COL774 – Machine Learning

Assignment – 2

Report

Submitted By:
Siddharth Gupta
Entry No. - 2021EE10627

Submitted To:
Parag Singla

Indian Institute of Technology, Delhi

Hauz Khas, New Delhi

Question – 1:

1. **Overview:**

   The problem is sentiment analysis of some Coronavirus tweets. We use Naïve Baye's Algorithm to classify the tweets into three sentiments – positive, negative, or neutral.

   Hereafter, I refer to tweet and sentiment as document and label respectively.

   - The vocabulary is formed using the training data only. In case, we come across a word in the validation set that is not included in the vocabulary, we skip over that word (i.e., this word does not affect the prediction)
   - The distribution of the labels in the generative process is taken to be multinoulli distribution.
   - Each position in a document is treated as a feature. This feature can take any one word from the vocabulary. The probability of any word occurring at a particular position is modelled using **multinoulli distribution**.
   - We implement the **bag-of-words model**. This model assumes that the parameters of the multinoulli distribution are same for every position.

   Different text preprocessing techniques are followed in different parts of the question.

2. **Assignment Questions:**

   a) In this part, no preprocessing is done on the text. The words from the document are extracted using the $split()$ method.

   The multinoulli distribution for y is define by three parameters:

   $$\Phi = [\phi_1 \quad \phi_2 \quad \phi_3]^T$$

   Each position is associated with three multinoulli distributions – one corresponding to each label. These distributions are assumed to be same for all the positions. Each distribution is defined by $|V|$ parameters where $|V|$ is the number of words in the vocabulary.

   Therefore, we have $3 * |V|$ parameters. These are represented as:

   $$\theta_{jl|y=k}$$

   where,

   $j = position\ index$

   $l = word\ index$

   $k = label$

The value of smoothening parameter α is kept 0.6 in this model.

After estimating the parameters using the results derived, we get the following results:

i. **Accuracy Data**

	Accuracy
Training Dataset	88.82 %
Validation Dataset	67.38 %

ii. **Word Clouds for Each Label**

I have removed some common words from the data while creating the word clouds. These words were the most frequent for every label. Therefore, to better visualise the frequent words unique to a particular sentiment, such words are removed.

Removed Words: variations of coronavirus and covid, single letters, will, t, co.

NOTE: Word Removal is NOT done while estimating the parameters. It is only done while creating word clouds.

The resulting word clouds are:

Fig: WC – Positive Tweets

Fig: WC-Negative Tweets

Fig: WC – Neutral Tweets

b) Some baseline methodologies to predict the label for each document are – randomly picking the label or labelling each document as positive. Accuracy on the validation set for each methodology is:

	Methodology	**Accuracy**
Naïve Baye's Model	Multinoulli Distribution	$67.38\ \%$
Baseline	Random Guess	$32.70\ \%$
	Always Positive	$43.85\ \%$

Our Naïve Baye's model gives significant improvement over the baseline models. Indeed, random guesses is a very naïve way to make predictions and gives an accuracy of about 1/3 (as expected). Always predicting the label to be positive is a completely wrong approach and is, as expected, extremely less accurate.

c) Confusion matrices are used to visualise the classification correctness of our model. Here, it will be a 3×3 matrix depicting the number of examples matched correctly/incorrectly for each label.
We can observe that in every confusion matrix, $(0, 0)^{th}$ diagonal entry is the largest. This means that our model is most successful in identifying positive tweets. Negative and neutral tweets are mis-identified more than positive tweets.

Naïve Baye's Model – Training

Actual / **Predicted**	Positive	Negative	Neutral
Positive	15738	668	196
Negative	771	13207	188
Neutral	1390	1018	4688

Naïve Baye's Model – Validation

Actual / **Predicted**	Positive	Negative	Neutral
Positive	1151	256	37
Negative	275	926	31
Neutral	292	183	142

Random Guess – Training

Actual / Predicted	Positive	Negative	Neutral
Positive	5541	5486	5575
Negative	4844	4712	4610
Neutral	2377	2388	2331

Random Guess – Validation

Actual / Predicted	Positive	Negative	Neutral
Positive	480	501	463
Negative	401	402	429
Neutral	213	209	195

Always Positive Label – Training

Actual / Predicted	Positive	Negative	Neutral
Positive	16602	0	0
Negative	14166	0	0
Neutral	7096	0	0

Always Positive Label – Validation

Actual / Predicted	Positive	Negative	Neutral
Positive	1444	0	0
Negative	1232	0	0
Neutral	617	0	0

d) **Text Pre-Processing**

i) In this part, text is pre-processed using multiple techniques.

1. Punctuation marks are replaced by a single space from the document.
2. The entire document is converted into lowercase.
3. Now, each word is lemmatised using $WordNetLemmatizer$ of $nltk$ library.
4. Stopwords are removed from the document. Along with the list of stopwords provided by $nltk$ library, words like $coronavirus, covid, covid19, http, https, will, t, co\ and\ english\ alphabet\ letters$ are also considered as stopwords.

Lemmatization converts a word into its base dictionary word. However, it ignores the part of speech the word belongs to. This results in some words to not be correctly changed to their base form.

Hyperlink text is not removed from the document. This also introduces noise into the document and hence reduces the accuracy.

The value of the smoothening parameter α is taken to be 0.4 in this model.

ii) Word clouds formed after the preprocessing are:

Fig: WC – Positive Tweets

Fig: WC-Negative Tweets

Fig: WC – Neutral Tweets

iii) Accuracy on the validation set of the new data is –

	Accuracy
Training Dataset	89.15 %
Validation Dataset	69.11 %

iv) The accuracy of the model increases after lemmatisation. However, the increase in the accuracy is marginal. This is because of the lack of presence of words which can be transformed into same root words. This is because the tweets are typed poorly and have a lot of spelling mistakes and grammatical errors.

Another reason for this is that lemmatisation is not very good in transforming words when it does not know which part of speech the word belongs to.

e) Feature Engineering

i) Along with using each position as a feature, we can also consider two consecutive positions as a feature. Such features are called bi-grams. The changes which need to be done to our basic implementation are:

- Consider consecutive words while creating the vocabulary.
- While computing $p(x|y)$ during inference, in addition to doing it for every word position, we have to do it for consecutive word positions also.

Rest of the implementation remains the same.

The value of the smoothening parameter α is taken to be 1.2 in this model.

The accuracy of this new model is:

	Accuracy
Training Dataset	89.97 %
Validation Dataset	65.29 %

We see that the training accuracy has increased while validation accuracy has decreased. This implies that adding bigrams as features results is overfitting of the model to the training data.

ii) Similar to the previous case, I now use 3-gram as features. This means that in addition to single and consecutive words positions as features, I also take 3 consecutive word positions as features. Intuitively, this results in learning of features common to a particular sentiment.

The value of the smoothening parameter α is taken to be 0.8 in this model.

The new accuracies are:

	Accuracy
Training Dataset	97.77 %
Validation Dataset	65.44 %

We can see that though this feature addition results in overfitting, it also increases the validation accuracy slightly. This indicates that this is a better feature than bigrams.

iii) For the three different feature schemes, we get:

Features	**Training Accuracy**	**Validation Accuracy**
Unigrams	89.15 %	69.11 %
Bigrams	89.97 %	65.29 %
3-grams	97.77 %	65.44 %

We conclude that:

- Bigrams are not a good feature in this scenario. Since, the data is extremely noisy, bigrams result in overfitting of the model.
- 3-grams improve the validation accuracy as compared to bigrams. However, this feature also results in overfitting.

Therefore, I conclude that for this dataset, unigrams are the best feature. N-grams results in overfitting of the model. Among N-grams, 3-grams are better than 2-grams. This is probably because that 3-grams capture the presence of phrases in the data (too long to be captured in 2-grams).

However, since the data requires much more pre-processing than we do, no feature engineering or basic text pre-processing results in a significant change in validation accuracy.

f) **Domain Adaptation**

i) In this part, we use a technique called Domain Adaptation. We have two data sets available – coronavirus tweets dataset (source domain) and general-purpose tweets dataset (target domain).
We train our Naïve Baye's model on the entire coronavirus tweets dataset and some percentage of general-purpose tweets dataset.
We then use the general-purpose tweets as the validation set.
The validation set accuracy using this approach comes out to be:

Target Dataset Used	**Validation Set Accuracy**
1%	41.91 %
2%	42.80 %
5%	43.50 %
10%	44.20 %
25%	46.42 %
50%	49.90 %
100%	54.10 %

ii) We now learn the Naïve Baye's Model only on a small part of target dataset. This model has two effects:

- The training dataset is quite small now.
- However, some of the noise introduced by the source dataset is removed.

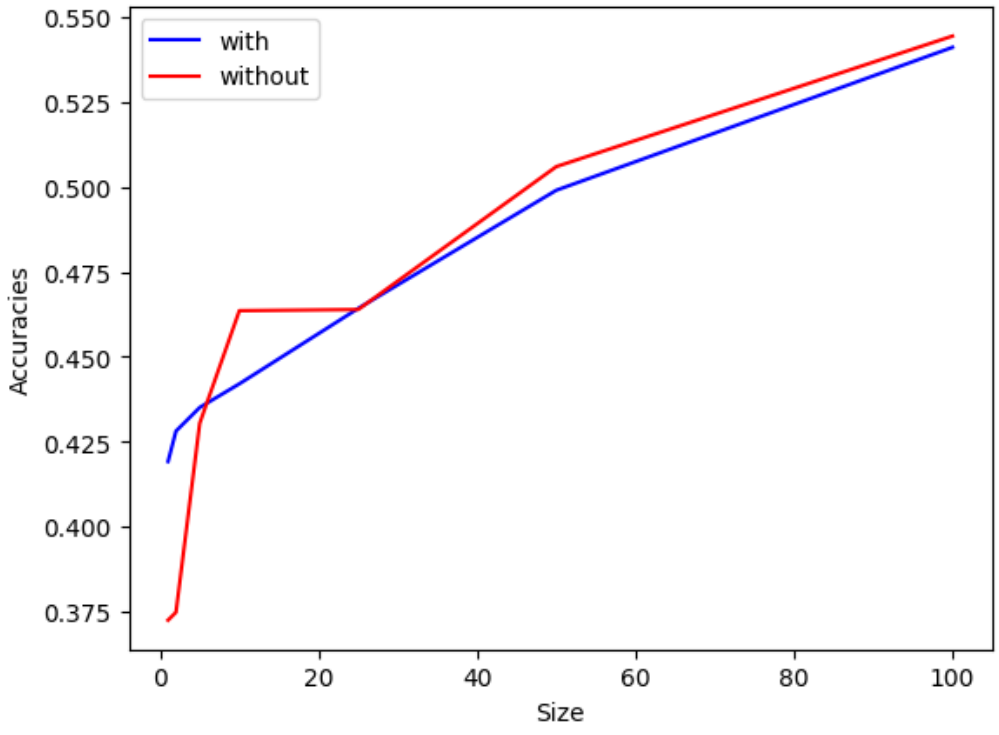
Using this approach, the accuracy on the validation set obtained are:

Target Dataset Used	Validation Set Accuracy
1%	37.24 %
2%	37.47 %
5%	43.04 %
10%	46.35 %
25%	46.38 %
50%	50.59 %
100%	54.44 %

iii) Analysing both the approaches:

Target Dataset Used	Accuracy Without Source Domain	Accuracy With Source Domain
1%	37.24 %	41.91 %
2%	37.47 %	42.80 %
5%	43.04 %	43.50 %
10%	46.35 %	44.20 %
25%	46.38 %	46.42 %
50%	50.59 %	49.90 %
100%	54.44 %	54.10 %

We can visualize the same graphically:



From the table and the graph, we conclude the following:

- With increasing size of target training data, the validation accuracy increases. This is to be expected since with more target data, the parameters are more tailored to the target domain data.
- When the target training data is less, the validation accuracy is more when we use the source data. This is because with less target training data and no source data, the model is underfit to the data. There is not enough data to learn the parameters sufficiently.
- When the target training data is more, the validation accuracy is more when we DO NOT use the source data. This is because the target training data is sufficient to estimate the parameters without the fear of underfitting. The source data now introduces noise and bias in the training data.

Therefore, we verify that Domain Adaptation is a useful machine learning technique when we do not have sufficient training data for the target domain.

However, if we have sufficient training data, then it is prudent to use only the target training data. Source data does more harm than good in this case.

Question – 2: Binary Classification

1. **Overview:**

This problem is of image classification. We use Support Vector Machines (SVMs) for classification. SVMs are discriminative models which find the separating hyperplane using the notion of functional margins. SVMs do not compute any probabilities and finally uses a small subset of training examples, called support vectors, to find the separating hyperplane.

For optimising the dual objective derived from the primal objective problem, we use two methods:

a. CVXOPT is a general-purpose package for solving convex constrained optimization problems.
b. LIBSVM is an algorithm to optimise convex constrained optimisation problems. $sklearn.svm.SVC()$ function of $scikit\ learn$ library optimises the SVM dual object using this algorithm.

We experiment with both linear and gaussian kernels. These kernels are given by:

$$K_{linear}(x, z) = x^T z$$
$$K_{gaussian}(x, z) = e^{-\gamma * ||x-z||^2}$$

2. Assignment Questions:

a) CVXOPT demands a particular form of the optimisation problem. This is:

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

$subject\ to:$

$$G\alpha \leq h$$

$$A\alpha = b$$

The dual optimisation problem of SVMs using linear kernel is:

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

$subject\ to:$

$$\alpha_i \leq C$$
$$-\alpha_i \leq 0$$
$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Manipulating the above expressions, we get:

$$P := P_{ij} = y^{(i)} * y^{(j)} * K\left(x^{(i)}, x^{(j)}\right)$$
$$b = 0$$
$$A = [y^{(1)} \quad y^{(2)} \quad \dots \quad y^{(m)}]$$
$$h = \begin{bmatrix} C \\ \vdots \\ C \end{bmatrix}_{m\times 1} \ and \ \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{m\times 1} \ stacked\ vertically$$
$$G = I_{m\times m} \ and \ -I_{m\times m} \ stacked\ vertically$$
$$q = \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix}_{m\times 1}$$

We fix the hyper-parameter C to be 1.0 in this question. We also need to set a threshold below which α_i is considered to be zero and another threshold above which α_i is considered to be 1. I keep this as:

$$\alpha_i = 0 \ for \ \alpha_i < 5 \times 10^{-9}$$
$$\alpha_i = 1 \ for \ \alpha_i > (1 - 10^{-10})$$

With this setting, CVXOPT optimises the α parameters. The results are:

i) Support vectors are those training examples corresponding to which $0 < \alpha_i < C$.
With this condition, the number of support vectors come out to be:

$$nSV = 770$$
$$\%\ SV = 16.17\ \%$$

ii) From the primal and dual objectives, we get:

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$b = \frac{1}{nSV}\left(\sum_{i \in SV} \left(y^{(i)} - w^T x^{(i)}\right)\right)$$

where,

$nSV = \#\ Support\ Vectors$

Since, while using CVXOPT, no α_i actually comes out to be exactly zero, I use all the examples to calculate w and b

Then,

$$b = \frac{1}{m}\left(\sum_{i=1}^{m}\left(y^{(i)} - w^T x^{(i)}\right)\right) = \frac{1}{m} \times \boldsymbol{Y} \times K(\boldsymbol{X}, \boldsymbol{X})$$

where,

$Y_{ij} = y^{(i)} y^{(j)}$

$K(\boldsymbol{X}, \boldsymbol{X}) = Kernel\ Matrix\ On\ Training\ Data$

Then, the final values are:

$$b = 3.8207$$

w is a vector of length 768. Hence, I omit to write it here. It can be seen in the code.

After getting w and b, prediction for feature x is given as:

$$\hat{y} = sign(w^T x + b)$$

Then, the accuracy comes out to be:

Data	**Accuracy**
Training Set	97.37 %
Validation Set	92 %

iii) After optimisation, we get the α vector. The examples whose corresponding α_i is large are the support vectors. The images for which $0 < a_i \leq C$, lie either on the margin boundary or between the two margin boundaries. Therefore, the predictions for these images are the least confident.

The $16 \times 16 \times 3$ images corresponding to 6 largest α_i correspond to the 6 weakest predictions. These images are as shown below:

Fig: Smallest α_i

Fig: 2nd Smallest α_i

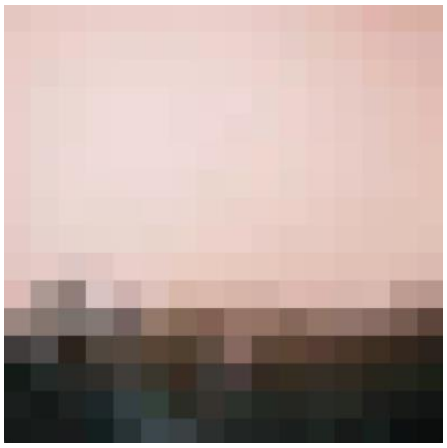


Fig: 3rd Smallest α_i

Fig: 4th Smallest α_i

Fig: 5th Smallest α_i

Fig: 6th Smallest α_i

We also make an image from the w vector obtained after optimisation. Since the values in this vector can exceed 1 (and therefore the de-normalised value will exceed 255), we first make a linear map of the vector such that all values are between 0 and 1. The physical significance of this image is as – the element of the w vector with large values indicate that the corresponding feature is more important for classifications. This means that the larger values of w correspond to the pixels which are more important for classification. This means that these pixels are different in separate classes.

Since larger RGB values correspond to a brighter pixel, we conclude that the parts of the image with brighter pixels are more important to classify the image than the rest of the image.

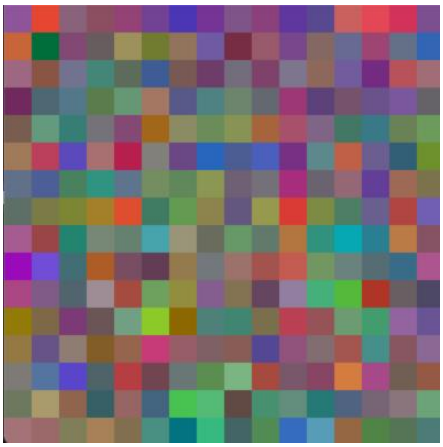


Fig: Image formed using 'w'

b) We now use a Gaussian Kernel instead of a Linear Kernel. The only difference is that we replace $(x^{(i)})^T x^{(j)}$ with $K(x^{(i)}, x^{(j)})$ where –

$$K(x, z) = e^{-\gamma * ||x-z||^2}$$

i) With this change, the new model results in:

$$nSV = 1898$$

$$\%\, nSV = 39.87\ \%$$

$$\#\ SVs\ common\ with\ linear\ case = 735$$

We observe that the number of support vectors required by the Gaussian Kernel is more than that required by Linear Kernel.

ii) The new accuracies are:

Data	**Accuracy**
Training Set	95.02 %
Validation Set	93.75 %

iii) The significance of the below images is the same as that for the images of linear kernel. The images with larger values of α_i lie on the margin boundary or between the 2 margin boundaries.
The $16 \times 16 \times 3$ images corresponding to 6 largest α_i are shown below:

Fig: Smallest α_i

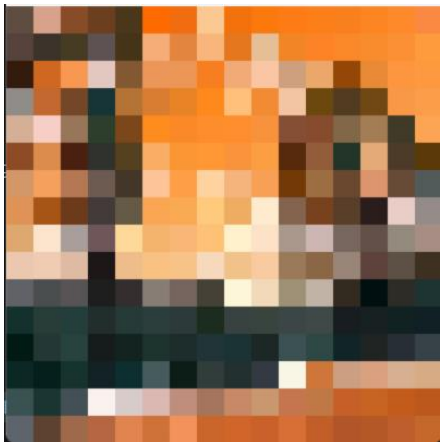


Fig: 2nd Smallest α_i

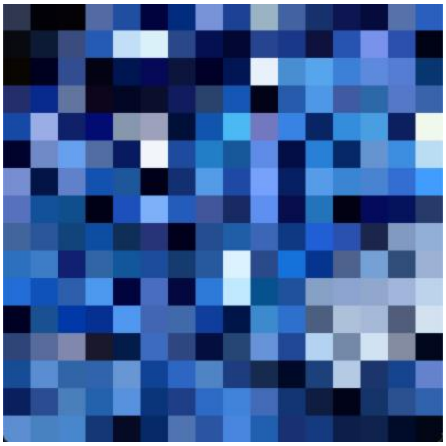


Fig: 3rd Smallest α_i

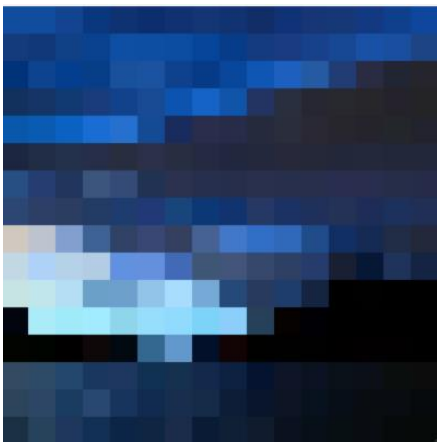


Fig: 4th Smallest α_i

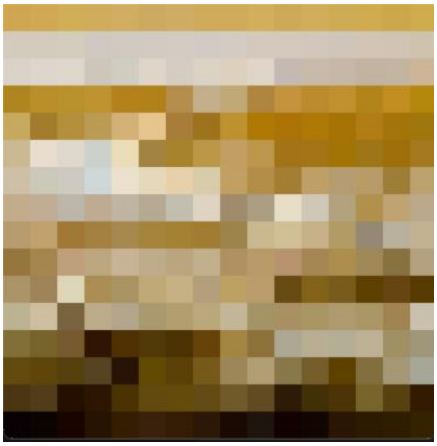


Fig: 5th Smallest α_i

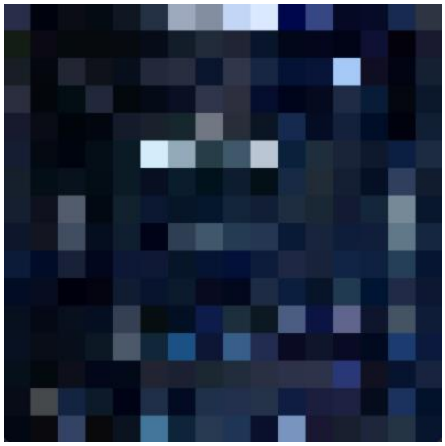


Fig: 6th Smallest α_i

iv) The accuracy for both validation and training set for Gaussian Kernel SVM is less than that of Linear Kernel SVM.
This implies that for the given set of images, Linear Kernel is a better choice for the SVM and the features corresponding to Linear Kernel are more suited to the problem that the ones corresponding to the Gaussian Kernel.

c) We now use $sklearn.svm.SVC()$ function to implement the SVMs.
This results in a much more concise implementation since most of the computations are done under-the-hood.

i) Support Vectors are now given by the $sklearn.svm.SVC()$ itself. Since we are not using any threshold here to determine the support vectors, the number of support vectors we get from here is much more accurate than those we found while using CVXOPT.
The number of support vectors found here are:

	No. of SVs
Linear Kernel	668
Gaussian Kernel	1085

The number of support vectors obtained using different models and the number of support vectors common to different pairs is summarised used the below table:

	CVXOPT	Scikit Learn	Common
Linear Kernel	770	668	668
Gaussian Kernel	1898	1085	1085
Common	735	558	

We make the following conclusions from this data:

- The number of support vectors required by gaussian kernel is more than that required by linear kernel.
- The number of support vectors required by CVXOPT is more than that required by Scikit Learn
- Since the number of support vectors common to CVXOPT and Scikit Learn models for a particular kernel is the same as that required by Scikit Learn model, we conclude that CVXOPT uses some extra support vectors which are probably redundant since they have been determined using a threshold fixed by me.

ii) The optimal bias obtained from here is:

$$b = 4.13$$

The w vector obtained using $scikit - learn$ is almost same as the one obtained using $CVXOPT$. The minute difference is that the absolute value of every element is slightly larger for w obtained using $CVXOPT$

iii) The accuracies in this case are:

	Training Accuracy	**Validation Accuracy**
Linear Kernel	$97.37\ \%$	92%
Gaussian Kernel	$95.12\ \%$	$93.75\ \%$

iv) The computational time required by scikit learn models is significantly less than that required by CVXOPT models. This is probably due to the fact that a lot of computations for the CVXOPT model are implemented by me and hence inefficient. The scikit learn models do most of the computations under the hood and probably uses better programming paradigms (like parallelization of operations etc.).

The computation times for training the different models are:

	CVXOPT	**Scikit Learn**
Linear Kernel	1 min 38 sec	4.2 sec
Gaussian Kernel	2 min 1.7 sec	7.6 sec

We also observe that the time taken for gaussian kernel models is more. This is because the computation of kernel matrix for gaussian kernel is more expensive than that for linear kernel.

Question – 2: Multi-Class Classification

1. Overview:

This problem is based on multi-class image classification.

We follow one-vs-one (OvO) scheme to classify the images. In this method, we first learn a classifier for each pair of classes (using the training data for those two classes only). Therefore, we get $\binom{k}{2}$ classifiers, where k is the number of classes.

For prediction, we get the predictions on the test dataset from each of the $\binom{k}{2}$ classifiers. Each test example has now $\binom{k}{2}$ labels, one from each classifier. The final label for each image is the one which occurs the maximum number of times.

This algorithm is extremely expensive since we are learning $\binom{k}{2}$ SVM models. The predictions also take a lot of time, again for the same reason.

However, it is fairly powerful and, therefore, used.

2. Assignment Questions:

a) We use CVXOPT here for multi-class classification. We separately train all 15 classifiers using the training data. Then, we pass the entire validation data through each of the classifiers. For any example x, the predicted label is the one which is predicted by the maximum number of classifiers.

With this technique, the accuracy is:

	Accuracy
Training Data	56.65 %
Validation Data	55.66 %

b) We now use $sklearn.svm.SVC()$ for classification. The function automatically handles multi-class classification. The default technique is one-vs-one, and we keep it as such.
The accuracy obtained here is:

	Accuracy
Training Data	$56.84\ \%$
Validation Data	$55.91\ \%$

The validation accuracy for the models implemented using scikit learn is slightly more than that for models implemented using CVXOPT. This is probably due to better selection of support vectors and more efficient computations.
The computational times for both the approaches are:

	CVXOPT	**Scikit Learn**
Gaussian Kernel	20 min 51 sec	3 min 37 sec

We again observe that the time taken by scikit learn models is significantly less compared to CVXOPT models.

c) We use $sklearn.metric.confusion\_matrix()$ function to create the confusion matrices.
The confusion matrices are as follows:

Confusion Matrix for Model Learnt using CVXOPT

Actual / **Predicted**	0	1	2	3	4	5
0	76	21	21	29	20	33
1	4	150	1	6	13	26
2	11	4	125	27	21	12
3	24	6	25	128	12	5
4	18	17	58	38	63	6
5	25	22	11	8	8	126

The number of wrong predictions for each class is summarised below:

Class	0	1	2	3	4	5
Wrong Predictions	82	70	116	108	74	82

We observe that classes 2 and 3 are the most mis-labelled ones.

- Class 2 images are most often mis-labelled as Class 4 images.
- Class 3 images are most often mis-labelled as Class 4 images.

Confusion Matrix for Model Learnt using Scikit Learn

Predicted \ Actual	0	1	2	3	4	5
0	76	22	18	26	24	34
1	4	150	1	6	12	27
2	11	4	125	26	22	12
3	24	6	25	126	14	5
4	18	17	58	33	68	6
5	25	23	10	8	8	126

The number of wrong predictions for each class is summarised below:

Class	0	1	2	3	4	5
Wrong Predictions	82	72	112	99	80	84

We observe that classes 2 and 3 are still the most mis-labelled ones.

- Class 2 images are most often mis-labelled as Class 4 images.
- Class 3 images are most often mis-labelled as Class 4 images.

Another frequent mis-labelling here is that Class 5 images are mis-labelled as Class 0 images.

From the above confusion matrices, we conclude that:

- Class 2 images are very similar to Class 4 images.
- Class 3 images are very similar to Class 4 images.
- Class 5 images are very similar to Class 0 images.

We show 12 mis-labelled images below. We indicate the true class of the image as well as the class of the image predicted by the SVM.

The 12 images are:

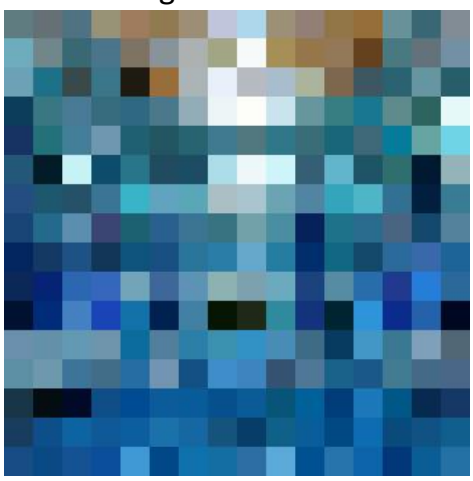


Fig: Actual-0, Predicted-1

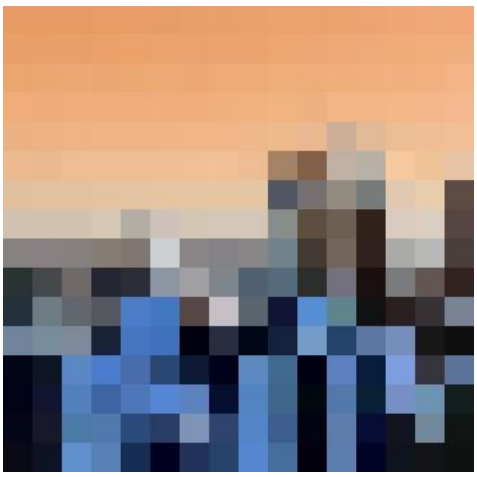


Fig: Actual-0, Predicted-3

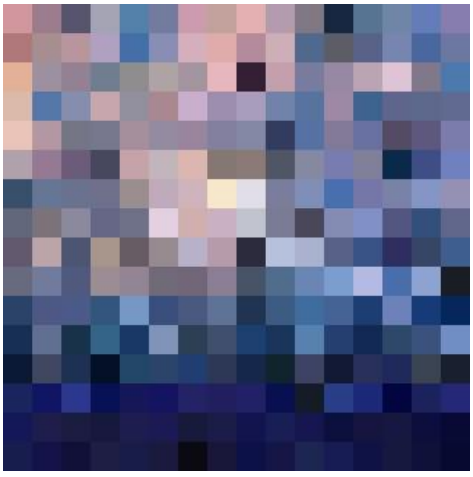


Fig: Actual-1, Predicted-4

Fig: Actual-2, Predicted-4

Fig: Actual-2, Predicted-5

Fig: Actual-3, Predicted-0

Fig: Actual-3, Predicted-5

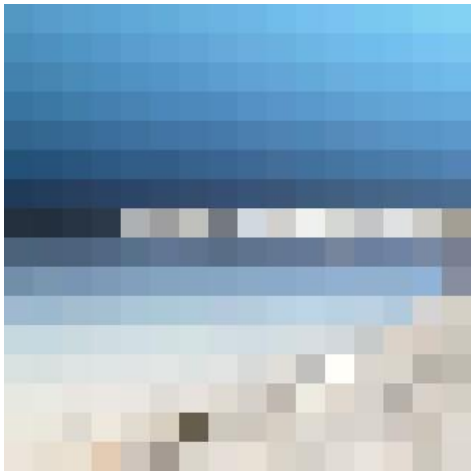


Fig: Actual-4, Predicted-3

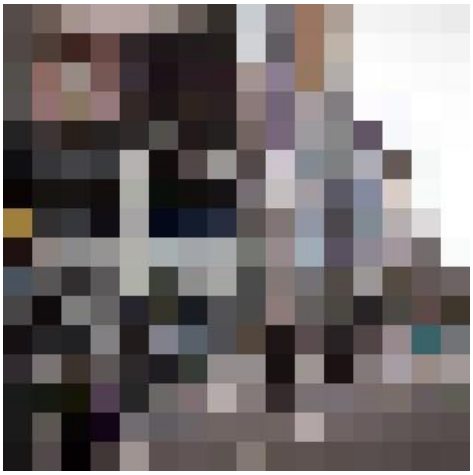


Fig: Actual-5, Predicted-0

Fig: Actual-5, Predicted-1

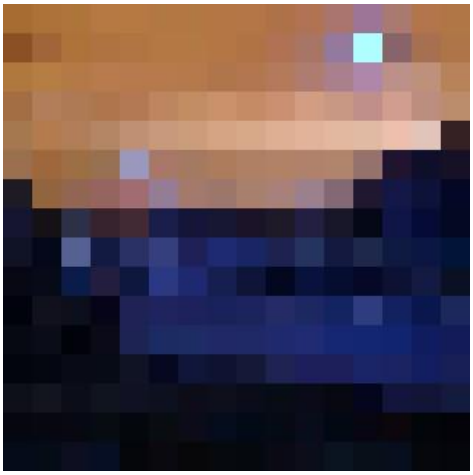


Fig: Actual-5, Predicted-3

Fig: Actual-5, Predicted-4

d) We implement 5-fold cross validation technique here to observe the change in validation accuracy as we change the hyper-parameter C.
We divide the data into 5 parts. Each part is considered to be the validation set once and the rest of the data is considered to be the training set. Therefore, we will train the SVM 5 times. The accuracy on the validation set obtained from these 5 models is averaged to get the 5-fold cross validation accuracy for a particular value of C.

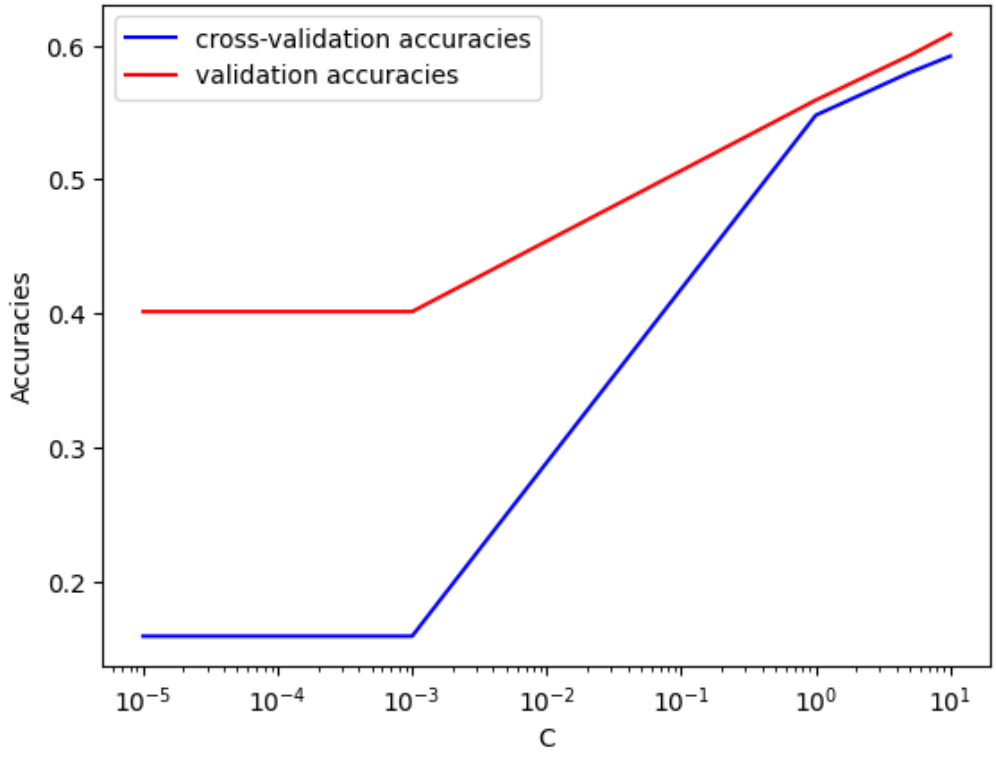
The accuracies obtained are:

C	5-Fold Cross Validation Accuracy
10^{-5}	15.98 %
10^{-3}	15.98 %
1	54.78 %
5	57.97 %
10	59.19 %

To better understand the behaviour of accuracy with change in parameter C, we also train our model on the whole training set for different values of C and find the validation accuracy for each model.
The accuracies obtained are:

C	Validation Accuracy
10^{-5}	40.16 %
10^{-3}	40.16 %
1	55.91 %
5	59.25 %
10	60.83 %

We show the plot for these variations for better visualisation:



We observe that larger C results in better 5-fold cross-validation accuracy as well as better validation accuracy. This is because larger C implies that we are giving more importance to the slack variables.
Another observation is that for $C = 10^{-5}\ and\ C = 10^{-3}$, there is absolutely no difference in the 5-fold cross-validation accuracy as well as validation accuracy. This implies that at small values of C, the slack variables do not affect the training of the model much.