ELL409 Assignment 2 Report

Kshitij Alwadhi (2019EE10577)

22nd October 2021

1. Introduction

In this assignment, we experiment with the use of SVMs for both binary and multiclass classification problems, and understand the effects of varying various hyperparameters therein.

2. Part 1A

2.1 Formulating the problem for CVXOPT.

We have the following optimization problem in the case of an SVM with L1 regularization:

$$\min_{\gamma, w, b} \quad \frac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{m} \xi_i$$

$$\text{s.t.} \quad y^i (w^T x^i + b) \geq 1 - \xi_i \quad , \quad i = 1 \dots m$$

$$\xi_i \geq 0$$

To solve this problem, we take the help of Lagrangian Multipliers and proceed using KKT conditions. (The following results are taken from the **CS229 SVM notes**)

The Lagrangian for the optimization problem is:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \cdot \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \left[y^i (x^T w + b) - 1 + \xi_i \right] - \sum_{i=1}^{m} r_i \xi_i$$

After applying the KKT conditions (taking the respective partial derivatives), we get the following optimization problem:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^i \cdot y^j \cdot \alpha_i \cdot \alpha_j \cdot \underbrace{\boxed{\langle x^i, x^j \rangle}}_{K(x^i, x^j)}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \ , \quad i = 1 \ldots m$$

$$\sum_{i=1}^{m} \alpha_i \cdot y^{(i)} = 0$$

This is equivalent to the following minimization problem:

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

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \ , \quad i = 1 \ldots m$$

$$\sum_{i=1}^{m} \alpha_i \cdot y^{(i)} = 0$$

To simplify things, we define the following:

We define a matrix H such that

$$H_{i,j} = y^i \cdot y^j \cdot K(x^i, x^j)$$

Now, our optimization problem looks like:

$$\min_{\alpha} \quad \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \cdot \alpha_j \cdot H_{ij} - \sum_{i=1}^{m} \alpha_i$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \ , \quad i = 1 \ldots m$$

$$\sum_{i=1}^{m} \alpha_i \cdot y^{(i)} = 0$$

Now this is a quadratic optimization problem but we need to convert it into a form which we can feed into CVXOPT.

CVX opt general form for QP:

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

$$\text{s.t.} \quad Gx \preceq h$$

$$Ax = b$$

$$x = \text{cvxopt.solvers.qp}(P, q, G, h, A, b)$$

We can convert our optimization problem to the following form so that it resembles the form of QP in CVXOPT:

$$\min_{\alpha} \quad \frac{1}{2} \alpha^T H \alpha + [-1 \ -1 \ \ldots \ -1]\, \alpha$$

$$\text{s.t.} \quad \begin{bmatrix} -I \\ I \end{bmatrix} \begin{array}{l} n\times n \\ n\times n \end{array} \alpha \preceq \begin{bmatrix} 0 \\ C \end{bmatrix} \begin{array}{l} n\times 1 \\ n\times 1 \end{array}$$

$$y^T \alpha = 0$$

On comparing the terms, we have the following:

$$P = H \qquad q = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$$

$$G = \begin{bmatrix} -I \\ I \end{bmatrix} \begin{matrix} \rightarrow n\times n \\ \rightarrow n\times n \end{matrix} \qquad h = \begin{bmatrix} 0 \\ C \end{bmatrix} \begin{matrix} \rightarrow n\times 1 \\ \rightarrow n\times 1 \end{matrix}$$

$$A = y^T \qquad b = 0$$

This can be solved using CVXOPT by the following:

$$\alpha = \text{cvxopt.solvers.qp}(P, q, G, h, A, b)$$

The following is the code which solves for α using CVXOPT:

```
        X = np.array(X)
        y = np.array(y)
        num_samples,num_features = X.shape
        K = np.zeros((num_samples,num_samples))

        for i in (range(num_samples)):
            for j in range(num_samples):
                if self.kernel == 'LINEAR':
                    K[i][j] = np.dot(X[i],np.transpose(X[j]))
                elif self.kernel == 'POLY':
                    K[i][j] = (np.dot(X[i],np.transpose(X[j])) + self.coeff) ** self.power
                elif self.kernel == 'RBF':
                    K[i][j] = np.exp(-1 * self.gamma*np.sum(np.square(X[i]-X[j])))

        self.K = K
```

```
            H = np.zeros((num_samples,num_samples))
            for i in (range(num_samples)):
                for j in range(num_samples):
                    H[i][j] = y[i]*y[j] * K[i][j]
            P = matrix(H)
            q = matrix(np.ones(num_samples) * -1)
            G = matrix(np.vstack(((np.identity(num_samples) * -1),np.identity(num_samples))))
            h = matrix(np.hstack((np.zeros(num_samples),np.ones(num_samples)*self.C)))
            A = matrix(y,(1,num_samples))
            b = matrix(0.0)

            solvers.options['show_progress'] = False
            soln = solvers.qp(P,q,G,h,A,b)
            alpha = np.array(soln['x'])
```

Now that we have solved for the value of α, we need to determine our supporting vectors, store the non-zero α values and also calculate the value of b so that we can make predictions.

The equation of separating hyperplane is given by:

$$y = \sum_{i=1}^{m} \alpha_i \cdot y^{(i)} \cdot \underbrace{< x^{(i)}, x >}_{K(x^{(i)}, x)} + b \; \Big] \rightarrow \text{Separating hyperplane}$$

$$y = \sum_{i=1}^{m} \alpha_i \cdot y^{(i)} \cdot K(x^{(i)}, x) + b$$

We know that the supporting vectors lie on the separating hyperplanes, so we exploit that and write the following:

(Sup_x, Sup_y) → Supporting Vectors

Let (sup_x[0], sup_y[0]) be a tuple from the supporting vectors which lies on our separating hyperplane.

$$sup\_y[0] = \sum_{i=1}^{m} \alpha_i \cdot y^{(i)} \cdot K(x^{(i)}, sup\_x[0]) + b$$

$$b = sup\_y[0] - \sum_{i=1}^{m} \alpha_i \cdot y^{(i)} \cdot K(x^{(i)}, sup\_x[0])$$

We find the value of b using the above formula. Also, we can directly find the indices of the supporting vectors by checking where the indices of α are non-zero. These steps are performed using the following code:

```
            self.sup_idx = np.where(alpha>1e-5)[0]
            self.ind =  np.arange(len(alpha))[self.sup_idx]

            self.sup_x = X[self.sup_idx,:]
            self.sup_y = y[self.sup_idx]
            self.alpha = alpha[self.sup_idx]
            self.b = self.sup_y[0]

            for i in range(len(self.alpha)):
                if self.kernel == 'LINEAR':
                    temp = np.dot(self.sup_x[i],np.transpose(self.sup_x[0]))
                elif self.kernel == 'POLY':
                    temp = (np.dot(self.sup_x[i],np.transpose(self.
↪sup_x[0]))+self.coeff)**self.power
                elif self.kernel == 'RBF':
                    temp = np.exp(-1 * self.gamma*np.sum(np.square(self.
↪sup_x[i]-self.sup_x[0])))
                self.b -= self.alpha[i] * self.sup_y[i] * temp
```

For deciding the class of the input X, we see which side of the separating hyperplane our data point is and give it a +1 class if it's above the hyperplane and -1 if it's below the hyperplane. This is performed by the following code snippet:

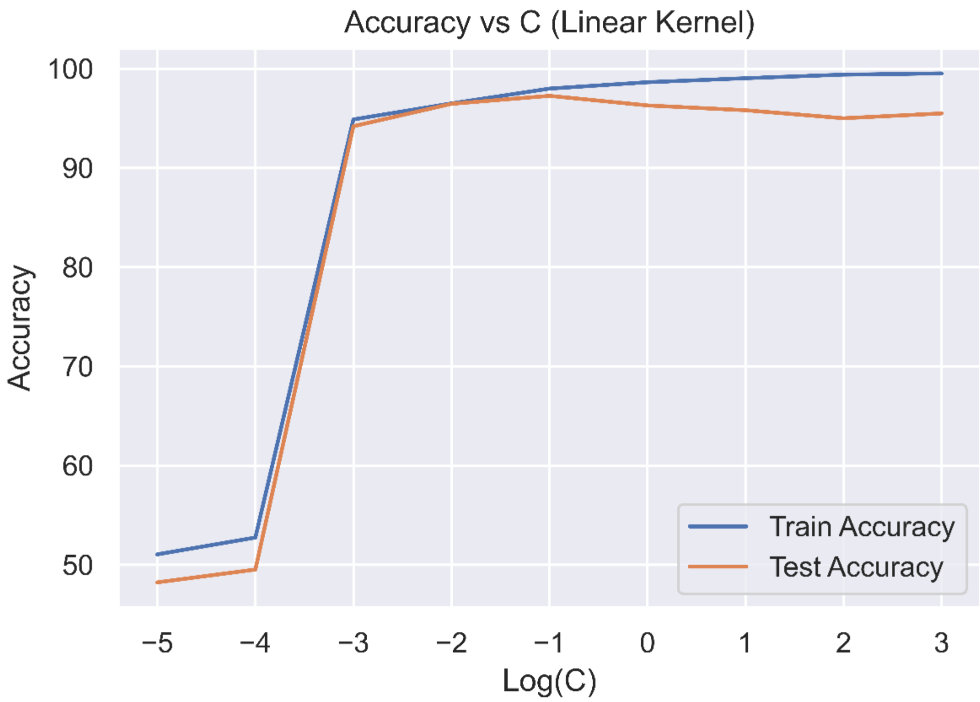
```
def predict(self,X):
    preds = []
    for x in X:
        pred = 0
        for i in range(len(self.alpha)):
            if self.kernel == 'LINEAR':
                temp = np.dot(self.sup_x[i],np.transpose(x))
            elif self.kernel == 'POLY':
                temp = (np.dot(self.sup_x[i],np.transpose(x)) + self.coeff)␣
↪** self.power
            elif self.kernel == 'RBF':
                temp = np.exp(-1 * self.gamma *np.sum(np.square(self.
↪sup_x[i]-x)))
            pred += self.alpha[i] * self.sup_y[i] * temp
        pred += self.b
        if pred>=0:
            preds.append(1.0)
        else:
            preds.append(-1.0)
    return np.array(preds)
```

2.2 Binary Classification

In all of the analysis, I have used 5-fold Cross Validation and then averaged out the train and test accuracy for comparing the results of the various hyperparameter settings.

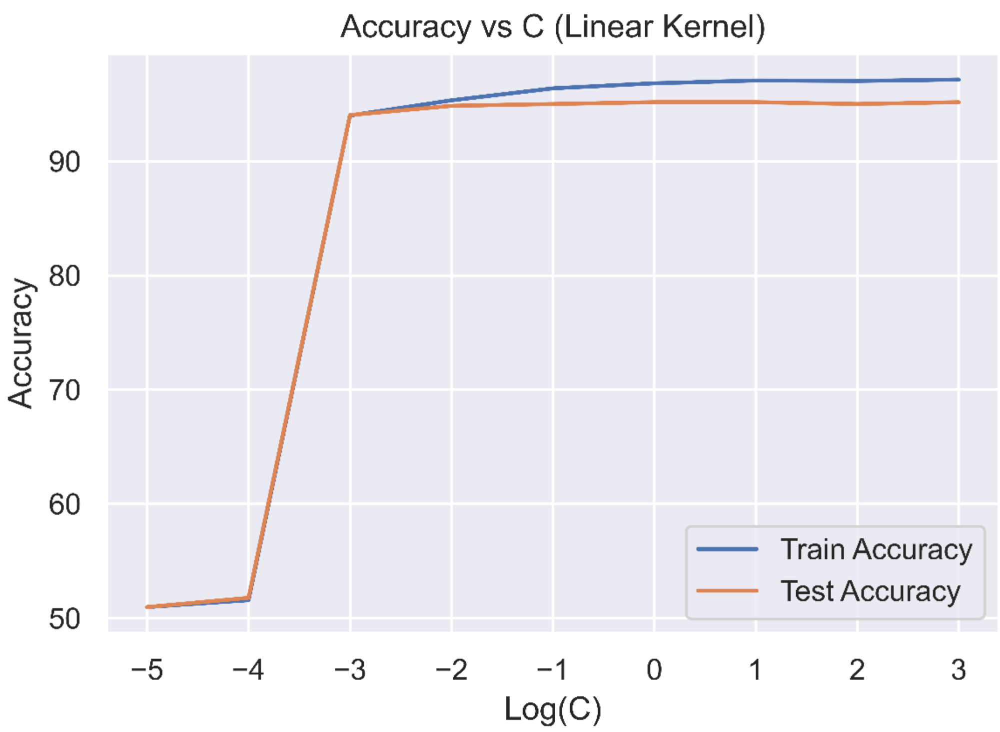
2.2.1 First we do the analysis for: **C1 = 1, C2 = 8**

Using **Linear Kernel** we get the following results:



(Accuracy vs C (Linear Kernel) for 25 features)

From the above plot, we can see that when the value of C is very low (<0.001), the model is underfitting the data but as the value of C increases, the model starts to fit the data (good fit) and we observe a peak Test accuracy at C = 0.1. Beyond this (C>0.1), we start to see that training accuracy creeps up to almost 100% and the test accuracy starts falling, hence indicating overfitting.



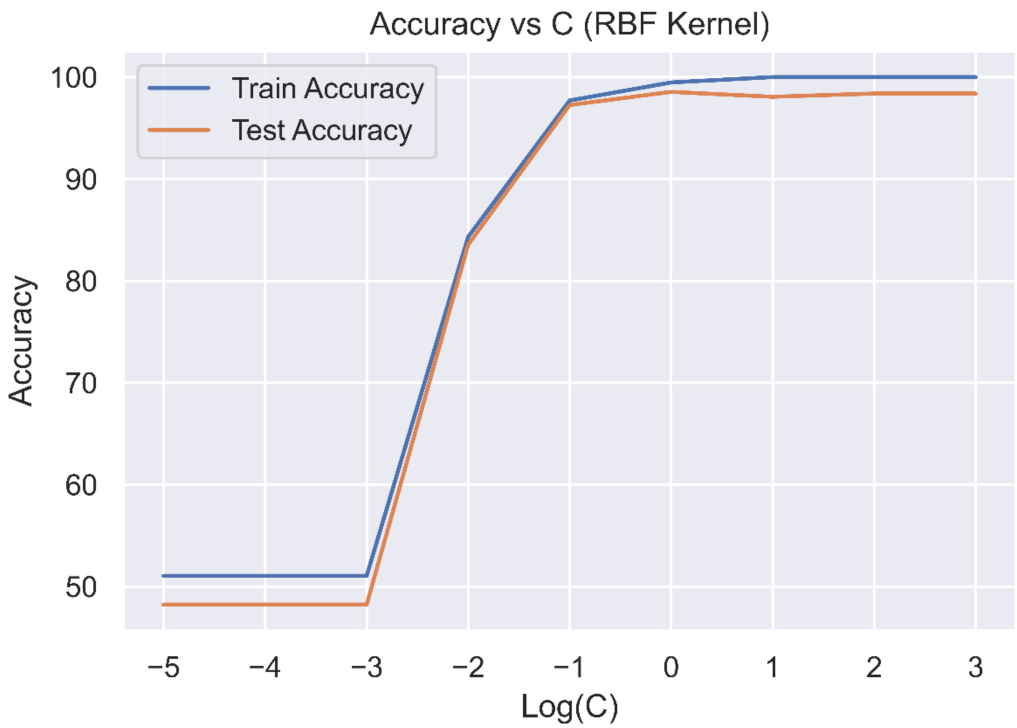
(Accuracy vs C (Linear Kernel) for 10 features)

Now, if we use only 10 features, we get a similar story, for C < 0.001, the model is underfitting the data and we get peak test accuracy at C = 0.01 beyond which the model starts overfitting the data.

There are two differences we observe here:

1) The peak test accuracy that we obtain when we use all 25 features is greater than when we use only 10 features.
2) The model is able to get a 100% train accuracy in the case of 25 features when C is significantly larger, however, this is not achievable in the case of 10 feature dataset.

Using **RBF kernel** we get the following results:

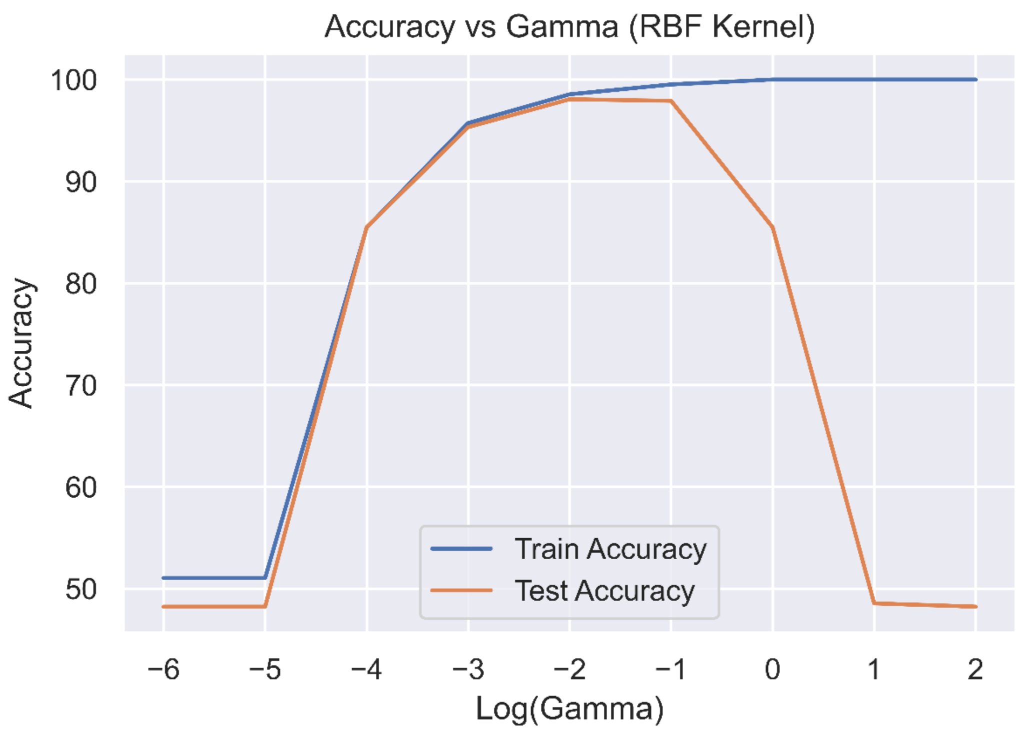


(Accuracy vs C (RBF Kernel) for 25 features)

In the case of RBF kernel, we have two parameters to play around with. For finding the appropriate value of C, we perform a logarithmic sweep while keeping the value of γ to be $\frac{1}{\#\ of\ features}$ as usually done by standard SVM libraries.

We observe the following:

C < 1	Underfitting
C = 1	Best fit
C > 1	Overfitting

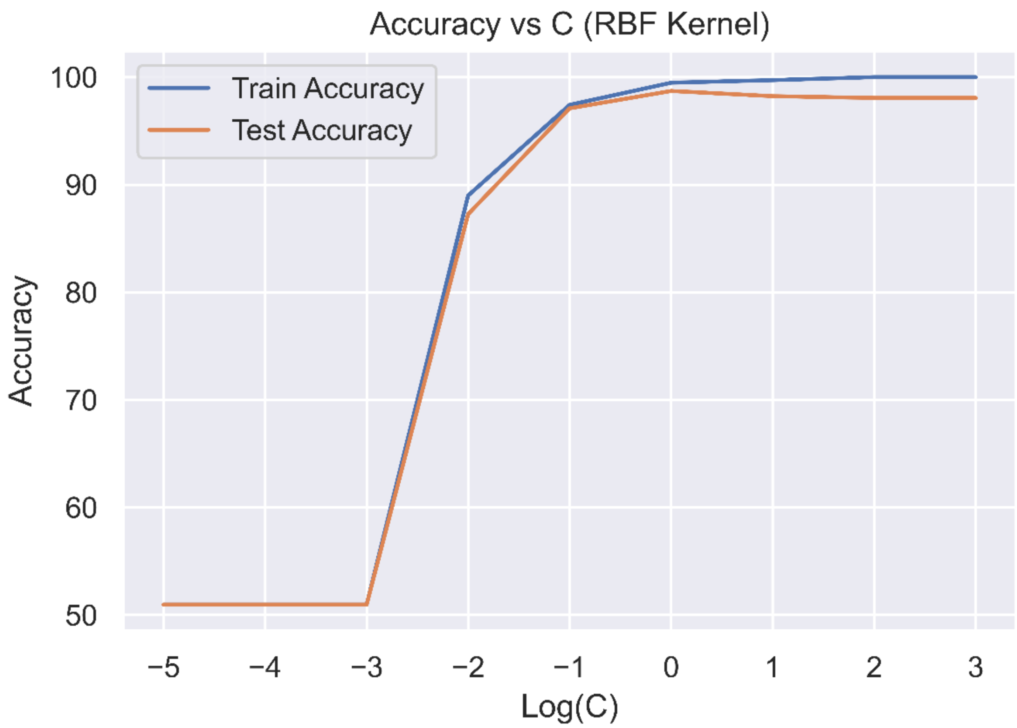


(Accuracy vs Gamma (RBF Kernel) for 25 features)

We also perform a logarithmic sweep over values of γ. We obtain the following result:

$\gamma < 0.01$	Underfitting
$\gamma = 0.01$	Best fit
$\gamma > 0.01$	Severe Overfitting

Now for the case of 10 features:

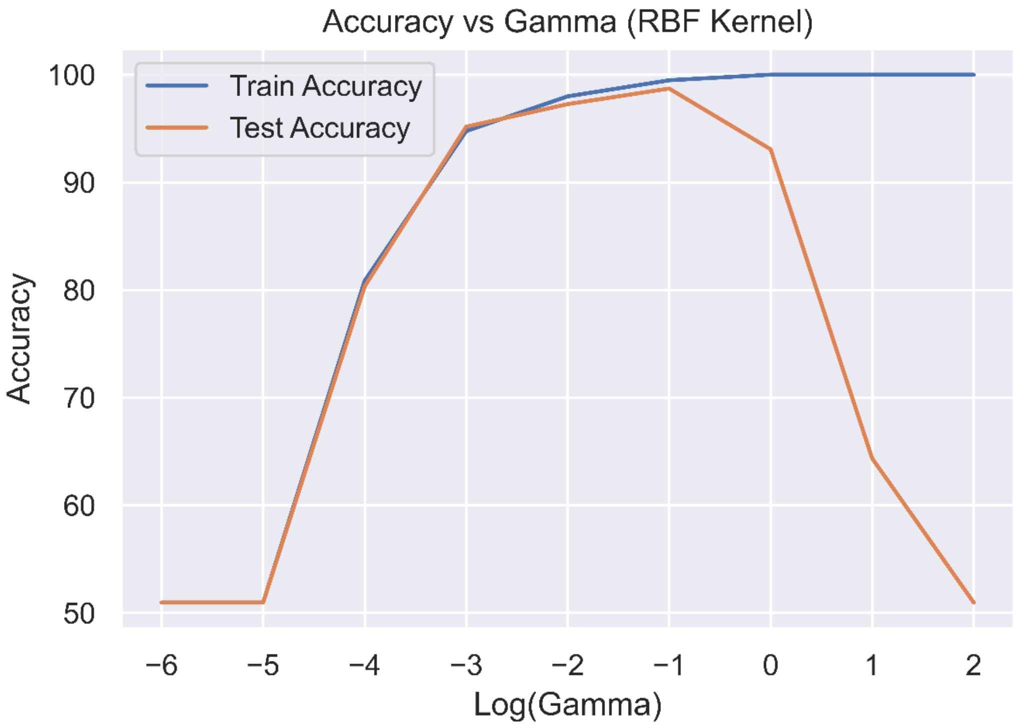


(Accuracy vs C (RBF Kernel) for 10 features)

Now, if we use only 10 features, we get a similar story, for C < 0.1, the model is underfitting the data and we get peak test accuracy at C = 1 beyond which the model starts overfitting the data.

C < 1	Underfitting
C = 1	Best fit
C > 1	Overfitting

The main difference that we observe here is that the peak test accuracy is lower when we have 10 features instead of 25.



(Accuracy vs γ (RBF Kernel) for 10 features)

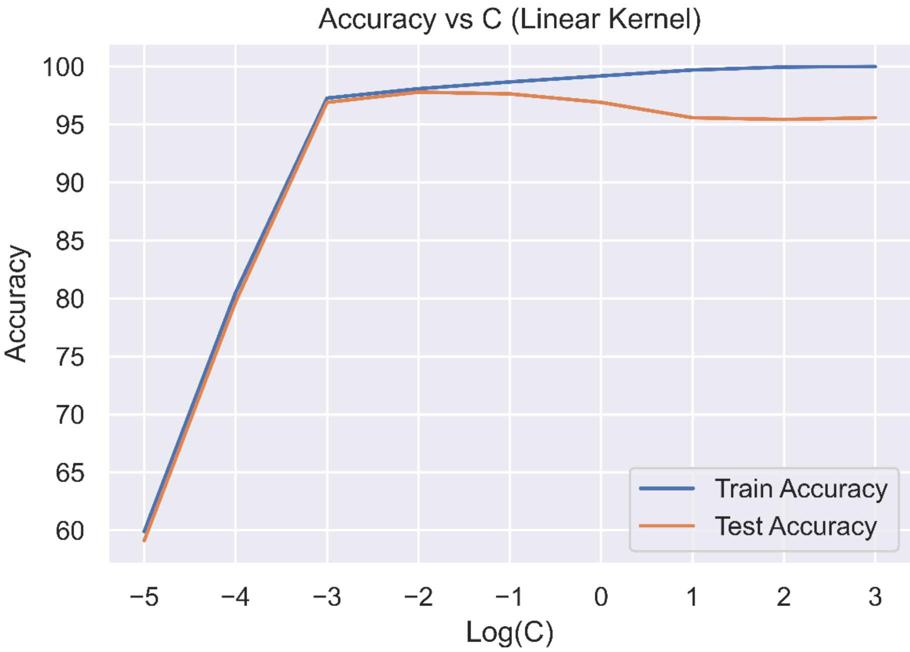
We get the following results after performing a logarithmic sweep over γ:

$\gamma < 0.1$	Underfitting
$\gamma = 0.1$	Best fit
$\gamma > 0.1$	Severe Overfitting

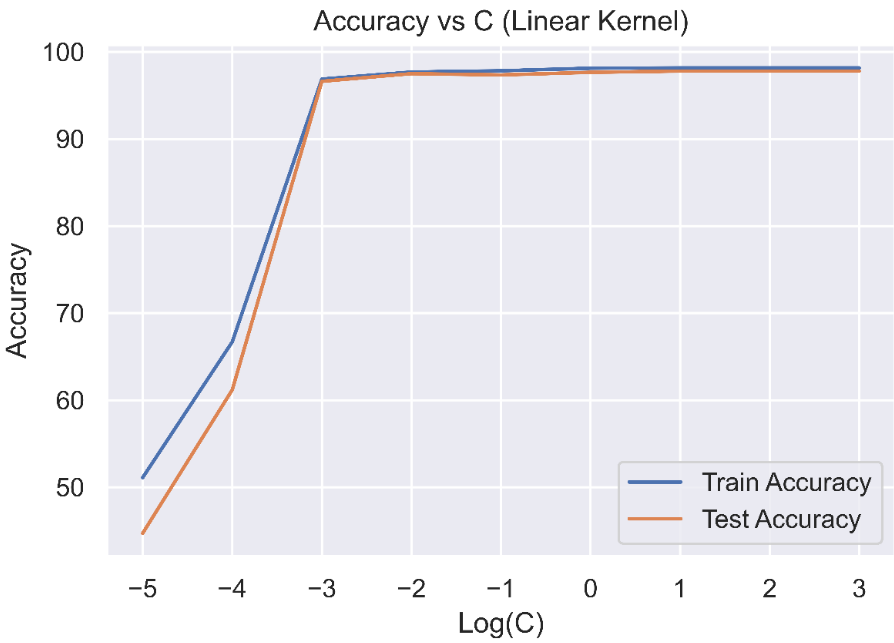
One thing to note here is that the value of γ at which peak test accuracy occurred has changed from 0.01 to 0.1, this change however is not very significant as the test accuracy is very close on both of these values.

2.2.2 Now we do the analysis for: **C1 = 3, C2 = 7**

Using **linear** kernel:



(Accuracy vs C (Linear Kernel) for 25 features)

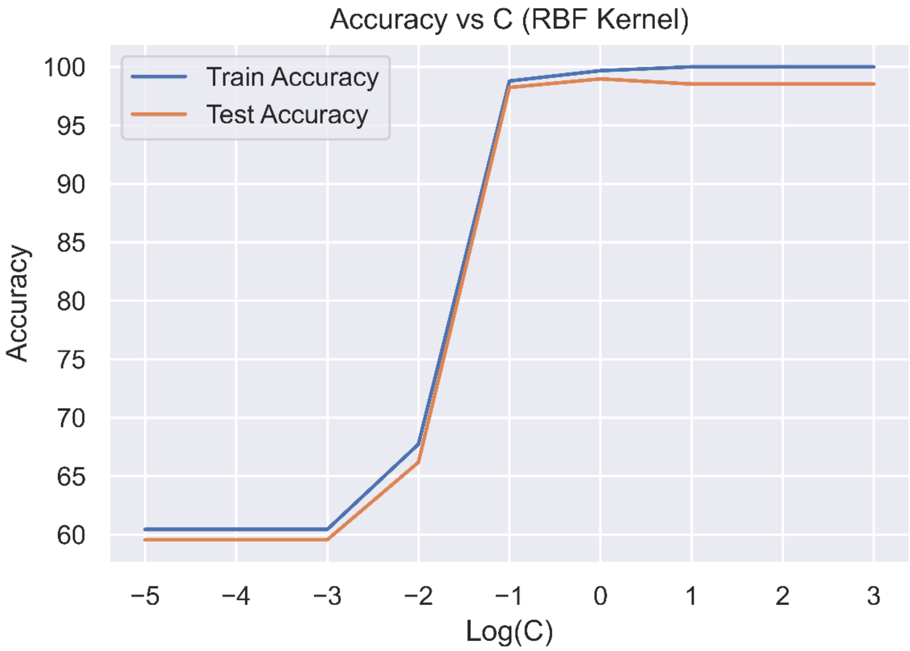


(Accuracy vs C (Linear Kernel) for 10 features)

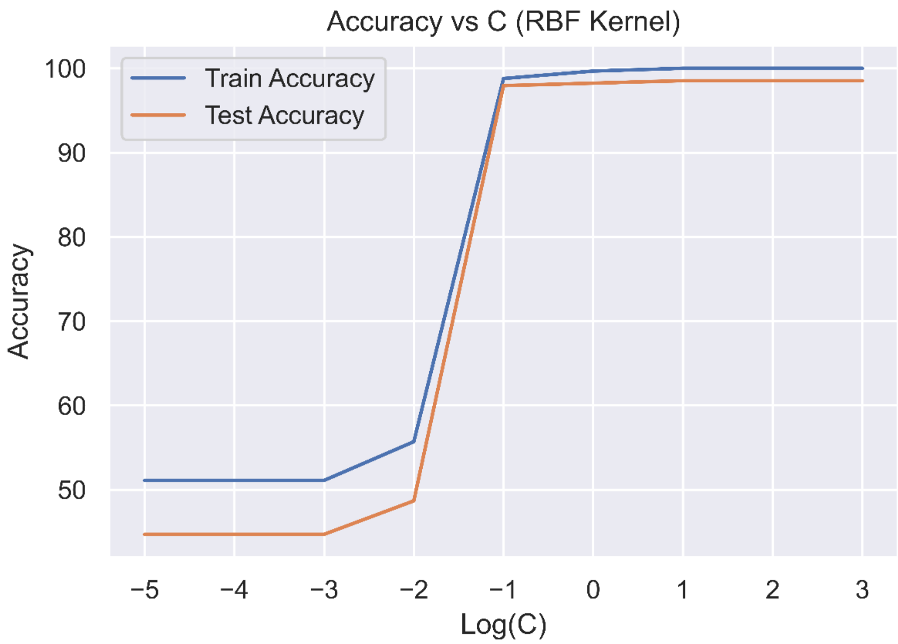
C < 0.01	Underfitting
C = 0.01	Best fit
C > 0.01	Overfitting

(Consistent results)

Using **RBF** kernel:



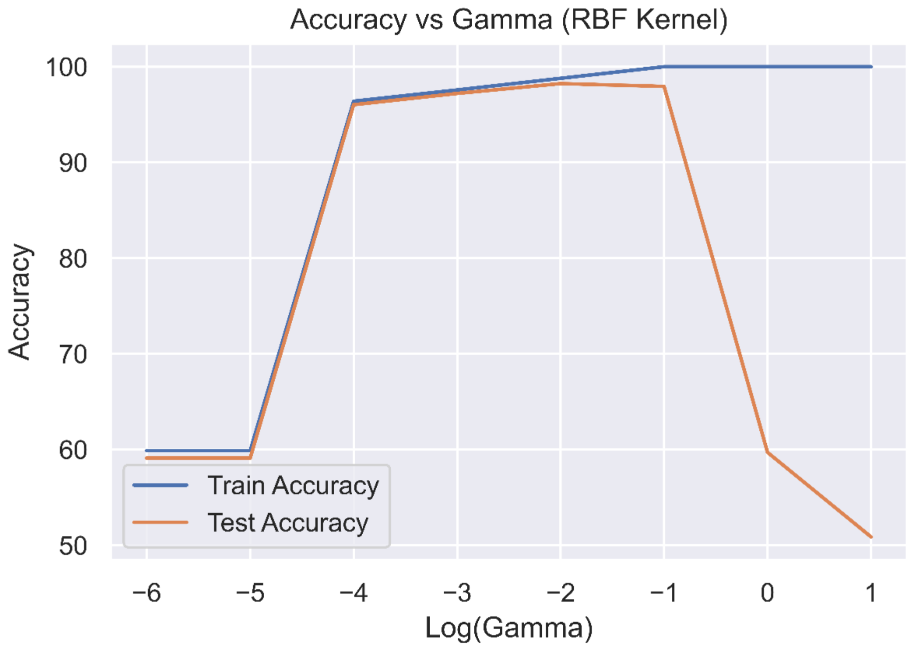
(Accuracy vs C (RBF Kernel) for 25 features)



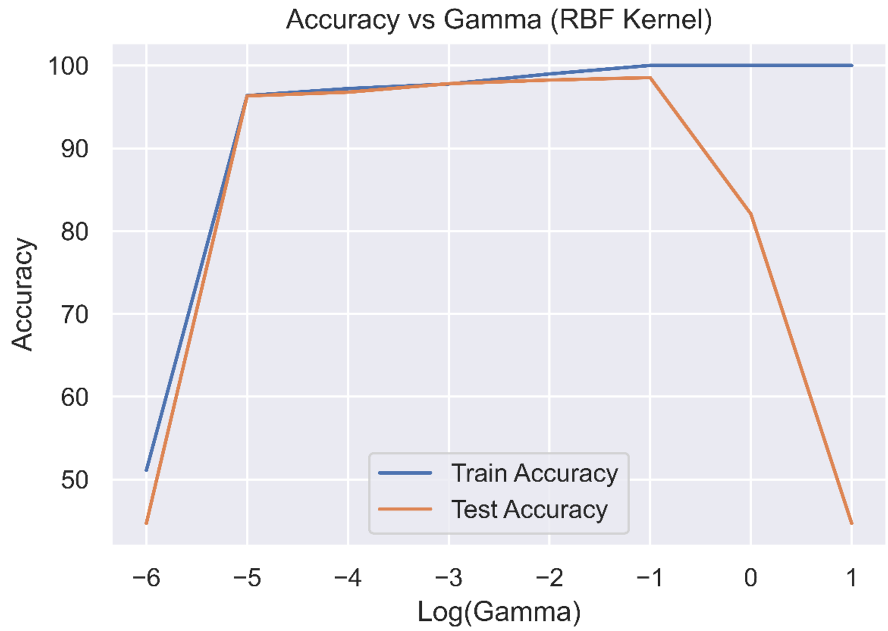
(Accuracy vs C (RBF Kernel) for 10 features)

C < 0.1	Underfitting
C = 0.1	Best fit
C > 0.1	Overfitting

(Consistent results)



(Accuracy vs γ (RBF Kernel) for 25 features)



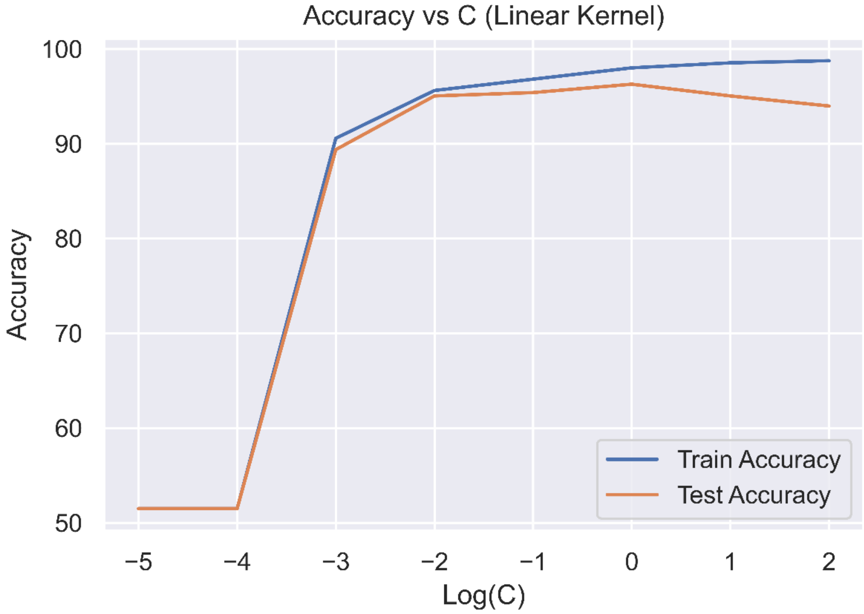
(Accuracy vs γ (RBF Kernel) for 10 features)

$\gamma < 0.1$	Underfitting
$\gamma = 0.1$	Best fit
$\gamma > 0.1$	Severe Overfitting

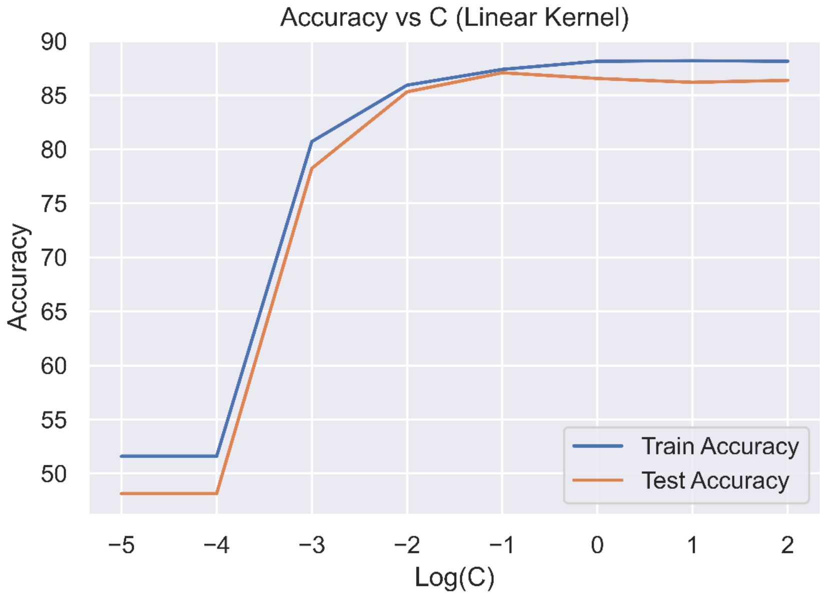
(Consistent results)

2.2.3 Now we do the analysis for: **C1 = 4, C2 = 9**

Using **linear** kernel:



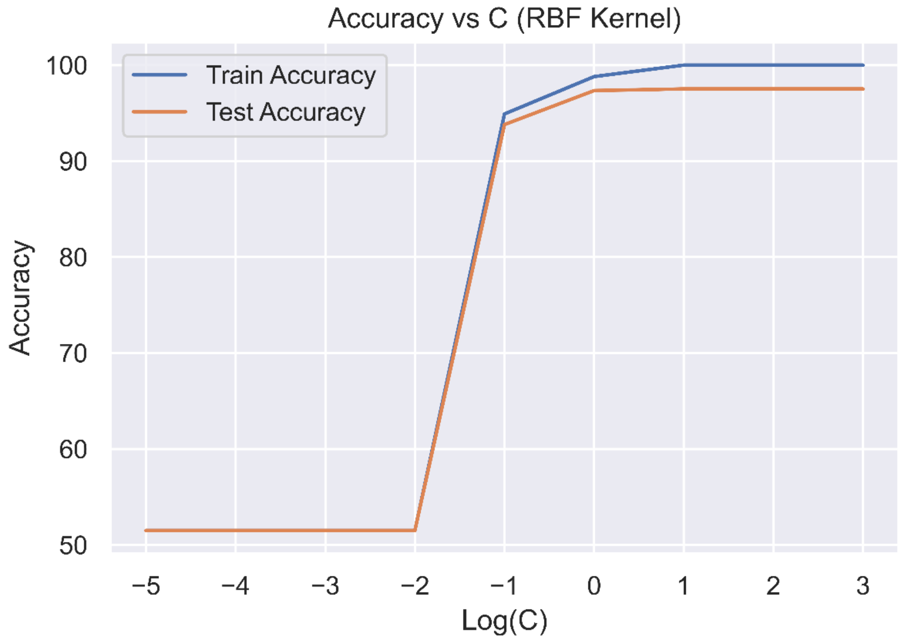
(Accuracy vs C (Linear Kernel) for 25 features)



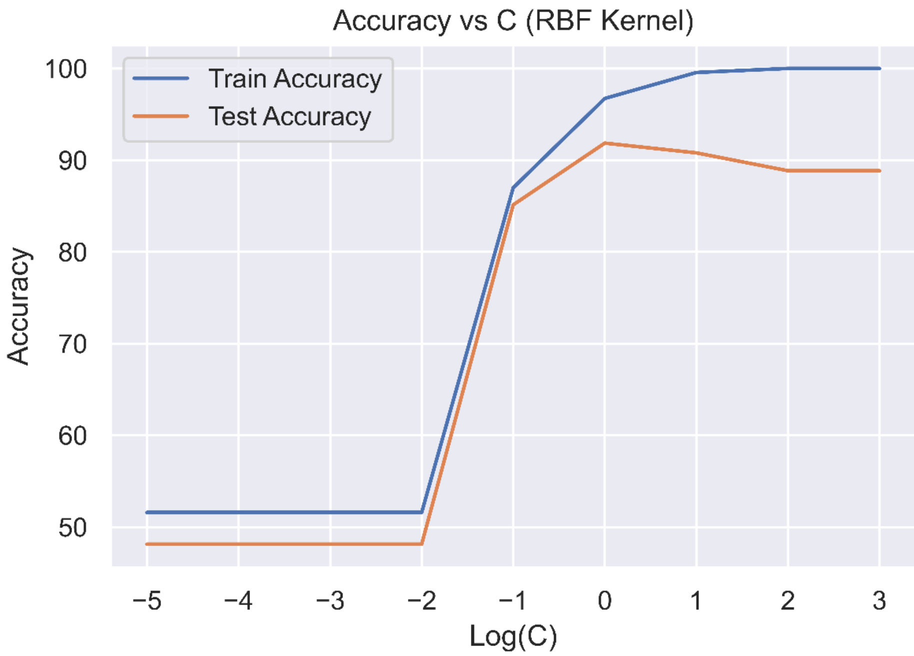
(Accuracy vs C (Linear Kernel) for 10 features)

25 features	10 features	
C < 1	C < 0.1	Underfitting
C = 1	C = 0.1	Best fit
C > 1	C > 0.1	Overfitting

Using **RBF** kernel:



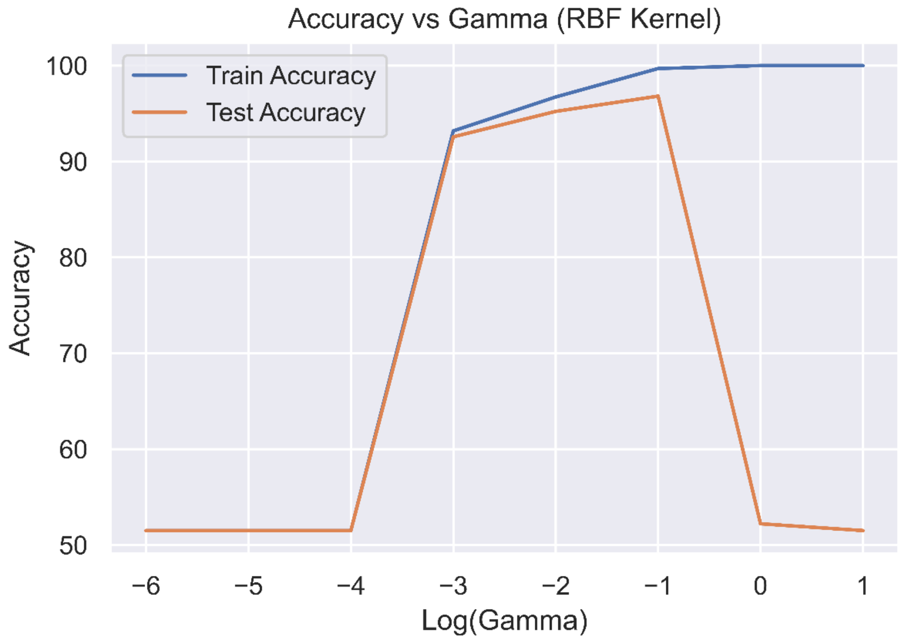
(Accuracy vs C (RBF Kernel) for 25 features)



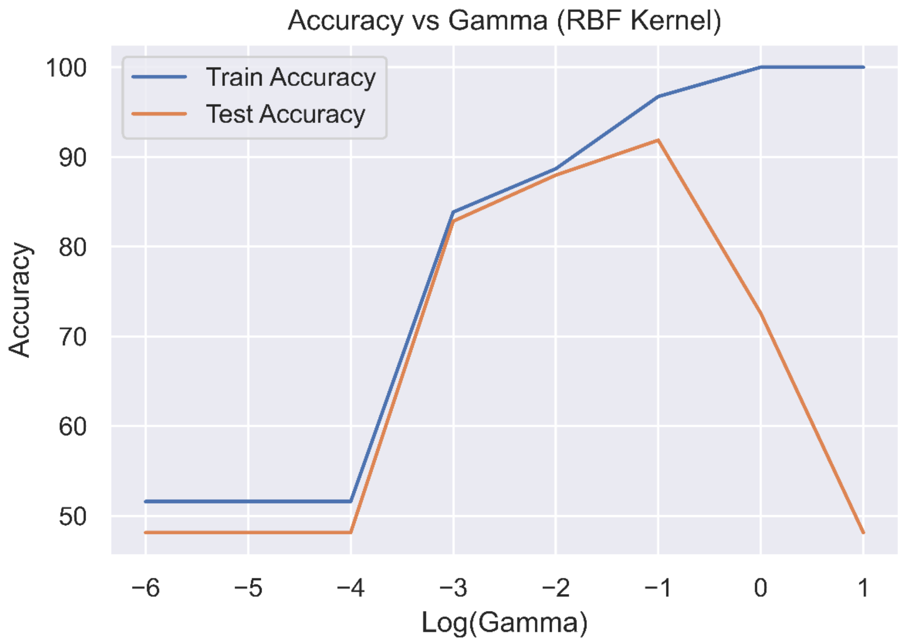
(Accuracy vs C (RBF Kernel) for 10 features)

C < 1	Underfitting
C = 1	Best fit
C > 1	Overfitting

(Consistent results)



(Accuracy vs γ (RBF Kernel) for 25 features)



(Accuracy vs γ (RBF Kernel) for 10 features)

$\gamma < 0.1$	Underfitting
$\gamma = 0.1$	Best fit
$\gamma > 0.1$	Severe Overfitting

(Consistent results)

2.2.4 Comparison of Hyperparameters

	Class C1,C2		
	(1,8)	(3,7)	(4,9)
Linear (C)			
RBF (C)			
RBF (γ)			