CS5691 Pattern Recognition and Machine Learning - Programming Assignment 2

Name: Atishay G & Aditya S
Roll Numbers: EE17B155 & EE17B102 **Team - 8**

December 28, 2020

1 Assignment

1.0 Implementation Details

We use Python 3 for experimenting the various ML methods. We use numpy for all the matrix and vector calculations and Matplotlib for the plotting. We used sklearn for the support vector machines.

1.1 Part A

1.1.1 Kernel Ridge Regression

This problem required us to fit a Ridge Regression model to the given data using both linear and polynomial kernels and to compare their performance.

The usual Ridge Regression solution is given by:

$$Y_{pred} = X_{test}^T W \tag{1}$$

$$W = (XX^T + \lambda I)^{-1} XY \tag{2}$$

where X is the input feature matrix of shape Mx N where N is the number of data points and we have M input features and W is the weight matrix.

This can be rewritten as:

$$Y_{pred} = X_{test}^T W = X_{test}^T (XX^T + \lambda I)^{-1} XY \tag{3}$$

We will now use the following result to change the above equation:

$$(XX^T + \lambda I)X = XX^T X + \lambda X = X(X^T X + \lambda I) \tag{4}$$

Which implies that (since the matrices after adding λ are invertible),

$$(XX^T + \lambda I)^{-1} X = X(X^T X + \lambda I)^{-1} \tag{5}$$

And therefore,

$$Y_{pred} = X_{test}^T X (X^T X + \lambda I)^{-1} Y \tag{6}$$

We can now use the kernel trick to replace all inner products in the above equation with the required kernel:

$$Y_{pred} = K(X_{test}, X)(K(X, X) + \lambda I)^{-1} Y \tag{7}$$

where K(A,B) is the inner product calculated in the higher dimensional space after applying the kernel. For example, the linear kernel corresponds to $K(A, B) = A^T B + c$ where c is a constant

We can now use the above equation to train our model. The results are given below:

Kernel	Training Error	Testing Error
Linear Kernel	24.167	29.187
Polynomial Kernel	4.557	156.091

The hyperparameters like the regularisation parameter λ and the constant in the kernels were varied for best performance. The value of λ for the linear kernel was 20 and 0.1 for the polynomial kernel. The constant was 1 for both kernels. The degree of the polynomial kernel used was 2.

The above results can be interpreted as follows: The polynomial kernel having more degrees of freedom than the linear kernel is able to fit the training data much better and hence has lesser training error. However, the polynomial kernel has hence grossly overfit the training data and this is apparent from the its testing error which is much higher than that of the linear kernel. On the other hand the training and testing error are quite similar for the linear kernel.

Thus we can conclude that the best kernel was the linear kernel as it has performed better on the test data (while the polynomial kernel overfit the training data and performed poorly on the test data)

1.1.2 Kernel Logistic Regression and Kernel SVM

For this problem, we are required to build a SVM and Logistic Regression classifier using both linear and polynomial kernels for the given dataset.

For SVM, the in-build sklearn SVM function was used, while for logistic regression we follow the approach given below:

For logistic regression, we solve the problem iteratively by repeating the following steps until convergence.

$$W_{k+1} = (A^T RA)^{-1} A^T Rb \tag{8}$$

Where A is the input feature matrix of shape (NxM) where the data is M-dimensional and we have N data points. W is the weight matrix of shape (Mx1). R is the (NxN) diagonal matrix with the i^{th} element given by $z_i(1 - z_i)$

where z_i is the i^{th} element of z given by

$$z_i = \sigma(AW_k) \tag{9}$$

and

$$b = AW_k - R^{-1}(z - y) \tag{10}$$

We will now write the above update equation in terms of inner products of data points (like we did in Ridge Regression) so that we can use the Kernel Trick.

Using the following,

$$(A^T RA)A^T R = A^T R(AA^T R) \tag{11}$$

And thus, by taking inverse (assuming R has no zero diagonal elements,

$$(A^T RA)^{-1} A^T R = A^T R(AA^T R)^{-1} \tag{12}$$

Hence, we can write

$$W_{k+1} = A^T R(K(A, A)R)^{-1} b = A^T \alpha \tag{13}$$

where K(A,B) is the kernel inner product evaluated in the higher dimensional space. For example, the linear kernel would give $K(A, B) = AA^T + c$ where c is a constant. (Note that the convention is different from the Ridge Regression case following the given reference)

And,

$$z = \sigma(AW_k) = \sigma(AA^T \alpha) = \sigma(K(A, A)\alpha) \tag{14}$$

$$b = AW_k - R^{-1}(z - y) = K(A, A)\alpha - R^{-1}(z - y) \tag{15}$$

After convergence, the final prediction is given by:

$$Y_{pred} = \sigma(AW) = \sigma(A_{test} A^T \alpha) = \sigma(K(A_{test}, A)\alpha) \tag{16}$$

We will now fit the SVM and Logistic Regression models to the training data. Since logistic regression only outputs the final probability, we will threshold it to predict any data point with output greater than 0.5 as belonging to class 1 and less than 0.5 as belonging to class 0

Model	Training Accuracy	Testing Accuracy
SVM Linear Kernel	0.531	0.545
SVM Polynomial Kernel	0.991	1.0
Logistic Regression Linear Kernel	0.522	0.52
Logistic Regression Polynomial Kernel	0.995	1.0

The decision boundary plots can be seen below:

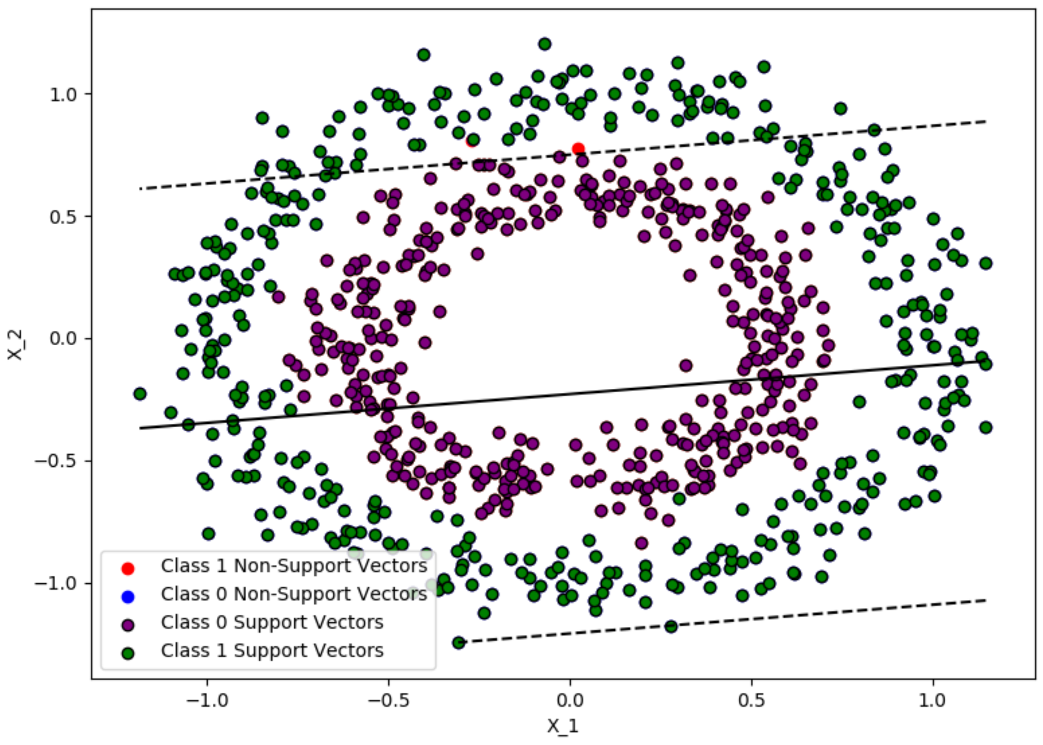

Figure 1: SVM with Linear Kernel

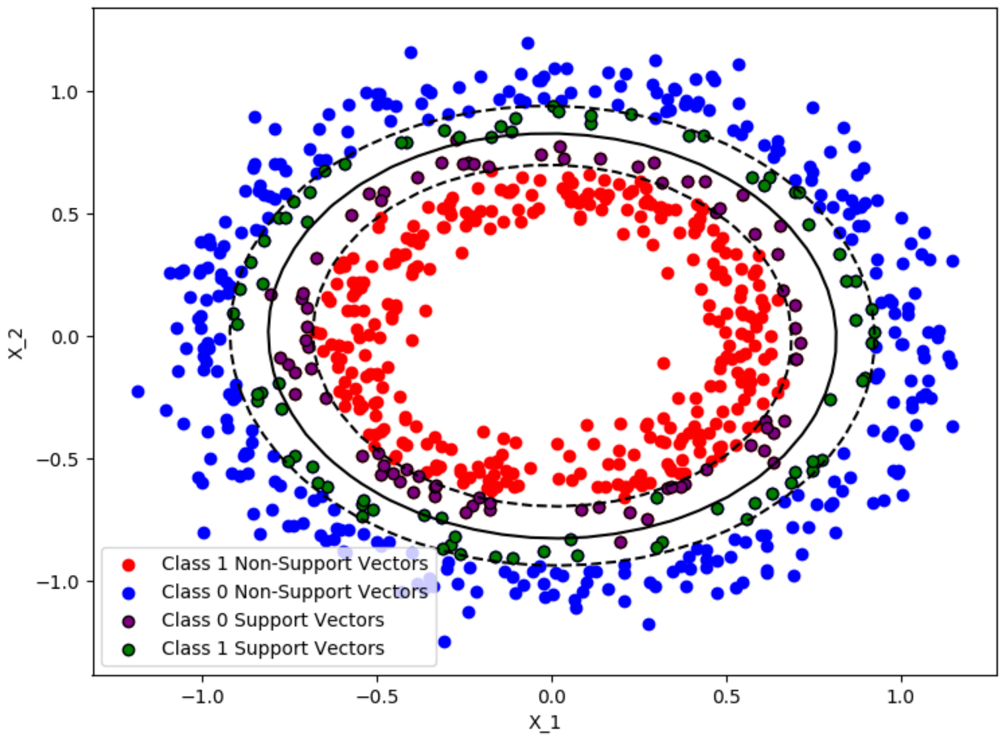

Figure 2: SVM with Polynomial Kernel

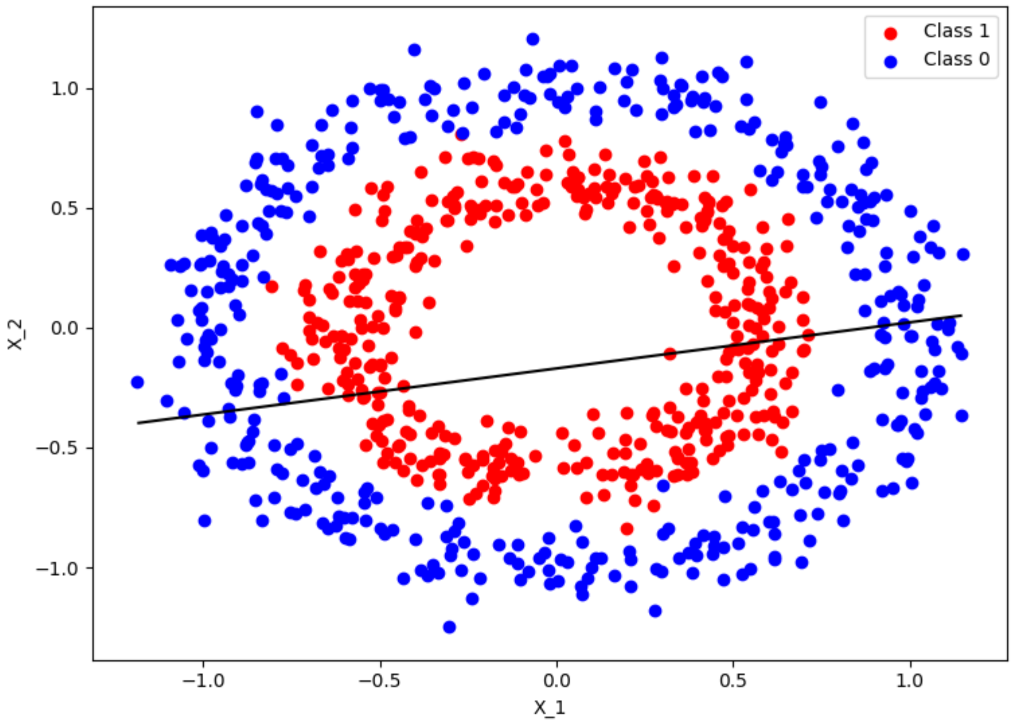

Figure 3: Logistic Regression with Linear Kernel

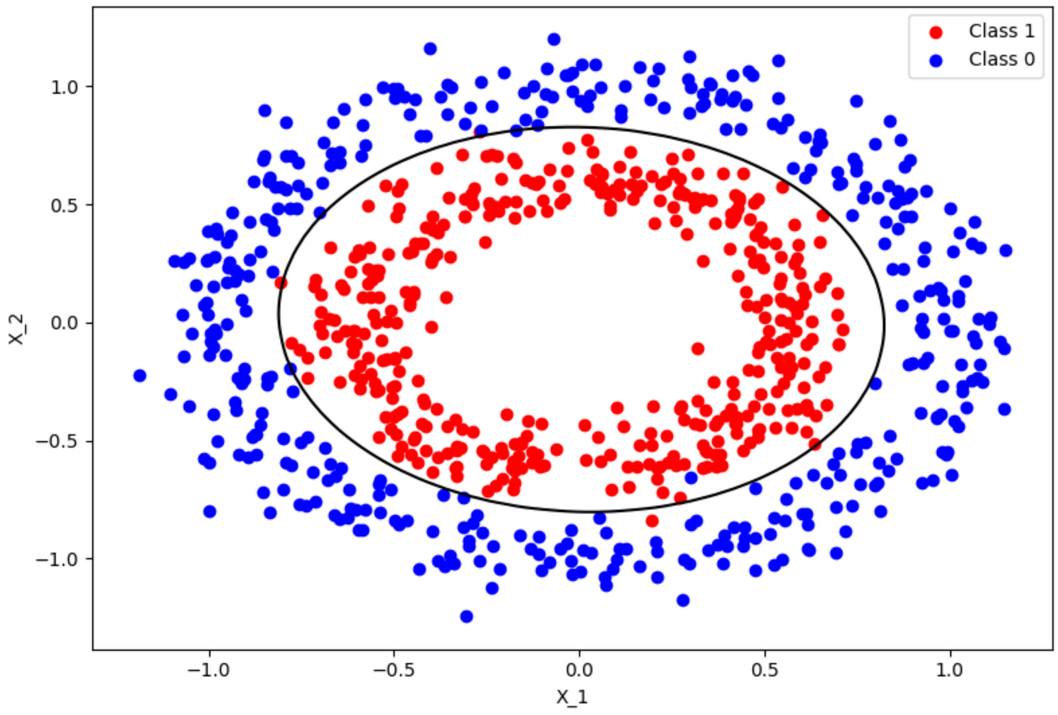

Figure 4: Logistic Regression with Polynomial Kernel

1.1.3 Analysis of Performance

We see that the given data is not linearly separable and hence the linear kernel performs very poorly. The polynomial kernel performs much better in both cases since the data is linearly separable in a higher dimensional space.

1.2 Part B (SVM)

After trying with various values of C, we found that C = 1 was as good as higher values of C and better than lower values of C. Hence we stick with C = 1 for the rest of this part.

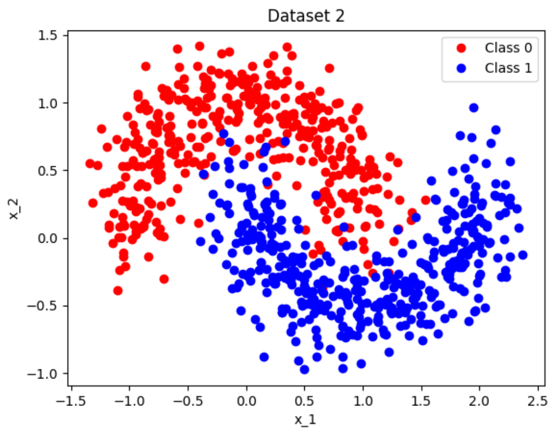

Figure 5: Dataset 2 - SVM Problem

As you can see, the data is not linearly separable, but we can still use a linear svm to achieve decently good results.

1.2.1 Classification Accuracy

Training Accuracy is 0.88.
Testing Accuracy is 0.795.

1.2.2 Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	87	21
Actual 1	20	72

1.2.3 Decision Boundary Plot, Margin and Support Vectors

Margin is 0.3463.
Total number of Support Vectors - 231.
Non marginal 0 support vectors - 114.
Non marginal 1 support vectors - 114.

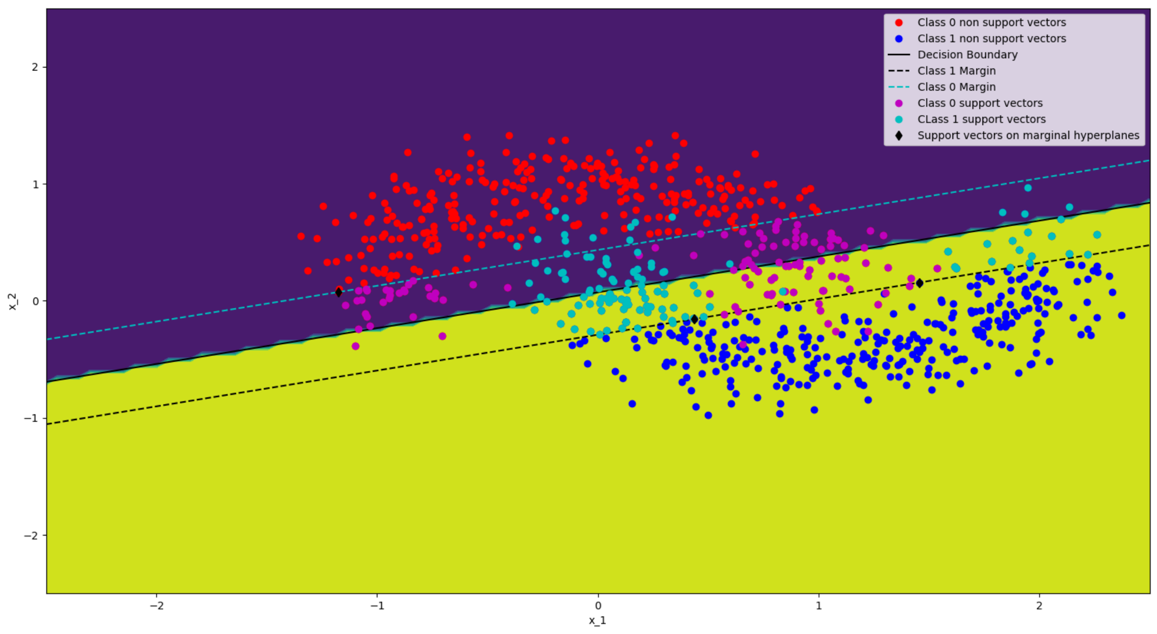

Figure 6: Figure for C =1 , k = 1

1.2.4 Number of Support Vectors

Number of support vectors on marginal hyperplanes is 3

1.2.5 Interpretation

The vast majority of support vectors are not on the marginal hyperplanes. This is because the svm is a soft margin svm and the data is not linearly seperable.

1.2.6 Varied class weights

In this subsection we explore the effect that varying of penalties for false negatives and false positives.

We wish to penalise false positive errors k - times more than a false negative error. We have varied the value of k between 1, 2, 4, 8 and 16. Since 1 would be the same as what was shown above, we now present the results for 2,4,8 and 16.

K factor 2
Training Accuracy is 0.865.
Testing Accuracy is 0.825.
Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	96	12
Actual 1	23	69

Margin is 0.3138
Total number of Support Vectors 237.
Non marginal 0 support vectors 77.
Non marginal 1 support vectors 157.
Number of support vectors on marginal hyperplanes 3

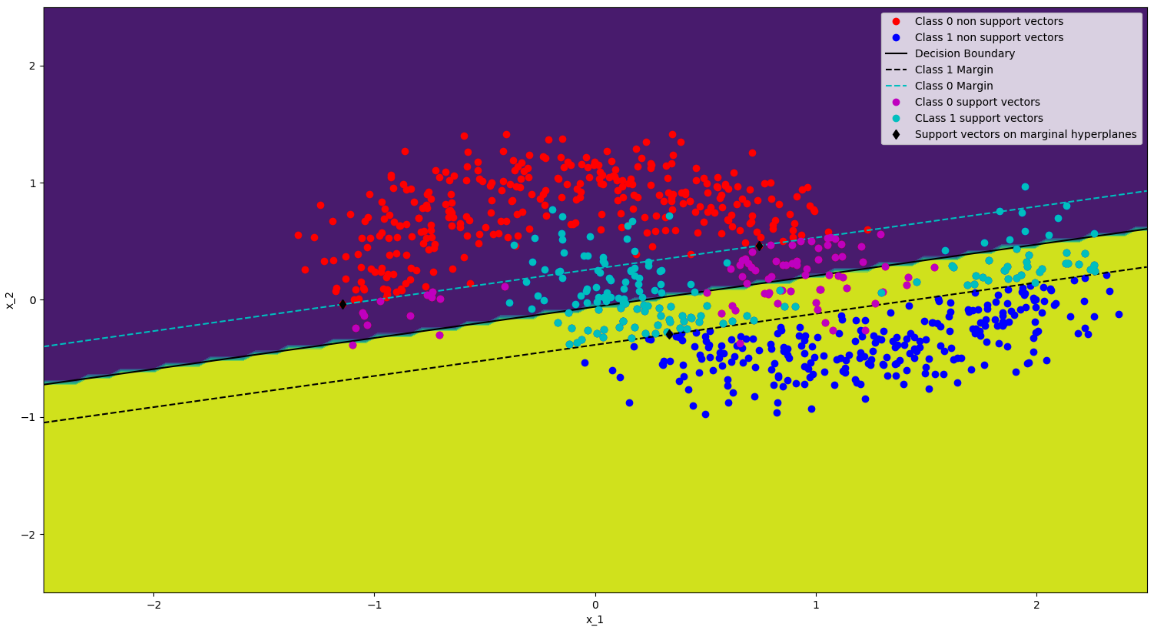

Figure 7: Figure for C =1 , k = 2

K factor 4
Training Accuracy is 0.86
Testing Accuracy is 0.795
Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	99	9
Actual 1	32	60

Margin is 0.3075
Total number of Support Vectors 261
Non marginal 0 support vectors 51

Non marginal 1 support vectors 207
Number of support vectors on marginal hyperplanes 3

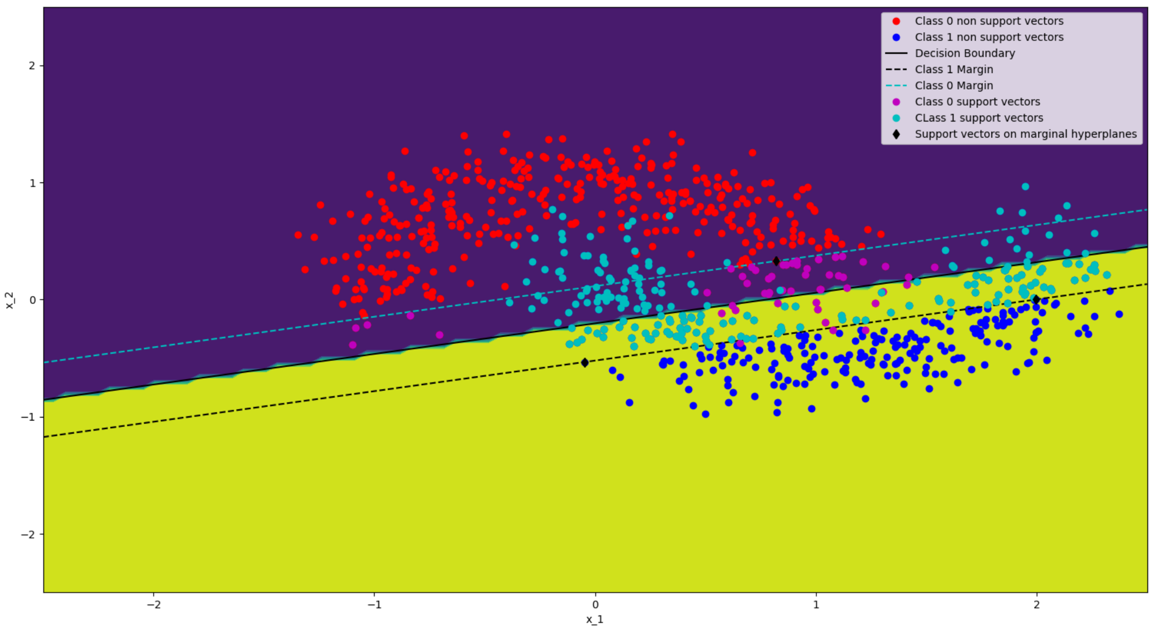

Figure 8: Figure for C =1 , k = 4

K Factor 8
Training Accuracy is 0.81375
Testing Accuracy is 0.795
Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	107	1
Actual 1	40	52

Margin is 0.2969
Total number of Support Vectors 297
Non marginal 0 support vectors 32
Non marginal 1 support vectors 263
Number of support vectors on marginal hyperplanes 2

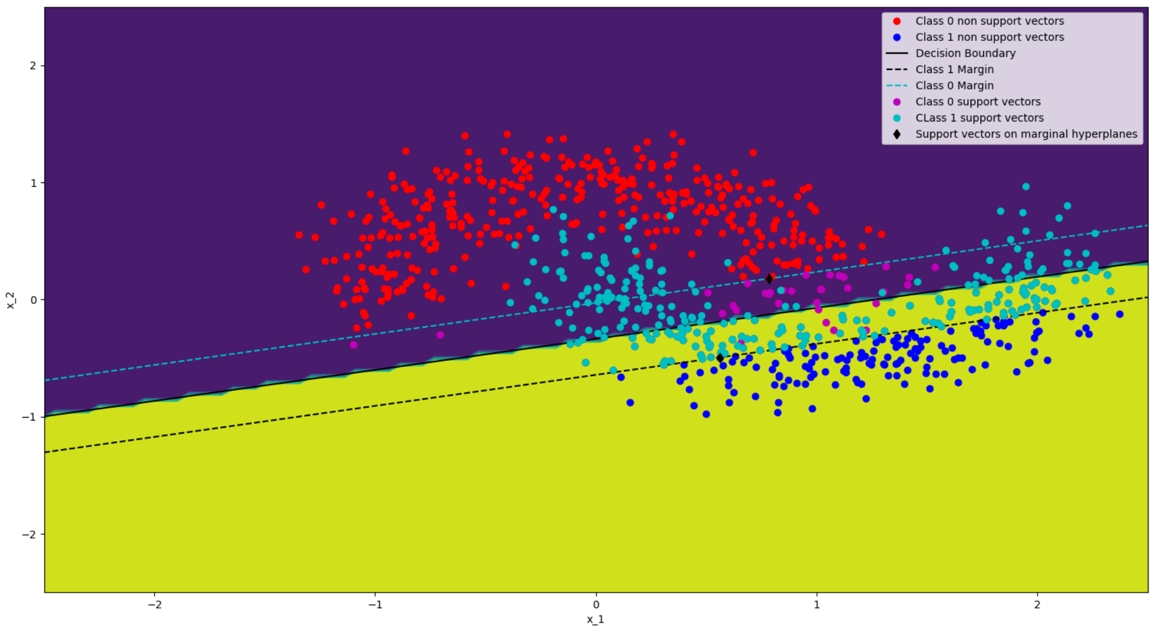

Figure 9: Figure for C =1 , k = 8

K Factor 16
Training Accuracy is 0.76125
Testing Accuracy is 0.76
Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	107	1
Actual 1	47	45

Margin is 0.2961
Total number of Support Vectors 338
Non marginal 0 support vectors 19
Non marginal 1 support vectors 316
Number of support vectors on marginal hyperplanes 3

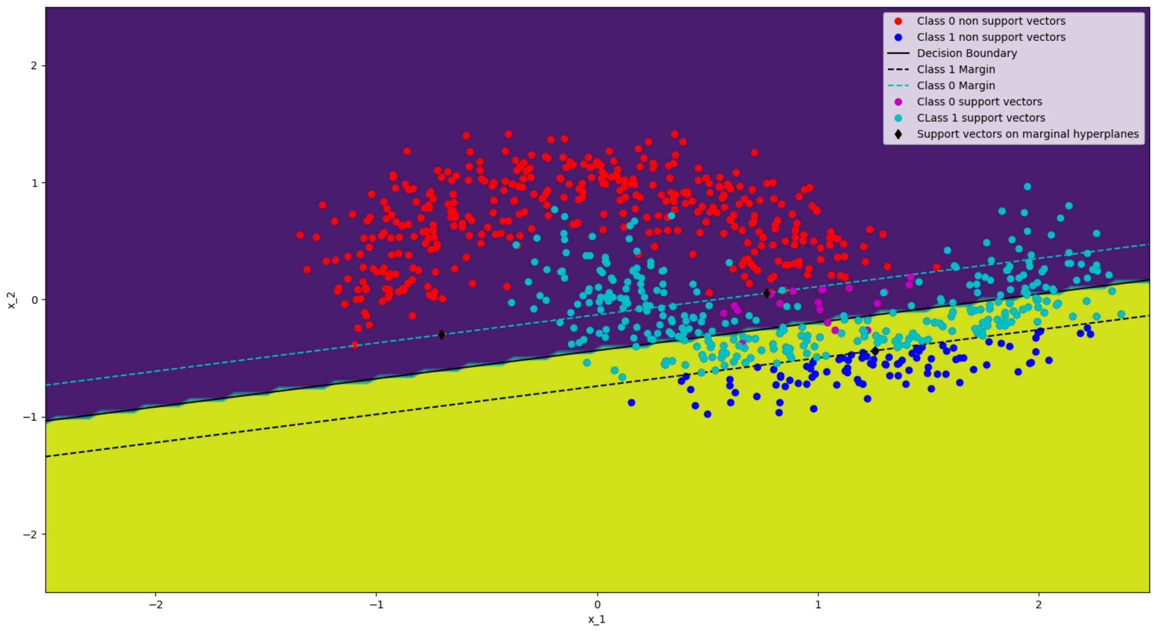

Figure 10: Figure for C =1 , k = 16

It can be observed that as the value of k increases,

- The number of points correctly classified as 0 increases.
- The number of points as support vectors for 0 decreases.
- The number of points correctly classified as 1 decreases.
- The number of points as support vectors for 1 increases.
- The decision boundary shifts ’down’ so to speak, heading towards smaller values of x_2
- The margin also decreases.

A final observation is that the RBF Kernel performs much better than a regular kernel, with an accuracy of upwards of 95%. This is because the data can be separated quite well using the RBF Kernel.

1.3 Part C (Perceptron)

In this problem we are required to implement a Perceptron based classifier using a linear and polynomial kernel.

Following the kernel version of perceptron, we implement the following algorithm:

```
alphas = np.zeros(T)
while(rat>thresh and epoch<max_epoch):
    err_old = err
    loss = 0
    for i in range(T):
        if np.sign(np.matmul(Y*alphas,kernel(X,X[i,:])))!= Y[i]:
            alphas[i] += 1
            loss+=1
        err = loss/T
        rat = abs(err-err_old)
        epoch += 1
```

The results can be seen below for both datasets:

Model	Training Accuracy	Testing Accuracy	Epochs	Number of Updates	Theoretical Bound
Linear Kernel: Dataset 1	1.0	1.0	1	1	12.64
Polynomial Kernel: Dataset 1	1.0	0.995	1	392	2204.90
Linear Kernel: Dataset 3	0.603	0.63	8	3103	191.30
Polynomial Kernel: Dataset 3	0.963	0.975	10	294	1.13

The decision boundary plots are given below:

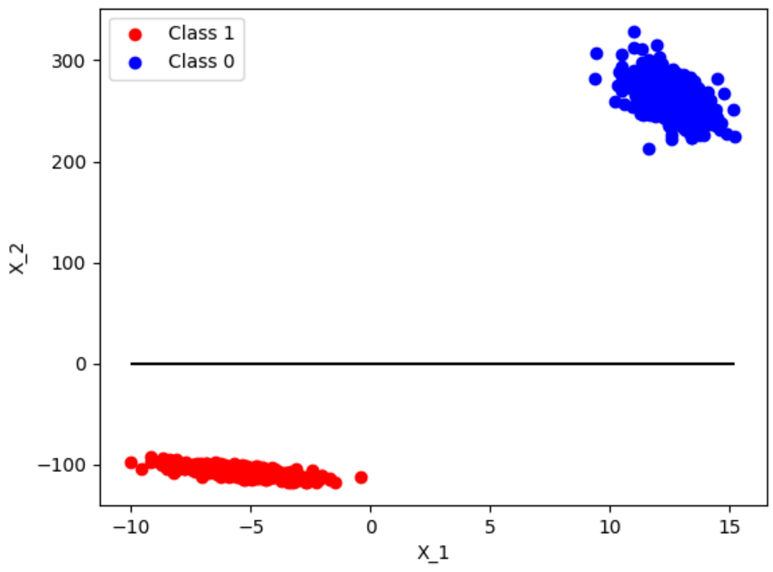

Figure 11: Linear Kernel used for Dataset 1

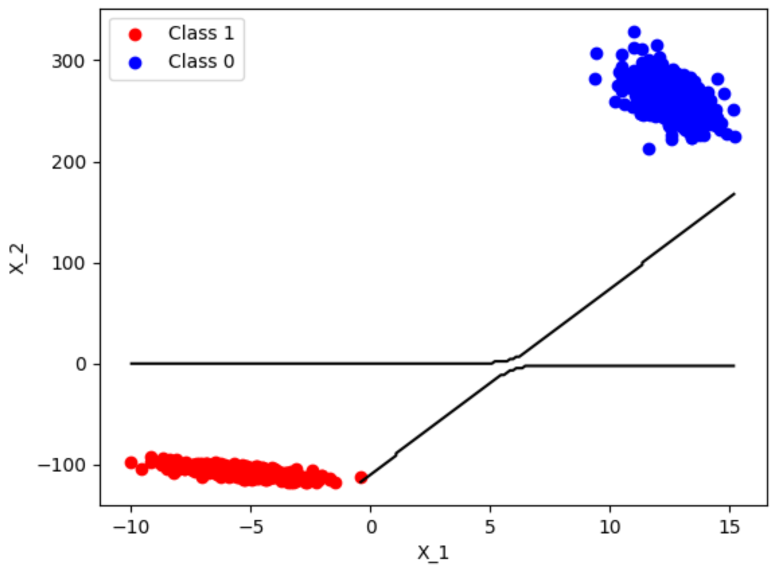

Figure 12: Polynomial Kernel used for Dataset 1

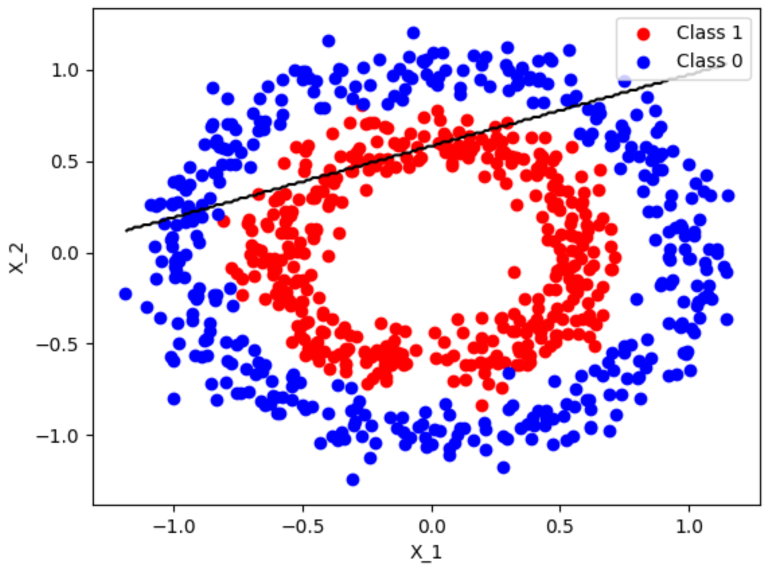

Figure 13: Linear Kernel used for Dataset 3

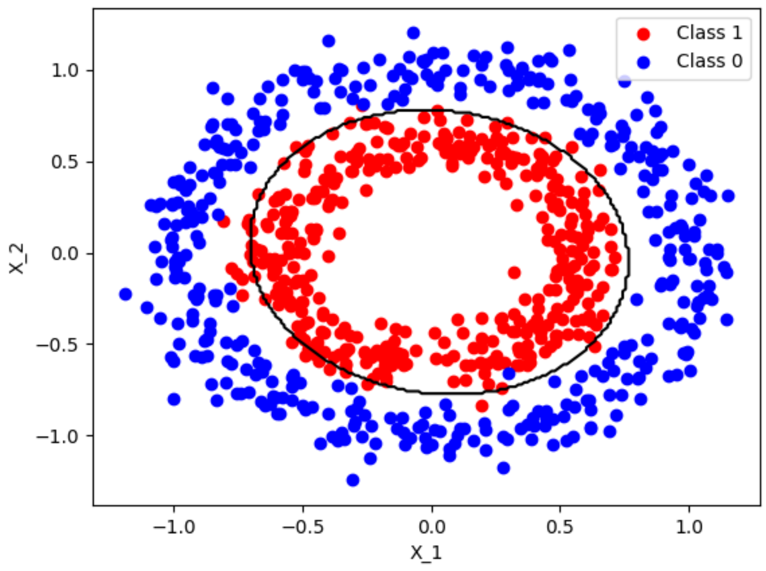

Figure 14: Polynomial Kernel used for Dataset 3

1.3.1 Analysis of performance

For dataset 1, we see that the data is already very easily linearly separable, hence the linear kernel performs very well while the polynomial kernel has overfit the data and has worse test accuracy. We also see that the linear kernel has converged with just one update while the polynomial one struggles to fit the data better and takes more updates. Hence the better kernel for this dataset was the linear kernel.

We also see that number of updates done by both models is less than the theoretical bound.

For dataset 3, we see that the data is not linearly separable and we do need a higher dimensional representation of the data to separate them satisfactorily. The polynomial kernel does just this and hence performs much better than the linear kernel as seen from the testing accuracy and the decision boundary it learns. We can also see that because of this it is able to converge with lesser number of updates while the linear kernel is not able to separate the data and undergoes more updates due to more misclassifications. Hence the better kernel for this dataset was the polynomial kernel.

Theoretical bound:: ADDDDDD

1.3.2 Comparison with SVM

On running the code used in part 1B for dataset 1, we see that the SVM is able to fit the polynomial kernel better than the perceptron, with a similar performance for the linear kernel as seen below:

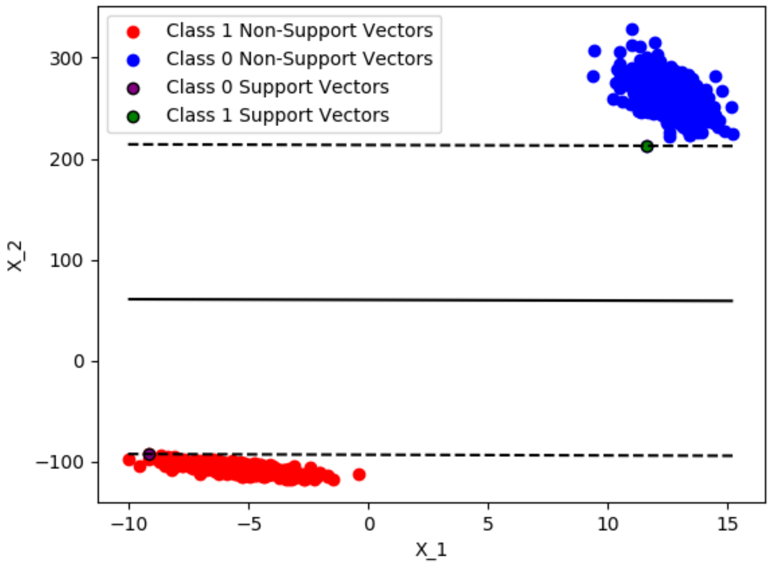

Figure 15: Linear Kernel used for fitting an SVM for Dataset 1

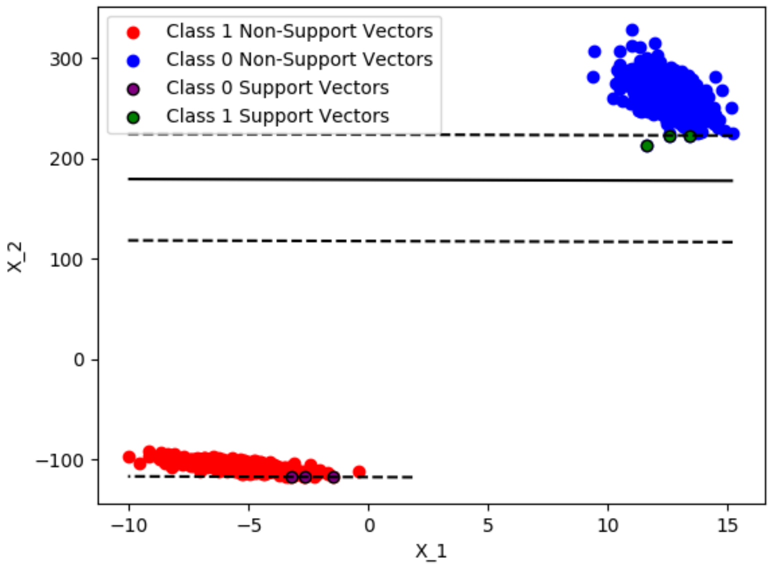

Figure 16: Polynomial Kernel used for fitting an SVM for Dataset 1

However, considering the simplicity of the perceptron algorithm with a linear kernel, that would be preferred for this dataset since the data very easily linearly separable.

For dataset 3, we had already fit an SVM in Q1B. We see that for the polynomial kernel, the SVM does better than the perceptron as expected since the SVM is a better classifier. However if we were to be receiving the data in an online fashion then the perceptron is an attractive model to be used since it does have high test accuracy with a polynomial kernel.

With a linear kernel however, the perceptron does better than the SVM although both models are unable to fit the data well.