ECE GY-6143 ML Homework 3 Solution

Yogya Sharma

(ys5250@nyu.edu)

1.

Answer 1)

(A)

Mercer Kernel $\Rightarrow$ $k(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle$

$$= \begin{cases} \phi(x)^T \phi(\tilde{x}) & \phi \to \text{finite} \\ \int_t \phi(x,t)\, \phi(\tilde{x},t)\, dt \end{cases}$$

Gram Matrix $\Rightarrow$ $K_{i,j} = k(x_i, x_j)$

Mercer kernel that we'll be using is:

$$k(x, \tilde{x}) = \phi(x)^T \phi(\tilde{x})$$

Inputs $S \to \{x_1, x_2, x_3 \cdots x_n\}$

$$\text{Gram Matrix } K = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ \vdots & k(x_2,x_2) & & k(x_2,x_n) \\ \vdots & & \ddots & \vdots \\ k(x_n,x_1) & \cdots & \cdots & k(x_n,x_n) \end{bmatrix}_{n \times n}$$

$$= \begin{bmatrix} \phi(x_1)^T \phi(\tilde{x}_1) & \phi(x_1)^T \phi(\tilde{x}_2) & \cdots & \phi(x_1)^T \phi(\tilde{x}_n) \\ \vdots & \phi(x_2)^T \phi(\tilde{x}_2) & & \phi(x_2)^T \phi(\tilde{x}_n) \\ \vdots & & \ddots & \vdots \\ \phi(x_n)^T \phi(\tilde{x}_1) & \cdots & \cdots & \phi(x_n)^T \phi(\tilde{x}_n) \end{bmatrix}$$

We know that $c \in R^n$, let c be a $n \times 1$, matrix

$$c = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} \quad \text{where } a_i \in R^n$$

$$C^T = \begin{bmatrix} a_1, a_2, a_3 \cdots\cdots a_{n-1}, a_n \end{bmatrix}$$

To prove kernel matrix K is semi definite

$$c^T K c \geq 0$$

$$c^T K = \begin{bmatrix} a_1, a_2 \cdots a_{n-1}, a_n \end{bmatrix} \begin{bmatrix} \phi(x_1)^T \phi(\tilde{x}_1) & \phi(x_1)^T \phi(\tilde{x}_2) \cdots\cdots & \phi(x_1)^T \phi(\tilde{x}_n) \\ \vdots & \phi(x_2)^T \phi(\tilde{x}_2) & \phi(x_2)^T \phi(\tilde{x}_n) \\ \vdots & \ddots & \vdots \\ \phi(x_n)^T \phi(\tilde{x}_1) & \cdots\cdots & \phi(x_n)^T \phi(\tilde{x}_n) \end{bmatrix}$$

$$= \left[\sum_{i=1}^{n} a_i \phi(x_i)^T \phi(\tilde{x}_1) \quad \sum_{i=1}^{n} a_i \phi(x_i)^T \phi(\tilde{x}_2) \quad \cdots\cdots \sum_{i=1}^{n} a_i \phi(x_i)^T \phi(\tilde{x}_n) \right]$$

$$C^T K C = \left[\sum_{i=1}^{n} a_i \phi(x_i)^T \phi(\tilde{x}_1) \quad \sum_{i=1}^{n} a_i \phi(x_i)^T \phi(\tilde{x}_2) \quad \cdots\cdots \quad \sum_{i=1}^{n} a_i \phi(x_i)^T \phi(\tilde{x}_m) \right]_{1\times m} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_m \end{bmatrix}_{m\times 1}$$

$$= \left[a_1 \phi(\tilde{x}_1) \sum_{i=1}^{n} a_i \phi(x_i)^T + a_2 \phi(\tilde{x}_2) \sum_{i=1}^{n} a_i \phi(x_i)^T + \cdots + a_m \phi(\tilde{x}_m) \sum_{i=1}^{n} a_i \phi(x_i)^T \right]_{1\times 1}$$

$$= \left[\sum_{j=1}^{n} a_j \phi(\tilde{x}_j) \sum_{i=1}^{n} a_i \phi(x_i)^T \right]_{1\times 1}$$

$$= \left\langle \sum_{j=1}^{m} a_j \phi(\tilde{x}_j) \sum_{i=1}^{n} a_i \phi(x_i)^T \right\rangle$$

$$= \left\| \sum_{j=1}^{n} a_j \phi(x_j) \right\|_H^2$$ which will always be ≥ 0.

The matrix K, hence, is semi definite, this proves the Mercer Theorem.

(a) $K(x, \tilde{x}) = \alpha K_1(x, \tilde{x}) + \beta K_2(x, \tilde{x})$ for $\alpha, \beta \geq 0$. — (1)

$K(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle = \phi(x)^T \phi(\tilde{x})$
where ϕ is finite

$\alpha K_1(x, \tilde{x}) = \alpha \, \phi_1(x)^T \phi_1(\tilde{x})$

$= \langle \sqrt{\alpha}\, \phi_1(x), \sqrt{\alpha}\, \phi_1(\tilde{x}) \rangle$ — (2)

$\beta K_2(x, \tilde{x}) = \beta \, \phi_2(x)^T \phi_2(\tilde{x})$

$\langle \sqrt{\beta}\, \phi_2(x), \sqrt{\beta}\, \phi_2(\tilde{x}) \rangle$ — (3)

Subsituting values from (2) and (3) in (1)

$K(x, \tilde{x}) = \alpha \, \phi_1(x)^T \phi_1(\tilde{x}) + \beta \, \phi_2(x)^T \phi_2(\tilde{x})$

$= \begin{bmatrix} \sqrt{\alpha}\, \phi_1(x)^T \\ \sqrt{\beta}\, \phi_2(x)^T \end{bmatrix} \begin{bmatrix} \sqrt{\alpha}\, \phi_1(\tilde{x}) & \sqrt{\beta}\, \phi_2(\tilde{x}) \end{bmatrix}$

$\phi_1(x)^T = \phi_1(x)$ as they are symmetric

$= \begin{bmatrix} \sqrt{\alpha}\, \phi_1(x) & \sqrt{\beta}\, \phi_2(x) \end{bmatrix}^T \begin{bmatrix} \sqrt{\alpha}\, \phi_1(\tilde{x}) & \sqrt{\beta}\, \phi_2(\tilde{x}) \end{bmatrix}$

$= \langle [\sqrt{\alpha}\, \phi_1(x)^T \; \sqrt{\beta}\, \phi_2(x)^T], [\sqrt{\alpha}\, \phi(\tilde{x}) \; \sqrt{\beta}\, \phi_2(\tilde{x})] \rangle$

Hence proved that this is also a mercer kernel of type $[\sqrt{\alpha}\, \phi_1(x), \sqrt{\beta}\, \phi_2(x)]$

(b) $k(x, \tilde{x}) = k_1(x, \tilde{x}) \times k_2(x, \tilde{x})$

$k(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle = \phi(x)^T \phi(\tilde{x})$
where ϕ is finite

$k_1(x, \tilde{x}) = \langle \phi_1(x), \phi_2(\tilde{x}) \rangle = \phi_1(x)^T \phi_1(\tilde{x})$
$k_2(x, \tilde{x}) = \langle \phi_2(x), \phi_2(\tilde{x}) \rangle = \phi_2(x)^T \phi_2(\tilde{x})$

$k_1(x, \tilde{x}) \times k_2(x, \tilde{x}) = \phi_1(x)^T \phi_1(\tilde{x}) \; \phi_2(x)^T \phi_1(\tilde{x})$

$= [\phi_1(x)^T \phi_2(x)^T][\phi_1(\tilde{x}) \; \phi_2(\tilde{x})]$

Given that all the above matrices are semi definite and symmetric

$= [\phi_1(x) \; \phi_2(x)]^T [\phi_1(\tilde{x}) \; \phi_2(\tilde{x})]$

$= \langle [\phi_1(x) \phi_2(x)] [\phi_1(\tilde{x}) \phi_2(\tilde{x})] \rangle$

Thus $k_1(x, \tilde{x}) \times k_2(x, \tilde{x})$ can be represented as a valid kernel.

(c) $k(x, \tilde{x}) = f(k_1(x, \tilde{x}))$ where f is any polynomial with positive coefficients.

$k(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle = \phi(x)^T \phi(\tilde{x})$
where ϕ is finite

As we know that f has all positive coefficients

(B) $k(x,y) = \exp\left(-\frac{1}{2}\|x-y\|^2\right) = \varphi(x) \cdot \varphi(y)$

Assuming x and y are scalars,

$$\|x-y\|^2 = (x-y)^2 = x^2 - 2xy + y^2$$

$$k(x,y) = \exp\left(-\frac{1}{2}(x^2 - 2xy + y^2)\right)$$

$$= e^{-x^2/2} e^{-y^2/2} e^{xy}$$

Taylor expansion of $e^x = 1 + \frac{x}{1!} + \frac{x^2}{3!} + \dots \infty$

So, Taylor expansion of e^{xy} can be written as:

$$1 + \frac{xy}{1!} + \frac{(xy)^2}{2!} + \frac{(xy)^3}{3!} \dots \infty$$

This can be represented as:
$$\begin{bmatrix} 1 \\ x \\ x^2/\sqrt{2!} \\ x^3/\sqrt{3!} \\ \vdots \\ \infty \end{bmatrix} \cdot \begin{bmatrix} 1 \\ y \\ y^2/\sqrt{2!} \\ y^3/\sqrt{3!} \\ \vdots \\ \infty \end{bmatrix}$$

$k(x,y)$ can be written as:

$$e^{-x^2/2} \cdot \begin{bmatrix} 1 \\ x \\ x^2/\sqrt{2!} \\ x^3/\sqrt{3!} \\ \vdots \\ \infty \end{bmatrix} \cdot \begin{bmatrix} 1 \\ y \\ y^2/\sqrt{2!} \\ y^3/\sqrt{3!} \\ \vdots \\ \infty \end{bmatrix} \cdot e^{-y^2/2}$$

$$= \begin{bmatrix} e^{-x^2/2} \\ x\, e^{-x^2/2} \\ \dfrac{x^2 e^{-x^2/2}}{\sqrt{2!}} \\ \dfrac{x^3 e^{-x^2/2}}{\sqrt{3!}} \\ \vdots \\ \infty \end{bmatrix} \cdot \begin{bmatrix} e^{-y^2/2} \\ y\, e^{-y^2/2} \\ \dfrac{y^2 e^{-y^2/2}}{\sqrt{2!}} \\ \dfrac{y^3 e^{-x^2/2}}{\sqrt{3!}} \\ \vdots \\ \infty \end{bmatrix} = \varphi(x) \cdot \varphi(y)$$

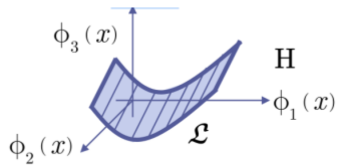
and these can be represented as $\varphi(x) \cdot \varphi(y)$

where $\varphi(x) = \begin{bmatrix} e^{-x^2/2} \\ x\, e^{-x^2/2} \\ \dfrac{x^2 e^{-x^2/2}}{\sqrt{2!}} \\ \dfrac{x^3 e^{-x^2/2}}{\sqrt{3!}} \\ \vdots \\ \infty \end{bmatrix}$ and $\varphi(y) = \begin{bmatrix} e^{-y^2/2} \\ y\, e^{-y^2/2} \\ \dfrac{y^2 e^{-y^2/2}}{\sqrt{2!}} \\ \dfrac{y^3 e^{-x^2/2}}{\sqrt{3!}} \\ \vdots \\ \infty \end{bmatrix}$

2. When we have a nonlinear SVM problem, we can translate the problem from L space to Hilbert space. This can be done using basis functions, phi(x) feature vectors.

For example, a quadratic classifier:

$$x_i \rightarrow \Phi(x_i) \quad via \quad \Phi(\vec{x}) = \begin{bmatrix} \vec{x} \\ vec(\vec{x}\vec{x}^T) \end{bmatrix}$$



This leads the problem to become as:

$$L_D : \max \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \quad s.t. \ \alpha_i \in [0, C], \sum_i \alpha_i y_i = 0$$

Which gives a nonlinear classifier in original space:

f(x) = $sign\left(\sum_i \alpha_i y_i \phi(x)^T \phi(x_i) + b\right)$

This involves solving for the inner product, which can be done using the help of kernels.

Mercer Kernel:

$$k(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle = \begin{cases} \phi(x)^T \phi(\tilde{x}) & for\ finite\ \phi \\ \int_t \phi(x,t)\phi(\tilde{x},t)\,dt & otherwise \end{cases}$$

Here we use linear, polynomial and RBF kernels:

Polynomial Kernel : $k(x, \tilde{x}) = \left(x^T \tilde{x} + 1\right)^p$

RBF Kernel : $k(x, \tilde{x}) = \exp\left(-\frac{1}{2\sigma^2} \left\| x - \tilde{x} \right\|^2\right)$

In python we import svm from the sklearn library, and use different kernels. Train the svm over the training data and then fit the test data over it to get predictions. We then compare these predictions with the true label in the test data and calculate the accuracy of the obtained classifier.

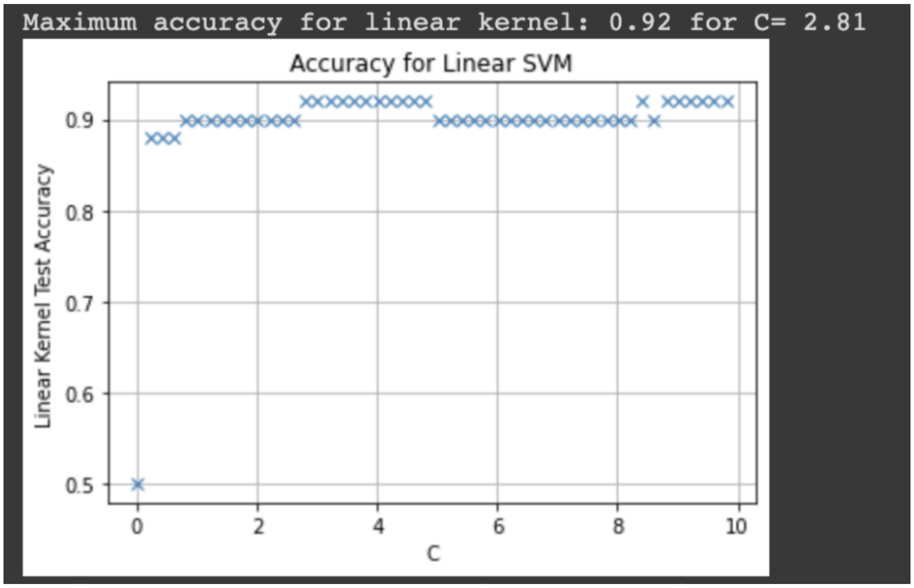
```
import scipy.io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#importing dataset
dataset = scipy.io.loadmat('dataset.mat')

#creating a dataframe
df1 = pd.DataFrame(dataset['X'] , columns = ['x1' , 'x2', 'x3', 'x4'])
df2 = pd.DataFrame(dataset['Y'] , columns = ['y'])

#Extractiong X and Y from the dataframe
X = pd.DataFrame(df1.iloc[ : , :4])
Y = pd.DataFrame(df2.iloc[ : , :1])

#splitting the data into data for train and test
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, random_state = 1, test_size
= 0.5)

test_len = len(Y_Test)

#Flattening the y_train and y_test vectors to pass to svm predict()
fl_Y_Train = np.ravel(Y_Train)
fl_Y_Test = np.ravel(Y_Test)

# List for testing out different C values
C = [0.01*i for i in range(1,1000,20)]

#Linear Kernel
linear_accuracy = list()

for c in C:
  linear_classifier = svm.SVC(kernel = 'linear', C = c, random_state = 1)
  #Training the Linear SVM model
  linear_classifier.fit(X_Train, fl_Y_Train)
  #Predicting using test data
  prediction = linear_classifier.predict(X_Test)
  #Calculating the accuracy
  linear_accuracy.append((np.sum(prediction == fl_Y_Test))/test_len)

#Polynomial Kernel
poly_accuracy = [[] for i in range(4)]
```

```
for c in C:
  for exp in range(2, 6):
    poly_classifier = svm.SVC(kernel = 'poly', C = c, degree = exp, random_state = 1)
    #Training the Polynomial SVM model
    poly_classifier.fit(X_Train, fl_Y_Train)
    #Predicting using test data
    prediction = poly_classifier.predict(X_Test)
    #Calculating the accuracy for each degree of the polynomial
    poly_accuracy[exp-2].append((np.sum(prediction == fl_Y_Test))/test_len)

#RBF Kernel
Gamma = [0.01, 0.1, 0.7, 1, 3, 7, 21]
rbf_accuracy = [[] for i in range(len(Gamma))]

for c in C:
  for sig in range(len(Gamma)):
    rbf_classifier = svm.SVC(kernel = 'rbf', C = c, gamma = Gamma[sig] , random_state
= 1)
    #Training the RBF SVM model
    rbf_classifier.fit(X_Train, fl_Y_Train)
    #Predicting using test data
    prediction = rbf_classifier.predict(X_Test)
    #Calculating the accuracy for each gamma value
    rbf_accuracy[sig].append((np.sum(prediction == fl_Y_Test))/test_len)

#Plotting for Linear Kernel
print('Maximum accuracy for linear kernel:', np.max(linear_accuracy), 'for C=',
C[linear_accuracy.index(np.max(linear_accuracy))] )
fig1, linear = plt.subplots(1, 1)
linear.plot(C, linear_accuracy,'x')
linear.grid()
linear.set(xlabel='C', ylabel='Linear Kernel Test Accuracy')
linear.set_title('Accuracy for Linear SVM')
plt.show()

#Plotting for Polynomial Kernel with degree 2
fig2, poly1 = plt.subplots(1, 1)
print('Maximum accuracy for polynomial kernel with degree: 2
is',np.max(poly_accuracy[0]), 'for C=',
C[poly_accuracy[0].index(np.max(poly_accuracy[0]))])
poly1.plot(C, poly_accuracy[0],'x')
poly1.grid()
poly1.set(xlabel='C', ylabel='Polynomial Kernel Test Accuracy')
poly1.set_title('Accuracy for Polynomial SVM with degree 2')
plt.show()

#Plotting for Polynomial Kernel with degree 3
fig3, poly2 = plt.subplots(1, 1)
```

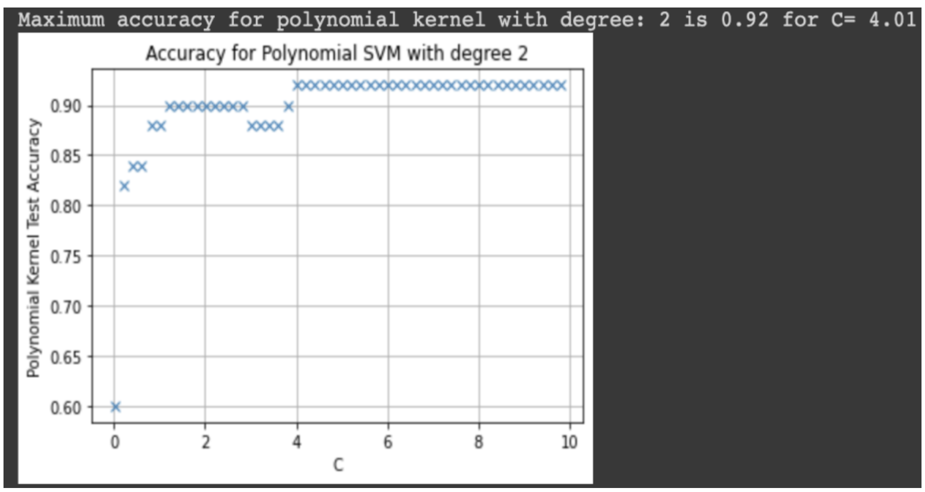
```
print('Maximum accuracy for polynomial kernel with degree: 3
is',np.max(poly_accuracy[1]), 'for C=',
C[poly_accuracy[1].index(np.max(poly_accuracy[1]))])
poly2.plot(C, poly_accuracy[1],'x')
poly2.grid()
poly2.set(xlabel='C', ylabel='Polynomial Kernel Test Accuracy')
poly2.set_title('Accuracy for Polynomial SVM with degree 3')
plt.show()

#Plotting for Polynomial Kernel with degree 4
fig4, poly3 = plt.subplots(1, 1)
print('Maximum accuracy for polynomial kernel with degree: 4
is',np.max(poly_accuracy[2]), 'for C=',
C[poly_accuracy[2].index(np.max(poly_accuracy[2]))])
poly3.plot(C, poly_accuracy[2],'x')
poly3.grid()
poly3.set(xlabel='C', ylabel='Polynomial Kernel Test Accuracy')
poly3.set_title('Accuracy for Polynomial SVM with degree 4')
plt.show()

#Plotting for Polynomial Kernel with degree 5
fig5, poly4 = plt.subplots(1, 1)
print('Maximum accuracy for polynomial kernel with degree: 5
is',np.max(poly_accuracy[3]), 'for C=',
C[poly_accuracy[3].index(np.max(poly_accuracy[3]))])
poly4.plot(C, poly_accuracy[3],'x')
poly4.grid()
poly4.set(xlabel='C', ylabel='Polynomial Kernel Test Accuracy')
poly4.set_title('Accuracy for Polynomial SVM with degree 5')
plt.show()

#Plotting for RBF Kernel gamma 0.01
fig2, rbf1 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 0.01 is',np.max(rbf_accuracy[0]),
'for C=', C[rbf_accuracy[0].index(np.max(rbf_accuracy[0]))])
rbf1.plot(C, rbf_accuracy[0],'x')
rbf1.grid()
rbf1.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf1.set_title('Accuracy for RBF SVM with gamma 0.01')
plt.show()

#Plotting for RBF Kernel gamma 0.1
fig3, rbf2 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 0.1 is',np.max(rbf_accuracy[1]),
'for C=', C[rbf_accuracy[1].index(np.max(rbf_accuracy[1]))])
rbf2.plot(C, rbf_accuracy[1],'x')
rbf2.grid()
rbf2.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf2.set_title('Accuracy for RBF SVM with gamma 0.1')
plt.show()
```

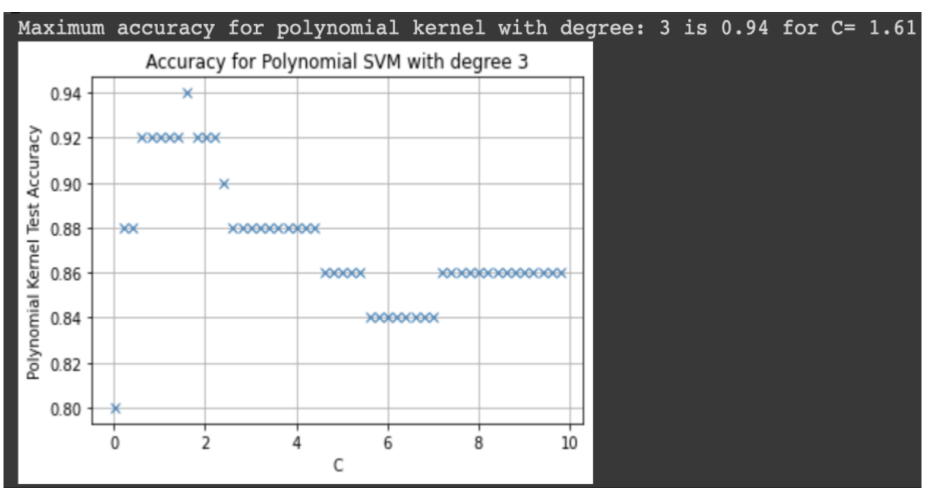
```
#Plotting for RBF Kernel gamma 0.7
fig4, rbf3 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 0.7 is',np.max(rbf_accuracy[2]),
'for C=', C[rbf_accuracy[2].index(np.max(rbf_accuracy[2]))])
rbf3.plot(C, rbf_accuracy[2],'x')
rbf3.grid()
rbf3.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf3.set_title('Accuracy for RBF SVM with gamma 0.7')
plt.show()

#Plotting for RBF Kernel gamma 1
fig5, rbf4 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 1 is',np.max(rbf_accuracy[3]), 'for
C=', C[rbf_accuracy[3].index(np.max(rbf_accuracy[3]))])
rbf4.plot(C, rbf_accuracy[3],'x')
rbf4.grid()
rbf4.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf4.set_title('Accuracy for RBF SVM with gamma 1')
plt.show()

#Plotting for RBF Kernel gamma 3
fig6, rbf5 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 3 is',np.max(rbf_accuracy[4]), 'for
C=', C[rbf_accuracy[4].index(np.max(rbf_accuracy[4]))])
rbf5.plot(C, rbf_accuracy[4],'x')
rbf5.grid()
rbf5.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf5.set_title('Accuracy for RBF SVM with gamma 3')
plt.show()

#Plotting for RBF Kernel gamma 7
fig7, rbf6 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 7 is',np.max(rbf_accuracy[5]), 'for
C=', C[rbf_accuracy[5].index(np.max(rbf_accuracy[5]))])
rbf6.plot(C, rbf_accuracy[5],'x')
rbf6.grid()
rbf6.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf6.set_title('Accuracy for RBF SVM with gamma 7')
plt.show()

#Plotting for RBF Kernel gamma 21
fig7, rbf7 = plt.subplots(1, 1)
print('Maximum accuracy for RBF kernel with gamma 21 is',np.max(rbf_accuracy[6]),
'for C=', C[rbf_accuracy[6].index(np.max(rbf_accuracy[6]))])
rbf7.plot(C, rbf_accuracy[6],'x')
rbf7.grid()
rbf7.set(xlabel='C', ylabel='RBF Kernel Test Accuracy')
rbf7.set_title('Accuracy for RBF SVM with gamma 21')
plt.show()
```

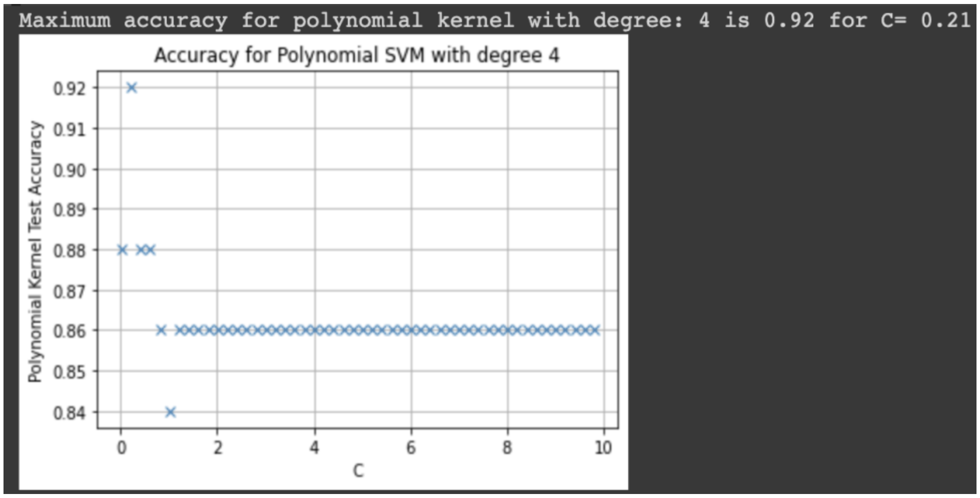
Plot for Linear SVM:

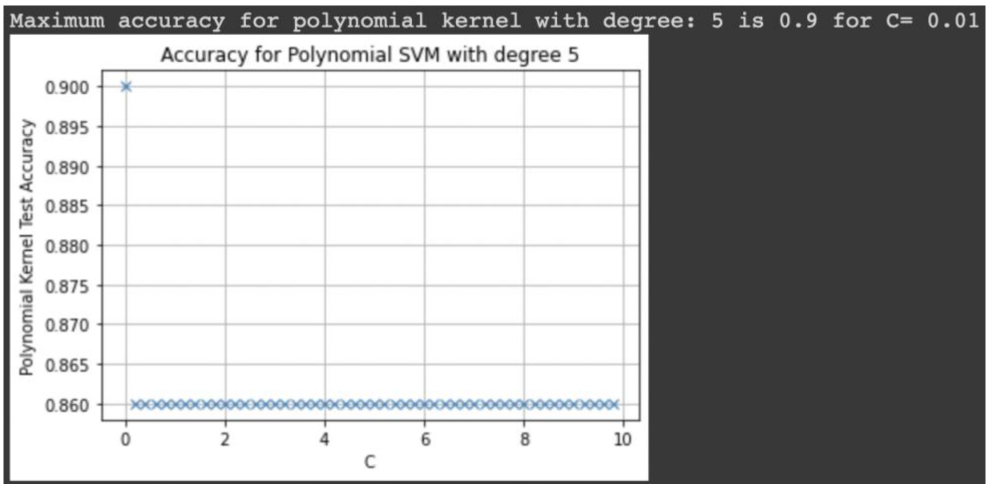


Plots for Polynomial SVM:

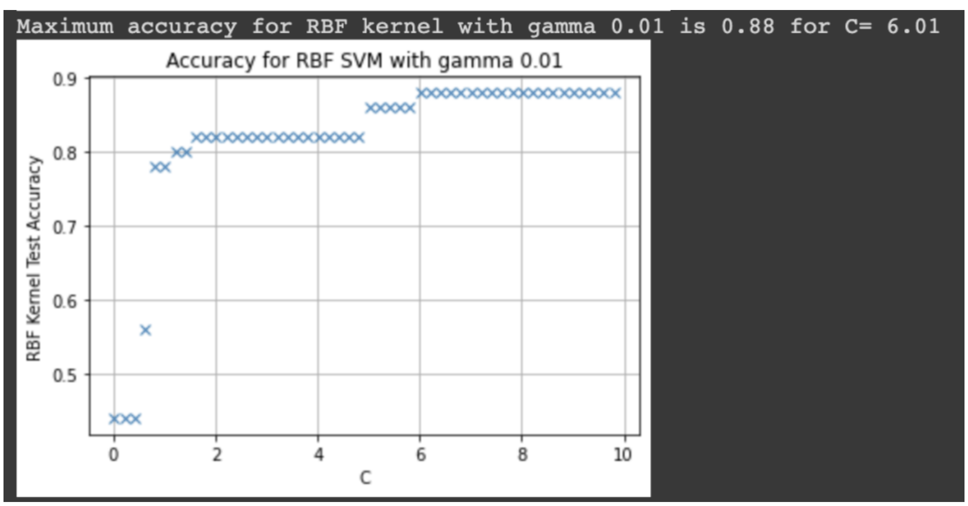




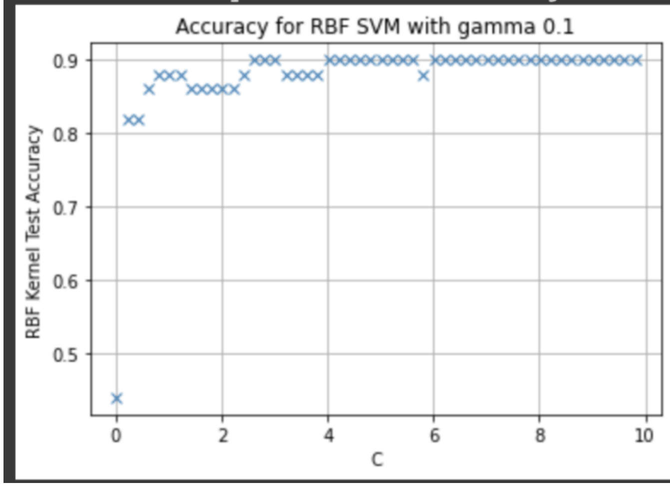




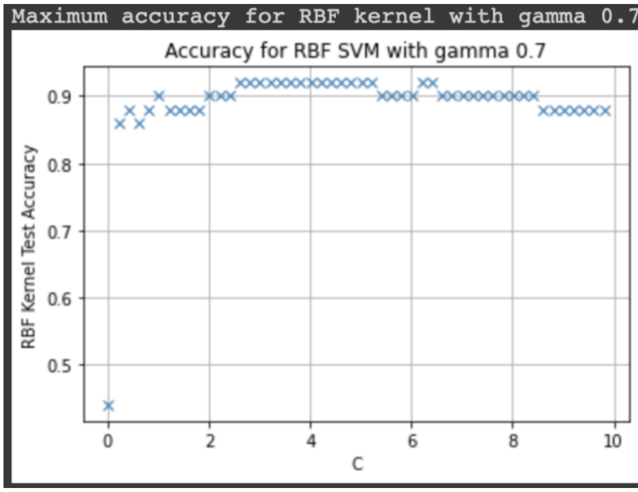
Plots for RBF SVM for various gamma values:



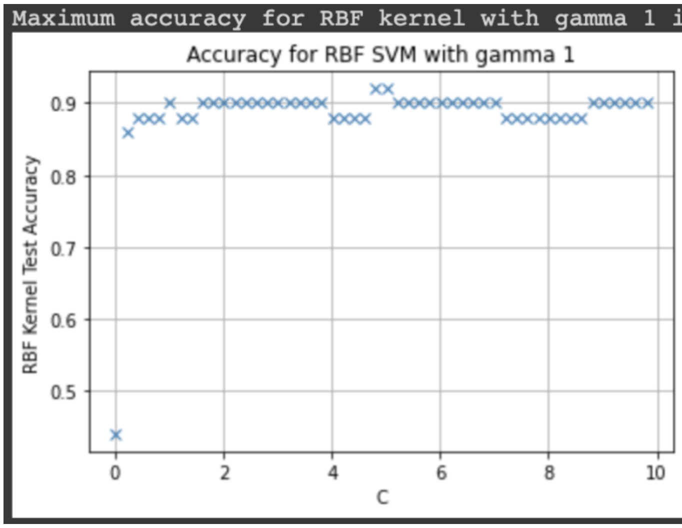
Maximum accuracy for RBF kernel with gamma 0.1 is 0.9 for C= 2.61

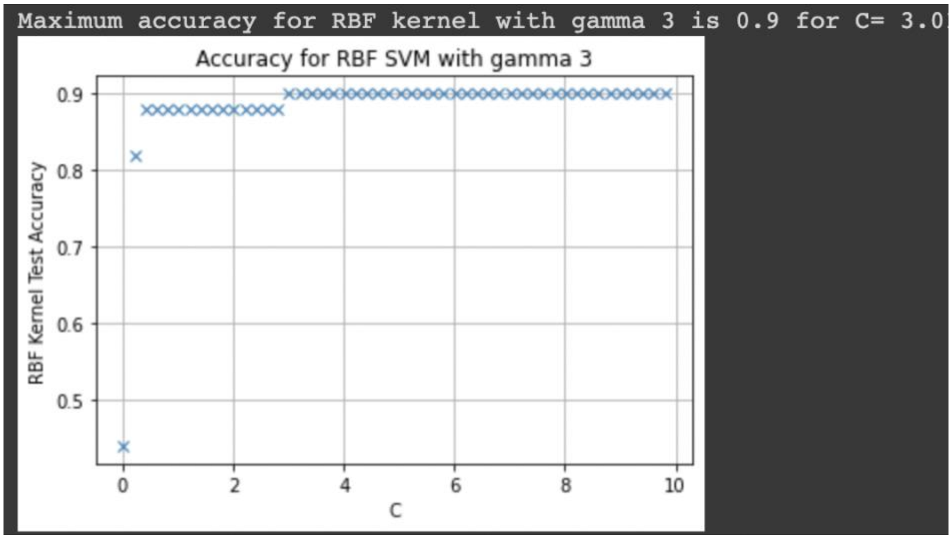


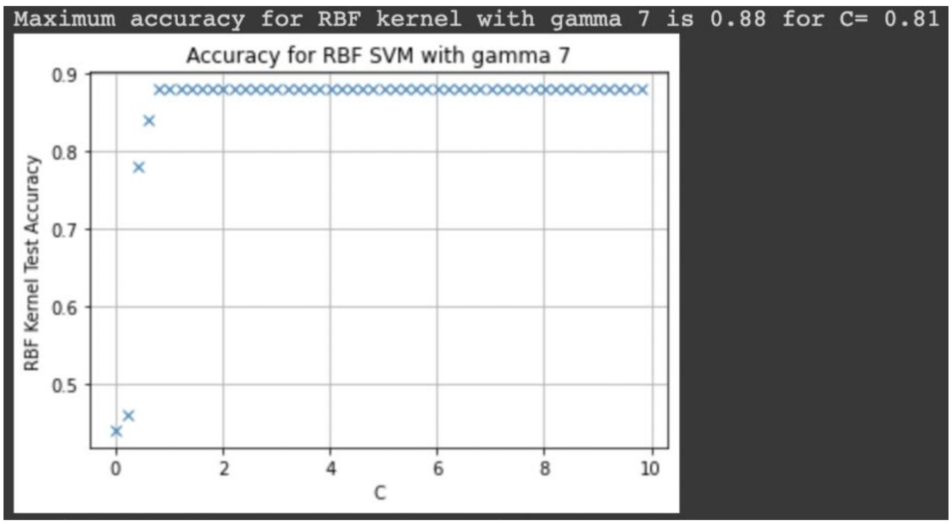
Maximum accuracy for RBF kernel with gamma 0.7 is 0.92 for C= 2.61

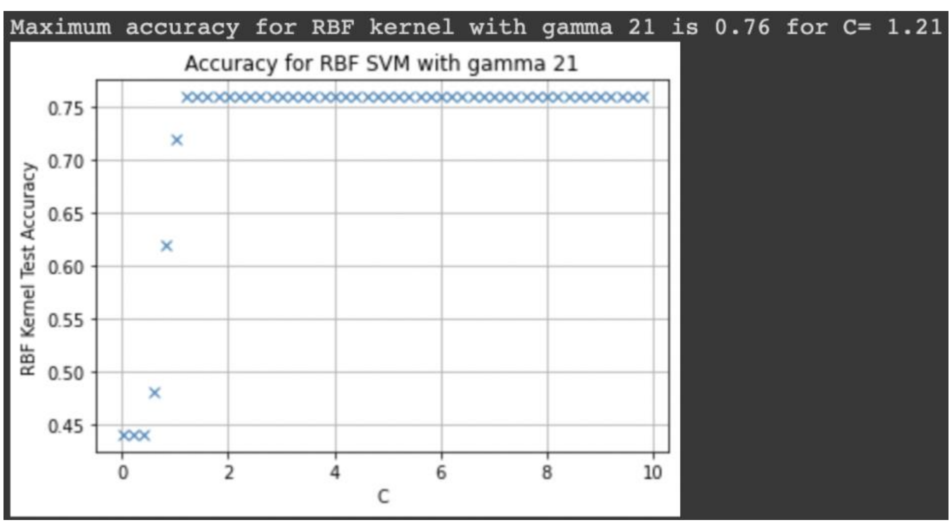


Maximum accuracy for RBF kernel with gamma 1 is 0.92 for C= 4.81









Hence, we have plotted the performance for SVM for Linear Kernel, RBF Kernel for varying values of C and Gamma (Gamma is inversely proportional to sigma) and Polynomial Kernel with varying C values and degrees ranging from 2 to 5.

From the above plots we can see that the maximum accuracy of 0.94 is obtained for the Polynomial Kernel of degree 3.

3.

Answer 3)

$X_1, X_2, X_3 \ldots X_N$ are iid samples from population with the probability distribution function (PDF) as:

$$f(x/\alpha) = \alpha^{-2} x e^{-x/\alpha}$$
where $x > 0$, $\alpha > 0$

To find the maximum likelihood estimator for the given PDF:

$$\ell = \prod_{i=1}^{N} f(x/\alpha) = \left(\alpha^{-2} x_1 e^{-x_1/\alpha}\right)\left(\alpha^{-2} x_2 e^{-x_2/\alpha}\right) \ldots \left(\alpha^{-2} x_N e^{-x_N/\alpha}\right)$$

$$= \alpha^{-2N} e^{\left(-\sum_{i=1}^{N} x_i\right)/\alpha} \prod_{i=1}^{N} x_i$$

Calculating log likelihood, $L = \log(\ell)$

$$L = -2N \log \alpha - \frac{1}{\alpha} \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} \log x_i$$

Differentiating the above equation and equating to 0, to maximise the likelihood.

$$\frac{\partial L}{\partial \alpha} = -\frac{2N}{\alpha} + \frac{1}{\alpha^2} \sum_{i=1}^{N} x_i = 0$$

$$\frac{1}{\alpha^2}\sum_{i=1}^{N} x_i = \frac{2N}{\alpha}$$

$$\alpha = \frac{1}{2N}\sum_{i=1}^{N} x_i \rightarrow (1)$$

This is the maximum likelihood estimator for the given probability distribution function.

$$\frac{\partial L}{\partial \alpha} = -\frac{2N}{\alpha} + \frac{1}{\alpha^2}\sum_{i=1}^{N} x_i$$

$$\frac{\partial^2 L}{\partial \alpha^2} = +\frac{2N}{\alpha^2} - 2 \times \frac{1}{\alpha^3}\sum_{i=1}^{N} x_i$$

Substituting the value of α from (1)

$$= 2N \Big/ \left(\frac{1}{2N}\sum_{i=1}^{N} x_i\right)^2 - 2 \times \sum_{i=1}^{N} x_i \Big/ \left(\frac{1}{2N}\sum_{i=1}^{N} x_i\right)^3$$

$$= -\frac{8N^3}{\left(\sum_{i=1}^{n} x_i\right)^2}$$, which will always be negative.

Now, for $x_1 = 0.25$, $x_2 = 0.75$, $x_3 = 1.50$, $x_4 = 2.5$, $x_5 = 2.0$

the value of α,

$$\alpha = \frac{1}{2N}\sum_{i=1}^{N} x_i \quad , N = 5$$

$$\alpha = \frac{1}{2 \times 5}(0.25 + 0.75 + 1.50 + 2.5 + 2.0)$$

$$\alpha = \frac{7}{10} = 0.7.$$