PART-A

Problem 1.1 - *Kernel Ridge Regression*
*(Note: **MATLAB** was used for this problem)*

Methodology:

- *Training and test data for "Regression_dataset" was taken as 80% - 20%, respectively.*
- *Function definition for Kernel (ref. to: "polykernel.m")*

$$K(x_i, x_j) = (< x_i, x_j > + c)^d$$

for linear kernel: c = 0, d = 1, *for polynomial kernel: c = 1, d = 2*

Calculation of α^* *for minimum mean squared error :*

- *For linear kernel:* $\alpha^* = K^\dagger Y$

"† " denotes Moore-Penrose inverse

- *For polynomial kernel:* $\alpha^* = (K + \lambda I)^{-1} Y$

- *Prediction is given by:* $H = K\alpha^*$

K *is the Kernel matrix with elements* $K(x_i, x_j)$, λ *is the regularization constant,* Y *is the output values*

Mean Squared Error

$$J = \frac{1}{n} ||H - Y||^2$$

The mean squared error (as defined above) for both the models were taken as a measure of accuracy. The table below shows us the values as follows:

"Regression_dataset.csv"		
Mean squared error	**Linear Kernel**	**Polynomial Kernel**
Training Error (J_{train})	22.74	5.66
Test Error (J_{test})	33.31	235.68

(for least training error in the Polynomial case, $\lambda = 2$ *)*

From the above table, we can clearly observe that the **Linear Kernel Model** is the **better choice** for this particular dataset. The reason being, that although the J_{train} is lower for the Polynomial case, we can see a significant increase in J_{test}, which clearly implies that there has been a case overfitting.

Problem 1.2 - *Kernel Logistic Regression & Kernel SVM*

*(Note: **MATLAB** was used for this problem)*

Methodology:

- *Training and test data for "Dataset_3_Team_18.csv" was taken as 80% - 20%, respectively.*
- *Function definition of Kernel (ref. to: "polykernel.m") {same as before}.*
- *Training of α using gradient descent on the following loss function:*

$$L(w) = -\sum_{i=1}^{m} \{ y_i (log(\sigma(w^T x_i)) + (1 - y_i) log(1 - \sigma(w^T x_i)) \}$$

$$w^T x_i = \alpha^T K(:, i)$$

$$\therefore L(\alpha) = -\sum_{i=1}^{m} \{ y_i (log(\sigma(\alpha^T K(:, i))) + (1 - y_i) log(1 - \sigma(\alpha^T K(:, i)) \}$$

Taking gradient of $L(\alpha)$ *w.r.t* α *:*

$$\nabla_\alpha L = \sum_{i=1}^{m} \{(\sigma(\alpha^T K(:, i)) - y_i) K(:, i)\}$$

K *is the Kernel matrix with i-th column* $K(:, i)$*, and* y_i *is the i-th class label.*

*From the class notes, we could see that the above loss function is **strictly convex,** w.r.t to* w *.*

Therefore, the above gradient in terms of the hyperparameter α *was used for our gradient descent algorithm.*

Table of Classification accuracies

As you can see below the table of classification accuracies for both the models and both the kernels:

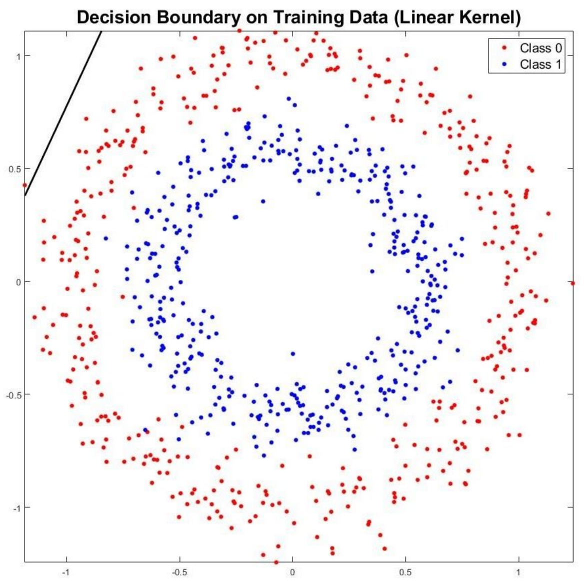
"Dataset_3_Team_18.csv"				
Classification accuracies	**Logistic Regression model**		**SVM model**	
	Train	Test	Train	Test
Linear Kernel	49.88%	50.00%	46.38%	44.00%
Polynomial Kernel	98.75%	98.50%	98.75%	100.00%

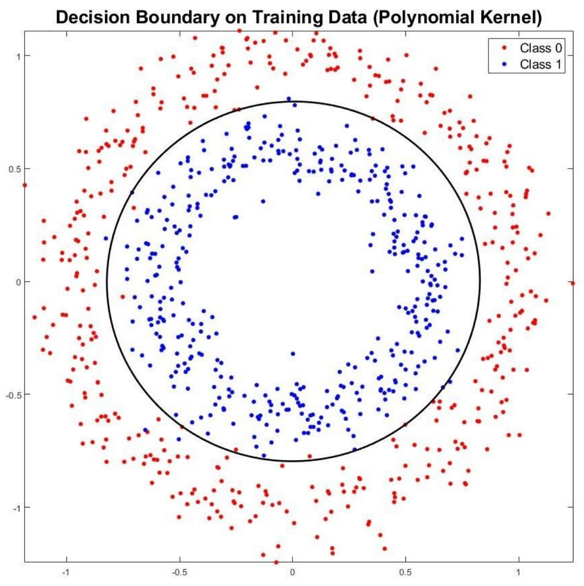
In both the models, we can see that the polynomial kernels outperform the linear kernels as the dataset is in a circular non-linear pattern. However, we can see that the use of the polynomial kernel in the SVM model works better than the logistic regression model as it is a more sophisticated algorithm.

(For more information about tuning of hyperparameters for the above models, refer to the included code)

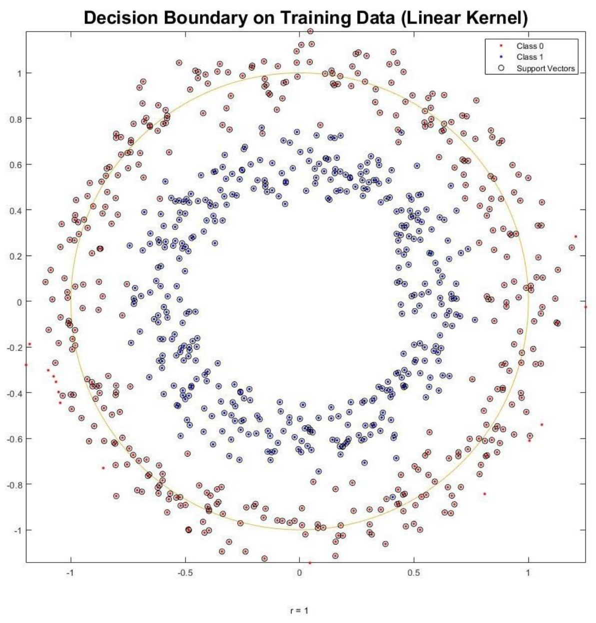
Plots of Decision Boundaries

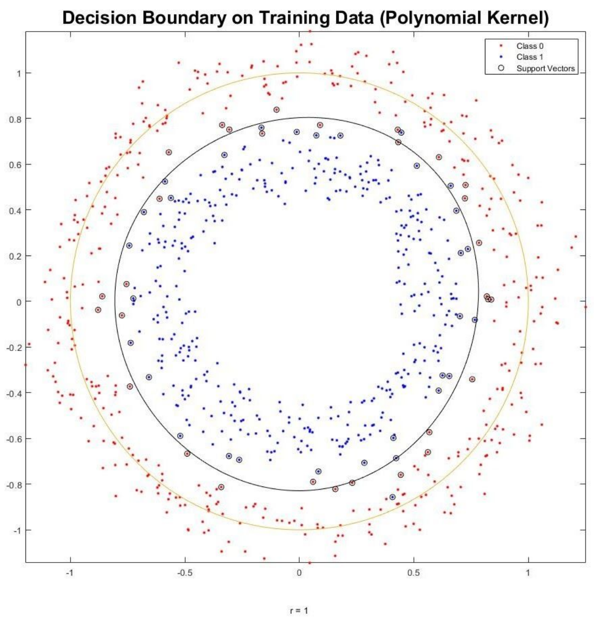
The Decision Boundary plots for the models corresponding to the table mentioned before are given below:





Above plots show us the decision boundaries generated by the Logistic Regression Model





Above plots show us the decision boundaries generated by the SVM Model

We can see that the decision boundary for the linear case for both the models is highly inaccurate, which was expected as the data resembles a circular distribution. In both cases, the Polynomial Kernel does perform better.

PART-B

Problem 2 –*SVM-based Classifier*
*(Note: **Python** was used for this problem)*

Methodology:

- *Training and test data for "Dataset_2_Team_18.csv" was taken as 80% - 20%, respectively.*
- *Different values were tried for the regularisation parameter C, and finally the best test accuracy was chosen as the best model.*

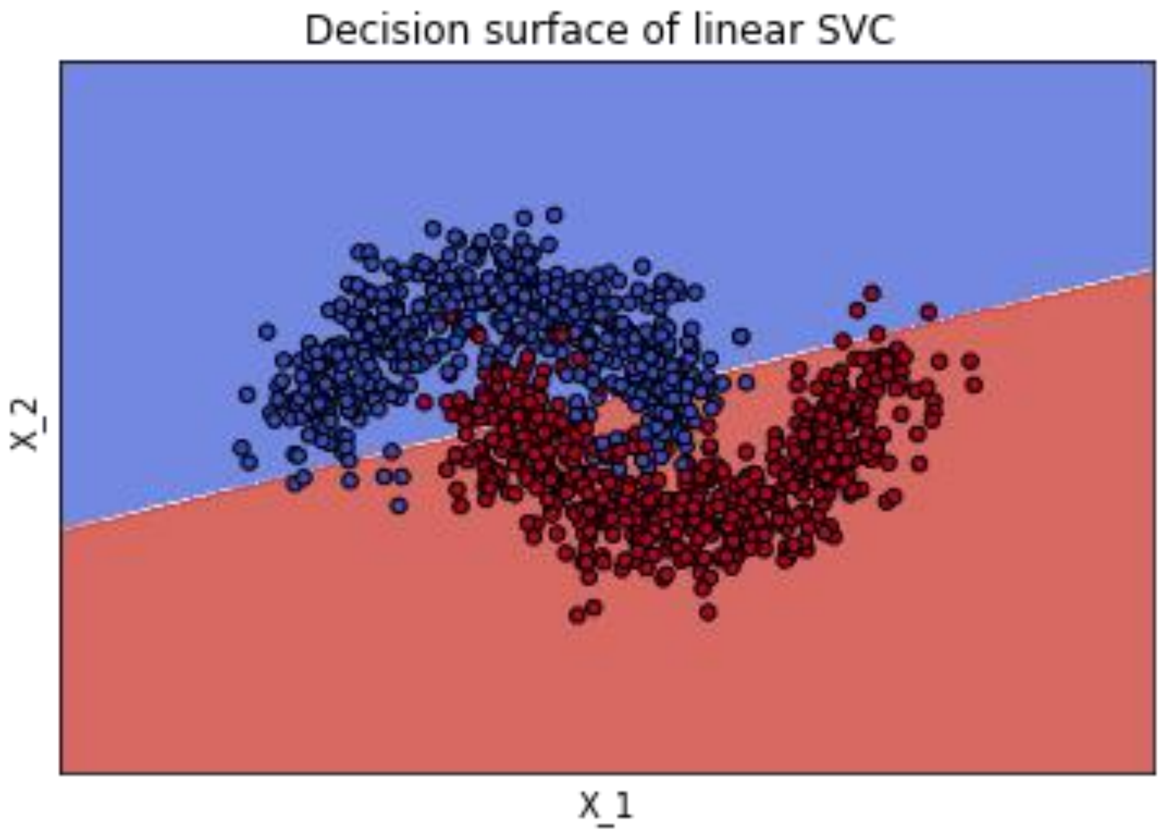
Classifier: SVM with linear kernel and regularisation parameter C=10

1. The classification accuracy of the train and the test data are as follows:
Train Accuracy: 0.8725
Test Accuracy: 0.9

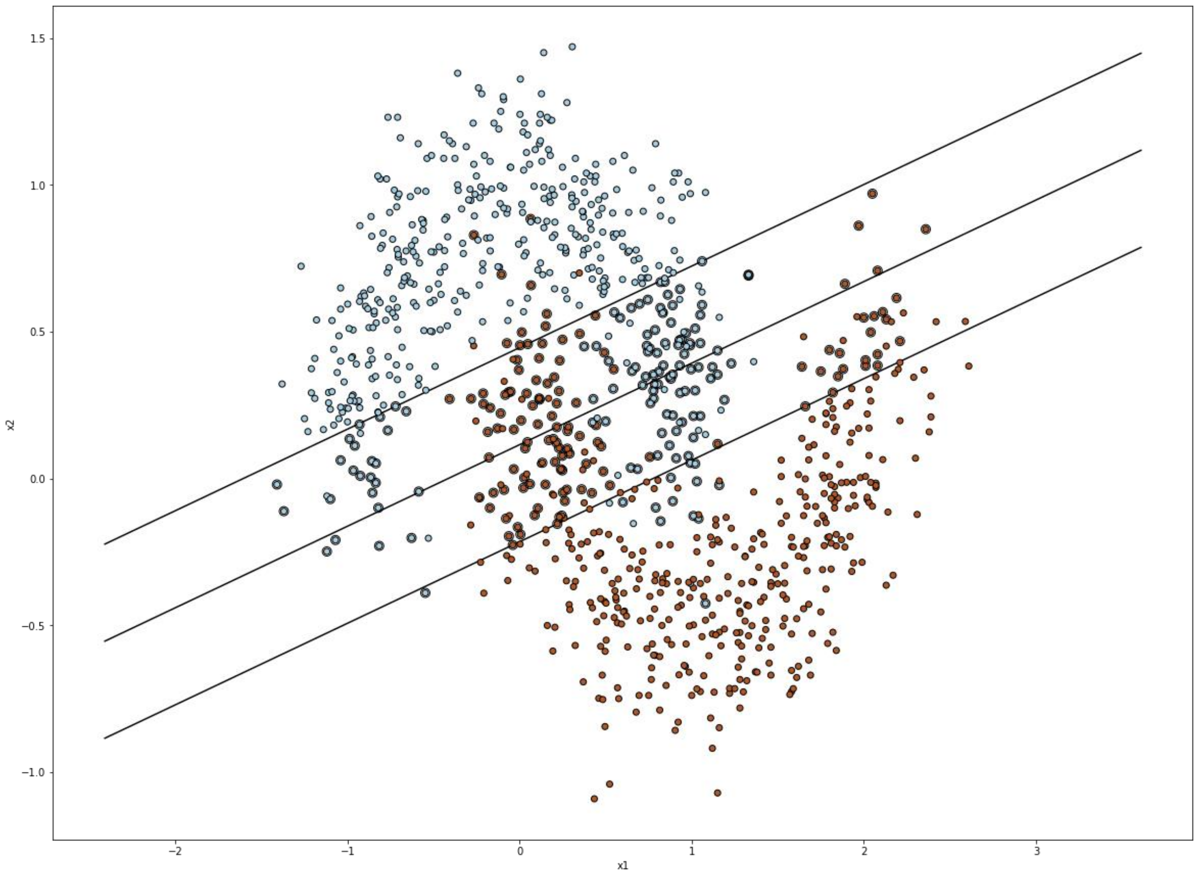
2. Confusion Matrix for the Test Data:

	Predicted NO	*Predicted YES*
Actual NO	*85*	*12*
Actual YES	*8*	*95*

3. Decision boundary plot:



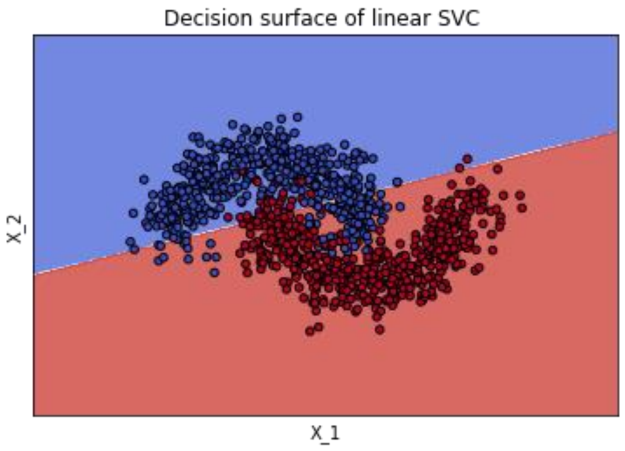
Plot along with the margin:

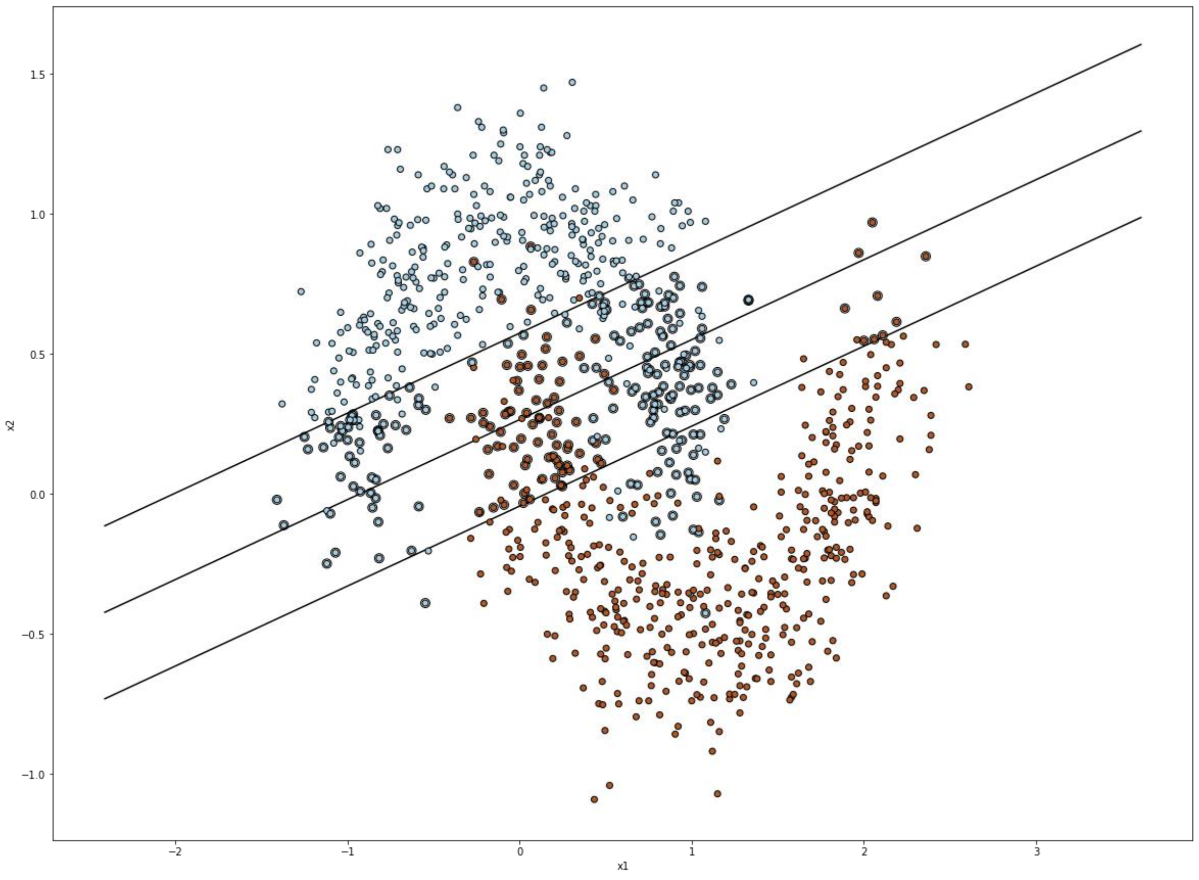


4. In the above figure we can see, the circles which are bold, they are the support vectors and they are the ones which lie on the hyperplane or on the wrong side of the hyperplane, i.e., which have been misclassified.

When we penalise the misclassification differently, the following changes are noticed in the boundary:
k=2

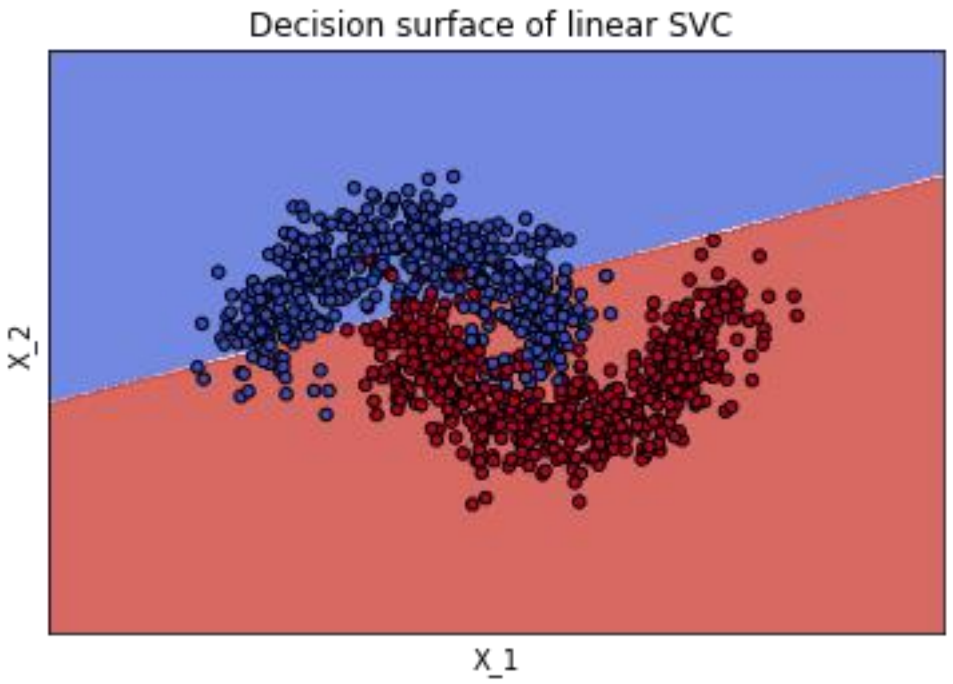
```
Train Accuracy:  0.85375 Test  Accuracy :  0.88
[[79 18]
 [ 6 97]]
```



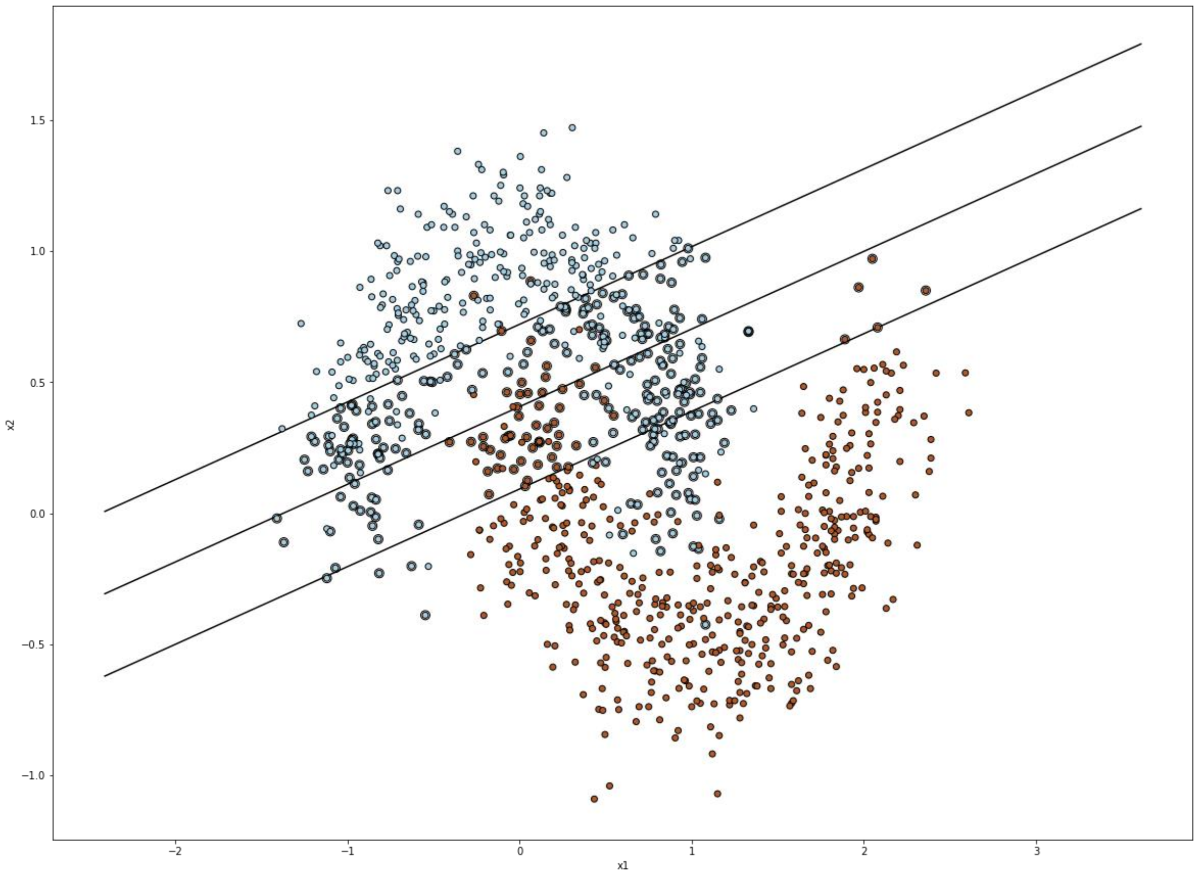


k=4

```
Train Accuracy :  0.845
Test  Accuracy :  0.865
[[ 73  24]
 [  3 100]]
<Figure size 2880x1440 with 0 Axes>
```

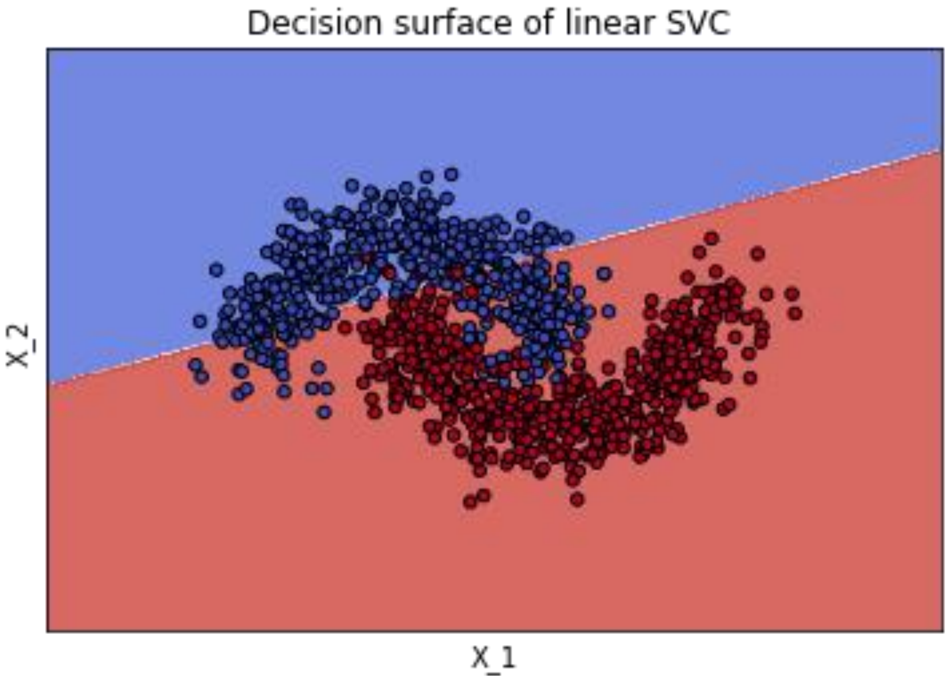


```
<Figure size 2880x1440 with 0 Axes>
```

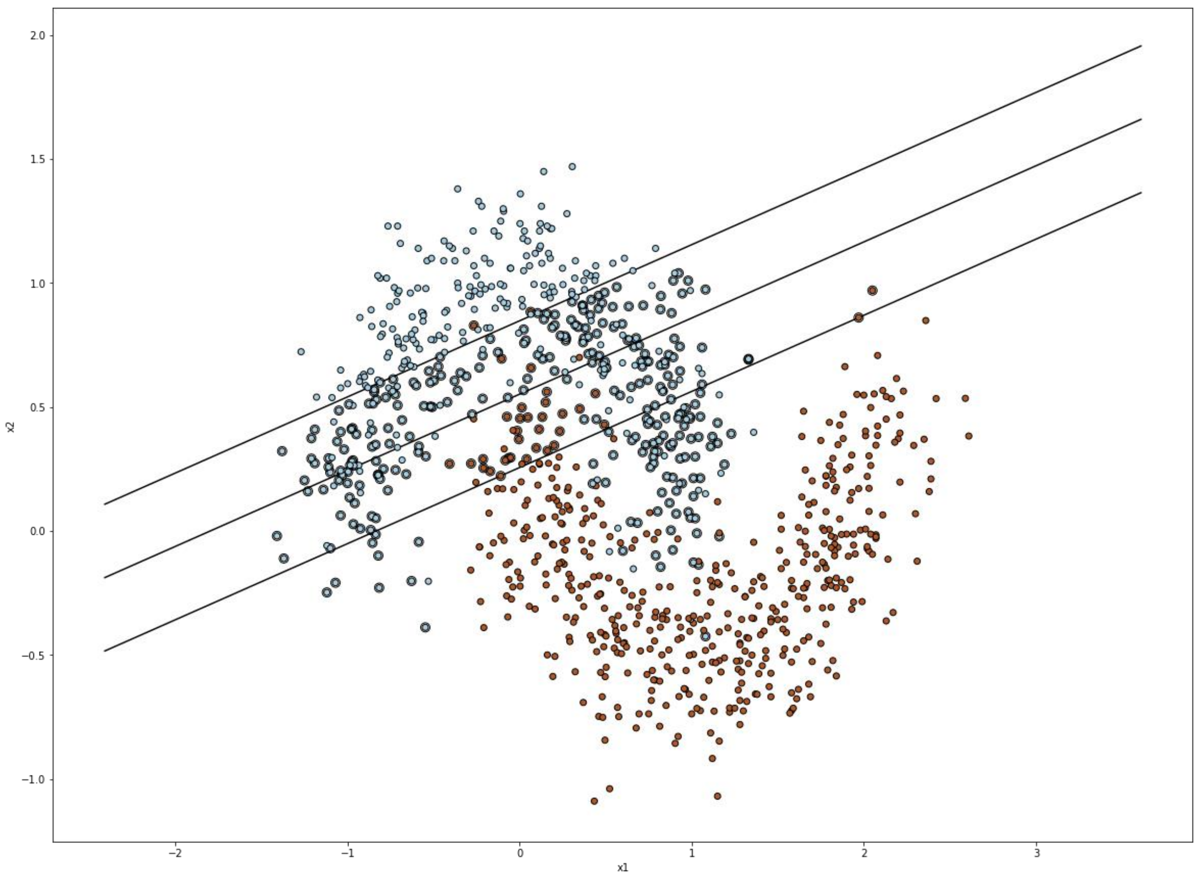


k=8

```
Train Accuracy :  0.8075
Test  Accuracy :  0.8
[[ 58  39]
 [  1 102]]
<Figure size 2880x1440 with 0 Axes>
```

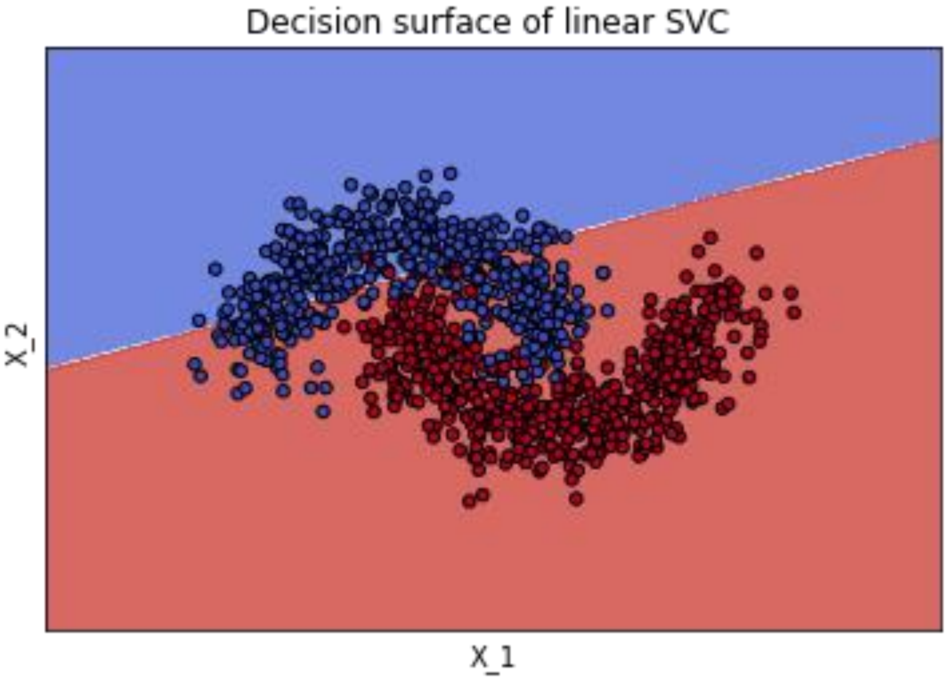


```
<Figure size 2880x1440 with 0 Axes>
```

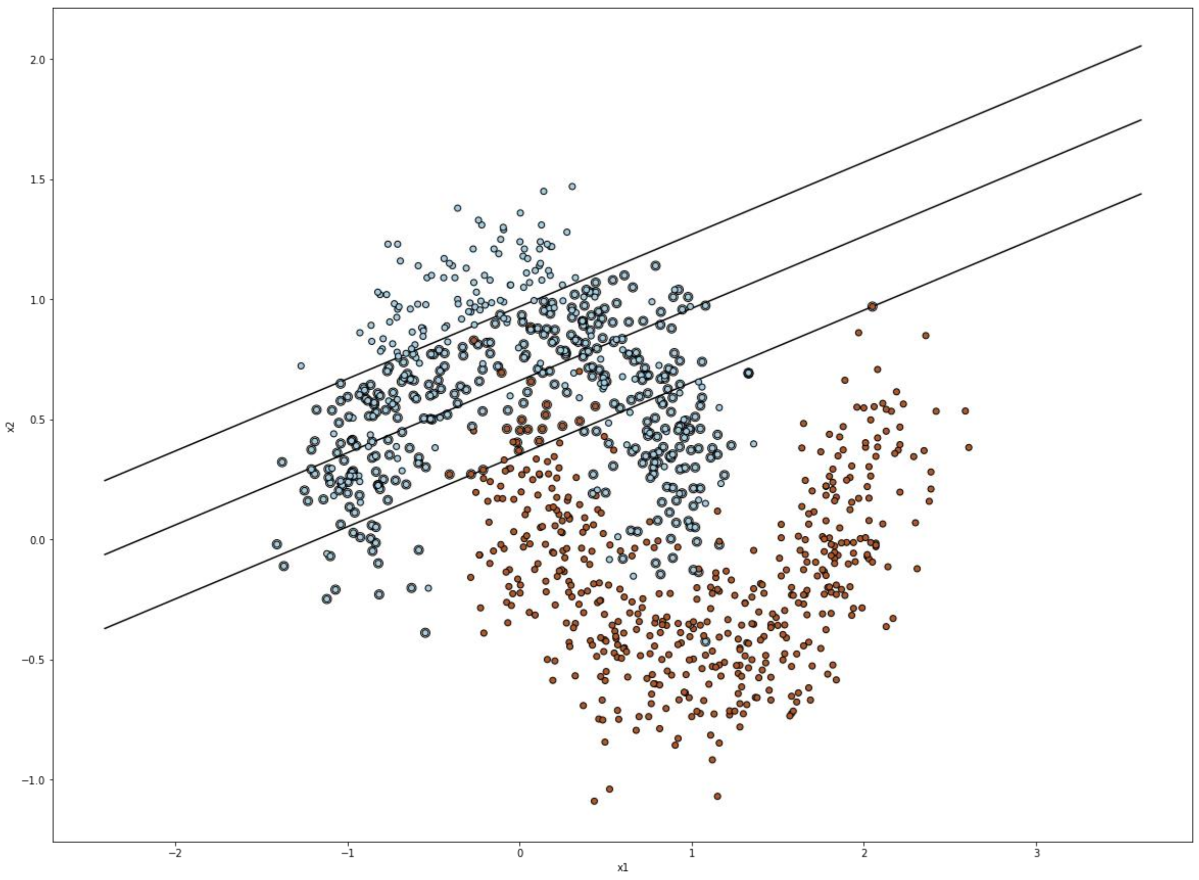


k=16

```
Train Accuracy :  0.7575
Test  Accuracy :  0.75
[[ 47  50]
 [  0 103]]
<Figure size 2880x1440 with 0 Axes>
```



```
<Figure size 2880x1440 with 0 Axes>
```



<u>Interpretations:</u>

As we can see, in the original classification, there were nearly equal number of misclassifications (which were beyond the margins). As, we increased the penalising for a particular class, the classifier tried to shift the boundary so that the higher weight class is not misclassified. This can be clearly seen from the plots for various values of k. As a result the accuracy of the model has decresed.

PART-C

Problem 3 - *Kernel Perceptron Classifier*
*(Note: **MATLAB** was used for this problem)*

Methodology:

- *Training and test data for “Dataset_1_Team_18.csv” and “Dataset_3_Team_18.csv” was taken as 80% - 20%, respectively.*
- *Function definition for Kernel (ref. to: “polykernel.m”)*

$$K(x_i, x_j) = (< x_i, x_j > + c)^d$$

for linear kernel: c = 0, d = 1, for polynomial kernel: c = 1, d = 2

- *Initial weights for the mistake-counter* α *were initialized to a zero-vector.*
- *A maximum number of epochs were kept as 10^4, however different stopping criteria were kept to manage computational time. (refer to the code for more information)*

Table of Classification accuracies

As you can see below the table of classification accuracies for both the models and both the kernels:-

"Dataset_1_Team_18.csv"		
Classification accuracies	**Kernel Perceptron Model**	
	Train	Test
Linear Kernel	100.00%	100.00%
Polynomial Kernel	100.00%	100.00%

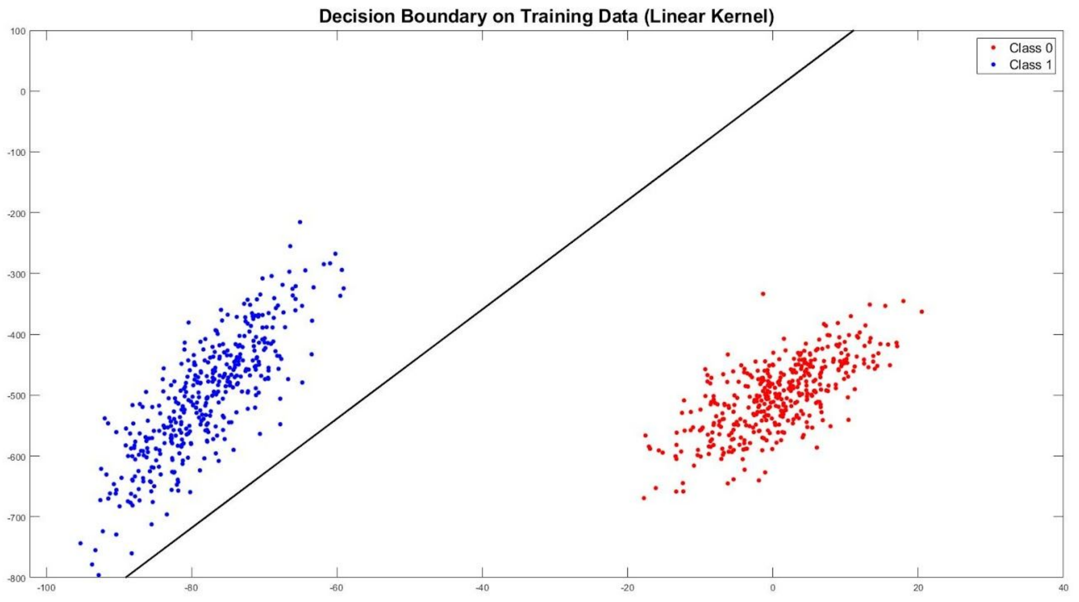
"Dataset_3_Team_18.csv"		
Classification accuracies	**Kernel Perceptron Model**	
	Train	Test
Linear Kernel	50.00%	50.00%
Polynomial Kernel	98.88%	98.00%

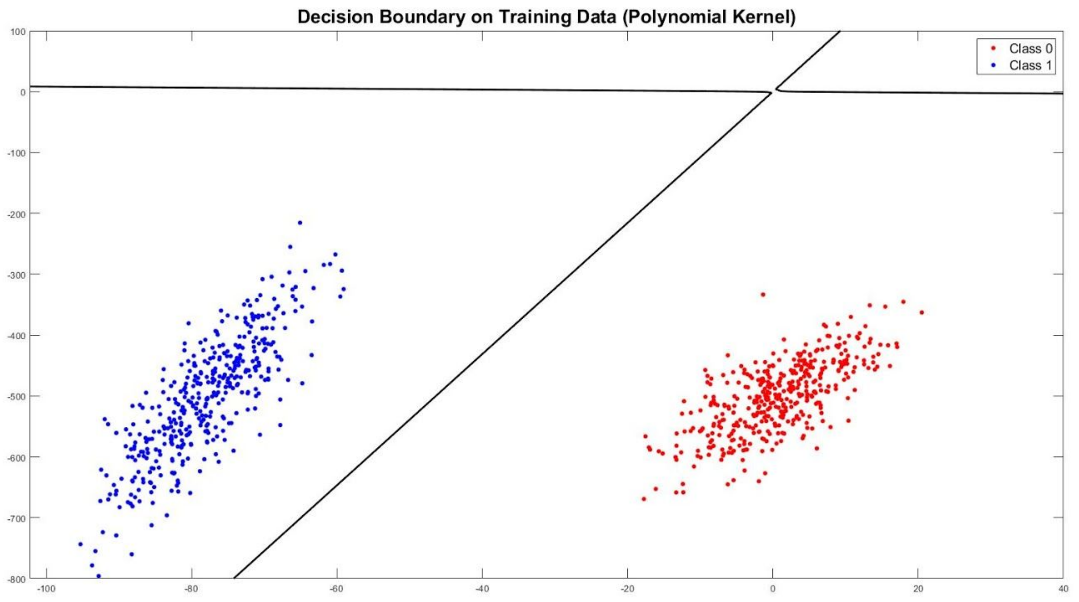
We can see that both the kernels for *"Dataset_1_Team_18.csv"* work perfectly including the polynomial kernel, in which case the decision boundaries resemble a joint line equation or a ‘sharp’ hyperbola. (refer the figures given below)

This is because the data is linearly separable unlike *"Dataset_3_Team_18.csv"*.

Plots of Decision Boundaries

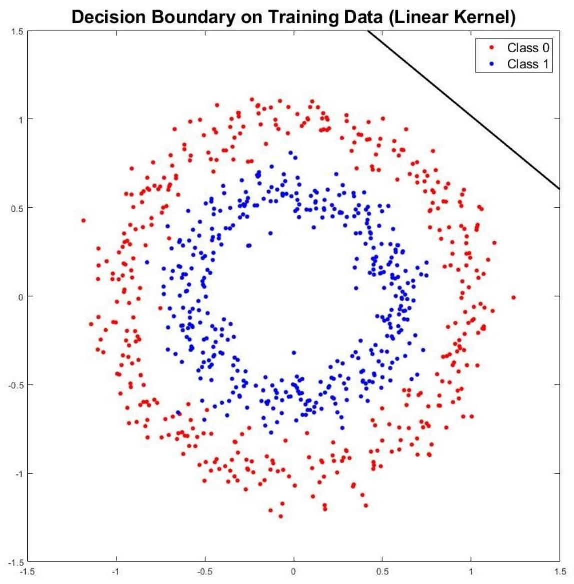
The Decision Boundary plots for the models corresponding to the table mentioned before are given below:

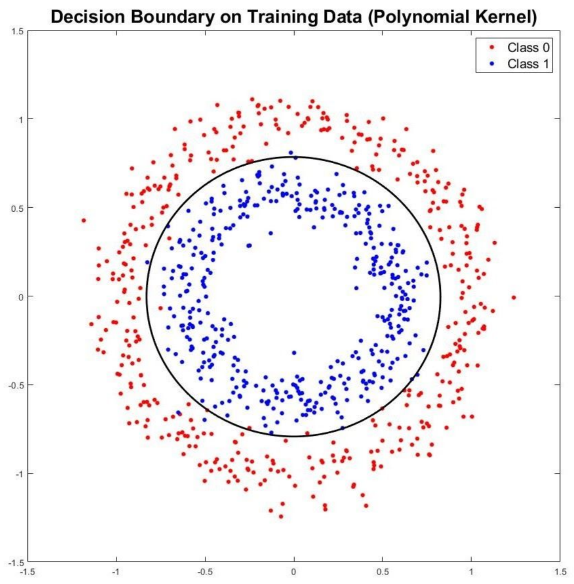




The plots above correspond to "Dataset_1_Team_18.csv" for both the kernels

As mentioned before, we can see that the polynomial kernel creates a hyperbola-type of decision boundary.





The plots above correspond to "Dataset_3_Team_18.csv" for both the kernels

Here we can see that the polynomial kernel does a much better job as compared to the linear kernel as the data has a circular distribution. This same observation was seen for Part-A of the assignment.

Analysis on the iteration bounds

As seen from the book *"Foundations of Machine Learning"*, the bound for the number of iterations is given as:

$$M < \frac{r^2}{p^2}$$

Where M is the number of iterations, r is the maximum norm of our input feature vectors, and p is the distance of the point closest from our decision boundary. For this bound to make sense, our data has to be linearly separable.

Let us first see our bound criterion for *"Dataset_1_Team_18.csv"*. For the linear kernel, it was found that the number of correction updates M = 24, and the upper bound was found to be 50714. Hence, the inequality was satisfied. For the polynomial kernel, M = 7 and the upper bound was 2371. Therefore, in both the cases the inequality was satisfied.

For *"Dataset_3_Team_18.csv"*, we know that the data is not linearly separable for both the kernels, and hence the main assumption for the bound criterion does not hold.

<u>Comparison with hard-margin SVMs</u>

The file *"Kernel_SVM.m"* was used to get our hard-margin SVM classifiers for the two datatsets. Below are the following results:

"Dataset_1_Team_18.csv"				
Classification accuracies	**Perceptron model**		**SVM model (hard-margin)**	
	Train	Test	Train	Test
Linear Kernel	100.00%	100.00%	100.00%	100.00%
Polynomial Kernel	100.00%	100.00%	50.00%	50.00%

"Dataset_3_Team_18.csv"				
Classification accuracies	**Perceptron model**		**SVM model (hard-margin)**	
	Train	Test	Train	Test
Linear Kernel	50.00%	50.00%	50.00%	50.00%
Polynomial Kernel	98.88%	98.00%	59.50%	58.50%

We can see that the hard-margin SVM model for *"Dataset_3_Team_18.csv"* performs very poorly as the data is not linearly separable.