ELL-409 – Assignment-2

Name: Guruvu Surya Sai Prakash

Entry No: 2019EE10481

Part-1A

1)

First, I got Familiarised with LIBSVM for python and convex optimization package CVXOPT of python.

2)

Binary Classification:

a)

First, I chose **labels 2.0 and 5.0** for Binary Classification and took all the 25 feature vectors for classification.

For this, I have done classification using **Linear Kernel and RBF Kernel.**

I have solved the problem using both **LIBSVM and CVXOPT** for each of the kernels.

I have used the **C-Support Vector Classification** in all parts. Here we generally need to solve the following primal optimization problem.

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, l, \end{aligned}$$

Where xi 's are training vectors and yi's are the labels and ϕ(xi) maps xi into a higher-dimensional space and C > 0 is the regularization parameter.

Due to the possible high dimensionality of the vector variable w, usually we solve the following dual problem.

Equations-1:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \\ & 0 \leq \alpha_i \leq C, \qquad i = 1, \ldots, l, \end{aligned}$$

where **e = [1, . . ., 1]** ^{T} is the vector of all ones, Q is an l-by-l positive semidefinite matrix,

Qij ≡ yi*yj*K (xi, xj), and K (xi, xj) ≡ ϕ(xi) ^{T} ϕ (xj) is the kernel function.

After problem is solved, using the primal-dual relationship, the optimal w satisfies,

$$\boldsymbol{w} = \sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i)$$

The value of b can be calculated as

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

Where S represent the support vectors. Ns represent total number of support vectors.

The decision function is:

$$\operatorname{sgn}\left(\boldsymbol{w}^T \phi(\boldsymbol{x}) + b\right) = \operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\right).$$

I have used the same method to predict the output while solving using CVXOPT.

Using Linear Kernel:

Total no of samples having labels 2.0 and 5.0 are 577.

In this, I split the such that first 177 samples for training and the other 300 samples for validation.

Now, first I used all the 25 feature vectors for classification.

Using this data, I trained an SVM.

Using LIBSVM:

First, create a svm problem using **'svm_problem'.** In case of Linear kernel, we have only one Hyper parameter. So, I need to tune the value C. for, that first I trained the svm using the training data using **'svm_train'** function for different values of C and predicted the labels of validation data and found the accuracy of the model using **'svm_predict'** function.

First, I have done coarse tuning by changing value of C from 2^-20 to 2^20. Later, I did the fine tuning, depending on the value of C.

Plot of Accuracy vs $\log_2$(C):

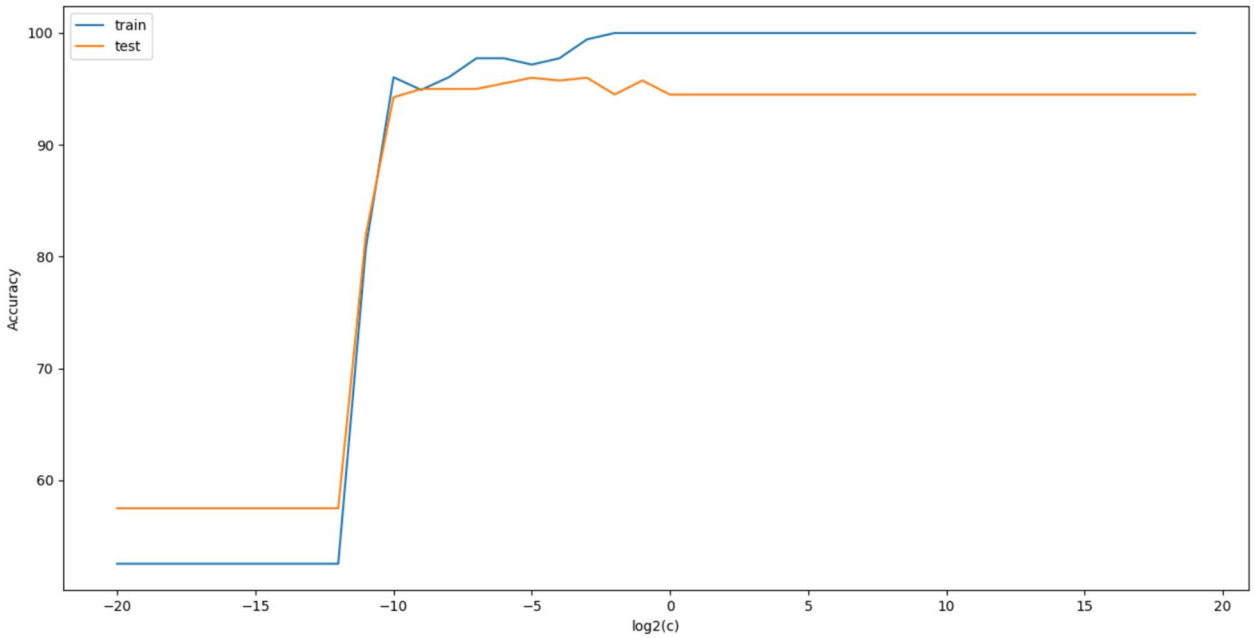
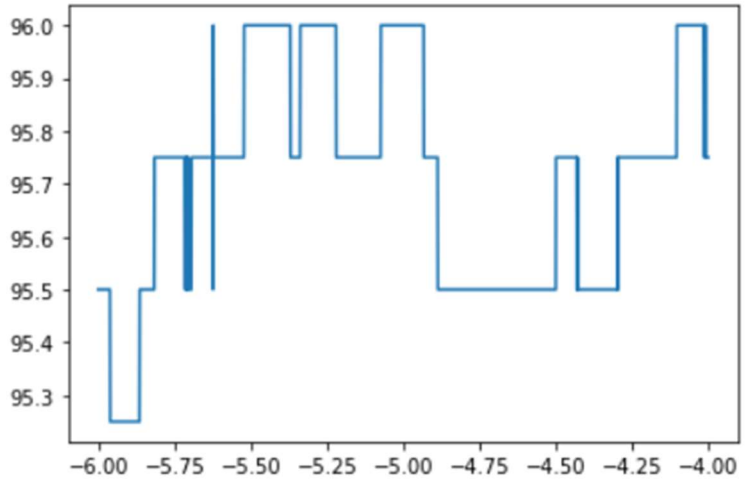


Figure-1

We could see from the above plot that we get a best fit for $\log_2$(C)= -5, and for $\log_2$(C)< 10 it was under fitting and for $\log_2$(C) > 5 it was kind of over-fitting.

Now, after this I have done fin tuning and we could see the plot below.

Plot of Accuracy vs $\log_2$(C) for testing data:



From, the plot we have many values which the accuracy is the same and maximum. But we consider the smallest value of C as it gives a simpler model. So, the value of C turns out to be -5.623. The accuracy at this value of C is 96%.

Using CVXOPT:

Since, I am solving using C-SVC, we need to solve quadratic programming in CVXOPT.

The standard form of a QP (following CVXOPT notation) is:

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2} x^\top P x + q^\top x \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b \end{aligned}$$

Which is similar to that Equations 1. But, in Equations-1 we have 0<= alphai <= C. But, here we have only one inequality, for this we change the condition as (-alphai<=0) and alphai <= C and now combine this to form matrices G and h using the following code.

```
#Generating conditions (i.e., o<=lambd_i<=c)
G1 = np.diag(np.ones(N) * -1)
G2 = np.identity(N)
G = cvxopt.matrix(np.vstack((G1, G2)))

h1 = np.zeros(N)
h2 = np.ones(N) * C
h = cvxopt.matrix(np.hstack((h1, h2)))
```

We need to also convert the labels into +1 and -1, to solve the dual problem.

Remaining all the other variables are very much same as that of Equations-1.

Now, I ran the code for C=-5.623, which was calculated using LIBSVM. I got the accuracy as 95.75 % which is very close to the value, we got from LIBSVM.

Now, I trained the SVM for a RBF kernel(non-Linear).

Using RBF Kernel:

Kernel function K (xi, xj) for RBF Kernel is given by **exp(-gamma*(|xi-xj|^2)).**

Using LIBSVM:

The t parameter for LIBSVM is 2.

The hyper parameters here are gamma, C.

Now, again as before I have done coarse tuning first by changing gamma from 2^-10 to 2^10 and C from 2^-10 to 2^10.

From, which I got gamma=1/32 and C=2.

Now on fine tuning I got C=2^0.6 and gamma=1/32 with 98.75% which is far better than that of Linear Kernel.

```
Accuracy = 98.5% (394/400) (classification)
Value of coarse log2(c), for maximum accuracy: 0.6
Value of coarse log2(gamma), for maximum accuracy: -5.0000000000000036
The percentage of predicted values is equal to 98.75 %
```

Using CVXOPT:

Now, taking C=2^0.6 and gamma=1/32. I got the accuracy percentage equals to 98.75%. Which is the same which we got using LIBSVM.

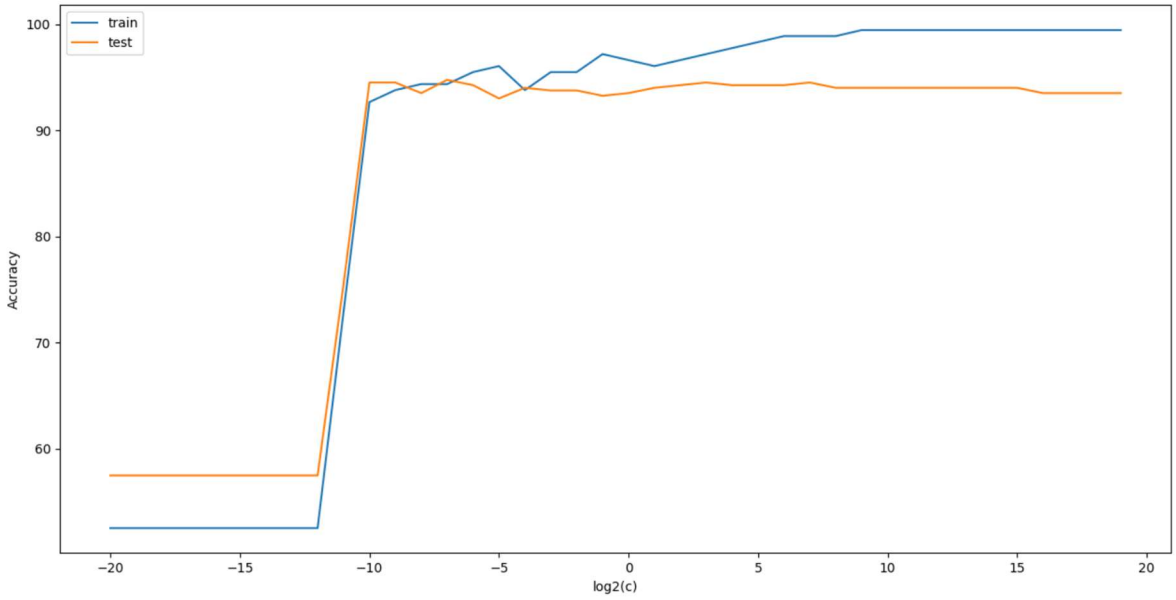
Using only 10 feature vectors:

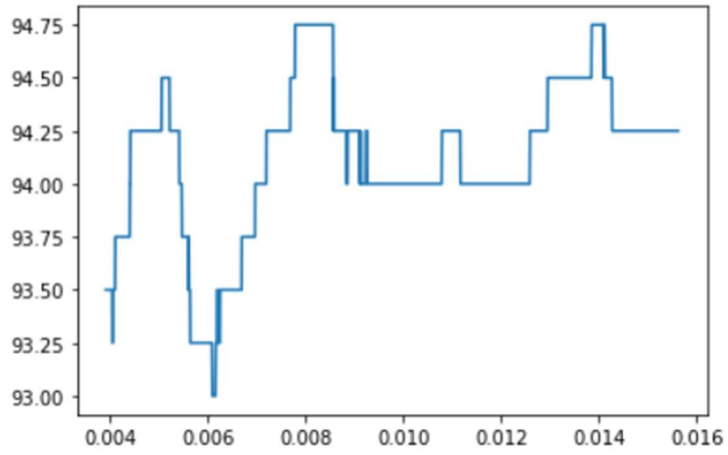
Linear Kernel:

Using LIBSVM:

I did the same analysis as before.

Plots of Accuracy vs $\log_2(C)$:





From, the above plots we get C=2^-7 as the best hyper parameter and accuracy of 94.75 %. We could see that using only 10 feature we have the optimal value of C is smaller than that of before case. Hence, now our SVM is more concentrated on margin (i.e., getting good margin). So, the accuracy also decreases lightly. But, as the accuracy is almost the same, we can conclude that it enough to use 10 feature vectors than using all the 25 feature vectors for classifying labels 2.0 and 5.0.

Using CVXOPT:

Now, taking C=2^-7. I got the accuracy percentage equals to 94.5%, which is very close to that of LIBSVM.

RBF Kernel:

Using LIBSVM:

On coarse tuning we get C=1, gamma= 1/8.

On fine tuning we get C= 2^-0.8, gamma= 2^-3.6 with an accuracy percentage equals to 98.5%. Here also the accuracy is almost same as with 25 feature vectors.

Using CVXOPT:

Now, taking C= 2^-0.8, gamma= 2^-3.6. I got the accuracy percentage equals to 98.25%, which is very close to that of LIBSVM.

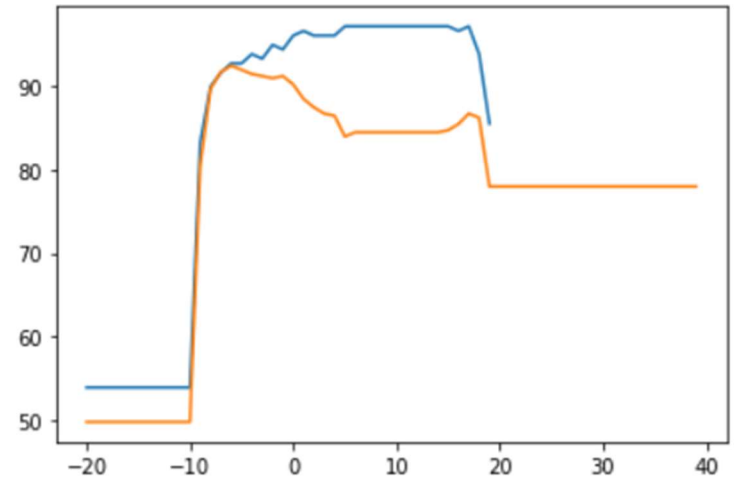
Similarly, I have repeated this for two more pairs of labels (Though only one pair more, I have mentioned in the Report, the other one you could see the code).

b)

Labels used are 4.0 and 9.0.

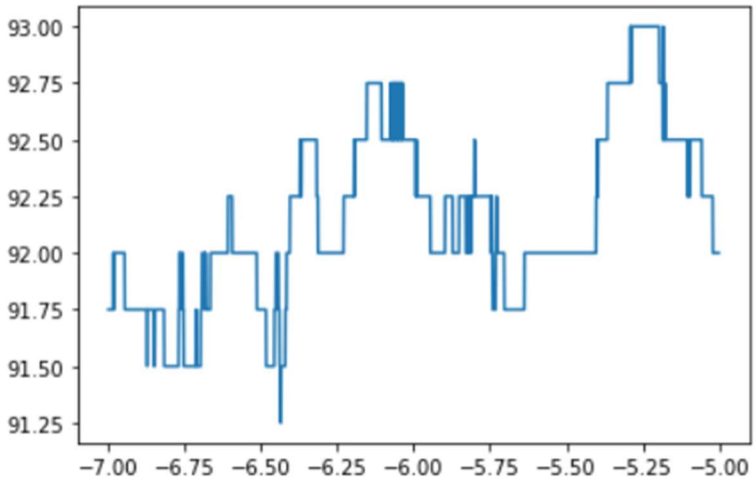
Linear Kernel:

Plots of Accuracy vs $\log_2(C)$:




```
Value of coarse log2(c), for maximum accuracy: -6
```

We could see that for log2(c) <-7 and log2(c)>18 it is underfitting. The best fit is for log2(C)=-6.




```
Value of fine log2(c), for maximum accuracy: -5.289999999999429
The percentage of predicted values: 93.0
```

The best fit is for log2(C)=-5.29 with accuracy of 93%.

Using CVXOPT:

Now, taking C= 2^-5.29. I got the accuracy percentage equals to 92.75%, which is very close to that of LIBSVM.

RBF Kernel:

Using LIBSVM:

On coarse tuning we get C=8, gamma= 2^-7.

On fine tuning we get C= 2^2.8, gamma= 2^-6.8 with an accuracy percentage equals to 94.25%. Here also the accuracy is almost same as with 25 feature vectors.

Using CVXOPT:

Now, taking C= 2^2.8, gamma= 2^-6.8. I got the accuracy percentage equals to 97.25%.

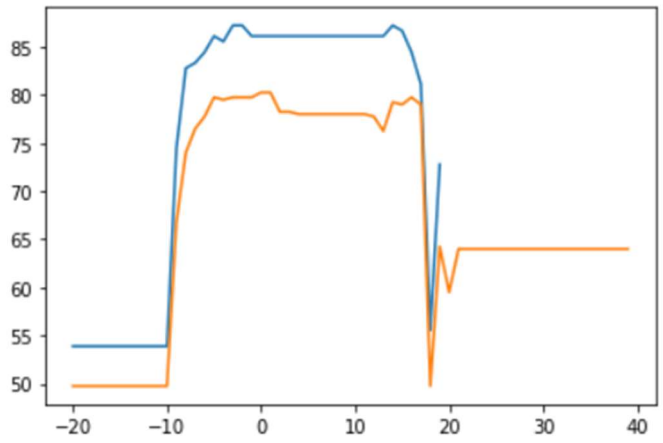
Using only 10 feature vectors:

Linear Kernel:

Using LIBSVM:

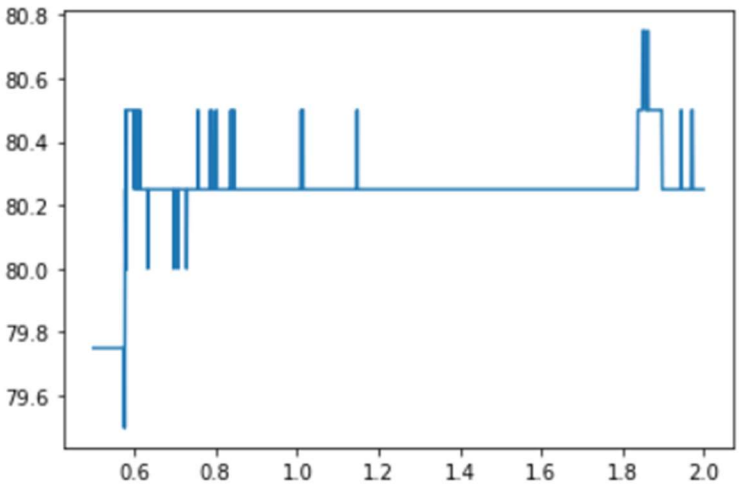
Plots of Accuracy vs $\log_2$(C):

Coarse-tuning:




```
Value of coarse log2(c), for maximum accuracy: 0
```

Fine-Tuning:




```
Value of fine log2(c), for maximum accuracy: 0.8870000000000016
```

The accuracy is 80.75% which is very low, this says that the output is dependent on the other feature vectors more.

Using CVXOPT:

Now, taking C= 2^0.88. I got the accuracy percentage equals to 79.25%.

RBF Kernel:

Using LIBSVM:

On coarse tuning we get C=2, gamma= 2^-5.

On fine tuning we get C= 2^ 0.5, gamma= 2^-4.7 with an accuracy percentage equals to 83.25%. Again, the accuracy is too low which again states that the output depends on remaining feature vectors too.

Using CVXOPT:

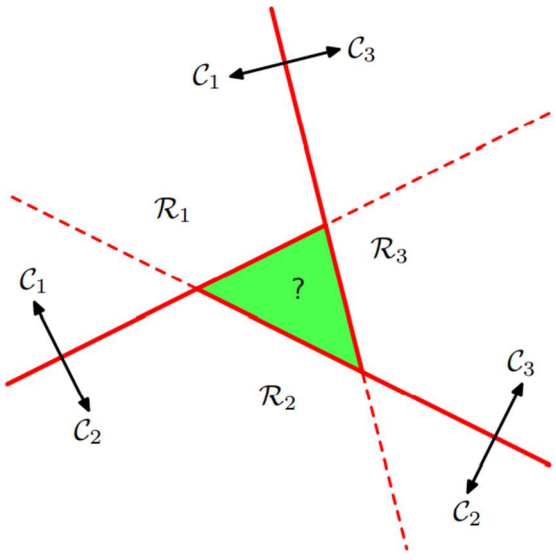
Now, taking C= 2^0.5, gamma= 2^-4.7. I got the accuracy percentage equals to 81.5%.

From, all the above cases we could see that CVXOPT and LIBSVM, both were giving the similar outputs, though LIBSVM was little fast because it uses SMO algorithm.

3)

Multiclass classification:

For Multi Class Classification, LIBSVM uses one vs one classification where we introduce K (K − 1)/2 binary discriminant functions, one for every possible pair of classes. The implementation for a 3 class is shown in the below figure.



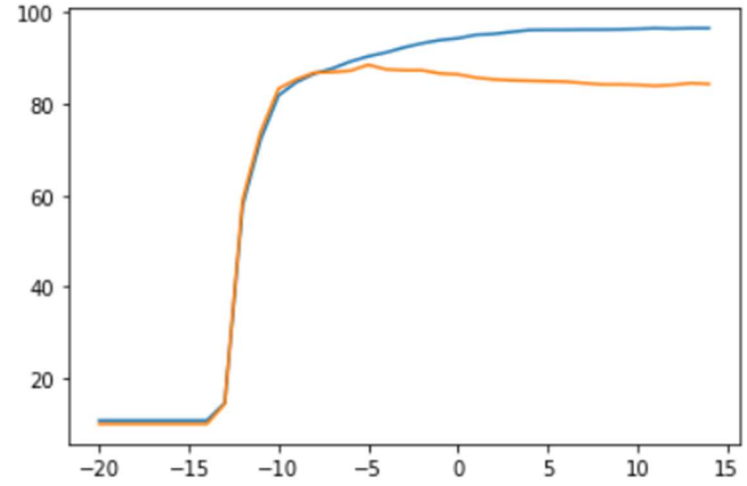
Here, also I have solved using LIBSVM for both Linear Kernel and RBF Kernel.

Linear Kernel:

Using LIBSVM:

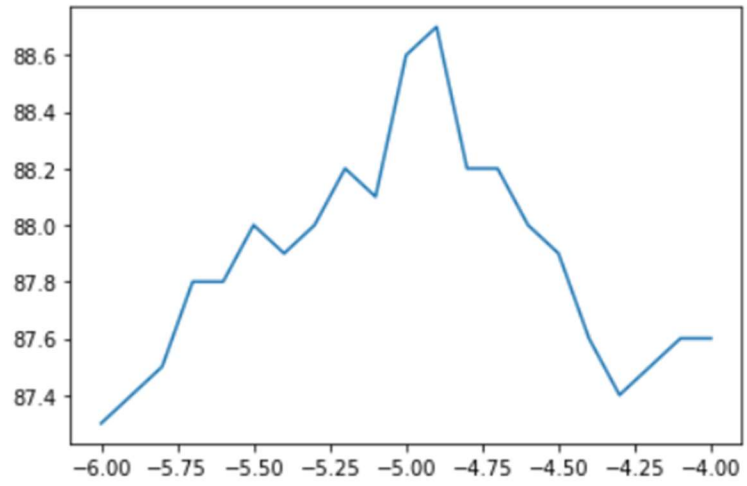
Plots of Accuracy vs $\log_2$(C):

Coarse-Tuning:




```
Value of coarse log2(c), for maximum accuracy: -5
```

Fine-Tuning:




```
Value of fine log2(c), for maximum accuracy: -4.900000000000004
Percentage of Samples for which output is predicted correctly: 88.7
```

RBF Kernel:

Using LIBSVM:

On coarse tuning we get C=2, gamma= 2^-4.

On fine tuning we get C= 2, gamma= 2^-4.24 with an accuracy percentage equal to 94.2% which is far better than linear kernel.

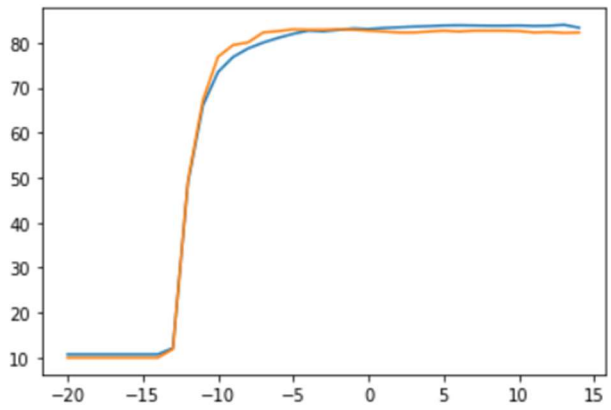
Using only 10 feature vectors:

Linear Kernel:

Using LIBSVM:

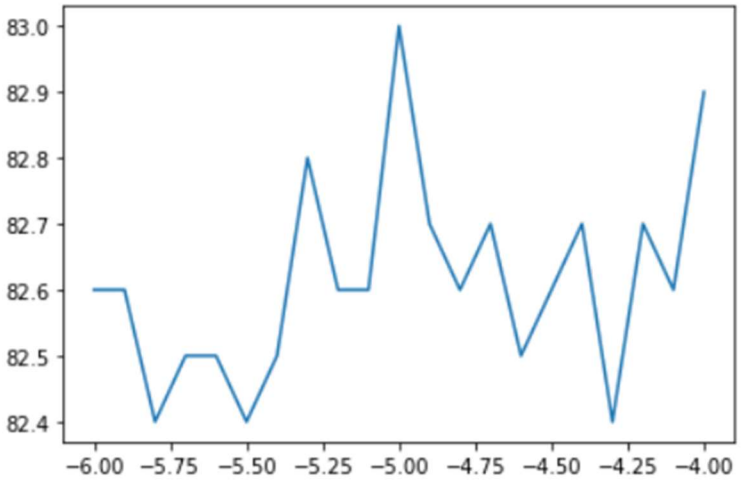
Plots of Accuracy vs $\log_2(C)$:

Coarse-Tuning:




```
Value of coarse log2(c), for maximum accuracy: -5
```

Fine-tuning:




```
Value of fine log2(c), for maximum accuracy: -5.0000000000000036
Percentage of Samples for which output is predicted correctly: 83.0
```

The accuracy is less, which means that the output depends on the other feature vectors too.

Part-1B:

I have implemented the Simplified SMO algorithm according to the SMO pdf of CS229 given in the question. Here, we initially take all the alphai's as zeroes and later, randomly select two alphai's and make their values positive and we continue this process until alphai's converge.

I have written the code step by step as given in the pdf. You could refer my python code and also jupyter notebook (Part-1B(SMO) (here, it is easier to understand).

Now, I tested this for the data given in Label-1A for Linear kernel (We could also extend this method to other kernels by slight modification in the code) using all the 25 feature vectors for binary classification.

I set the tolerance as 1e-7 and max passes as 20, which I have decided after rigorous testing.

I took the regularisation parameter C as -5.63 (the value we got in part1A). I got the accuracy equals to 94.25% which was good estimate as with LIBSVM (96%) and CVXOPT (95.75%). But the SMO algorithm was a bit slow compared to that of with that of LIBSVM and CVXOPT, which is most probably because of the fact that the random number generation takes time, and in Simplified SMO we need have to generate random, alphai's. Also, for more accuracy we need even more no. of max_passes, but it would even more time. So, Simplified SMO is not a better idea according to me.

To, over come this Problem, we could use Full SMO (which LIBSVM) uses, here we don't have any randomness, there is a way to select the alphai's, which would be give better and faster output compared to that of Simplified one.

Part-2:

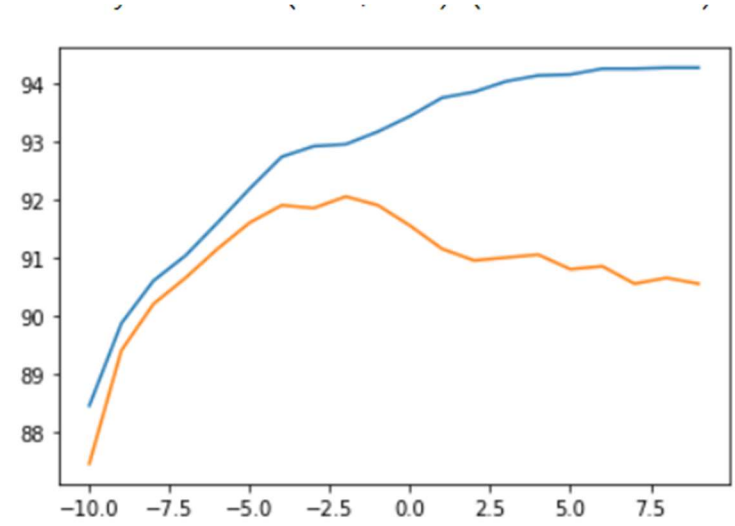
Given, 8000 instances. I first divided 6000 samples into training set and remaining as Testing (Validation) set. (I have decided this splitting after rigorous testing).

First, I trained this Multi Class SVM using Linear Kernel and later using RBF Kernel, both using LIBSVM (The method is same as before which I have explained already. So, here I am writing the final outputs).

Linear-Kernel:

Plots of Accuracy vs $\log_2(C)$:

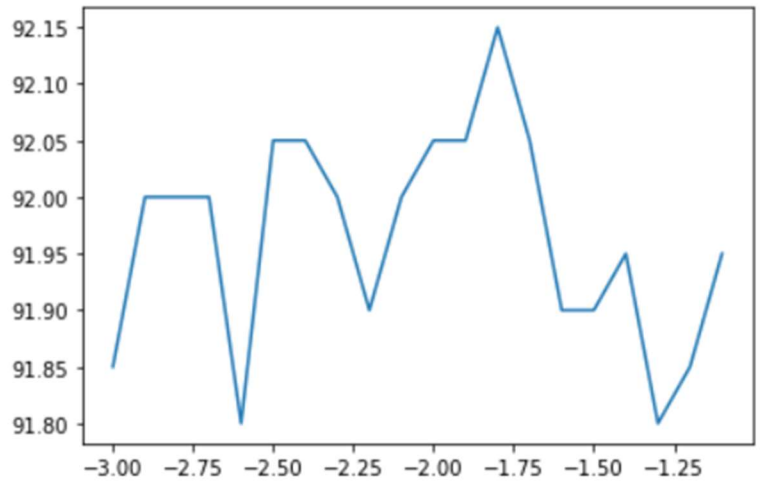
Coarse-Tuning:




```
Value of coarse log2(c), for maximum accuracy: -2
```

From, the above plot we could see that log2(C)<-5 it is under fitting and log2(C) >0 it is overfitting.

Fine-Tuning:




```
Value of coarse log2(c), for maximum accuracy: -1.799999999999999
Percentage of Samples for which output is predicted correctly: 92.15
```

We get an accuracy of 92.15 %, which is not a good estimate. So, I switched to RBF Kernel.

RBF Kernel:

Using LIBSVM: (Method, already written in Part1A)

On coarse tuning we get C=4, gamma= 2^-4.

On fine tuning we get C= 4, gamma= 2^-3.95 with an accuracy percentage equals to **96.75%.**

This is far better than the previous case, so RBF Kernel is better Kernel than Linear. I have also submitted my predictions on Kaggle (foo3.csv is the file).