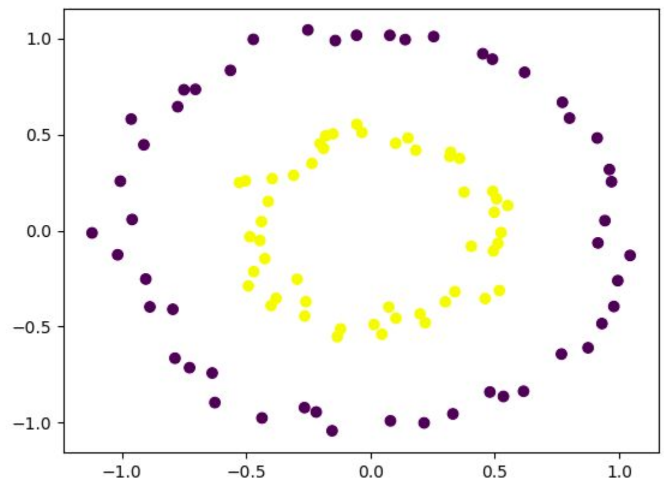
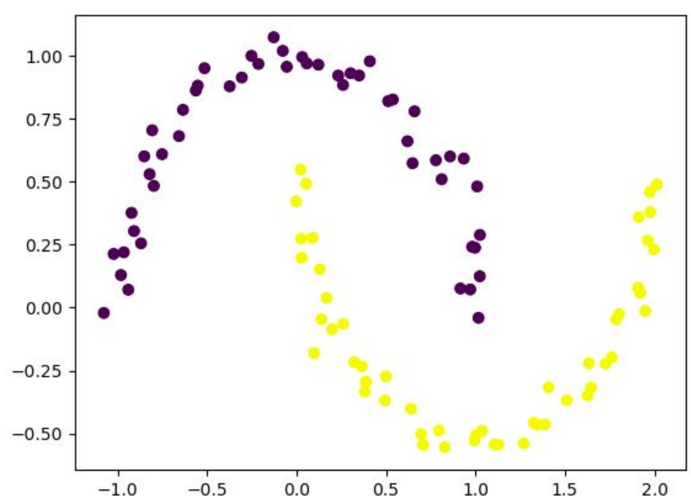
Assignment 2

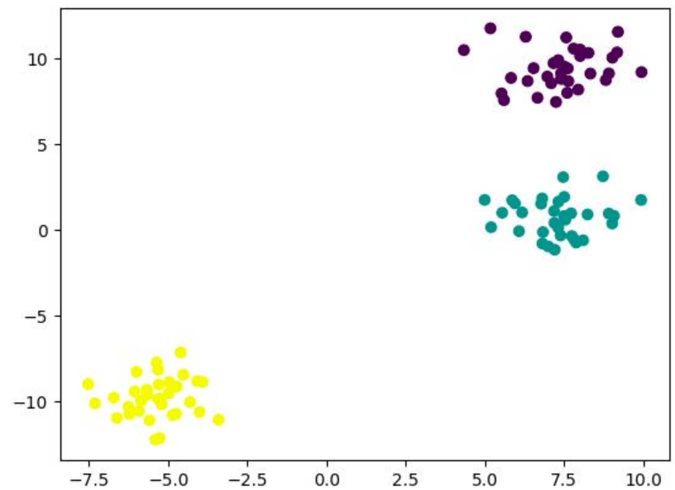
1.



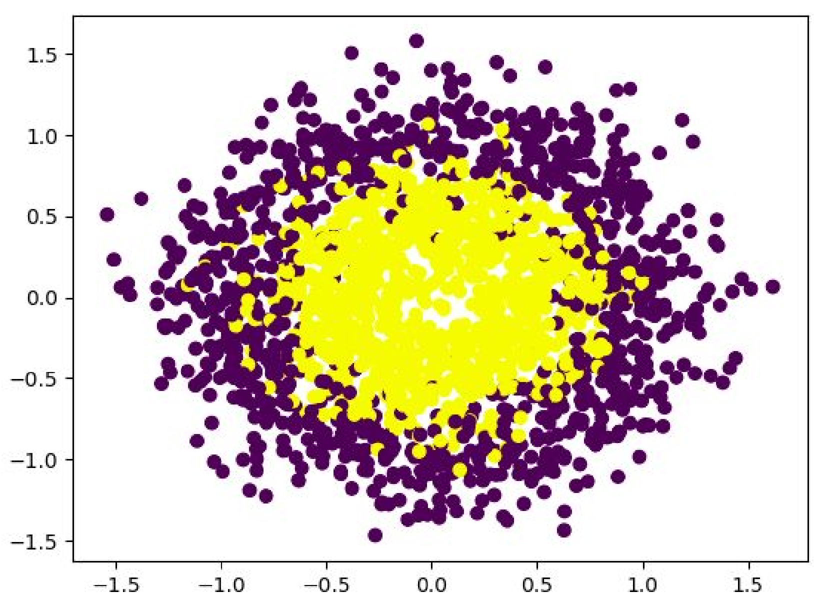
This dataset is not linearly separable but it can be separated using non-linear boundary, without any outliers



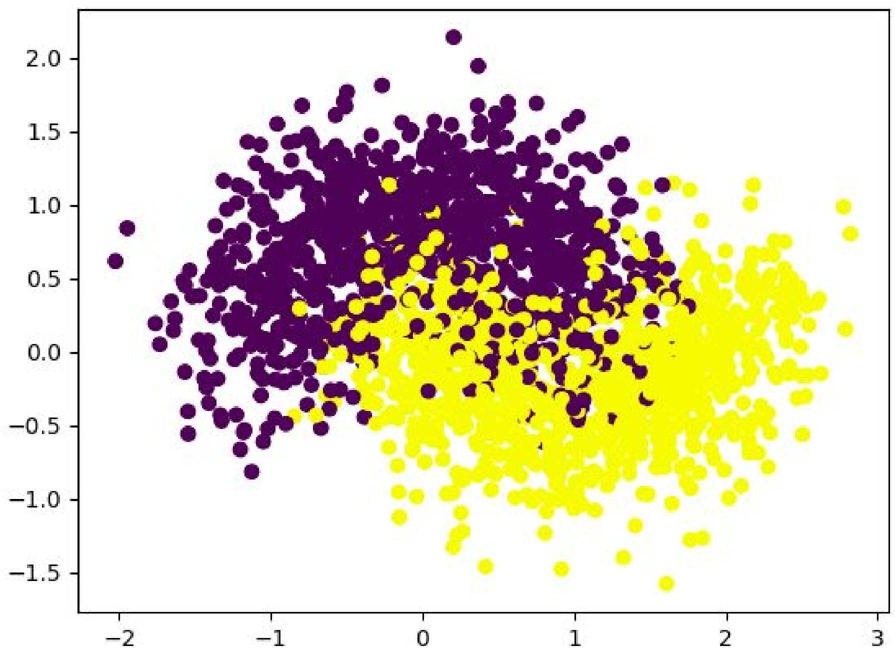
This dataset is also not linear separable but without noise



Linearly separable without any noise

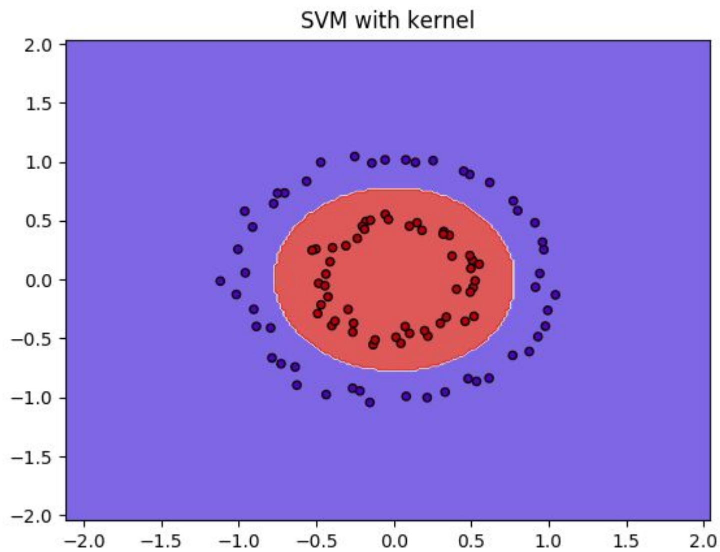


Non-Linearly separable with many outliers

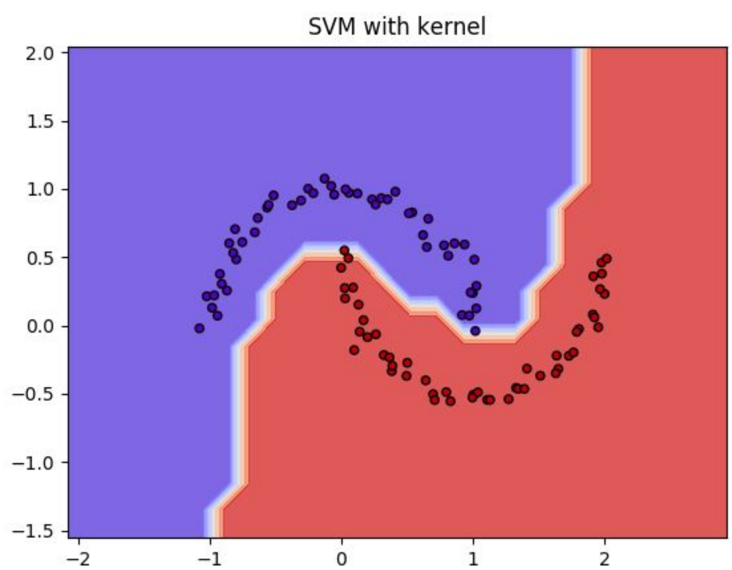


Non-Linearly separable with many outliers

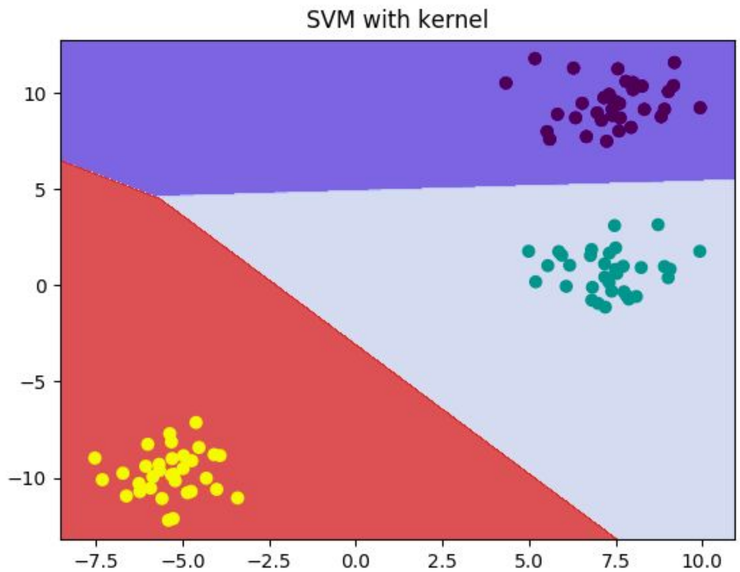
2.



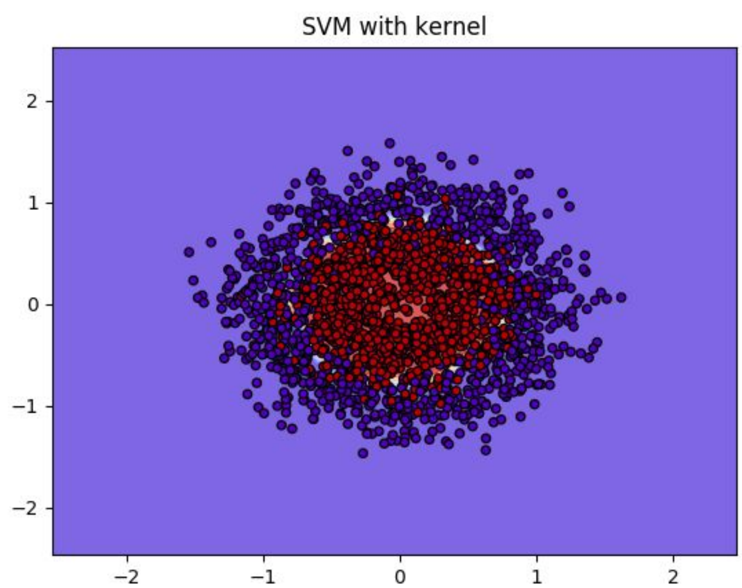
Data 1



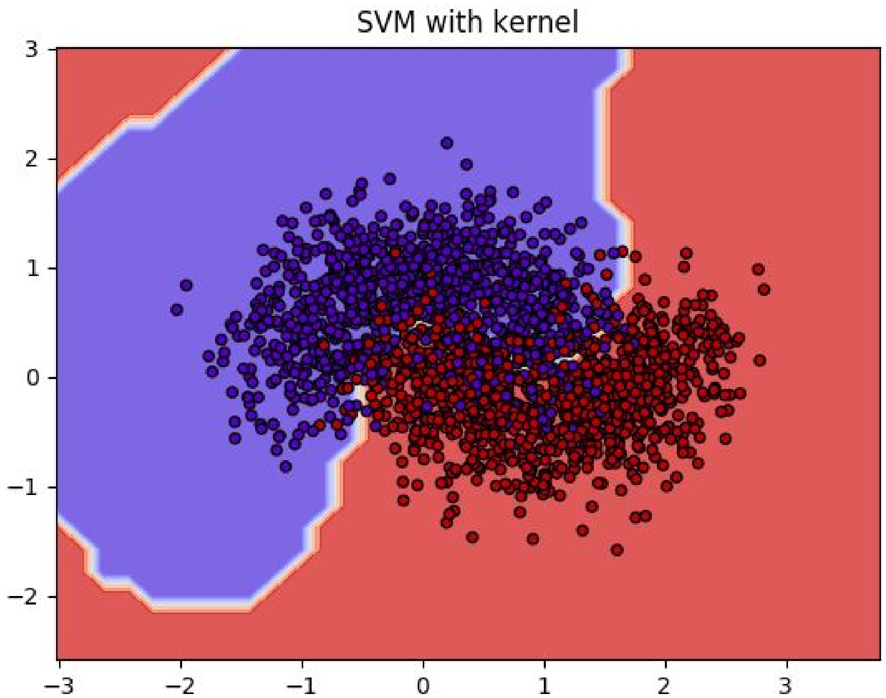
Data 2


SVM with kernel
10
5
0
−5
−10
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Data 3


SVM with kernel
2
1
0
−1
−2
−2
−1
0
1
2

Data 4

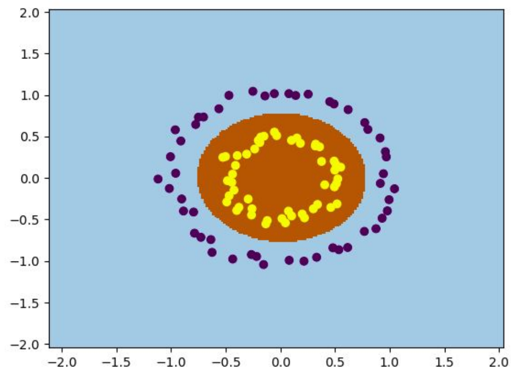


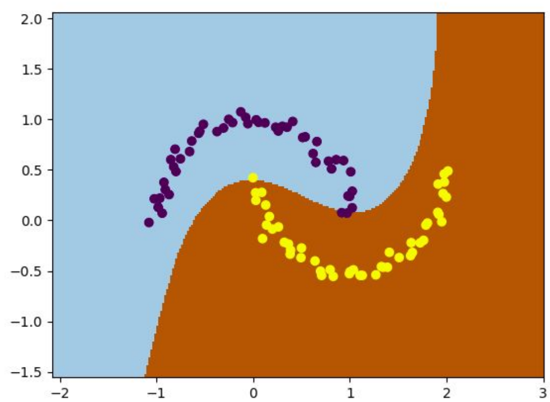
Data 5

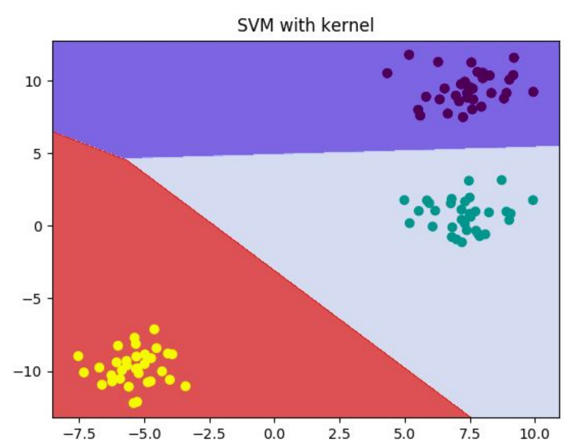
In <u>data 1,2,4 and 5</u>: Kernel used is rbf, because it is non-linearly separable data. The data can not be separated by a linear line, therefore we use rbf kernel which divides the data non-linearly and smoothly.

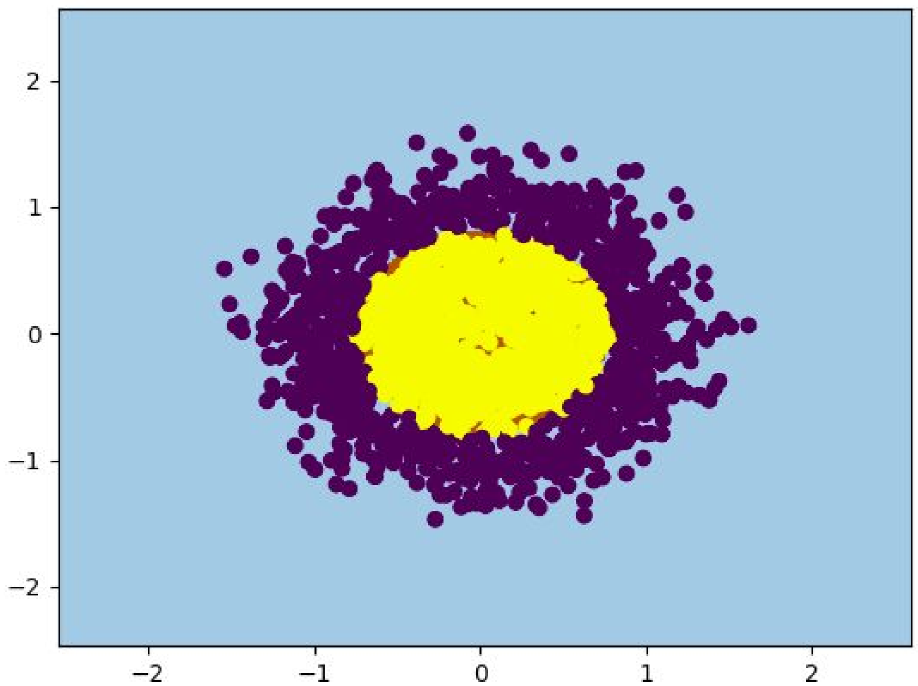
In <u>data 3</u>, it can be easily separated by linear lines, there linear kernel of svm is used.

3.

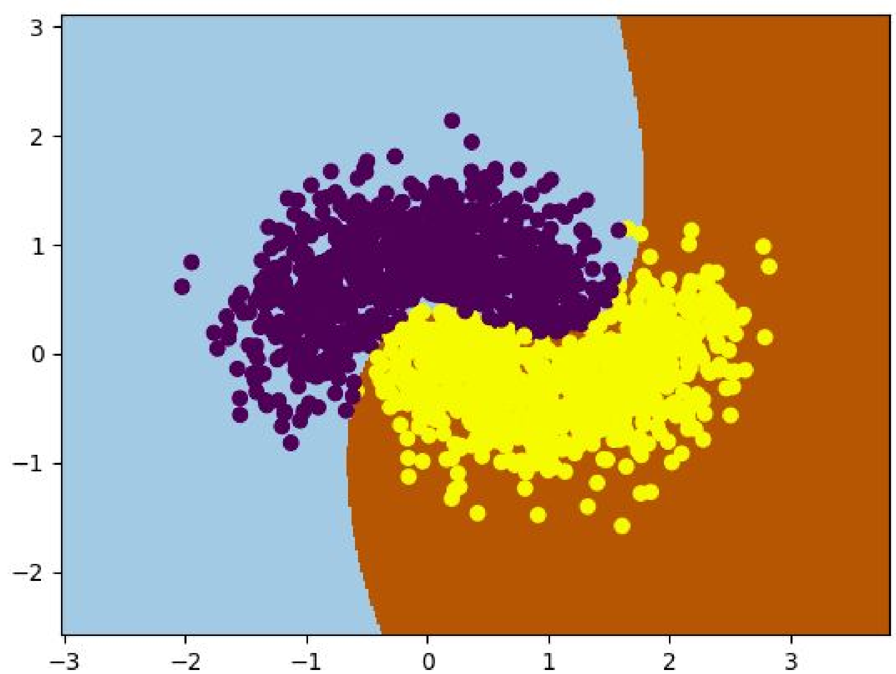








2
1
0
−1
−2
−2
−1
0
1
2



3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

SVM

1. Linear Kernel

I have used C hyperparameter. Large value of C (like 10) will Try to overfit the data, and make smaller margins. It will try To avoid the outliers at the cost of margin size. Whereas, Smaller size of C will underfit the data but make larger margins.

a.) One-vs-Rest

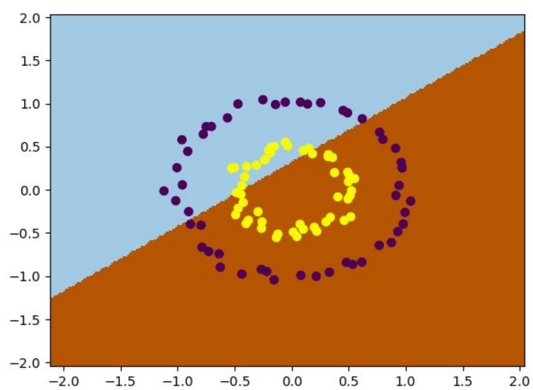
- **Data 1**

K =	1	2	3	4	5
c=1	50.0	35.0	40.0	40.0	30.0
c=0.1	50.0	45.0	40.0	45.0	35.0
c=10	50.0	35.0	40.0	40.0	40.0

As, we can see in most of the cases c=0.1 is giving the best result. Therefore, best result is when margin is big and no overfitting. There was no noise or outliers in data, therefore small c was best.

<u>Confusion matrix </u>(I have taken only for c=1 and for all K-folds)

```
[[ 17.  28.]
 [ 33.  22.]]
```



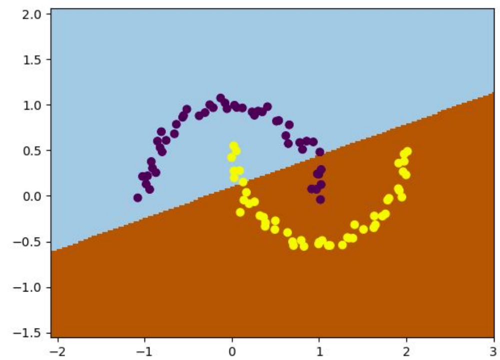
- **Data 2**

K =	1	2	3	4	5
c=1	75.0	90.0	85.0	90.0	85.0
c=0.1	70.0	90.0	85.0	90.0	85.0
c=10	75.0	90.0	85.0	95.0	85.0

The trend shows, very tiny value of c gives less accuracy. c=1 would fit best for this because it has relatively more noise.

<u>Confusion matrix</u> (I have taken only for c=1 and for all K-folds)

[[42.   7.]
 [8. 43.]]



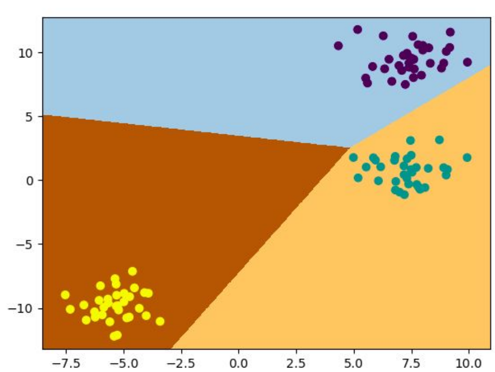
- Data 3

K =	1	2	3	4	5
c=1	100.0	100.0	100.0	100.0	100.0
c=0.1	100.0	100.0	100.0	100.0	100.0
c=10	100.0	100.0	95.0	100.0	100.0

Only once accuracy fallen down, i.e. at c = 10. There overfitting is there which results in less accuracy, due to larger value of c.

Confusion matrix (I have taken only for c=1 and for all K-folds)

[[34. 0. 0.]
[0. 33. 0.]
[0. 0. 33.]]



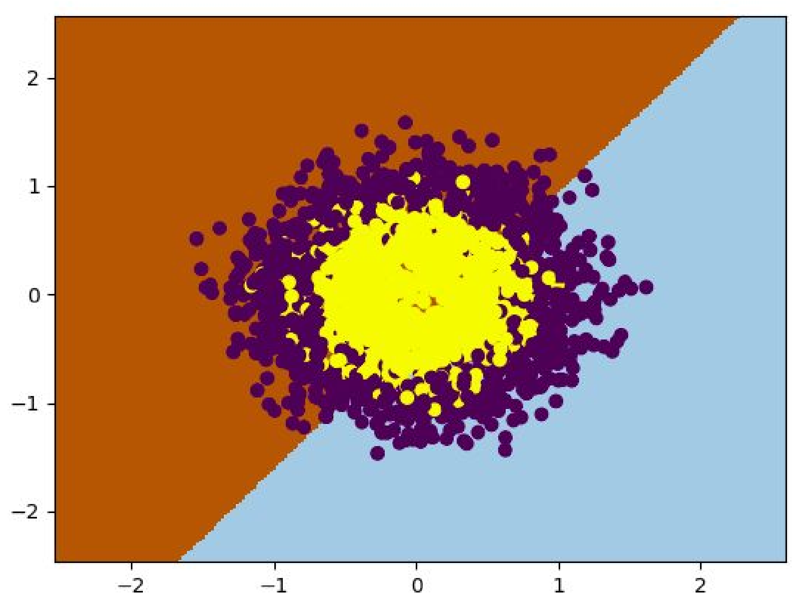
- **Data 4**

K =	1	2	3	4	5
c=1	52.75	56.75	51.0	54.75	47.25
c=0.1	53.0	58.75	51.75	48.0	47.25
c=10	52.75	55.75	50.5	54.75	47.25

The accuracy is fluctuating with c. But c=1, is consistently giving best results. Therefore, moderate level of overfitting will give good result in this case.

Confusion matrix (I have taken only for c=1 and for all K-folds)

[[478. 428.]
 [522. 572.]]



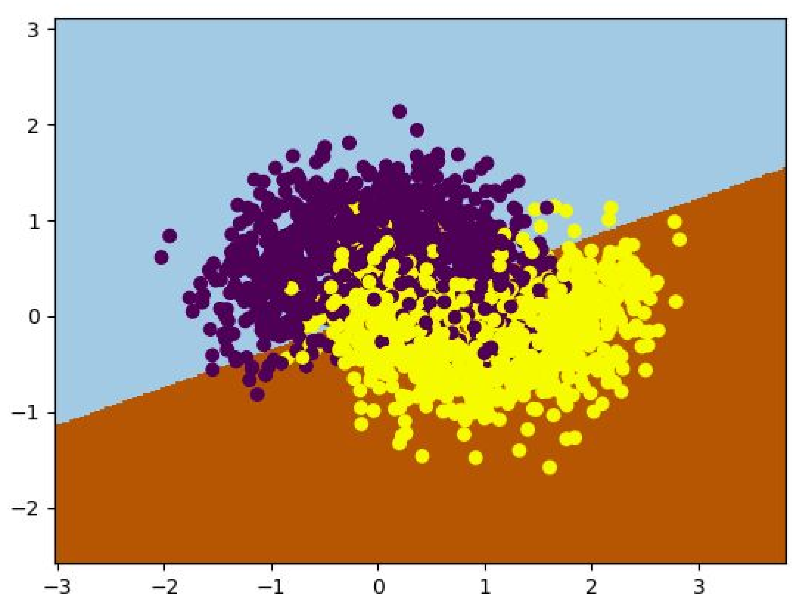
- Data 5

K =	1	2	3	4	5
c=1	80.75	86.5	85.5	86.5	82.75
c=0.1	81.5	86.25	85.0	85.25	82.75
c=10	80.75	86.5	85.5	86.5	82.75

The accuracy is fluctuating with c. But c=1, is consistently giving best results. Therefore, moderate level of overfitting will give good result in this case.

<u>Confusion matrix </u>(I have taken only for c=1 and for all K-folds)

[[837. 149.]
 [163. 851.]]



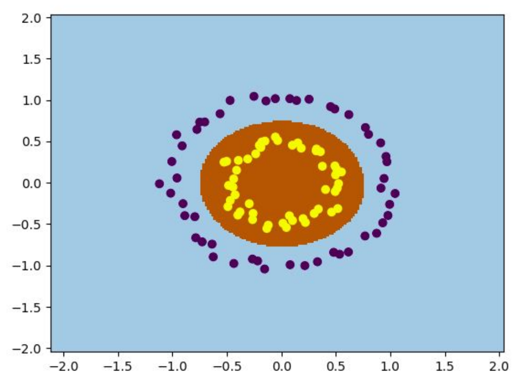
2. RBF Kernel

-data 1

K=	1	2	3	4	5
	100.0	100.0	100.0	100.0	100.0

<u>Confusion matrix</u>

[[50.   0.]
 [0. 50.]]

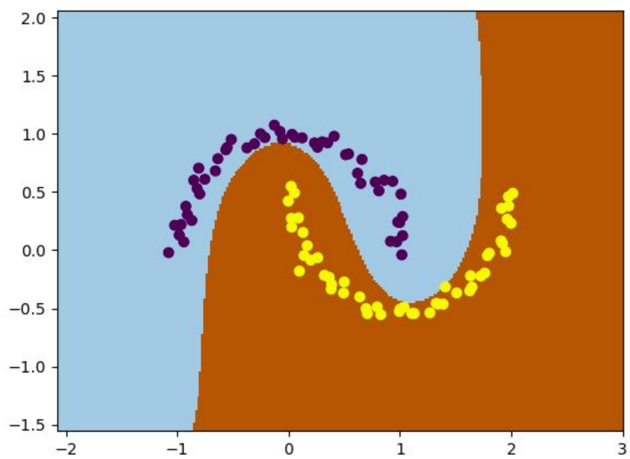


- Data 2

K=	1	2	3	4	5
	100.0	100.0	100.0	100.0	100.0

Confusion matrix

[[50.   0.]
 [0. 50.]]

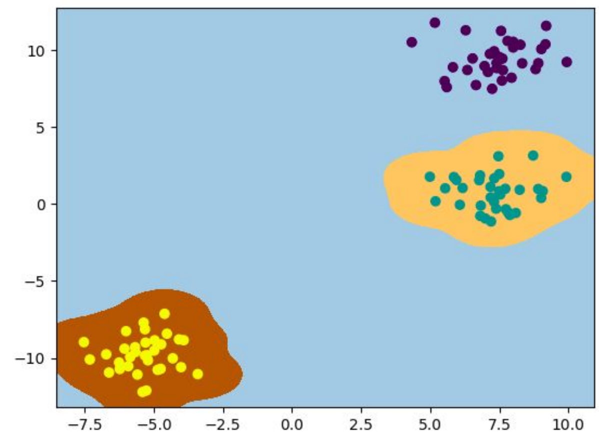


- **Data 3**

K=	1	2	3	4	5
	100.0	100.0	100.0	100.0	100.0

Confusion Matrix

[[34.   0.   0.]
 [0. 33.   0.]
 [0.   0. 33.]]

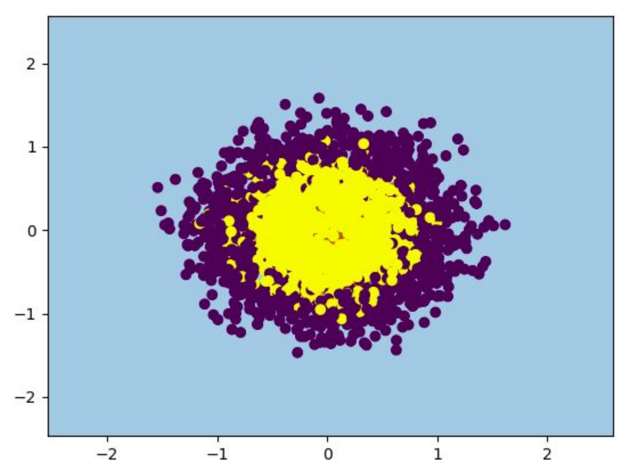


- Data 4

K=	1	2	3	4	5
	85.5	87.75	89.25	88.75	89.25

<u>Confusion Matrix</u>

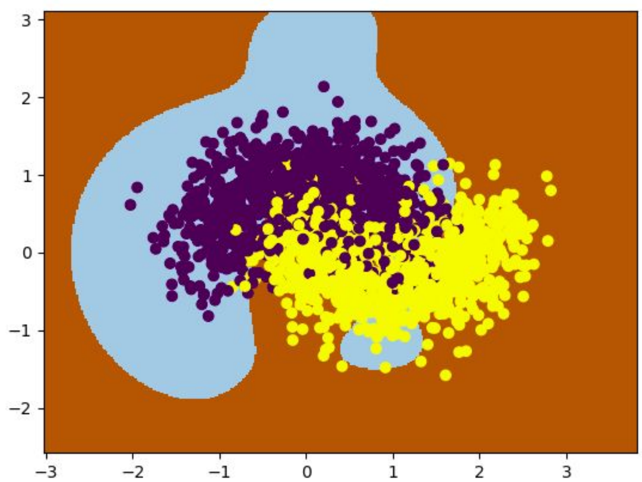
[[901. 139.]
 [99. 861.]]



- **Data 5**

K=	1	2	3	4	5
	80.0	85.75	83.75	84.0	79.0

[[814. 164.]
 [186. 836.]]



1. Linear kernel

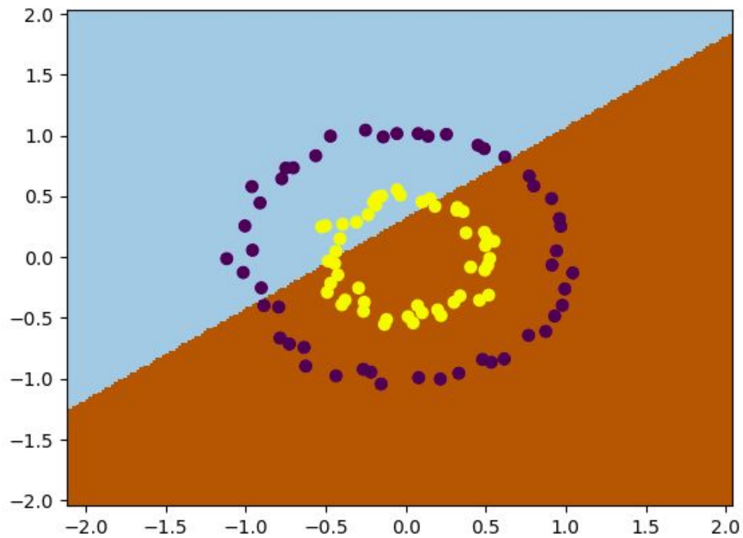
b) One-vs-One

```
k = 1, C = 1
50.0
k = 1, C = 0.1
50.0
k = 1, C = 10
50.0
k = 2, C = 1
35.0
k = 2, C = 0.1
35.0
k = 2, C = 10
35.0
k = 3, C = 1
40.0
k = 3, C = 0.1
40.0
k = 3, C = 10
40.0
k = 4, C = 1
40.0
k = 4, C = 0.1
45.0
k = 4, C = 10
40.0
k = 5, C = 1
30.0
k = 5, C = 0.1
35.0
k = 5, C = 10
35.0
[[ 17.  28.]
 [ 33.  22.]]
```

Data 1

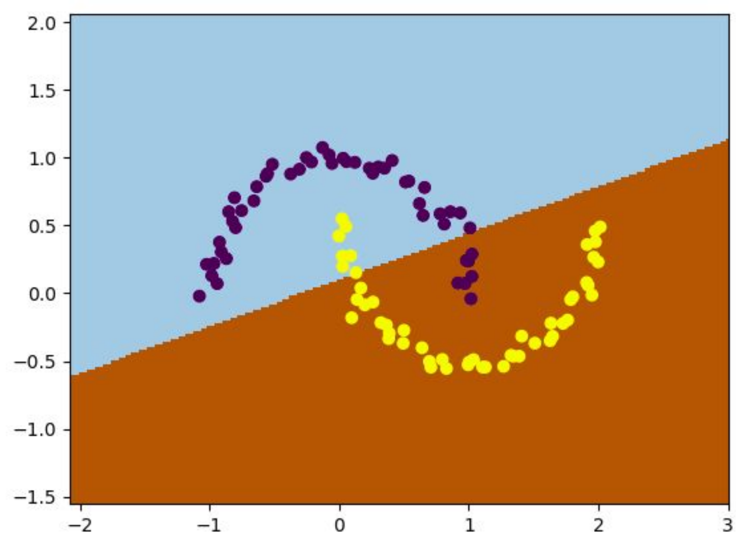
```
k = 1, C = 1
75.0
k = 1, C = 0.1
75.0
k = 1, C = 10
75.0
k = 2, C = 1
90.0
k = 2, C = 0.1
90.0
k = 2, C = 10
90.0
k = 3, C = 1
85.0
k = 3, C = 0.1
85.0
k = 3, C = 10
85.0
k = 4, C = 1
90.0
k = 4, C = 0.1
90.0
k = 4, C = 10
90.0
k = 5, C = 1
85.0
k = 5, C = 0.1
85.0
k = 5, C = 10
85.0
[[ 42.   7.]
 [  8.  43.]]
```

Data 2



2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0

Data 1



2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2 −1 0 1 2 3

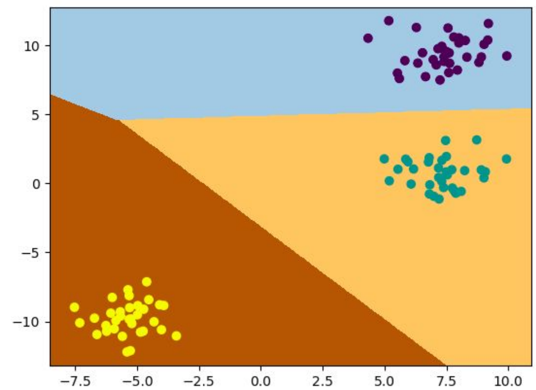
Data 2

- Data 3

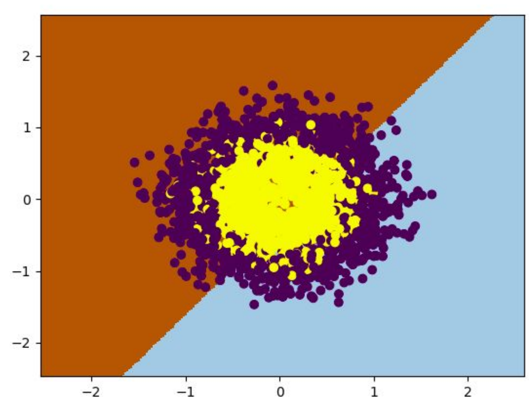
K =	1	2	3	4	5
c=1	100.0	100.0	100.0	100.0	100.0
c=0.1	100.0	100.0	100.0	100.0	100.0
c=10	100.0	100.0	100.0	100.0	100.0

<u>Confusion Matrix</u>

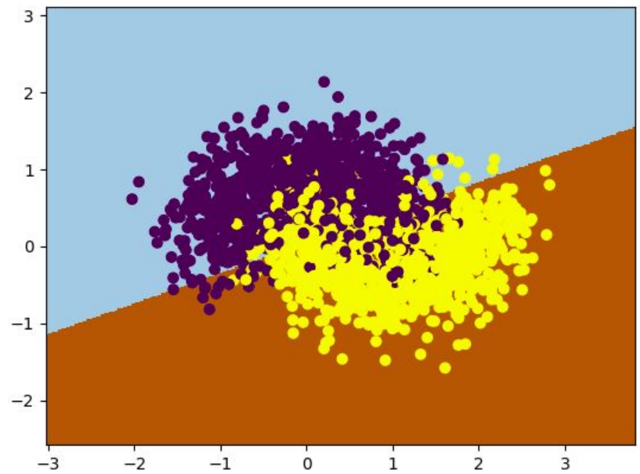
[[34.   0.   0.]
 [0. 33.   0.]
 [0.   0. 33.]]



- Data 4



- **Data 5**




```
k = 1, C = 1
80.75
k = 1, C = 0.1
80.75
k = 1, C = 10
80.75
k = 2, C = 1
86.5
k = 2, C = 0.1
86.5
k = 2, C = 10
86.5
k = 3, C = 1
85.5
k = 3, C = 0.1
85.0
k = 3, C = 10
85.5
k = 4, C = 1
86.5
k = 4, C = 0.1
86.5
k = 4, C = 10
86.5
k = 5, C = 1
82.75
k = 5, C = 0.1
82.75
k = 5, C = 10
82.75
[[ 837.  149.]
 [ 163.  851.]]
```

- **For rbf, graph is plotted.**

3. Kaggle

I have used tf idf vectorizer because our features were not of same dimension, e.g.

X1 = 22 342 52 213
X2 = 34 23

Both have different number of features. Thus, we converted the X1, X2….Xn into a string and passed them in tf idf vectorizer.

Then, I used SVM classifier. I tried many values of C, but C=0.3529 best fitted the data and predicted the best value. This is a small value of C, therefore, the classifier doesn't work too hard to avoid misclassification during our training. That is overfitting was avoided. And because of tiny value of C it is a large margin svm classifier.

For removing outliers, I trained the classifier with X,Y. Then predicted the values for same X. Then I compared the predicted values from original Y. If they matches then I kept it, Else I removed it from X and Y. This gave us a new X and Y which I further used to fitting the classifier.

<u>**OLD DATA SET**</u>

Part C

```
k = 1, C = 1
99.375
k = 1, C = 0.1
100.0
k = 1, C = 10
99.375
k = 2, C = 1
100.0
k = 2, C = 0.1
100.0
k = 2, C = 10
100.0
k = 3, C = 1
100.0
k = 3, C = 0.1
100.0
k = 3, C = 10
100.0
k = 4, C = 1
100.0
k = 4, C = 0.1
100.0
k = 4, C = 10
100.0
k = 5, C = 1
100.0
k = 5, C = 0.1
100.0
k = 5, C = 10
100.0
[[ 392.    0.]
 [   1.  407.]]
```

Earlier, DT- 95

Logistic - 100

Gaussian Naive Bayes- 96

SVM IS BETTER

Part A

```
k = 1, C = 0.1
81.0714285714
k = 1, C = 10
81.0714285714
k = 2, C = 1
82.1428571429
k = 2, C = 0.1
82.1428571429
k = 2, C = 10
82.1428571429
k = 3, C = 1
83.2142857143
k = 3, C = 0.1
83.2142857143
k = 3, C = 10
83.2142857143
k = 4, C = 1
81.4285714286
k = 4, C = 0.1
81.4285714286
k = 4, C = 10
81.4285714286
k = 5, C = 1
81.6666666667
k = 5, C = 0.1
81.6666666667
k = 5, C = 10
81.6666666667
[[ 366.    1.    5.    3.    2.    5.    4.    4.    8.    5.]
 [   0.  464.   12.    1.    4.    2.    3.    4.   15.    7.]
 [   3.    3.  306.   17.    3.    4.    7.    9.   15.    4.]
 [   2.    1.   16.  341.    1.   18.    1.    9.   13.   11.]
 [   2.    1.   12.    2.  337.    7.    3.    6.    6.   34.]
 [  10.    3.    3.   27.    7.  301.   16.    2.   35.    5.]
 [   6.    2.    4.    1.    2.   16.  397.    1.    9.    0.]
 [   2.    3.   10.    7.    3.   13.    1.  358.    4.   36.]
 [   6.   14.   22.   17.   13.   23.    5.    4.  288.    9.]
 [   3.    2.    3.    8.   44.    9.    4.   34.   17.  282.]]
```

Earlier,

DT - 75%
Logistic- 82% ; Gaussian Naive Bayes - 57% ;SVM IS BETTER

One-One VS One-Every

Accuracy - one vs one
Time Consuming- one vs one