better. So, along with the normal words, the bigrams were also added in the vocabulary, and bigram features from the test set were compared against these to obtain better results. Bigrams improved the accuracy to 67.042%.

Thus, both the above features increased the accuracy over the vanilla bag of words method. While the first one reduces the effect of outliers, the second one provides a more contextual representation of the text. Bigrams provide much better results compared to the first features, meaning that there a better representation is more important than an outlier-free weaker representation.

(f) **F1-score**

Precision and recall are two commonly considered metrics for evaluating classification. While precision measures how many of the predicted values are actually correct, recall measures how many of the actual class were predicted correctly. In terms of true positives (TP), false positives (FP) and false negatives (FN), these two metrics are defined as:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{3}$$

In usual scenarios, a high precision leads to a lower recall, and vice versa. Thus, there is a complex tradeoff between the two metrics. Generally, evaluating two separate metrics is more complicated than a single one, and so, a new metric combining these two is commonly used. Thus the F1-score encodes both the precision and recall in it. The F1-score is formulated as:

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{4}$$

which is basically the harmonic mean of precision and recall. The class wise and macro F1 scores of the models previously considered are shown in the table below.

Method	Class 1	Class 2	Class 3	Class 4	Class 5	Macro	Accuracy
Vanilla Naive Bayes	0.13	0.12	0.25	0.28	0.71	0.30	50.971%
Random Classifier	0.03	0.04	0.12	0.22	0.30	0.14	20.364%
Majority Classifier	0	0	0	0	0.79	0.15	66.085%
Naive Bayes Stemming	0.14	0.12	0.25	0.30	0.73	0.31	54.114%
Frequency Threshold	0.21	0.13	0.25	0.33	0.79	0.34	63.307%
Bigrams	0.28	0.18	0.25	0.38	0.81	0.38	67.042%

Table 1: Class wise and macro F1 scores of different methods, along with their accuracies

It can be seen that the majority classifier has a very high accuracy, but a very low F1 score. The given dataset is highly imbalanced, and that is why the majority classifier obtains a very high accuracy. However, since macro F1 score contains metrics for each class in an equal footing, it would be a stricter and thus, better metric for this data. Thus, for this highly class imbalanced data, macro F1 score is a better metric.

(g) **Summary**

The summary is a more compact version of the review text. Thus, it is expected to give better performance. The accuracy with the summary alone was 14.771%, while that with both summary and review text was 54.707% (using only stemming with bag of words). Thus, there is an improvement of 0.7% in the latter case. The reason that the summary alone is not sufficient could be that the length of the summary is too less giving a vocabulary of only 14631 words, compared to 219955 in review text, and thus, it alone is not sufficient for the naive bayes model to classify correctly.

2. **MNIST Digit Classification**

(a) **Binary Classification**

i. **Linear Kernel**

The dual objective of SVM is:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}x^{(i)T}x^{(j)} - \sum_{j=1}^{m}\alpha_j \\ \text{s.t.} \quad & \alpha_i \leq C \qquad \forall i \in [1,m], \\ & -\alpha_i \leq 0 \qquad \forall i \in [1,m], \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned}$$

Clubbing the two inequality constraints together, a form identical to that required by cvxopt package can be observed. The matrices and vectors P, q, G, h, A, b are obtained as follows:

$$P = \begin{bmatrix} x_1^{(1)}y^{(1)} & x_1^{(2)}y^{(2)} & ... & x_1^{(m)}y^{(m)} \\ . & . & & . \\ . & . & & . \\ x_n^{(1)}y^{(1)} & x_n^{(2)}y^{(2)} & ... & x_n^{(m)}y^{(m)} \end{bmatrix} \begin{bmatrix} x_1^{(1)}y^{(1)} & x_1^{(2)}y^{(2)} & ... & x_1^{(m)}y^{(m)} \\ . & . & & . \\ . & . & & . \\ x_n^{(1)}y^{(1)} & x_n^{(2)}y^{(2)} & ... & x_n^{(m)}y^{(m)} \end{bmatrix}^T \qquad q = \begin{bmatrix} -1 & -1 & ... & -1 \end{bmatrix}^T \tag{5}$$

$$G = \begin{bmatrix} I_{m\times m} & -I_{m\times m} \end{bmatrix}^T \qquad h = \begin{bmatrix} C & ... & C & 0 & ... & 0 \end{bmatrix}^T \qquad A = y^T \qquad b = 0 \tag{6}$$

The SVM model was implemented using the above equations and CVXOPT package. With this, a total of 163 support vectors were obtained with the support vector being identified as the vector with corresponding $\alpha < 10^{-8}$. This small threshold was kept because almost no alphas were exactly 0, and alphas with such a less value can be neglected. The SVM obtained an accuracy of **100%** on the training data, and an accuracy of **98.84%** on the test data. The weight and bias are calculated using the equations,

$$w = \sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)} \qquad\qquad b = -\frac{1}{2}(\max_{i|y^{(i)}=-1} w^T x^{(i)} + \min_{i|y^{(i)}=1} w^T x^{(i)}) \tag{7}$$

ii. **Gaussian Kernel**

For the Gaussian kernel, simply the $x^{(i)T}x^{(j)}$ terms from the linear kernel get replaced by $\phi(x^{(i)})^T\phi(x^{(j)})$.

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}\phi(x^{(i)})^T\phi(x^{(j)}) - \sum_{j=1}^{m}\alpha_j \\ \text{s.t.} \quad & \alpha_i \leq C \qquad \forall i \in [1,m], \\ & -\alpha_i \leq 0 \qquad \forall i \in [1,m], \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned}$$

Thus, only the P matrix changes, while the other q, G, h, A and b matrices and vectors remain the same as in part (i) linear kernel. Now, it can be seen that $P_{ij} = y^{(i)}y^{(j)}e^{-\gamma|x^{(i)}-x^{(j)}|^2}$. The matrix form of this is given by the equation,

$$P = YY^T.(diag(XX^T, 0) + diag(XX^T, 1) - 2XX^T) \tag{8}$$

where diag(M, 0) takes the diagonal elements of matrix M and puts them along the rows, and then repeats them along the columns. diag(M, 1) does a similar thing, but on the columns instead of rows.
Using this formulation, the model was trained with CVXOPT package. This obtained an accuracy of **99.825%** on the training dataset and an accuracy of **98.615%** on the test dataset. These are very similar to the accuracies obtained using the linear kernel.

iii. **Comparision with LIBSVM**

The same dataset was used for learning a model of LIBSVM. The parameters used were: t 0 or 2 for linear or gaussian, c 1, g 0.05 for γ. The accuracies on the training and test sets, along with the time taken and the number of support vectors obtained are summarised in the table below.

Method	Kernel	Train Accuracy	Test Accuracy	Training Time	support vectors
CVXOPT	Linear	100%	98.84%	100.8s	163
LIBSVM	Linear	100%	99.03%	1.2s	158
CVXOPT	Gaussian	99.82%	98.61%	69.2s	848
LIBSVM	Gaussian	99.97%	99.58%	3.4s	877

Table 2: Accuracies, time and number of support vectors for different versions of SVMs.

As can be seen in the table above, LIBSVM and the implementation using CVXOPT both give very similar results. The accuracy differences for linear kernels on the training set is 0, while on the test set is only 0.2%. The accuracy differences for gaussian kernels on the training set is only 0.15%, while on the test set is 0.9%. The difference in the number of support vectors is 5 (3% of CVXOPT) in the linear case and 29 (≈3% of CVXOPT), which is quite small. On the other hand, the training time of the CVXOPT implementation is almost 10 that of LIBSVM in the case of linear, and more than 20 times in the case of gaussian kernels. Thus, even though the accuracies, weights and number of support vectors may be very close, the LIBSVM version is much more efficient compared to used CVXOPT implementation version.

(b) **Multi-Class Classification**

i. **One vs One Multiclass SVM**

Herer, the SVMClassifier class implemented in the previous part is reused directly, and the higher level functions of train and test are used directly, being abstracted away to the previous part. Thus, there are 45 SVMClassifiers, each having data of only two classes, and the labels being transformed to -1 and 1. The predictions of each of these 45 classifiers are then parsed during test time, and the votes are used to give the final class prediction of each sample in the test set.
The implementation led to a training accuracy of **99.995%** and a test accuracy of **99.780%**.

ii. **Multiclass SVM using LIBSVM**

The LIBSVM classifier gave accuracies of **99.920%** on the training set and **97.230%** on the test set. Thus, the performance of custom implementation using CVXOPT is slightly larger for the training set, and more the 2% higher for the test set. Thus, the LIBSVM may have more variance compared to the custom implementation, because of which it performs slightly poorer in the test set compared to the train set.
In terms of computational cost, again, there is a big difference here. While LIBSVM takes only 322s ($\approx$ 5 minutes), the custom implementation took 2993s ($\approx$ 50 minutes), which is again around a 10 times difference. This ratio for the multiclass classification stage is as expected, since both implementations use a one vs one method, and the binary classification of each pair of classes follows this ratio, as observed in the training times of the previous parts.

iii. **Confusion Matrix and Visualisation**

The respective confusion matrices are shown in Fig 3 below

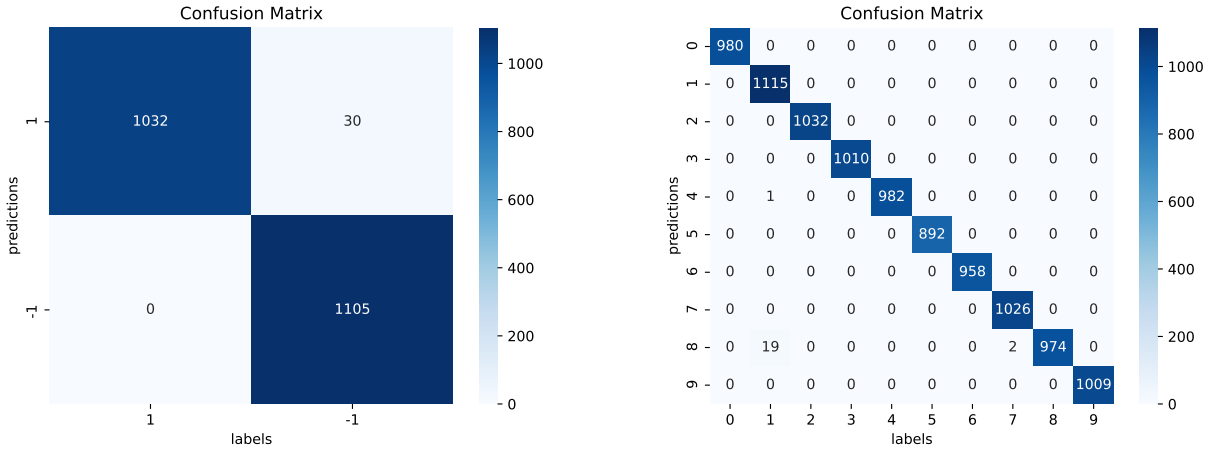
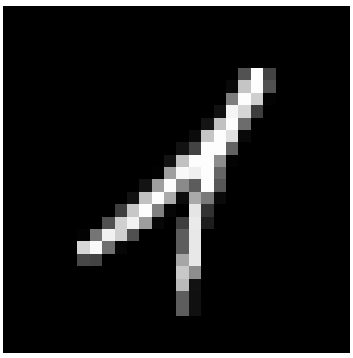


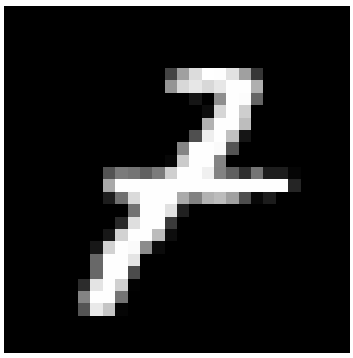
Figure 3: Confusion matrices for binary and multi-class SVM

As can be seen above, the digit 1 gets misclassified into the digit 8 most often, followed by the digit 7 being misclassified as 8 and 1 being misclassified as 4. Other than these, most of the samples are correctly classified, leading to a very high set of values in the diagonal of the confusion matrix.
In the Figure below, 10 samples from the test set that got misclassified are visualised as 28×28 pixel images, along with their true and predicted classes. The image processing library OpenCV was used to save these images in png image format.

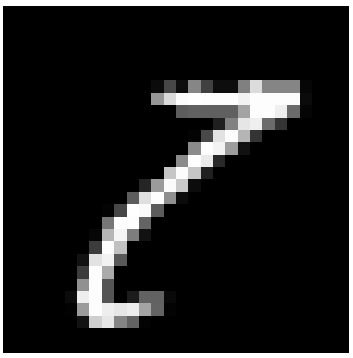
A majority of the misclassified samples are that where the true label is 1, but the classifier predicts 8, so this misclassification makes sense. Looking at Fig 4d, even a human may confuse it to be an 8. Although it is the case for this example, some other examples, like Fig 4j should have been easy to classify and are failure cases of the algorithm. Cases like Fig 4a and Fig 4c are also a bit ambiguous, and misclassification



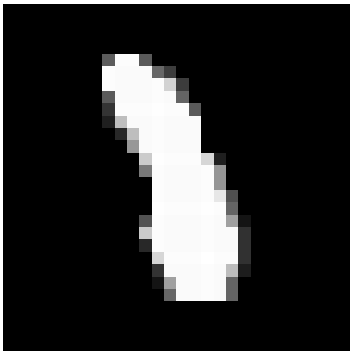
(a) True 1, predicted 4



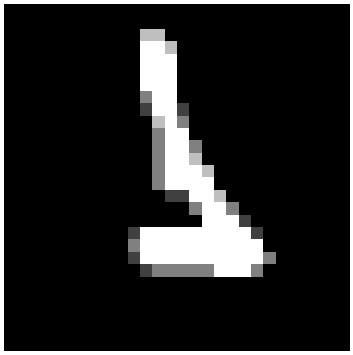
(b) True 7, predicted 8



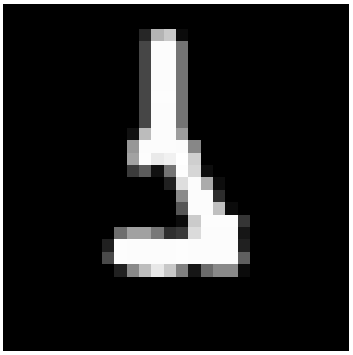
(c) True 7, predicted 8



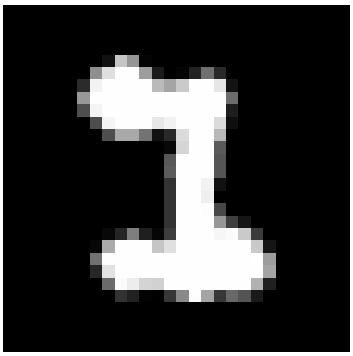
(d) True 1, predicted 8



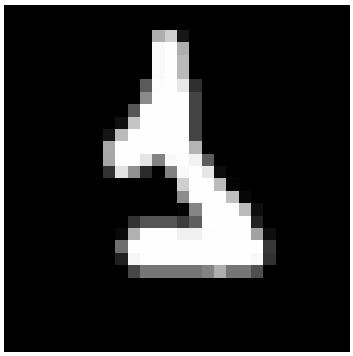
(e) True 1, predicted 8



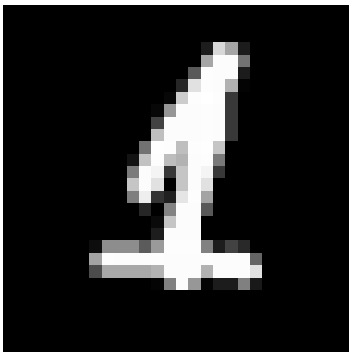
(f) True 1, predicted 8



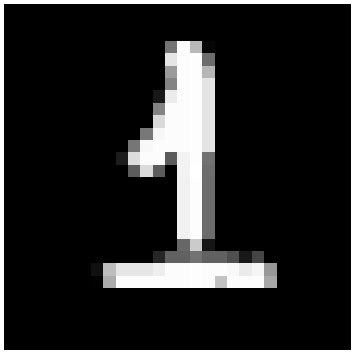
(g) True 1, predicted 8



(h) True 1, predicted 8



(i) True 1, predicted 8



(j) True 1, predicted 8

Figure 4: 10 examples of misclassified digits, along with their predicted and actual classes

in these examples makes complete sense.

iv. **k-fold cross validation**

The training data was initially randomly shuffled, and partitioned into k (5) equal parts. Taking k-1 parts for training, the LIBSVM model was evaluated on the remaining last part. Then this was rotated, to get k different folds, and k different accuracies, the average of which the the k-fold cross validation accuracy. Out of the k folds, the model having best performance was chosen, and tested upon the test set to get the test accuracy.

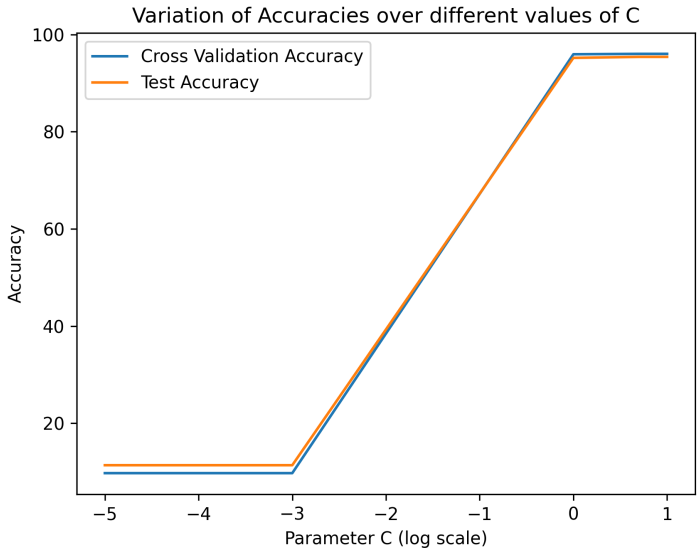


Figure 5: Accuracies of the validation and test sets for different values of C

The numerical values of these accuracies are as reported in the table below.

Value of C	Cross Validation accuracy	Test Accuracy
10^{-5}	11.35%	9.715%
10^{-3}	11.35%	9.715%
1	95.19%	95.945%
5	95.41%	96.03%
10	95.41%	96.03%

Table 3: Values of the 5 fold cross validation

Thus, the C values of 5 and 10 give the best cross validation accuracy, and these also give the best test accuracy. A clear increasing trend can be seen for both accuracies with increasing values of C. The accuracy at very low values of C is very poor compared to higher values of C. The jump from 10^{-3} to 1 is a big one, so the discrete change visible here, may look smoother if sampled at intermediate values.