Foundations of Machine Learning

Sakshi Badole - CS24MTECH11008

September 23, 2024

ASSIGNMENT - 2
Questions: Theory

---

1 Support Vector Machines

In Support Vector Machine, we need to find the maximum margin, and its boundaries are given by:

$$w \cdot x_i + b = +1$$

and

$$w \cdot x_i + b = -1$$

After replacing 1 with γ the new margin boundaries equations become:

$$w \cdot x_i + b = +\gamma$$

and

$$w \cdot x_i + b = -\gamma$$

, where γ is an arbitrary constant such that $\gamma > 0$

We can simply scale the equation such that

$$\frac{w}{\gamma} \cdot x_i + \frac{b}{\gamma} = +1$$

and

$$\frac{w}{\gamma} \cdot x_i + \frac{b}{\gamma} = -1$$

Let, $w' = \dfrac{w}{\gamma}$ and $b' = \dfrac{b}{\gamma}$

The Lagrange problem for this becomes:

$$p^* = \min_{w',b'} \max_{\alpha_i} \frac{1}{2}\|w'\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(w' \cdot x_i + b') - 1\right]$$

$$\text{subject to } \alpha_i \geq 0 \text{ , i = 1,2,..,N}$$

The Dual problem for this becomes:

$$d^* = \max_{\alpha_i} \min_{w',b'} \frac{1}{2}\|w'\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(w' \cdot x_i + b') - 1 \right]$$

$$\text{subject to } \alpha_i \geq 0 \text{ , i = 1,2,..,N}$$

After verifying that the KKT conditions are satisfied, the primal problem and the dual problems are equal, i.e. $d^* = p^*$, and the optimal solution is calculated.

Here,
The derivatives of the Lagrangian with respect to w' and b' are given by:

$$\frac{\partial L}{\partial w'} = w' - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \implies w' = \sum_{i=1}^{N} \alpha_i y_i x_i$$

Next, the derivative with respect to b':

$$\frac{\partial L}{\partial b'} = \sum_{i=1}^{N} \alpha_i y_i = 0 \implies \sum_{i=1}^{N} \alpha_i y_i = 0$$

The optimization problem becomes:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$\text{subject to: } \alpha_i \geq 0 \quad \text{for} \quad i \in [1, N]$$
$$\text{and } \textstyle\sum_{i=1}^{N} \alpha_i y_i = 0$$

Clearly even after replacing -1 and +1 in the equations of the boundary of the margin with $-\gamma$ and $+\gamma$, the solution for the maximum margin hyperplane is unchanged.

---

2 Support Vector Machine

The half-margin of maximum-margin SVM is defined by

$$\rho = \frac{1}{||w||} \tag{2.1}$$

To prove:

$$\frac{1}{\rho^2} = \sum_{i=1}^{N} \alpha_i$$

The Lagrangian problem for hard margin SVM is defined as :

$$p^* = \min_{w,b} \max_{\alpha_i} \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(w \cdot x_i + b) - 1 \right] \tag{2.2}$$

subject to $\alpha_i \geq 0$, i = 1,2,..,N

The Dual problem for hard margin SVM is defined as:

$$d^* = \max_{\alpha_i} \min_{w,b} \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(w \cdot x_i + b) - 1\right] \tag{2.3}$$

subject to $\alpha_i \geq 0$, i = 1,2,..,N

After verifying that the KKT conditions are satisfied, the primal problem and the dual problems are equal, i.e. $d^* = p^*$, and the optimal solution is calculated.

Here,
The derivatives of the Lagrangian with respect to w and b are given by:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{2.4}$$

Next, the derivative with respect to b:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i = 0 \implies \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.5}$$

After substituting these values in the Dual problem and simplifying:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{2.6}$$

subject to: $\alpha_i \geq 0 \quad \text{for} \quad i \in [1, N]$
and $\sum_{i=1}^{N} \alpha_i y_i = 0$

Using the equation $w \cdot x_i + b = y_i$:

$$b = y_i - \sum_{j=1}^{N} \alpha_j y_j (x_i \cdot x_j) \tag{2.7}$$

where $w = \sum_{j=1}^{N} \alpha_j y_j x_j$

Multiply both sides of (equation 2.7) by $\alpha_i y_i$

$$\sum_{i=1}^{N} \alpha_i y_i b = \sum_{i=1}^{N} \alpha_i y_i^2 - \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

Clearly $y_i^2 = 1$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$ (using partial derivative of L w.r.t b) . And $w \cdot w = ||w||^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$

Therefore,

$$0 = \sum_{i=1}^{N} \alpha_i - ||w||^2 \tag{2.8}$$

Hence using equations 2.1 and 2.8 , $\frac{1}{\rho^2} = ||w||^2 = \sum_{i=1}^{N} \alpha_i$

---

3 Kernels

(a) $k(x, z) = k_1(x, z) + k_2(x, z)$
Let the two kernels k_1 and k_2 get representation data from 1 and 2 :

$$k_1(x_1, z_1) + k_2(x_2, z_2) = \phi_1(x_1)^T \phi_1(z_1) + \phi_2(x_2)^T \phi_2(z_2)$$

Now, $x = [x_1, x_2]$, x is the concatenation of the representations x_1 and x_2.
Therefore, $\boldsymbol{\phi_1(x_1)^T \phi_1(z_1) + \phi_2(x_2)^T \phi_2(z_2) = \phi(x)^T \phi(z) = k(x, z)}$

(b) $k(x, z) = k_1(x, z)k_2(x, z)$
Let k_1 and k_2 be the Gram matrices corresponding to the kernels $k_1(x, z)$ and $k_2(x, z)$, respectively.
The kernel $k(x, z) = k_1(x, z) \cdot k_2(x, z)$ corresponds to the Hadamard product (element-wise product of the Gram matrices:

$$k = k_1 \cdot k_2$$

By the Schur product theorem, the Hadamard product of two PSD matrices is also PSD. Therefore, for any $\alpha \in R_n$:

$$\alpha(k_1 \cdot k_2)\alpha \geq 0$$

Hence, the kernel $k(x, z) = k_1(x, z) \cdot k_2(x, z)$ is valid.

(c) $k(x, z) = h(k_1(x, z))$, where h is a polynomial with positive coefficients
Let $k_1(x, z)$ be a valid kernel, so the corresponding Gram matrix k_1 is PSD(positvie semidefinite matrix).
A polynomial function $h(t) = a_0 + a_1 t^2 + ... + a_d t^d$ with positive coefficients applied to kernel $k_1(x, z)$ can be written as:

$$k(x, z) = h(k_1(x, z)) = a_0 + a_1 k_1(x, z) + a_2 k_1(x, z)^2 + ...$$

This corresponds to adding multiple valid kernels, as $k_1(x, z)^i$ is valid for any i (since it corresponds to the Hadamard product of i identical PSD matrices). Therefore, $h(k_1(x, z))$ is also a valis kernel, as it is sum of valid kernels.

(d) $k(x, z) = \exp(k_1(x, z))$
Let $k_1(x, z)$ be a valid kernel, so the corresponding Gram matrix k_1 is positive semi-definite (PSD).

The function $\exp(t)$ is positive and monotonically increasing. Applying this function to the kernel $k_1(x,z)$ gives the kernel

$$k(x,z) = \exp(k_1(x,z)),$$

which corresponds to applying the exponential function element-wise to the Gram matrix.

A result from matrix analysis shows that applying the exponential function element-wise to the entries of a PSD matrix preserves positive semi-definiteness. Therefore, the resulting Gram matrix

$$k = \exp(k_1)$$

is PSD. Hence, $k(x,z) = \exp(k_1(x,z))$ is a valid kernel.

(e) $k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$

This kernel is known as the Gaussian Radial Basis Function (RBF) kernel. It measures similarity based on the distance between x and z. The kernel is given by

$$k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right).$$

The RBF kernel corresponds to a feature mapping into an infinite-dimensional space, where the dot product in this space corresponds to the Gaussian function. By Mercer's theorem, which guarantees that if a kernel can be expressed as a series of dot products in a feature space, it is a valid kernel. Therefore,

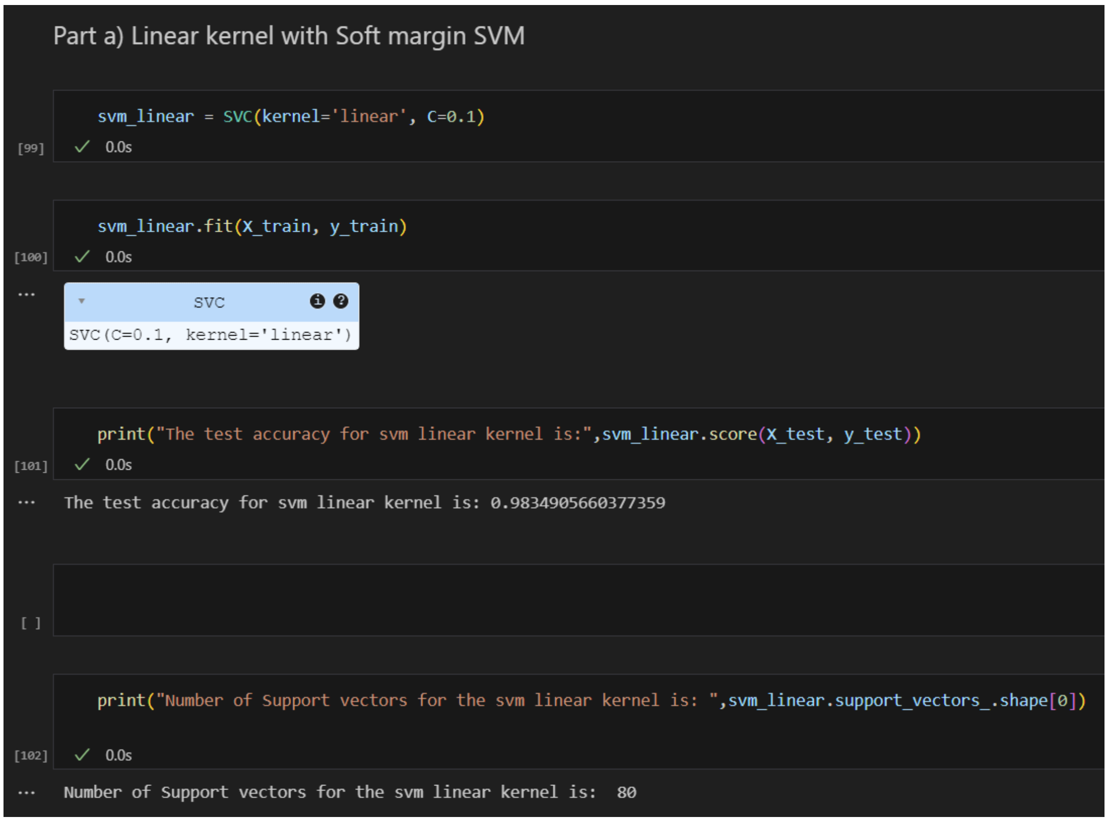
$$k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

is a valid kernel, and the Gram matrix associated with this kernel is positive semi-definite.

Programming Questions

4 SVMs

(a) For the part (a) these are results after running soft-margin SVM with C=0.1 the **test accuracy over the entire dataset is : 0.9834905660377359 or 98.349%** and **the number of support vectors are : 80**



Part a) Linear kernel with Soft margin SVM

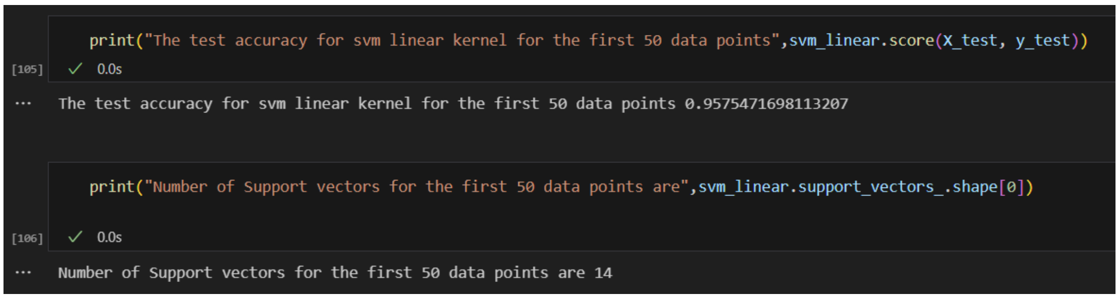
```
[99]  svm_linear = SVC(kernel='linear', C=0.1)
      ✓ 0.0s

[100] svm_linear.fit(X_train, y_train)
      ✓ 0.0s
...   SVC
      SVC(C=0.1, kernel='linear')

[101] print("The test accuracy for svm linear kernel is:",svm_linear.score(X_test, y_test))
      ✓ 0.0s
...   The test accuracy for svm linear kernel is: 0.9834905660377359

[ ]

[102] print("Number of Support vectors for the svm linear kernel is: ",svm_linear.support_vectors_.shape[0])
      ✓ 0.0s
...   Number of Support vectors for the svm linear kernel is:  80
```

Figure 1: Test Accuracy and number of support vectors in SVM with linear kernel

(b) For part (b) the linear SVM is trained on first 50,100,200,800 data points and then tested over the entire test dataset and noted the test accuracy and number of support vectors.

(i) For the **first 50 data points**:

- Test Accuracy: **0.9575471698113207 or 95.754%**
- Number of Support Vectors : **14**

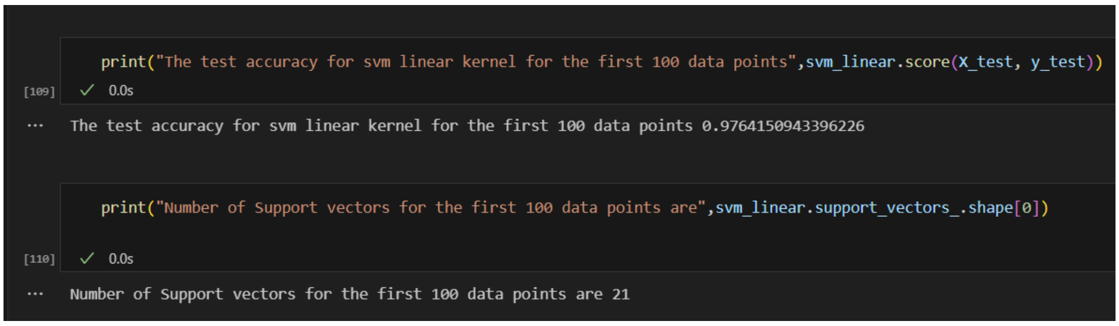


```
[105] print("The test accuracy for svm linear kernel for the first 50 data points",svm_linear.score(X_test, y_test))
      ✓ 0.0s
...   The test accuracy for svm linear kernel for the first 50 data points 0.9575471698113207

[106] print("Number of Support vectors for the first 50 data points are",svm_linear.support_vectors_.shape[0])
      ✓ 0.0s
...   Number of Support vectors for the first 50 data points are 14
```

Figure 2: Test Accuracy and Number of support vectors for first 50 data points

(ii) For the **first 100 data points**:

- Test Accuracy: **0.9764150943396226 or 97.642%**
- Number of Support Vectors : **21**

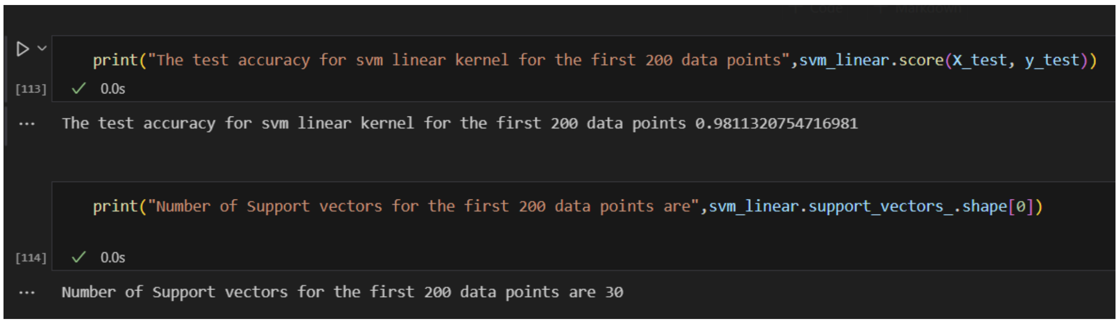


```
[109] print("The test accuracy for svm linear kernel for the first 100 data points",svm_linear.score(X_test, y_test))
      ✓ 0.0s
...   The test accuracy for svm linear kernel for the first 100 data points 0.9764150943396226

[110] print("Number of Support vectors for the first 100 data points are",svm_linear.support_vectors_.shape[0])
      ✓ 0.0s
...   Number of Support vectors for the first 100 data points are 21
```

Figure 3: Test Accuracy and Number of support vectors for first 100 data points

(iii) For the **first 200 data points**:

- Test Accuracy: **0.9811320754716981 or 98.113%**
- Number of Support Vectors : **30**

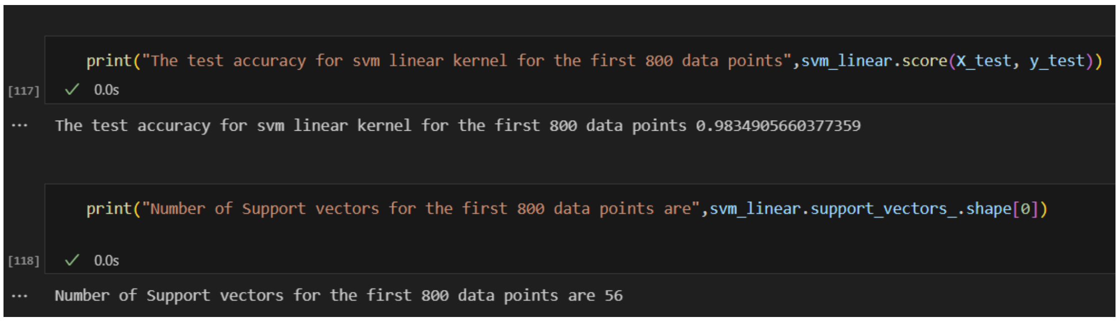


```
[113] print("The test accuracy for svm linear kernel for the first 200 data points",svm_linear.score(X_test, y_test))
      ✓ 0.0s
...   The test accuracy for svm linear kernel for the first 200 data points 0.9811320754716981

[114] print("Number of Support vectors for the first 200 data points are",svm_linear.support_vectors_.shape[0])
      ✓ 0.0s
...   Number of Support vectors for the first 200 data points are 30
```

Figure 4: Test Accuracy and Number of support vectors for first 200 data points

(iv) For the **first 800 data points**:

- Test Accuracy: **0.9834905660377359 or 98.349%**
- Number of Support Vectors : **56**

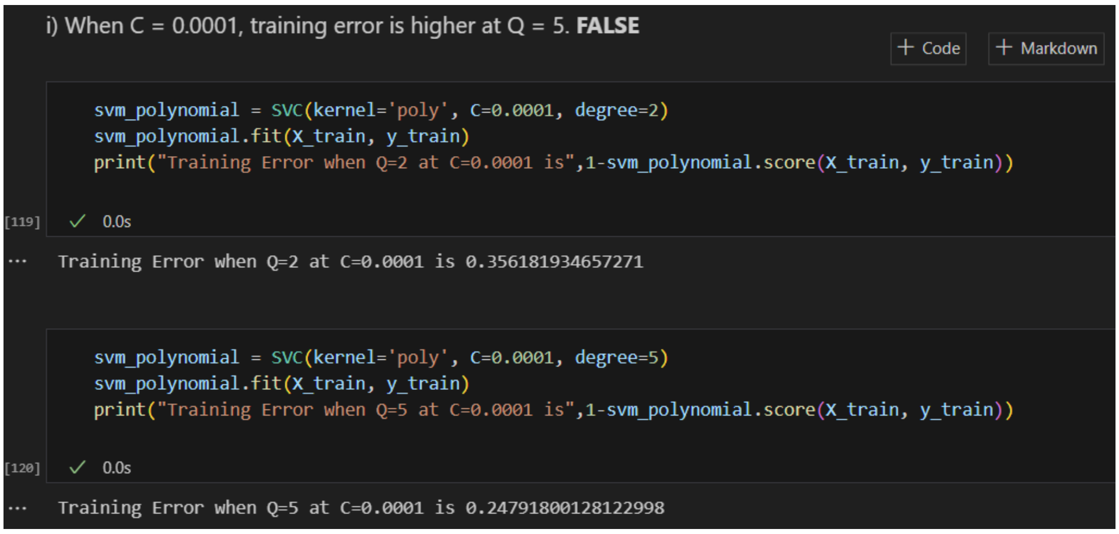


```
[117]
print("The test accuracy for svm linear kernel for the first 800 data points",svm_linear.score(X_test, y_test))
✓ 0.0s
The test accuracy for svm linear kernel for the first 800 data points 0.9834905660377359

[118]
print("Number of Support vectors for the first 800 data points are",svm_linear.support_vectors_.shape[0])
✓ 0.0s
Number of Support vectors for the first 800 data points are 56
```

Figure 5: Test Accuracy and Number of support vectors for first 800 data points

(c) For this part, polynomial kernels with degrees Q =2 and Q=5 are compared for different values of C.

(i) When C = 0.0001, training error is higher at Q = 5. **FALSE**

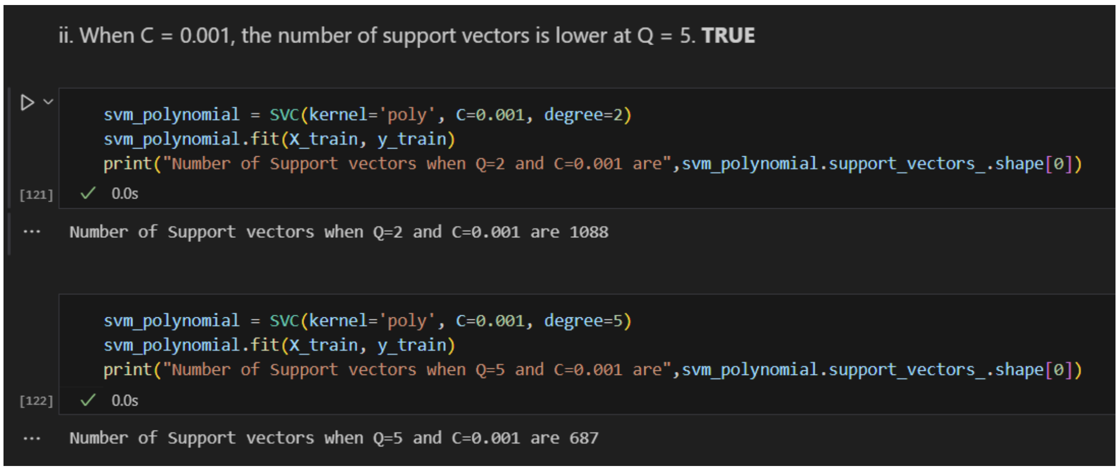


i) When C = 0.0001, training error is higher at Q = 5. **FALSE**

```
[119]
svm_polynomial = SVC(kernel='poly', C=0.0001, degree=2)
svm_polynomial.fit(X_train, y_train)
print("Training Error when Q=2 at C=0.0001 is",1-svm_polynomial.score(X_train, y_train))
✓ 0.0s
Training Error when Q=2 at C=0.0001 is 0.356181934657271

[120]
svm_polynomial = SVC(kernel='poly', C=0.0001, degree=5)
svm_polynomial.fit(X_train, y_train)
print("Training Error when Q=5 at C=0.0001 is",1-svm_polynomial.score(X_train, y_train))
✓ 0.0s
Training Error when Q=5 at C=0.0001 is 0.24791800128122998
```

Figure 6: Training Error for Q=2 and Q=5 at C=0.0001

(ii) When C = 0.001, the number of support vectors is lower at Q = 5. **TRUE**


ii. When C = 0.001, the number of support vectors is lower at Q = 5. **TRUE**

```
svm_polynomial = SVC(kernel='poly', C=0.001, degree=2)
svm_polynomial.fit(X_train, y_train)
print("Number of Support vectors when Q=2 and C=0.001 are",svm_polynomial.support_vectors_.shape[0])
```

[121] ✓ 0.0s

```
Number of Support vectors when Q=2 and C=0.001 are 1088
```

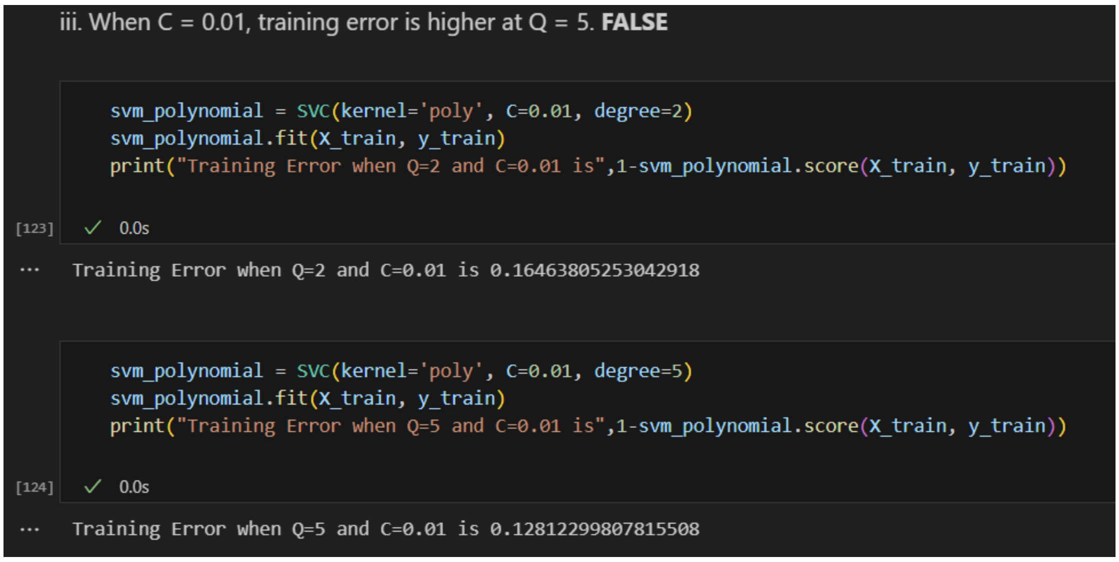
```
svm_polynomial = SVC(kernel='poly', C=0.001, degree=5)
svm_polynomial.fit(X_train, y_train)
print("Number of Support vectors when Q=5 and C=0.001 are",svm_polynomial.support_vectors_.shape[0])
```

[122] ✓ 0.0s

```
Number of Support vectors when Q=5 and C=0.001 are 687
```

Figure 7: Number of support vectors for Q=2 and Q=5 at C=0.001

(iii) When C = 0.01, training error is higher at Q = 5. **FALSE**


iii. When C = 0.01, training error is higher at Q = 5. **FALSE**

```
svm_polynomial = SVC(kernel='poly', C=0.01, degree=2)
svm_polynomial.fit(X_train, y_train)
print("Training Error when Q=2 and C=0.01 is",1-svm_polynomial.score(X_train, y_train))
```

[123] ✓ 0.0s

```
Training Error when Q=2 and C=0.01 is 0.16463805253042918
```

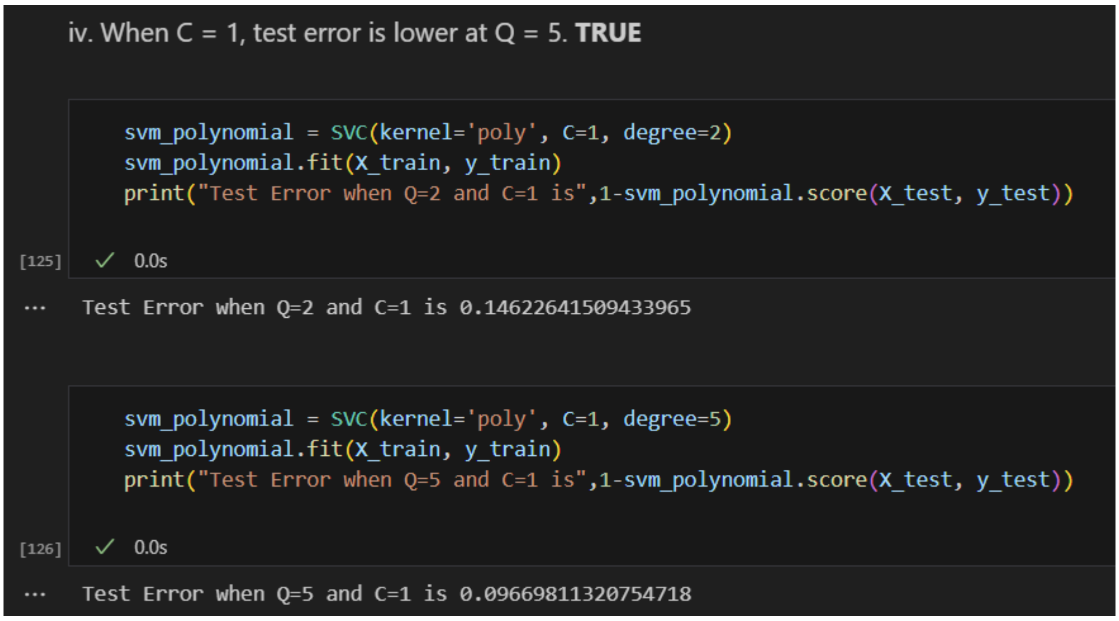
```
svm_polynomial = SVC(kernel='poly', C=0.01, degree=5)
svm_polynomial.fit(X_train, y_train)
print("Training Error when Q=5 and C=0.01 is",1-svm_polynomial.score(X_train, y_train))
```

[124] ✓ 0.0s

```
Training Error when Q=5 and C=0.01 is 0.12812299807815508
```

Figure 8: Training Error for Q=2 and Q=5 at C=0.01

(iv) When C = 1, test error is lower at Q = 5. **TRUE**



iv. When C = 1, test error is lower at Q = 5. **TRUE**

```
svm_polynomial = SVC(kernel='poly', C=1, degree=2)
svm_polynomial.fit(X_train, y_train)
print("Test Error when Q=2 and C=1 is",1-svm_polynomial.score(X_test, y_test))
```

[125] ✓ 0.0s

```
Test Error when Q=2 and C=1 is 0.14622641509433965
```

```
svm_polynomial = SVC(kernel='poly', C=1, degree=5)
svm_polynomial.fit(X_train, y_train)
print("Test Error when Q=5 and C=1 is",1-svm_polynomial.score(X_test, y_test))
```

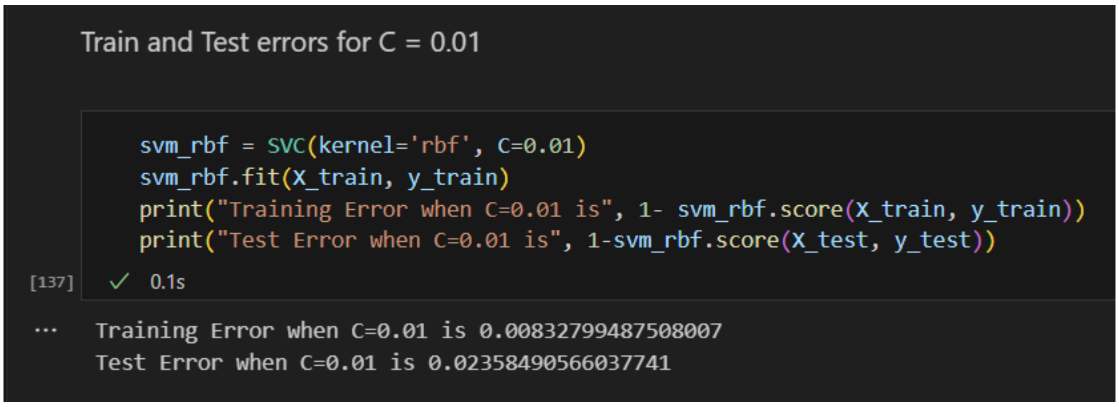
[126] ✓ 0.0s

```
Test Error when Q=5 and C=1 is 0.09669811320754718
```

Figure 9: Test Error for Q=2 and Q=5 at C=1

(d) For this part radial basis kernel function is used in the soft-margin SVM approach and training error and test error is compared for different values of C $\in \{0.01, 1, 100, 10^4, 10^6\}$

- **C=0.01**



Train and Test errors for C = 0.01

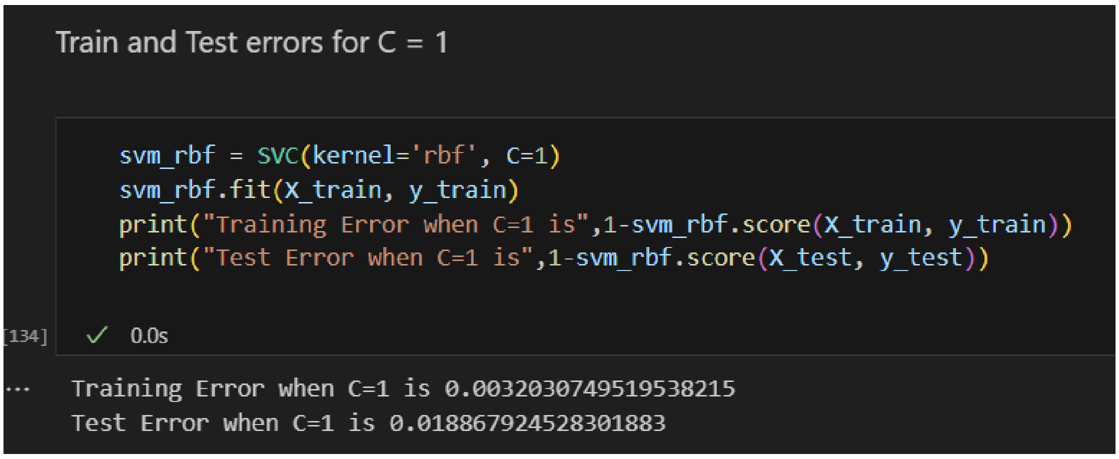
```
svm_rbf = SVC(kernel='rbf', C=0.01)
svm_rbf.fit(X_train, y_train)
print("Training Error when C=0.01 is", 1- svm_rbf.score(X_train, y_train))
print("Test Error when C=0.01 is", 1-svm_rbf.score(X_test, y_test))
```

[137] ✓ 0.1s

```
Training Error when C=0.01 is 0.008327994875080007
Test Error when C=0.01 is 0.02358490566037741
```

Figure 10: Training and Test error for C=0.01

- **C=1**


Train and Test errors for C = 1

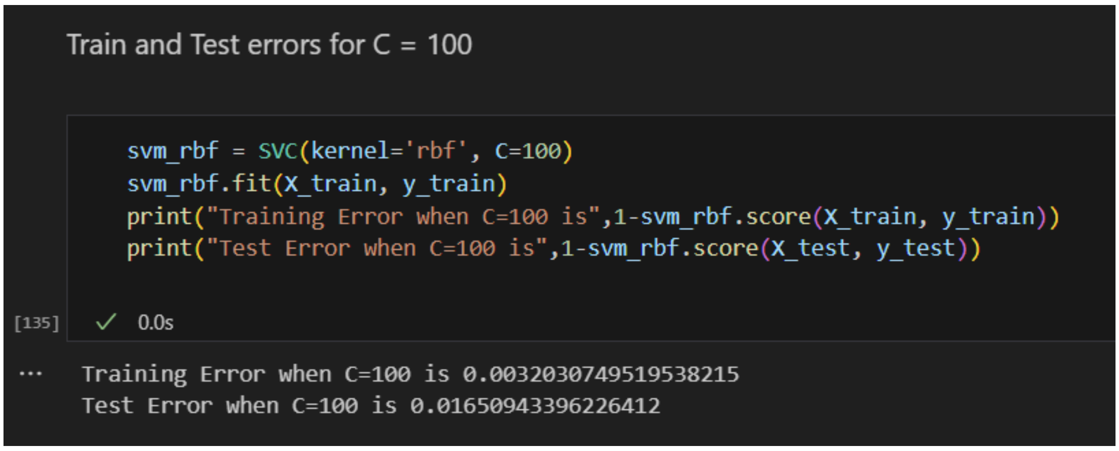
```
svm_rbf = SVC(kernel='rbf', C=1)
svm_rbf.fit(X_train, y_train)
print("Training Error when C=1 is",1-svm_rbf.score(X_train, y_train))
print("Test Error when C=1 is",1-svm_rbf.score(X_test, y_test))
```

[134] ✓ 0.0s

```
Training Error when C=1 is 0.0032030749519538215
Test Error when C=1 is 0.018867924528301883
```

Figure 11: Training and Test error for C=1

- **C=100**


Train and Test errors for C = 100

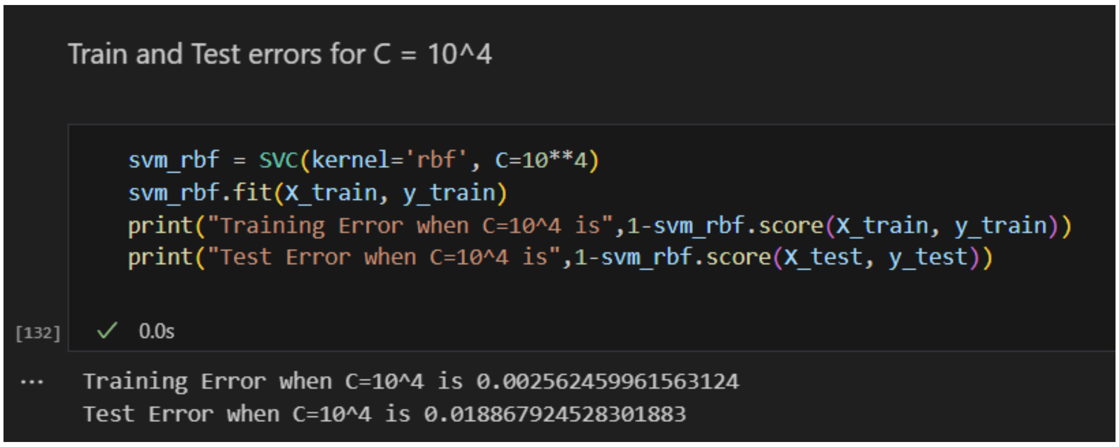
```
svm_rbf = SVC(kernel='rbf', C=100)
svm_rbf.fit(X_train, y_train)
print("Training Error when C=100 is",1-svm_rbf.score(X_train, y_train))
print("Test Error when C=100 is",1-svm_rbf.score(X_test, y_test))
```

[135] ✓ 0.0s

```
Training Error when C=100 is 0.0032030749519538215
Test Error when C=100 is 0.016509433962264412
```

Figure 12: Training and Test error for C=100

- $C = 10^4$


Train and Test errors for C = 10^4

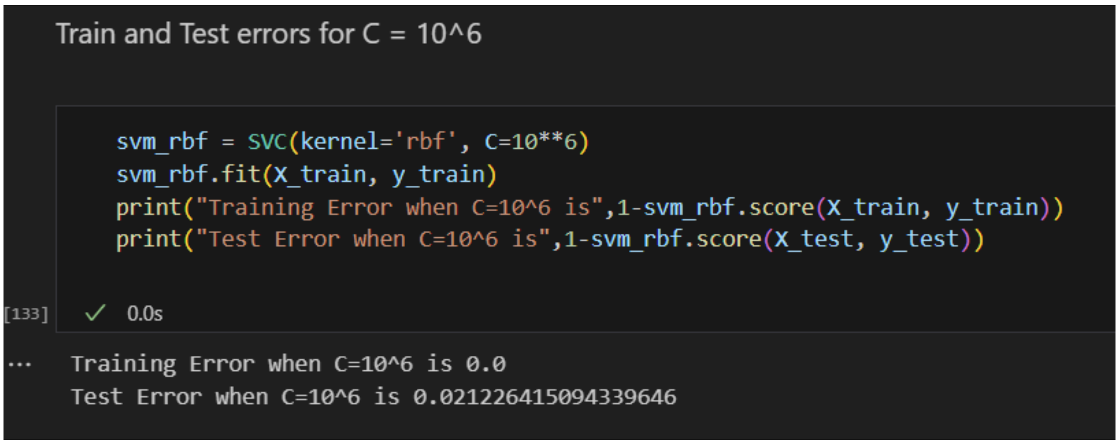
```
svm_rbf = SVC(kernel='rbf', C=10**4)
svm_rbf.fit(X_train, y_train)
print("Training Error when C=10^4 is",1-svm_rbf.score(X_train, y_train))
print("Test Error when C=10^4 is",1-svm_rbf.score(X_test, y_test))
```

[132] ✓ 0.0s

```
Training Error when C=10^4 is 0.002562459961563124
Test Error when C=10^4 is 0.018867924528301883
```

Figure 13: Training and Test error for $C = 10^4$

- $\boldsymbol{C = 10^6}$


Train and Test errors for C = 10^6

```
svm_rbf = SVC(kernel='rbf', C=10**6)
svm_rbf.fit(X_train, y_train)
print("Training Error when C=10^6 is",1-svm_rbf.score(X_train, y_train))
print("Test Error when C=10^6 is",1-svm_rbf.score(X_test, y_test))
```

[133] ✓ 0.0s

```
Training Error when C=10^6 is 0.0
Test Error when C=10^6 is 0.021226415094339646
```

Figure 14: Training and Test error for $C = 10^6$

Table containing summary of running rbf kernel SVM with different C values

Value of C	Training Error	Test Error
0.01	0.0083279	0.0235849
1	0.0032030	0.0188679
100	0.0032030	**0.0165094**
10^4	0.0025624	0.0188679
10^6	**0.0**	0.0212264

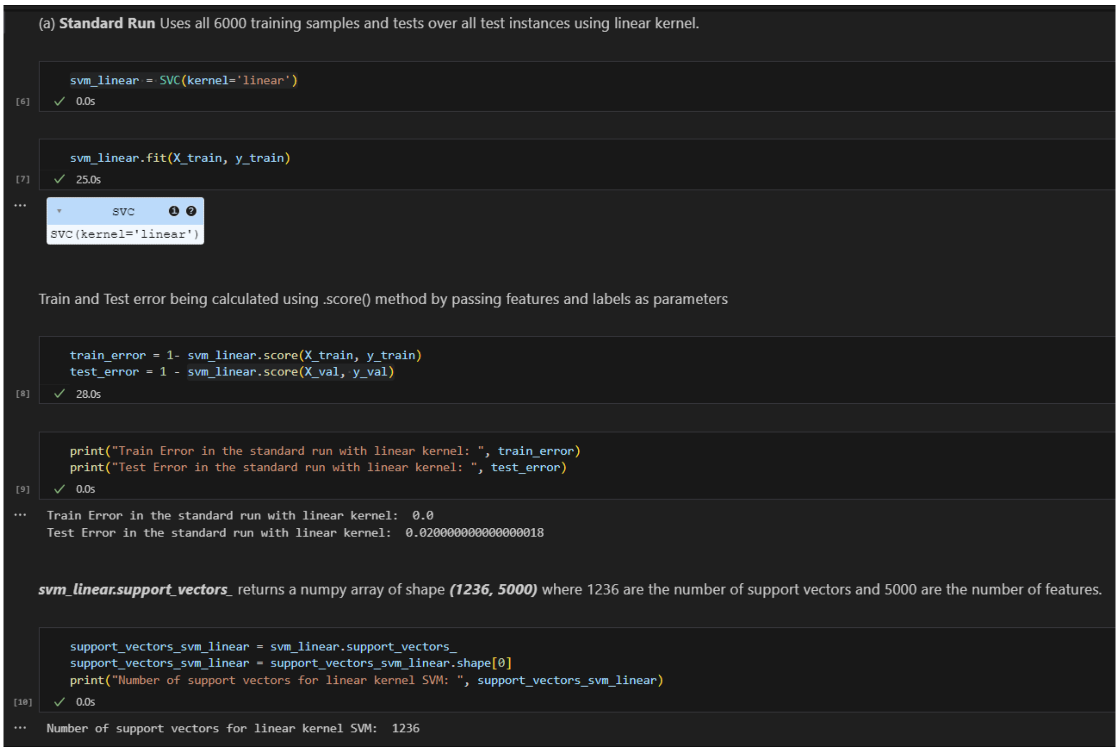
Lowest Training Error is for $\boldsymbol{C = 10^6}$
Lowest Test Error is for $\boldsymbol{C = 100}$

5 SVMs conti.

Here the GISETTE dataset is used for Handwritten digit recognition problem(for digits 4 and 9)

(a) **Standard run:** Here a linear kernel SVM is trained over the entire dataset of 6000 data samples and then tested over the test dataset consisting of 1000 data samples.
Linear Kernel Report:

- **Training Error: 0.0**
- **Test Error: 0.020**
- **No. of support vectors: 1236**



(a) **Standard Run** Uses all 6000 training samples and tests over all test instances using linear kernel.

```
[6]  svm_linear = SVC(kernel='linear')
     ✓ 0.0s
```

```
[7]  svm_linear.fit(X_train, y_train)
     ✓ 25.0s
...  SVC
     SVC(kernel='linear')
```

Train and Test error being calculated using .score() method by passing features and labels as parameters

```
[8]  train_error = 1- svm_linear.score(X_train, y_train)
     test_error = 1 - svm_linear.score(X_val, y_val)
     ✓ 28.0s
```

```
[9]  print("Train Error in the standard run with linear kernel: ", train_error)
     print("Test Error in the standard run with linear kernel: ", test_error)
     ✓ 0.0s
...  Train Error in the standard run with linear kernel:  0.0
     Test Error in the standard run with linear kernel:  0.020000000000000018
```

svm_linear.support_vectors_ returns a numpy array of shape ***(1236, 5000)*** where 1236 are the number of support vectors and 5000 are the number of features.

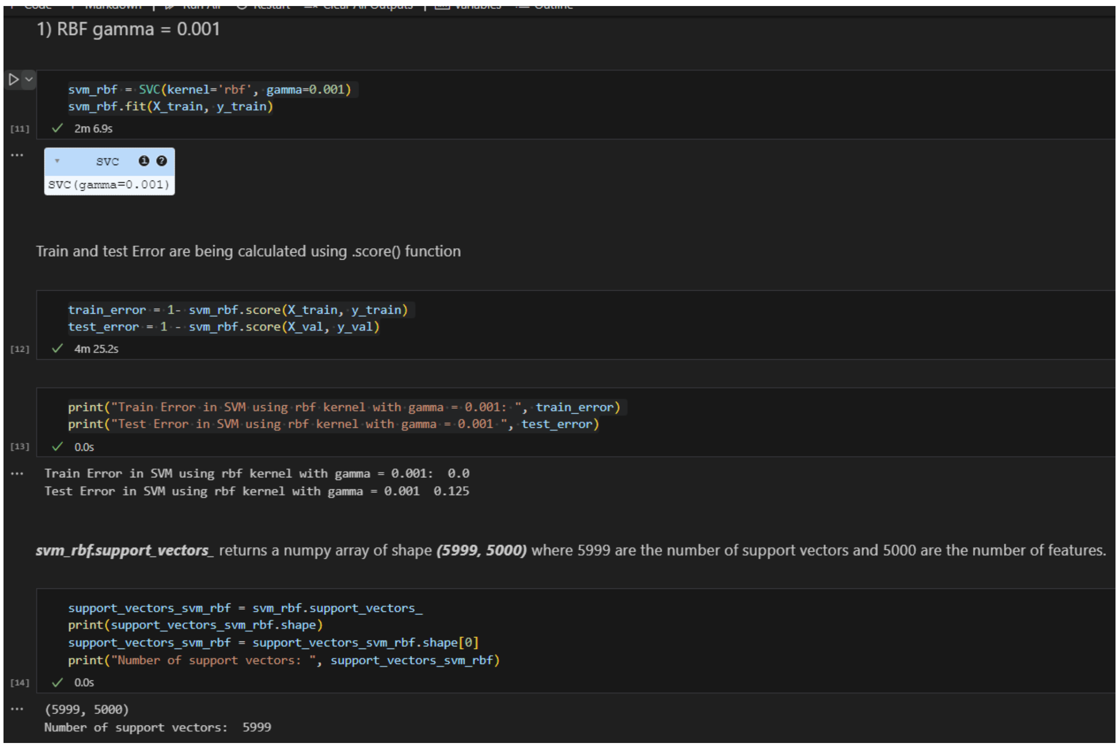
```
[10] support_vectors_svm_linear = svm_linear.support_vectors_
     support_vectors_svm_linear = support_vectors_svm_linear.shape[0]
     print("Number of support vectors for linear kernel SVM: ", support_vectors_svm_linear)
     ✓ 0.0s
...  Number of support vectors for linear kernel SVM:  1236
```

Figure 15: Linear Kernel SVM training error, test error and no. of support vectors.

(b) **Kernel Variations:** Here two different kernel variations are tried:

(i) **RBF (Gaussian) Kernel**: Gamma $\gamma = 0.001$
Gaussian Kernel Report:

- **Training Error: 0.0**
- **Test Error: 0.125**
- **No. of support vectors: 5999**



1) RBF gamma = 0.001

```
svm_rbf = SVC(kernel='rbf', gamma=0.001)
svm_rbf.fit(X_train, y_train)
```

```
SVC(gamma=0.001)
```

Train and test Error are being calculated using .score() function

```
train_error = 1- svm_rbf.score(X_train, y_train)
test_error = 1 - svm_rbf.score(X_val, y_val)
```

```
print("Train Error in SVM using rbf kernel with gamma = 0.001: ", train_error)
print("Test Error in SVM using rbf kernel with gamma = 0.001 ", test_error)
```

```
Train Error in SVM using rbf kernel with gamma = 0.001:  0.0
Test Error in SVM using rbf kernel with gamma = 0.001  0.125
```

svm_rbf.support_vectors_ returns a numpy array of shape ***(5999, 5000)*** where 5999 are the number of support vectors and 5000 are the number of features.

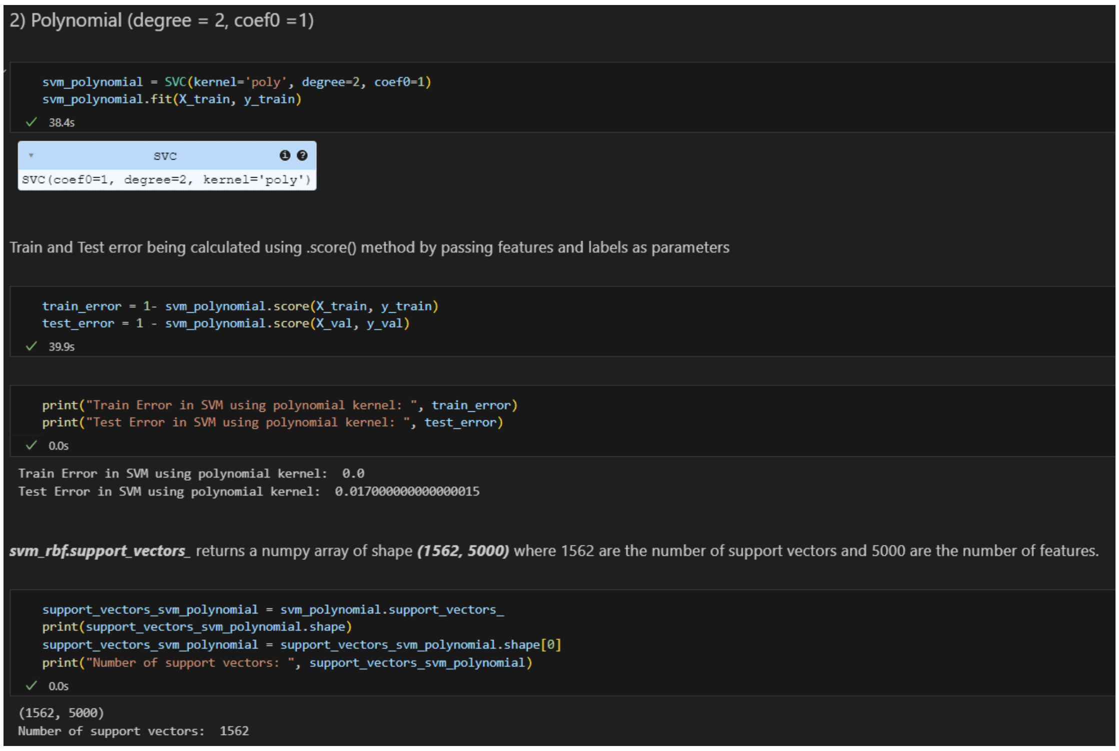
```
support_vectors_svm_rbf = svm_rbf.support_vectors_
print(support_vectors_svm_rbf.shape)
support_vectors_svm_rbf = support_vectors_svm_rbf.shape[0]
print("Number of support vectors: ", support_vectors_svm_rbf)
```

```
(5999, 5000)
Number of support vectors:  5999
```

Figure 16: Gaussian Kernel SVM training error, test error and no. of support vectors.

(ii) **Polynomial Kernel**: degree = 2 and coef0 = 1
Polynomial Kernel Report:

- **Training Error: 0.0**
- **Test Error: 0.017**
- **No. of support vectors: 1562**



2) Polynomial (degree = 2, coef0 =1)

```
svm_polynomial = SVC(kernel='poly', degree=2, coef0=1)
svm_polynomial.fit(X_train, y_train)
```
✓ 38.4s

```
SVC(coef0=1, degree=2, kernel='poly')
```

Train and Test error being calculated using .score() method by passing features and labels as parameters

```
train_error = 1- svm_polynomial.score(X_train, y_train)
test_error = 1 - svm_polynomial.score(X_val, y_val)
```
✓ 39.9s

```
print("Train Error in SVM using polynomial kernel: ", train_error)
print("Test Error in SVM using polynomial kernel: ", test_error)
```
✓ 0.0s

```
Train Error in SVM using polynomial kernel:  0.0
Test Error in SVM using polynomial kernel:  0.017000000000000015
```

svm_rbf.support_vectors_ returns a numpy array of shape ***(1562, 5000)*** where 1562 are the number of support vectors and 5000 are the number of features.

```
support_vectors_svm_polynomial = svm_polynomial.support_vectors_
print(support_vectors_svm_polynomial.shape)
support_vectors_svm_polynomial = support_vectors_svm_polynomial.shape[0]
print("Number of support vectors: ", support_vectors_svm_polynomial)
```
✓ 0.0s

```
(1562, 5000)
Number of support vectors:  1562
```

Figure 17: Polynomial Kernel SVM training error, test error and no. of support vectors.