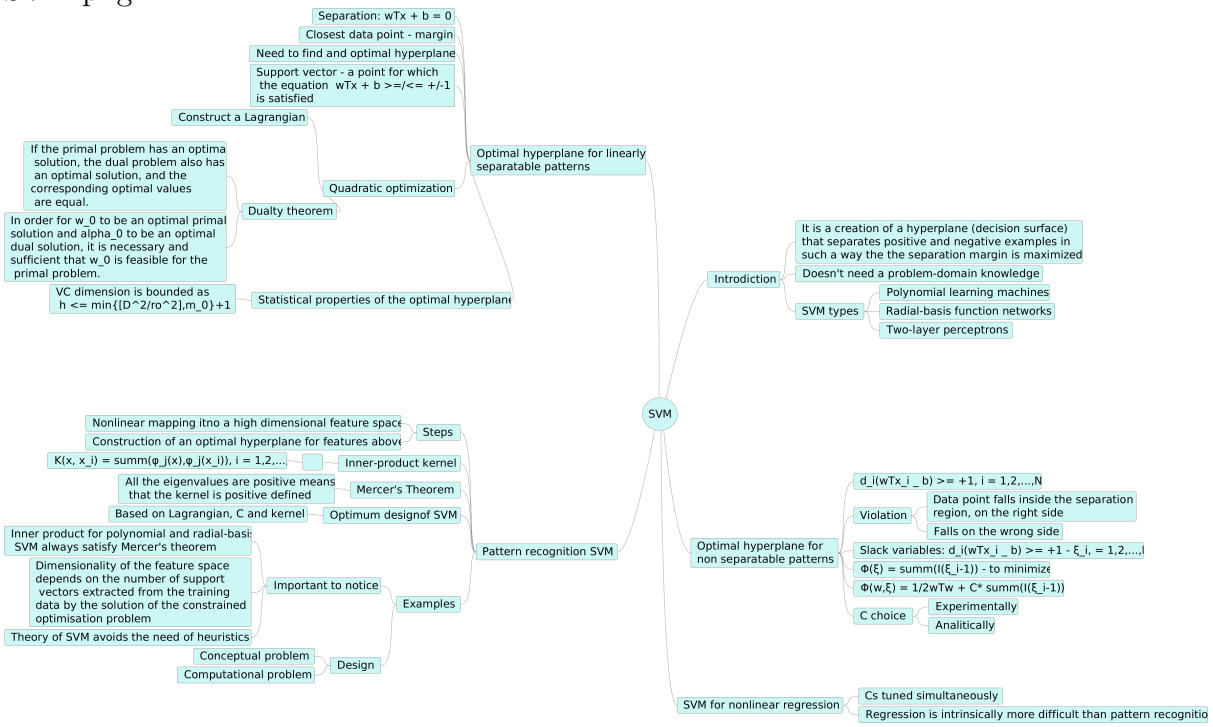
Neural Networks
- Homework 9 -

Petr Lukin, Evgeniya Ovchinnikova

Lecture date: 21 November 2016

1 Mind map

Figure 1: Mind map. Chapter 6 from Haykin's book. A zoomed version is attached as SVM.png



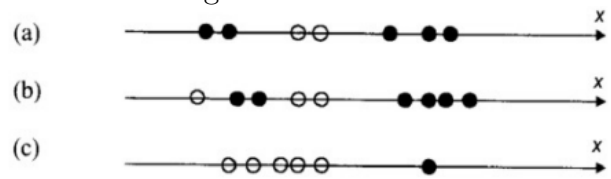
2 Exercises

2.1 Exercise 2

The graphs below represent three different one-dimensional classification (dichotomization) tasks (along a sketched x-axis, dash means "no data point"). What is the lowest-order polynomial decision function that can correctly classify the given data? Black dots d note class 1 with target function value y1 = +1 and white dots depict class 2 with targets y2 = -1. What are the decision boundaries?

If you wanted to classify the data sets (a), (b), (c) from Fig.2 using SVMs with Gaussian basis functions, how many hidden layer neurons would you need for each problem?

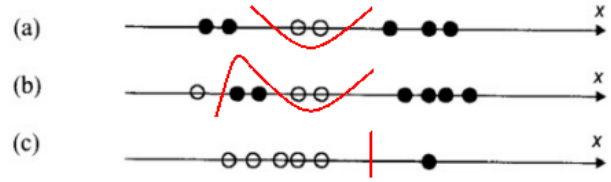
Figure 2: The dots



Solution:

From the positive and negative examples distribution we can assume that (a) requires a second order polynomial function, (b) - third degree and (c) - just a linear one (Fig. 3).

Figure 3: The dots with classification polynoms



To proof it we used the following code:

```
from itertools import product

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.svm import SVC
import sklearn.svm as svm

Xa = np.array([[1,0],[2,0],[5,0],[6,0],[9,0],[11,0],[12,0]])
ya = np.array([-1, -1, 1, 1, -1, -1,-1])
Xb = np.array
    ([[1,0],[3,0],[4,0],[6,0],[7,0],[10,0],[11,0],[12,0],[14,0]])
yb = np.array([1, -1, -1, 1, 1, -1, -1, -1, -1])
Xc = np.array([[1,0],[2,0],[3,0],[4,0],[5,0],[9,0]])
yc = np.array([1,1,1,1,1,-1])

def classify_and_plot(X,Y, degr):
    clf = svm.SVC(kernel='poly', coef0 = 1, probability=True,
        degree = degr, gamma=2)
    clf.fit(X, Y)
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    h = 0.1
    x, y = np.meshgrid(np.arange(x_min, x_max, h), np.arange(
        y_min, y_max, h))
    #print np.c_[xx.ravel(), yy.ravel()]
```

```
    Z = clf.predict(np.c_[x.ravel(), y.ravel()])
    Z = Z.reshape(x.shape)
    plt.pcolormesh(x, y, Z, cmap=plt.cm.Paired)
    plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
    plt.title('SVM classification')
    plt.axis('tight')
    plt.grid()
    plt.show()
```

First, we will try to classify all the cases with degree = 1:

```
classify_and_plot(Xa, ya, 1)
classify_and_plot(Xb, yb, 1)
classify_and_plot(Xc, yc, 1)
```

The results are shown in Fig. 4. We can see that the classification for (c) case is correct, so a linear function is fine for this case. Classifications for (a) and (b) are wrong.

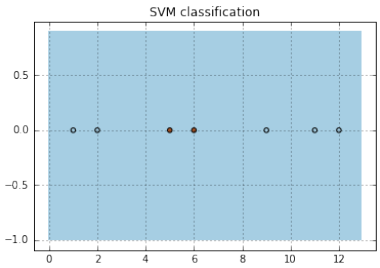


Figure 4: Classification for (a) case with degree = 1

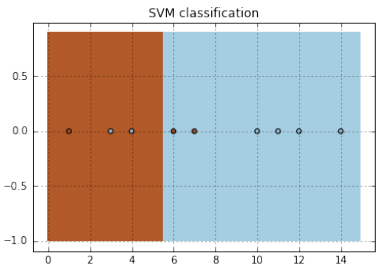


Figure 5: lassification for (b) case with degree = 1

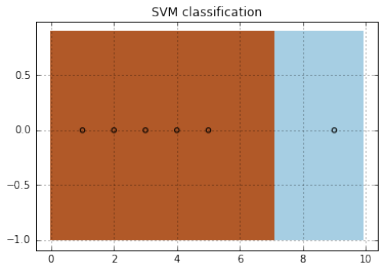


Figure 6: Alassification for (c) case with degree = 1

So, we will test (a) and (b) with degree = 2:

```
classify_and_plot(Xa, ya, 2)
classify_and_plot(Xb, yb, 2)
```

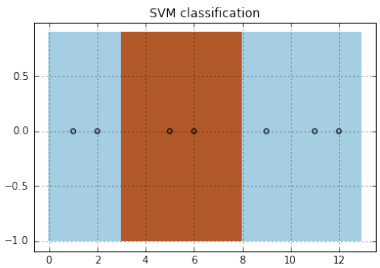


Figure 7: Classification for (a) case with degree = 2

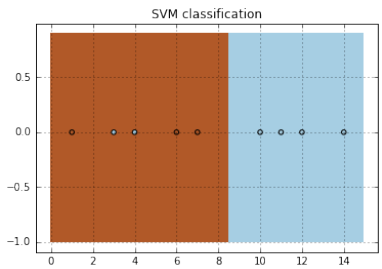
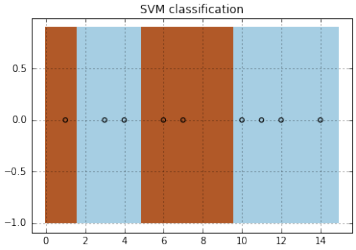


Figure 8: lassification for (b) case with degree = 2

In Fig. 7 we see that, as expected, classification for (a) is good, but classification for (b) still failing. So, we need to try degree = 3:

```
classify_and_plot(Xb, yb, 3)
```

Figure 9: Classification for (b) case with degree = 3



If we wanted to classify the data sets (a), (b), (c) using SVMs with Gaussian basis functions, we to use another - 'rbf' - kernel function:

```
clf = svm.SVC(kernel='rbf', coef0 = 1, probability=True, degree
    = degr, gamma=2)
```

We will obtain the following results:

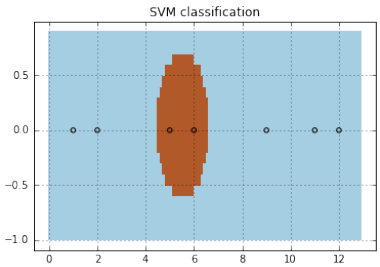


Figure 10: Classification for (a) case with Gaussian basis function

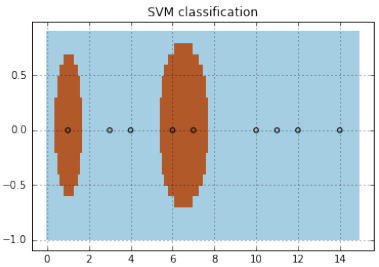


Figure 11: lassification for (b) case with Gaussian basis function

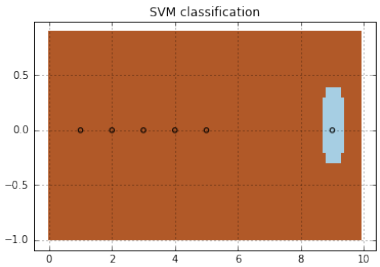


Figure 12: Alassification for (c) case with Gaussian basis function

To determine the number of neurons we need a number of features in a feature space that depends on the number of support vectors extracted from the training data. From Fig. 10 we can see that in (a) case we have 4 support vectors, in (b) - 6 and in (c) - 2. So, we should have 4, 5 and 2 neurons.

2.2 Exercise 3

Download LIVSVM from http://www.csie.ntu.edu.tw/ cjlin/libsvm/. Study how to use it with MATLAB or python. In this exercise you have to use this. Figure 1.8 on next page shows a pair of moons facing each other in an asymmetrically arranged manner. The moon labeled Region A is positioned symmetrically with respect to the y-axis, whereas the moon labeled Region B is displaced to the right of the y-axis by an amount equal to the radius r and below the x-axis by the distance d. The two moons have identical parameters:

- Radius of each moon, r = 10,

- Width of each moon, w = 6.

The vertical distance d separating the two moons is adjustable; it is measured with respect to the x-axis, as indicated in the figure 13.

- Increasingly positive values of d signify increased separation between the two moons;
- Increasingly negative values of d signify the two moons coming closer to each other.

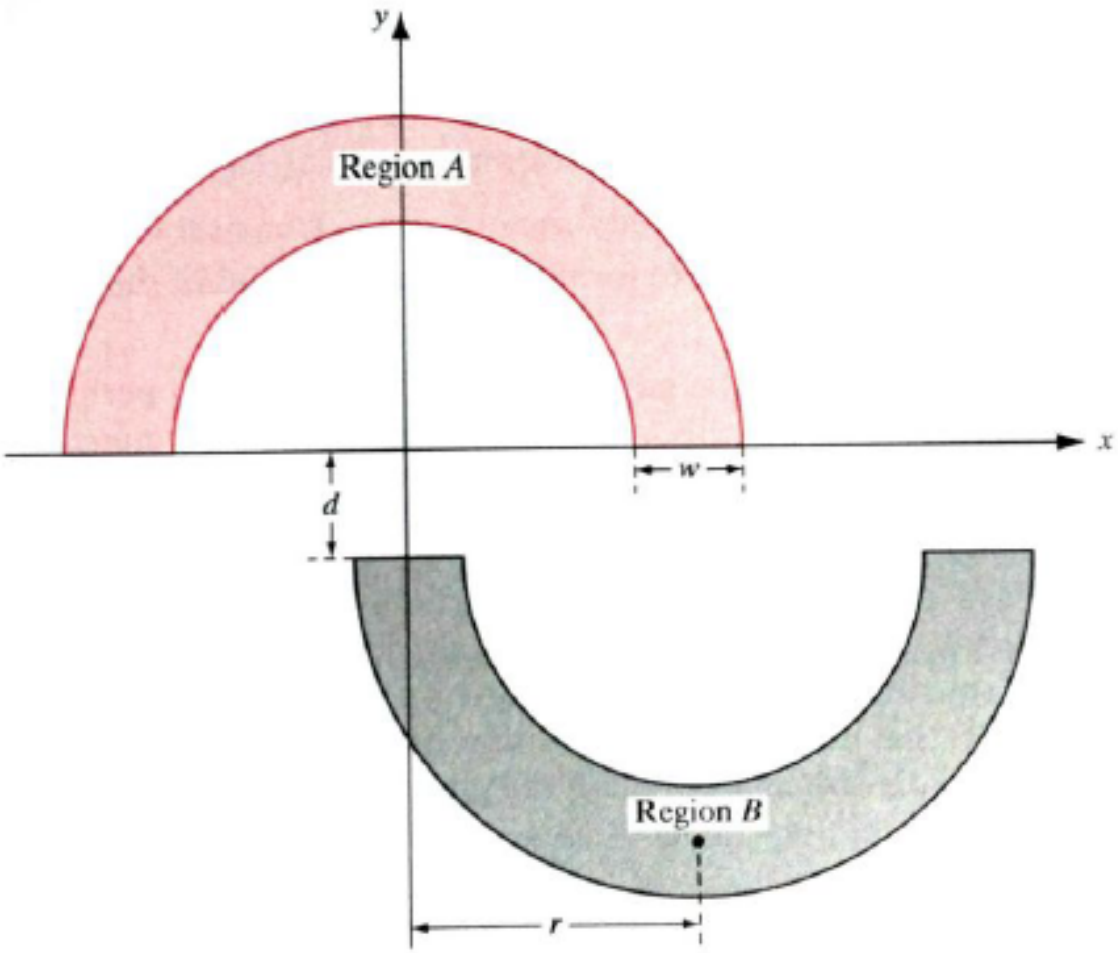


FIGURE 1.8 The double-moon classification problem.

Figure 13: Data distribution

The training sample consists of 1000 pairs of data points, with each pair consisting of one point picked from region A and another point picked from region B, both randomly. The test sample consists of 3,000 pairs of data points, again picked in a random manner. Tasks: Your task is to classify the dataset using SVM (Support Vector Machine) for some cases given below. Generate the dataset for each case and classify using different kernels (e.g. linear, polynomial, radial basis etc.) Show the decision boundary (Plotting the classified points using different color will be enough)

- Case 1: $d = 0$
- Case 2: $|d| = 1/2*$(radius of moons inner half-circle) and d is negative i.e. d is in the upper side of x-axis.
- Case 3: Increase d negatively such that both of the moons touch each other.
- Case 4: Both moons overlap each other
- Case 5: Add some noise in the training set

Try to experiment with different options in svmtrain (LIBSVM). Comment on your findings. This experiment is taken from Haykins book (3rd edition) which is introduced in the first chapter and continued throughout the book (chapter 2, 3, 4). For intuition, you can take a look in it.
Solution to this problem includes 2 MATLAB programs. One is used for double moon data generation. The user can choose number of data points, parameter d and add gaussian noise to data with variance.

```
function [ moons,labels ] = generate_moons( d,points,noise)
%% Exercise 3 Neural networks.
%Generate data points in shape of double moons
%Authors P.Lukin, E. Ovchinnikova

rA = 7+6*rand(points,1);
rB = 7+6*rand(points,1);
phiA = pi*rand(points,1);
phiB = pi*rand(points,1);
moons = [rA.*cos(phiA) rA.*sin(phiA); 10+rB.*cos(phiB) -d-rB.*
    sin(phiB)];

%Random noise addition
moons =moons+normrnd(0,noise,2*points,2);

%Data labelling
labels = ones(points,1);
labels = [labels; -ones(points,1)];

%Shuffle the data
shuffle = randperm(2*points);
moons = moons(shuffle,:);
labels = labels(shuffle,:);
end
```

The next function uses generated data to train SVM based on chosen kernel type. Then, classification is applied to test data set and finally, uniformly distributed datapoints are classified to find decision boundary.

```
%% Exercise 3 Neural networks.
%Train and test SVM
%Authors P.Lukin, E. Ovchinnikova
close all
```

```
clear all
clc
%Train and test data generation
d = -7;
[moons,labels] = generate_moons(d,1000,0);
[test_moons,test_labels] = generate_moons(d,3000,0);

%Plot train data
figure(1)
plot(moons(:,1),moons(:,2),'b.')
grid on
axis equal
title('Data distribution')
xlabel('x')
ylabel('y')

%Train SVM with defined kernel
model = svmtrain(labels, moons,'-t 3 -d 5');
[predicted_label, accuracy, decision_values] =svmpredict(
    test_labels, test_moons, model, [, 'libsvm_options']);
x = test_moons(:,1);
y = test_moons(:,2);

%Classification results
figure(2)
hold on
scatter(x(predicted_label == 1), y(predicted_label == 1), 'g', '
    .')
scatter(x(predicted_label == -1), y(predicted_label == -1), 'r',
     '.')
grid on
axis equal
title('Data classification')
xlabel('x')
ylabel('y')
hold off

%Decision boundary, generate uniform points and apply
    classification
uniform_points = [-15+40*rand(10000,1) -15+40*rand(10000,1)];
[predicted_label, accuracy, decision_values] =svmpredict(rand
    (10000,1), uniform_points, model, [, 'libsvm_options']);

x = uniform_points(:,1);
y = uniform_points(:,2);

figure(3)
hold on
```

```
plot(moons(:,1),moons(:,2),'b.')
scatter(x(predicted_label == 1), y(predicted_label == 1), 'g', '.')
scatter(x(predicted_label == -1), y(predicted_label == -1),'r', '.')
grid on
axis equal
title('Decision boundary')
xlabel('x')
ylabel('y')
hold off
```

Classification results:

2.2.1 Case $d = 0$

The moons can be easily distinguished and hence, separated even with linear kernel with high precision. Other kernels have good results as well.

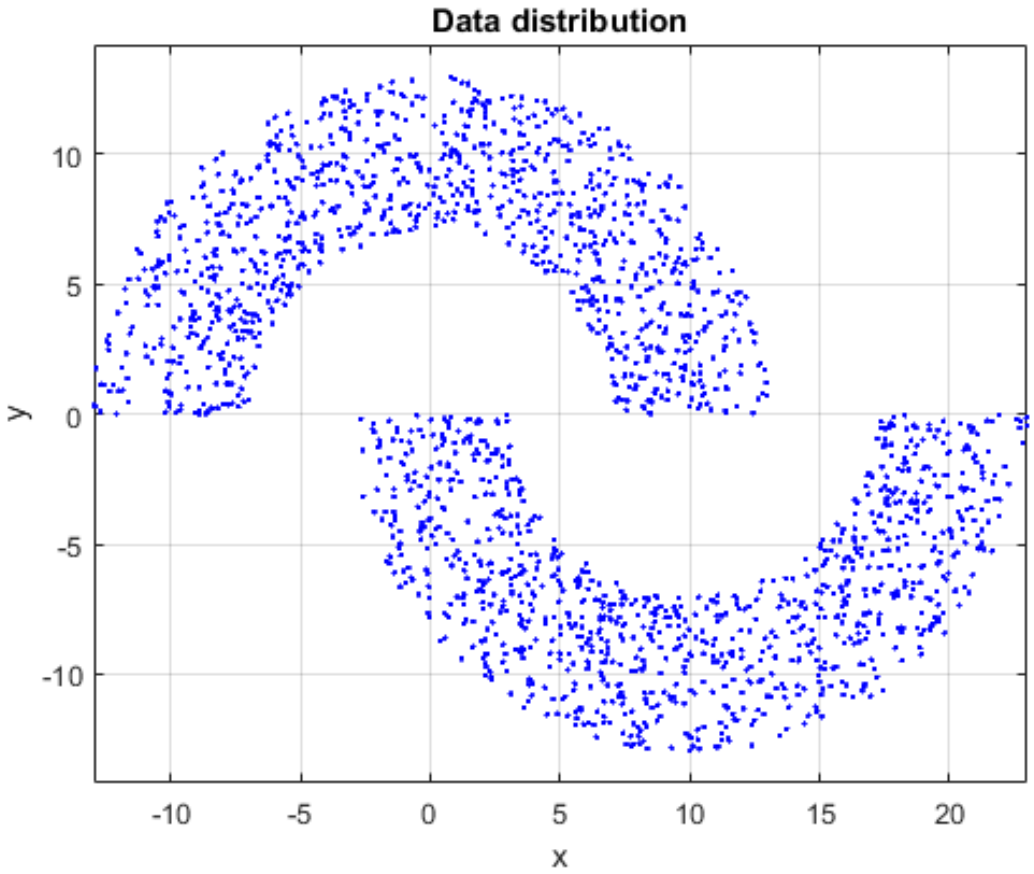


Figure 14: Data distribution

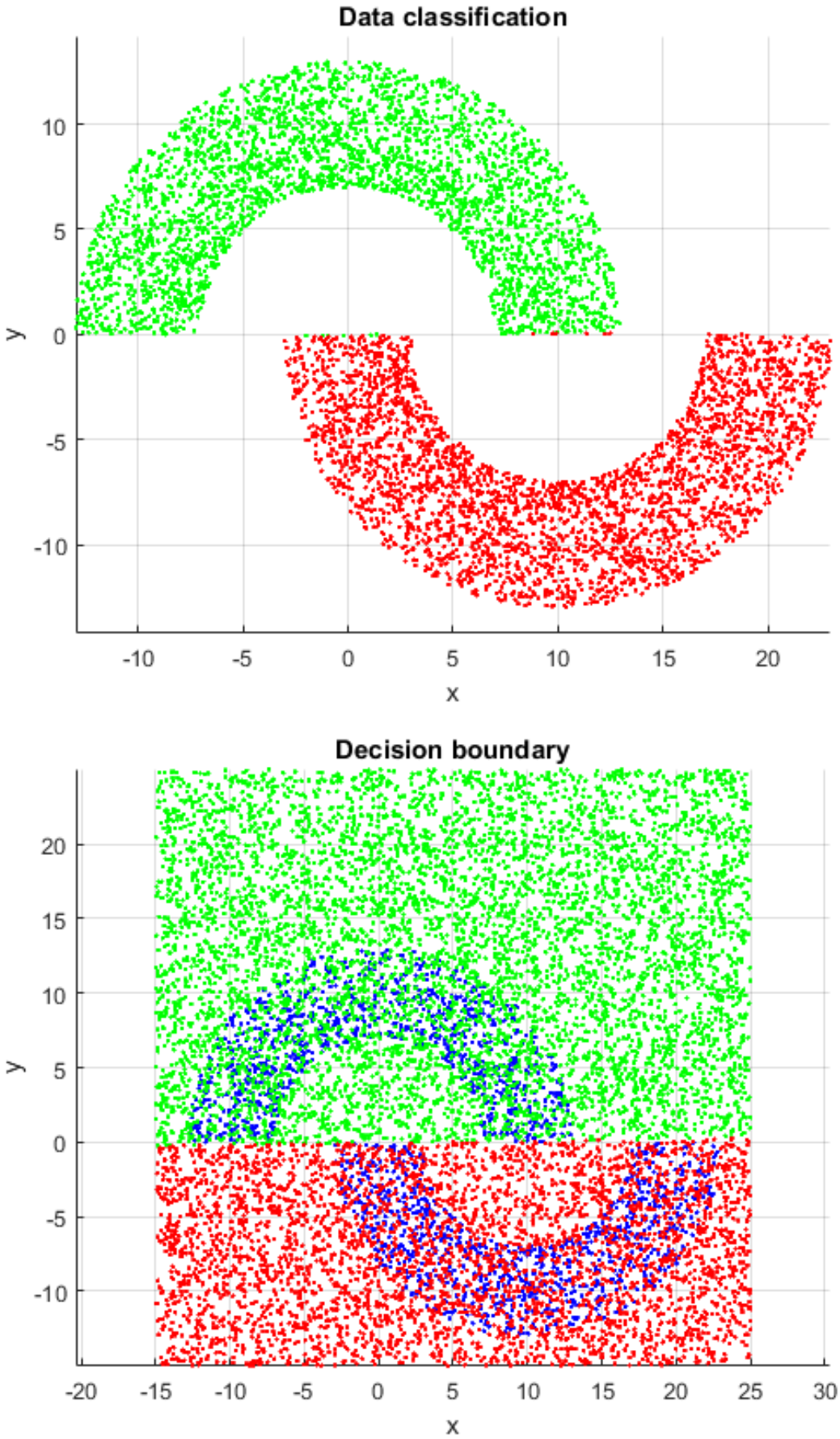


Figure 15: Linear kernel

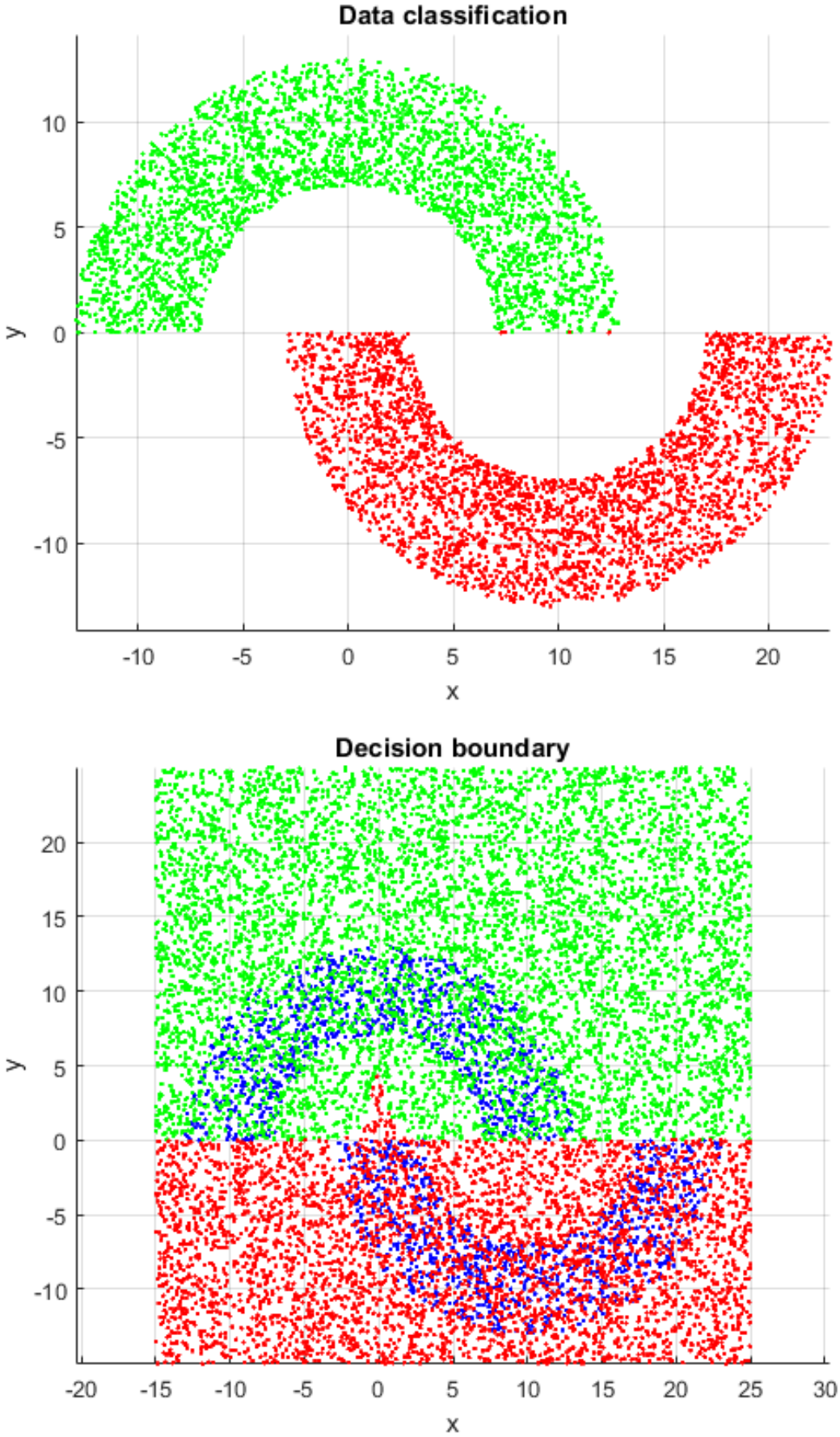
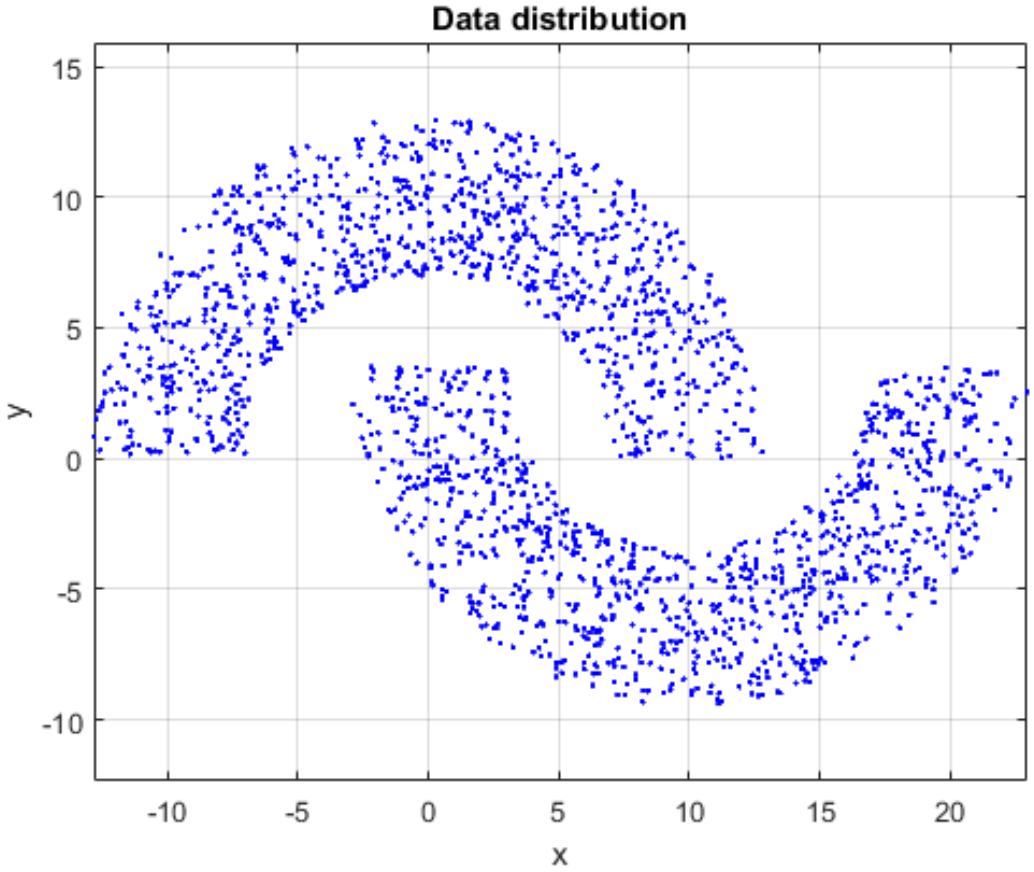


Figure 16: 3-rd order polynomial kernel

2.2.2 Case $d = -3.5$

In this case moons still can be distinguished, but there is no line that can separate them easily. So, linear kernel can not cope with it anymore. RBF kernel has the same good performance.



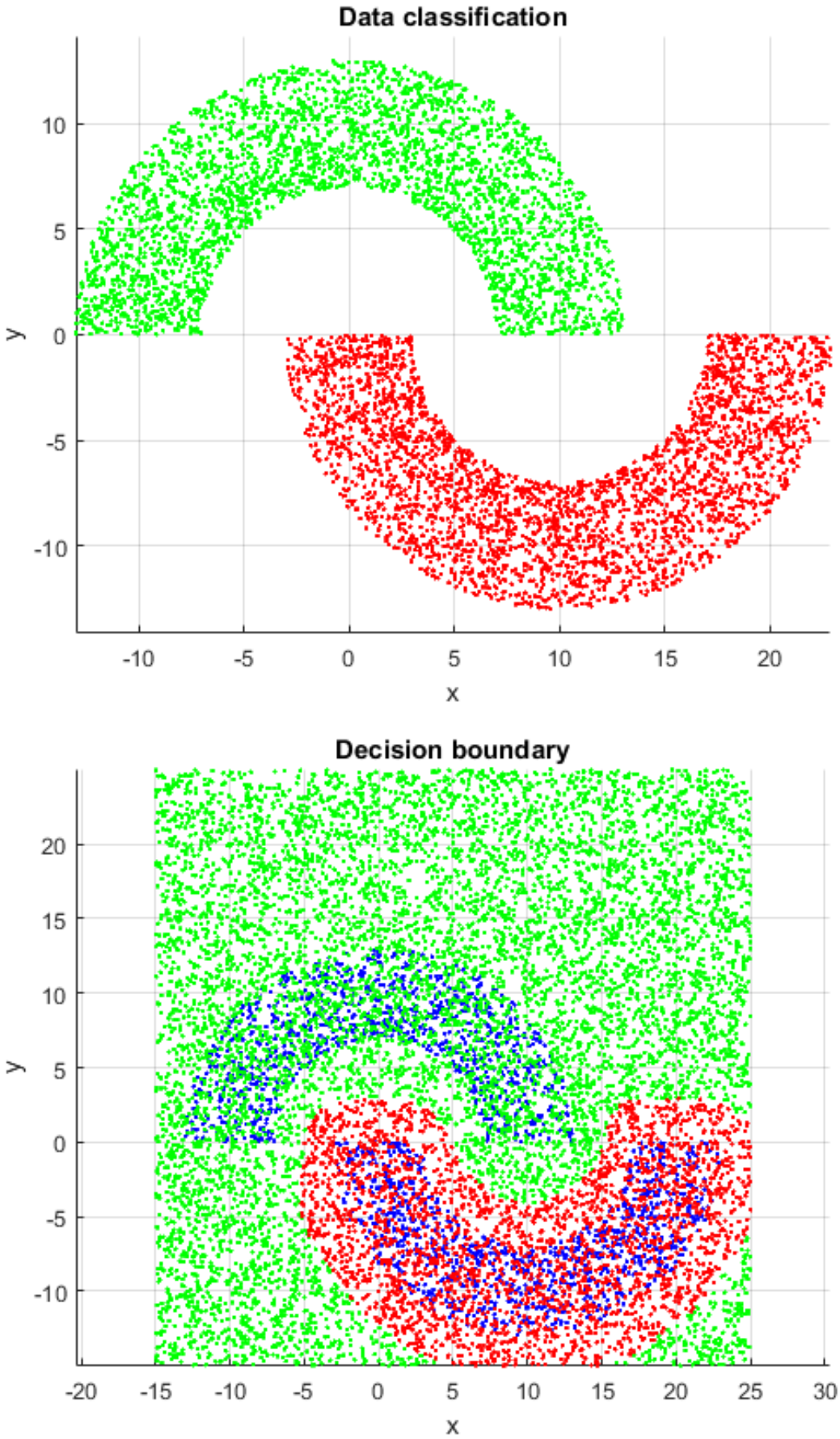


Figure 17: Gauss kernel

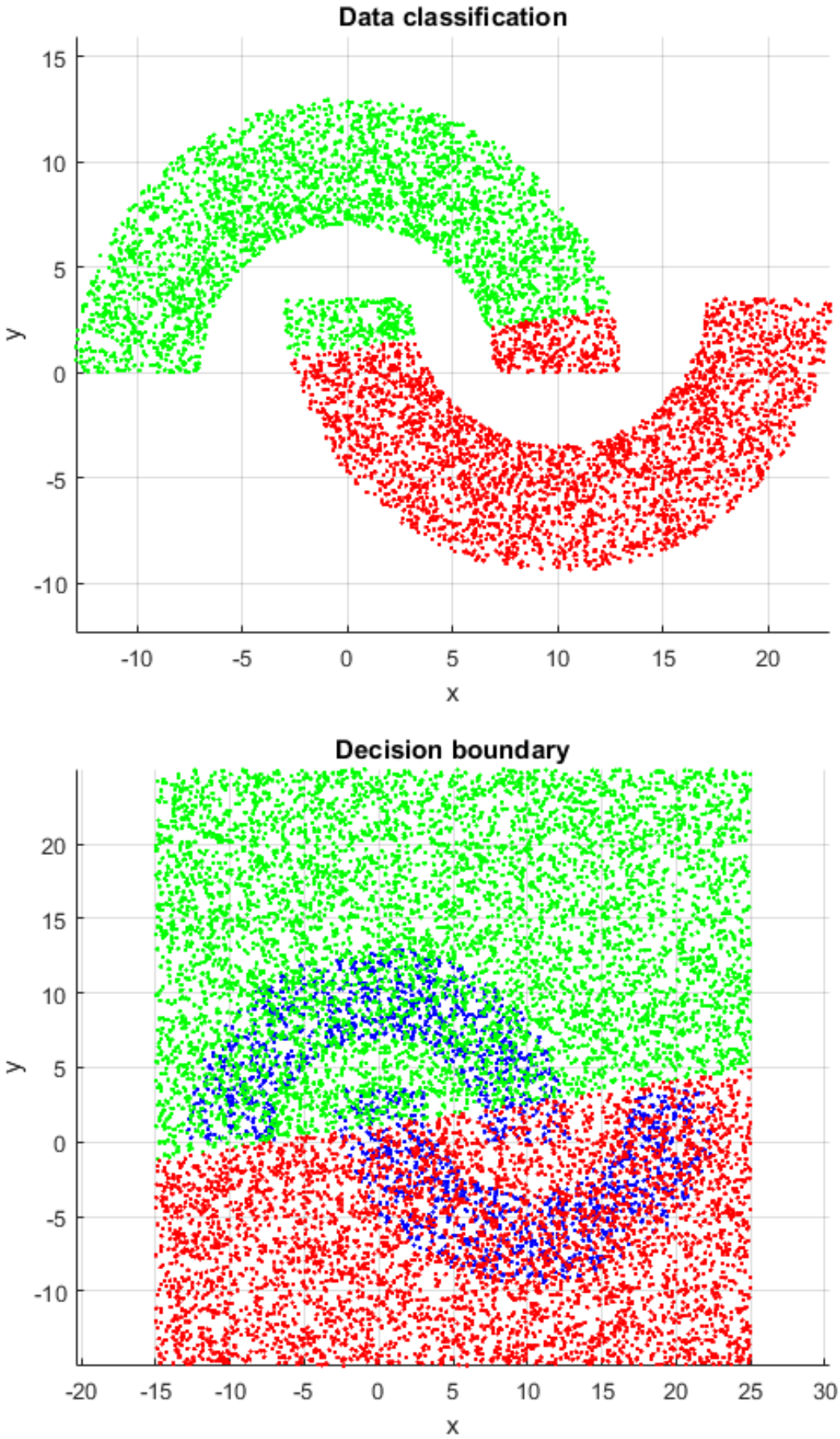


Figure 18: Linear kernel

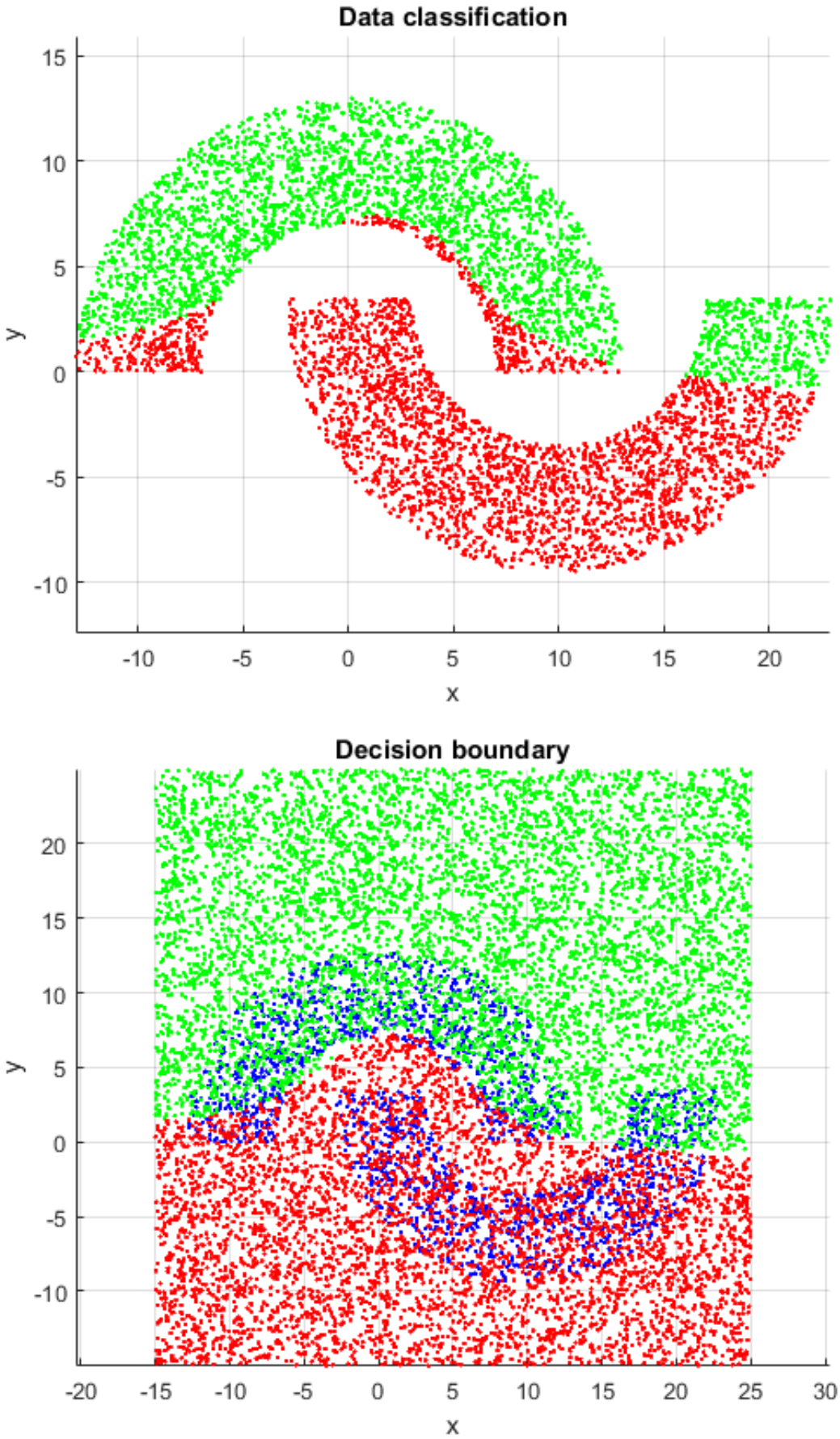
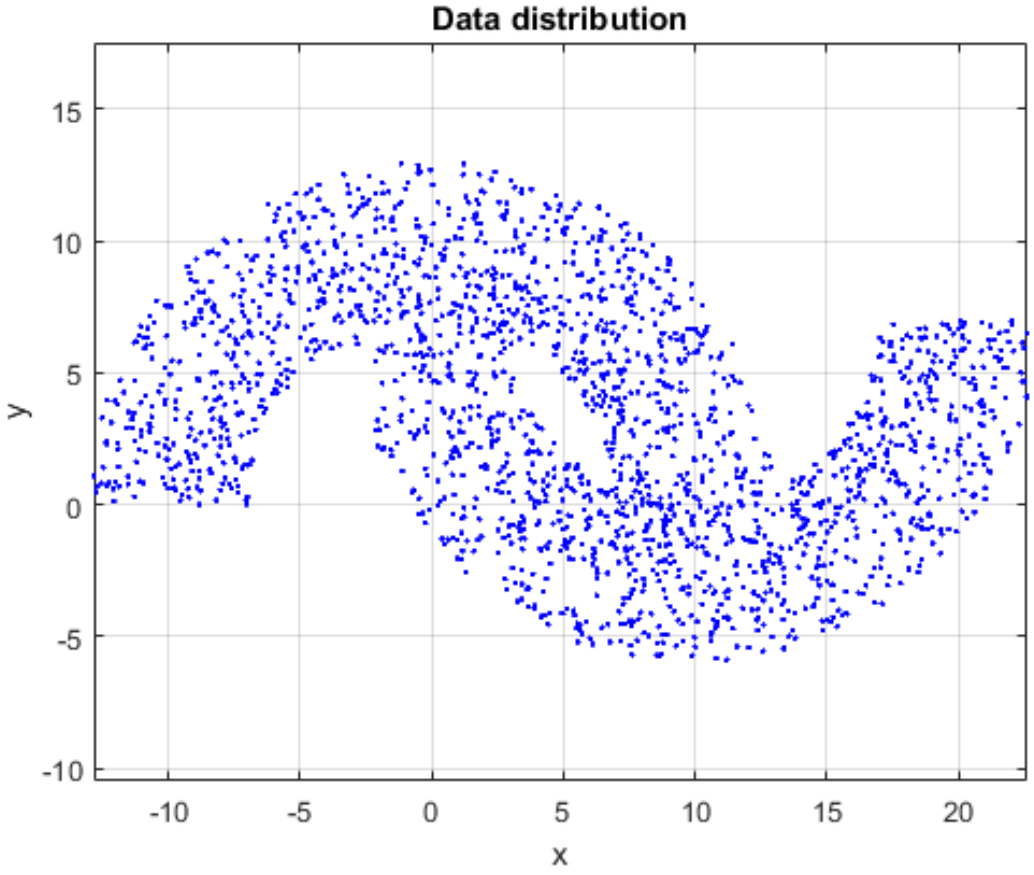


Figure 19: 3-rd order polynomial kernel

2.2.3 Case $d = -7$. Moons slightly connected

In this case moons are connected but still can be divided. Linear and polynomial kernel work worse that RBF kernel.



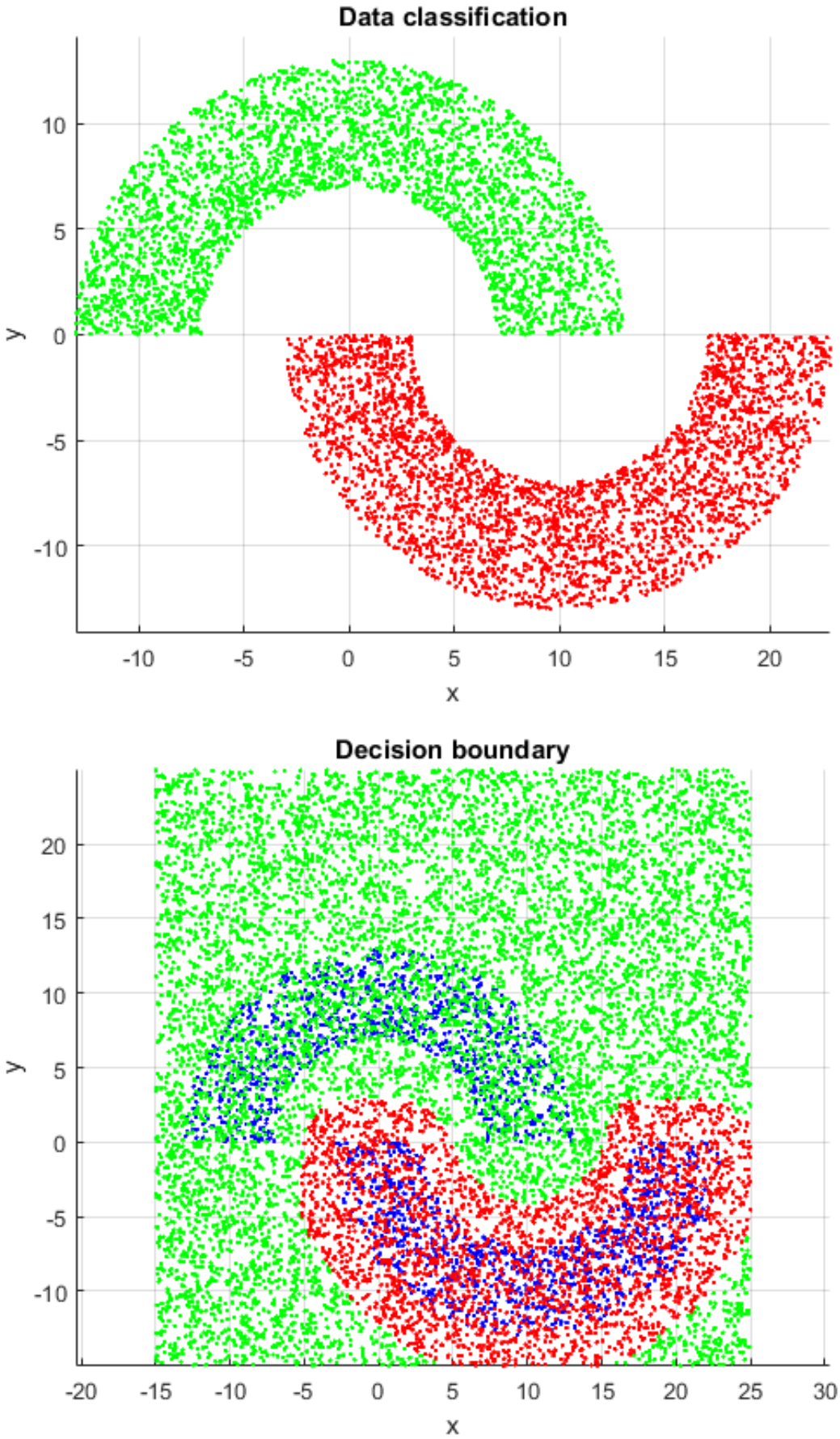


Figure 20: Gauss kernel kernel

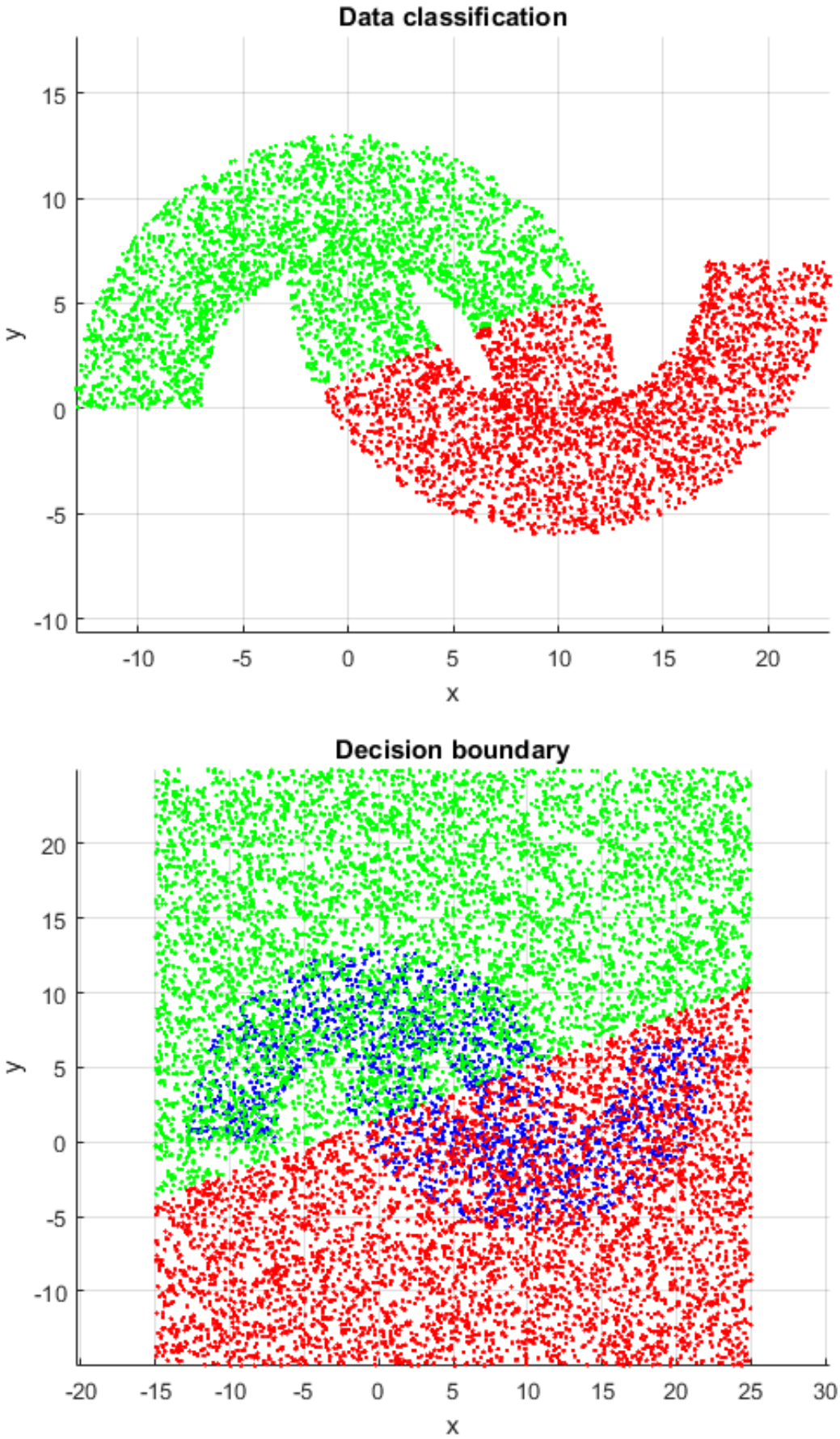


Figure 21: Linear kernel

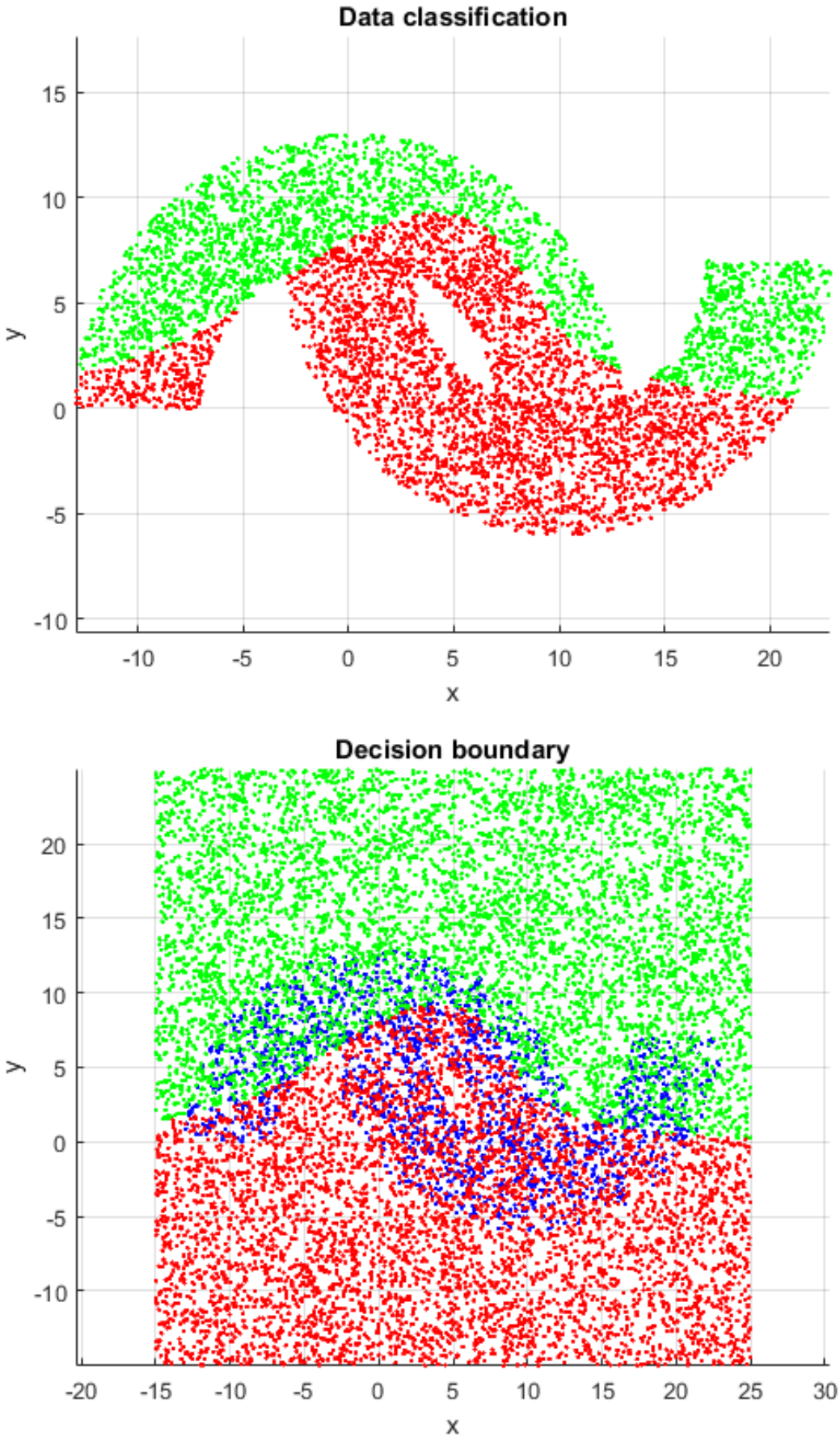
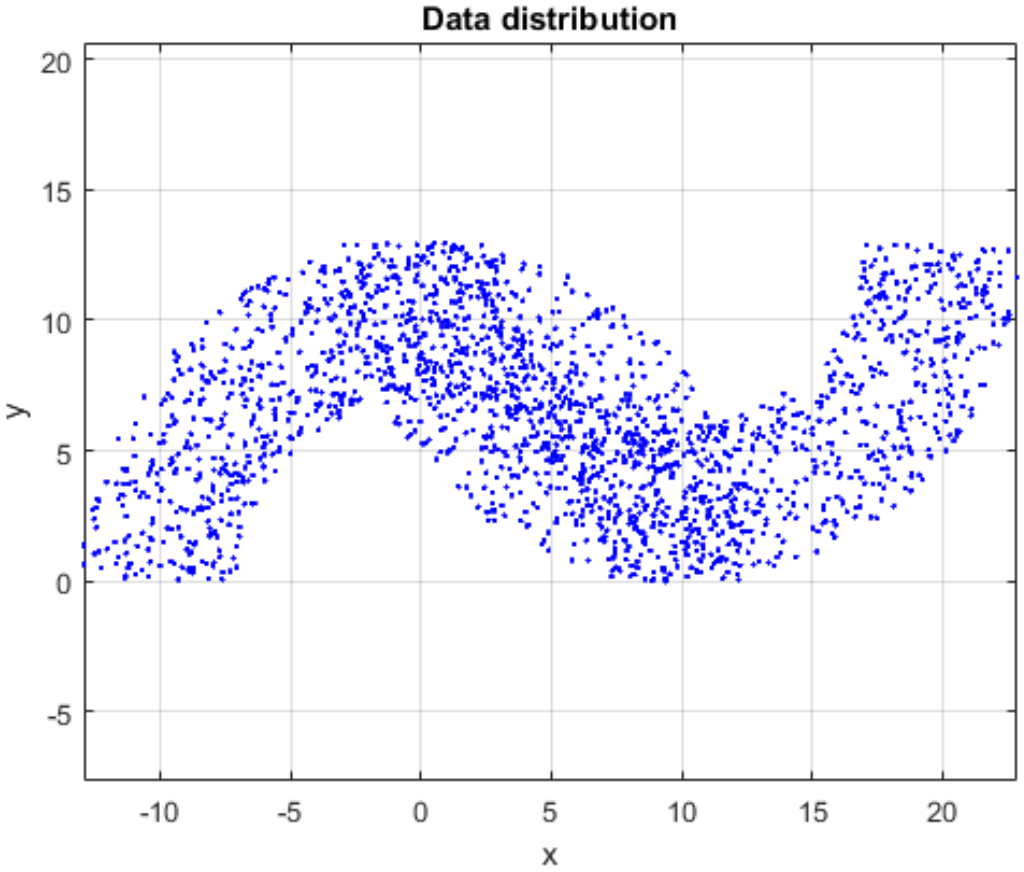


Figure 22: 3-rd order polynomial kernel

2.2.4 Case $d = -13$. Moons overlay

Moons can not be recognized by eyesight. All kernels, except RBF kernel show bad performance. RBF has accuracy around 78%. This accuracy do not grow with extra training data.



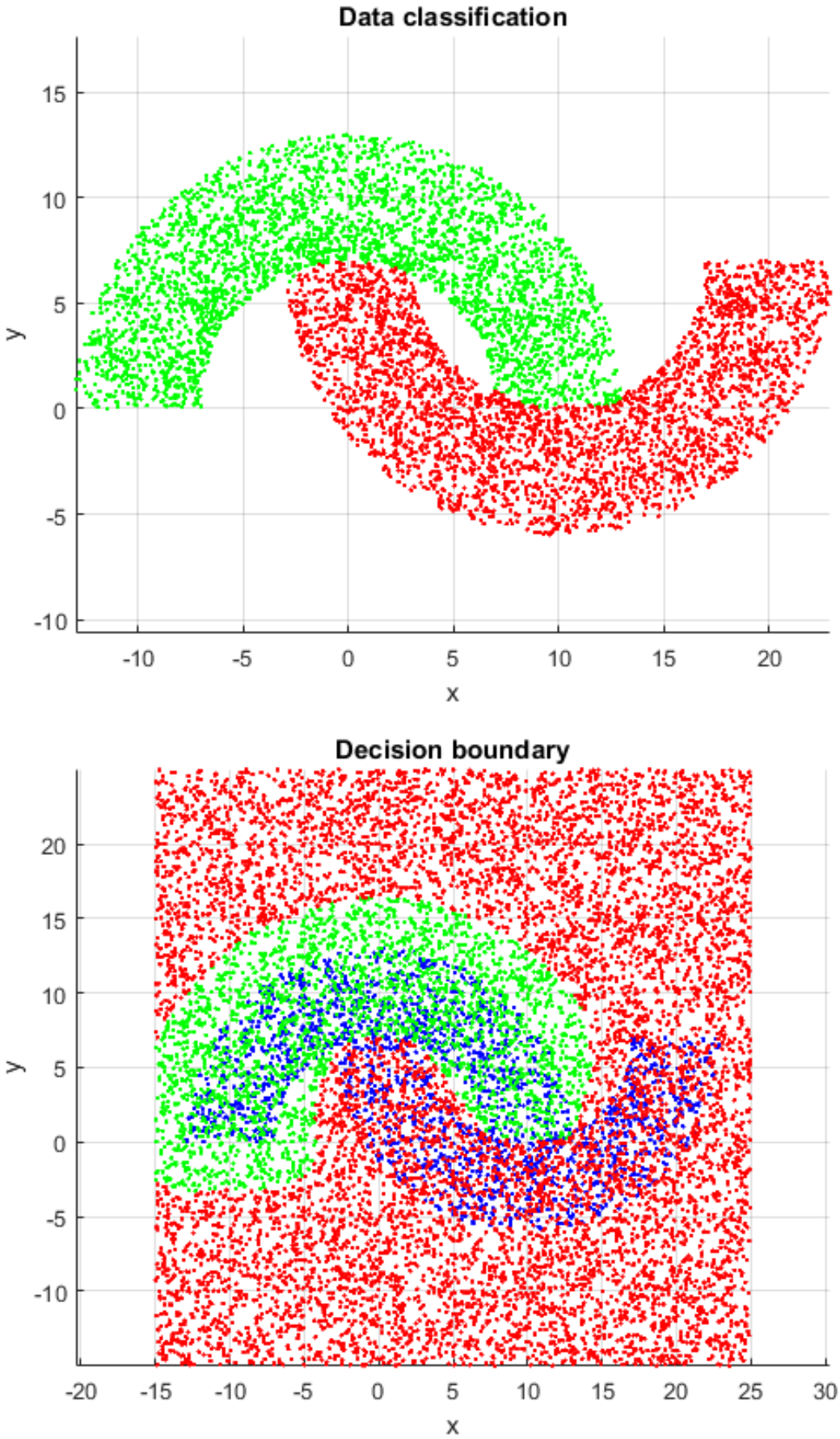


Figure 23: Gauss kernel

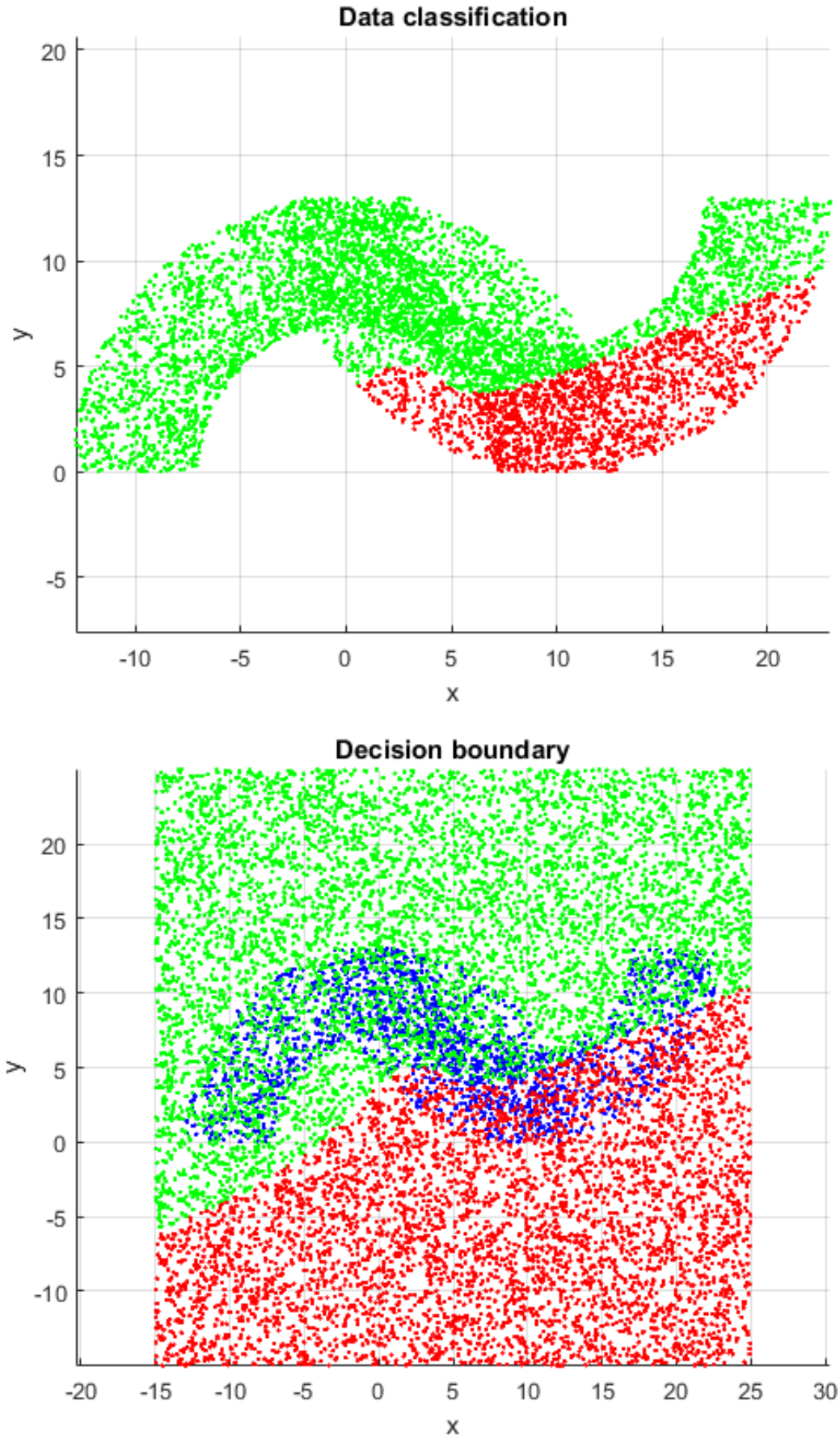
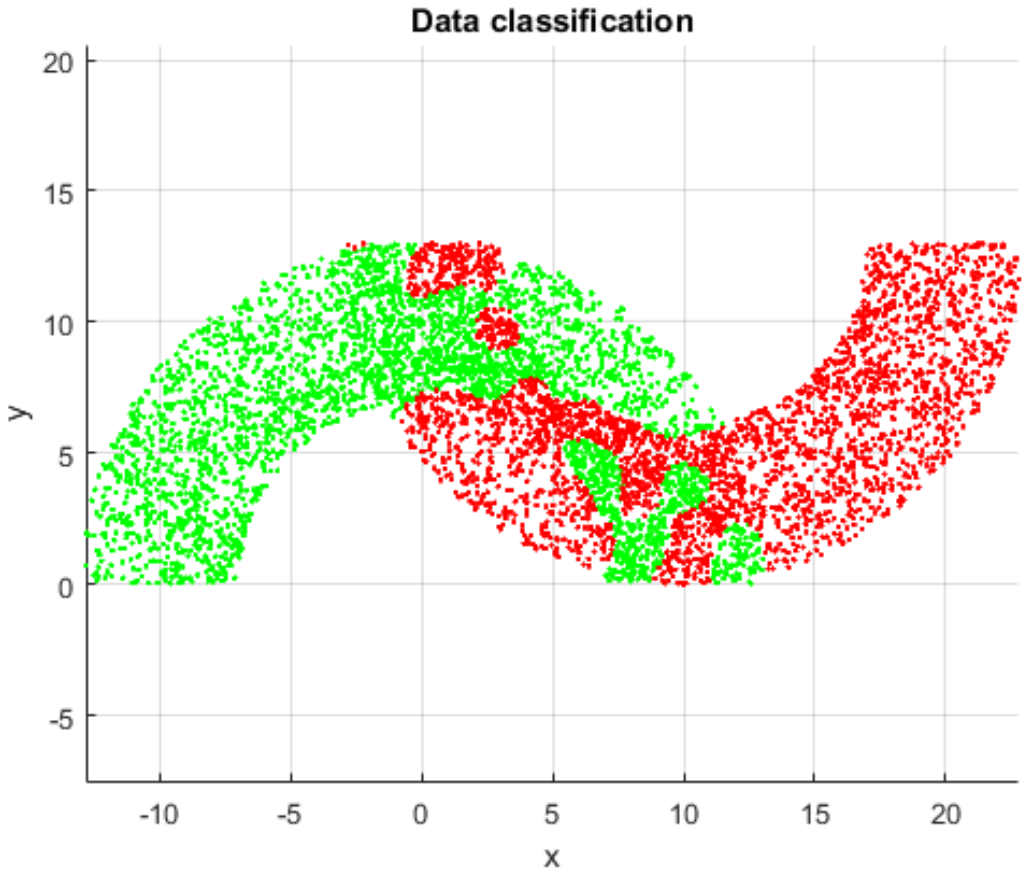
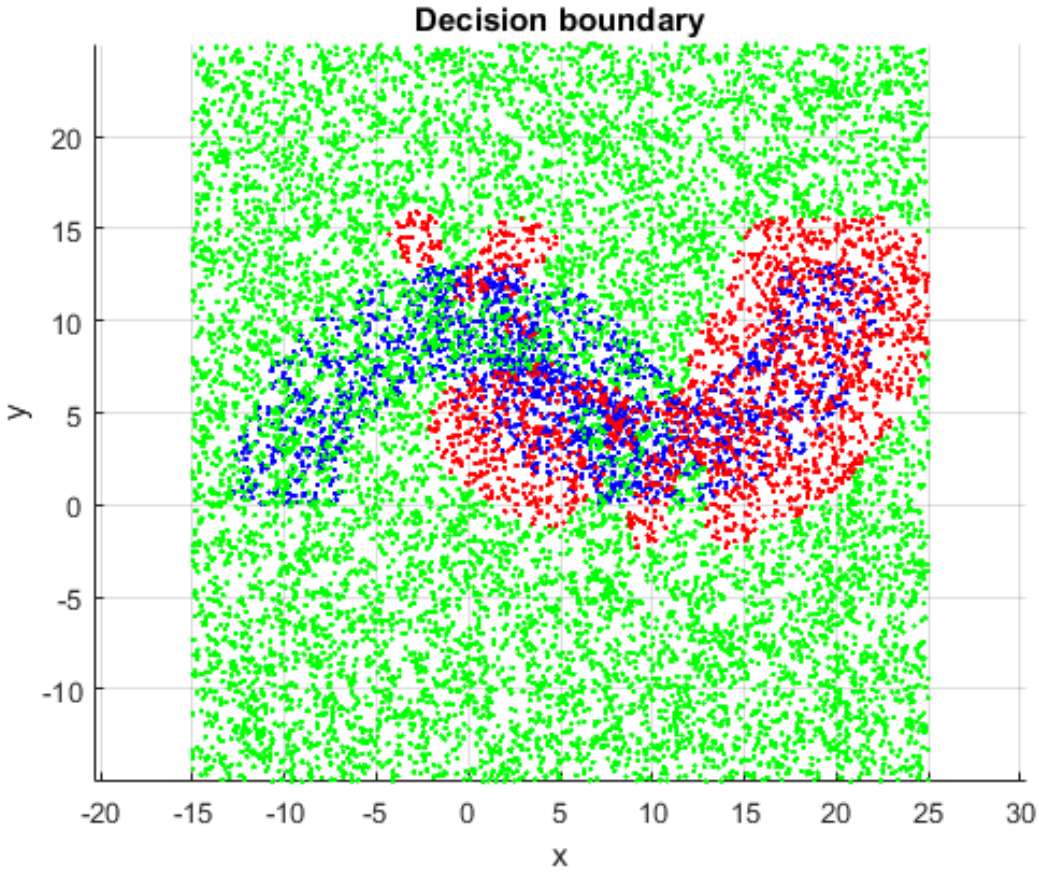


Figure 24: Linear kernel

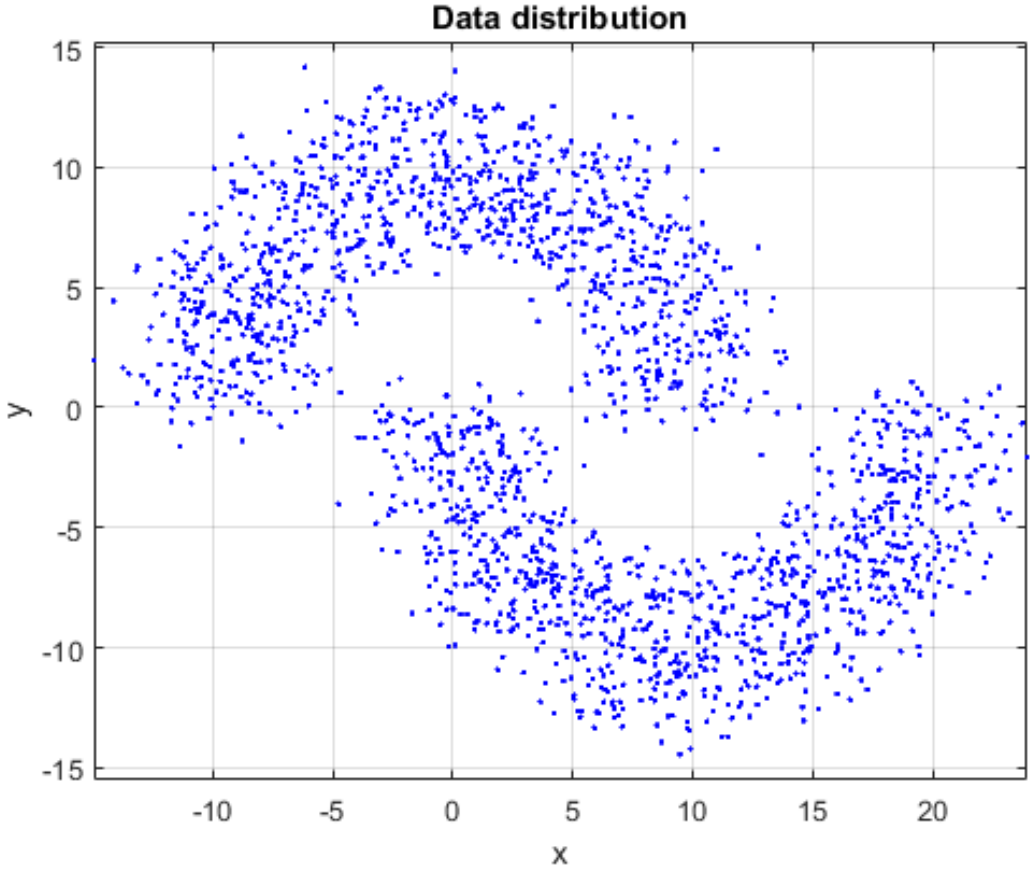


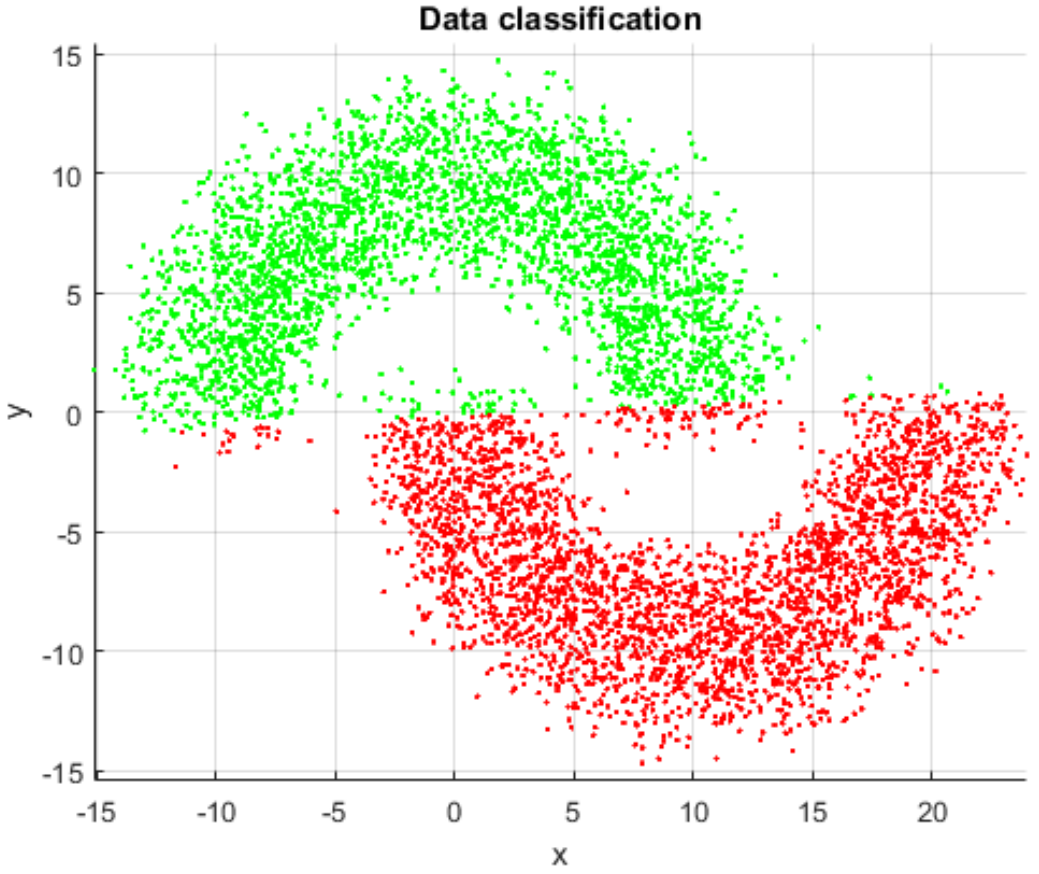
Gauss kernel.

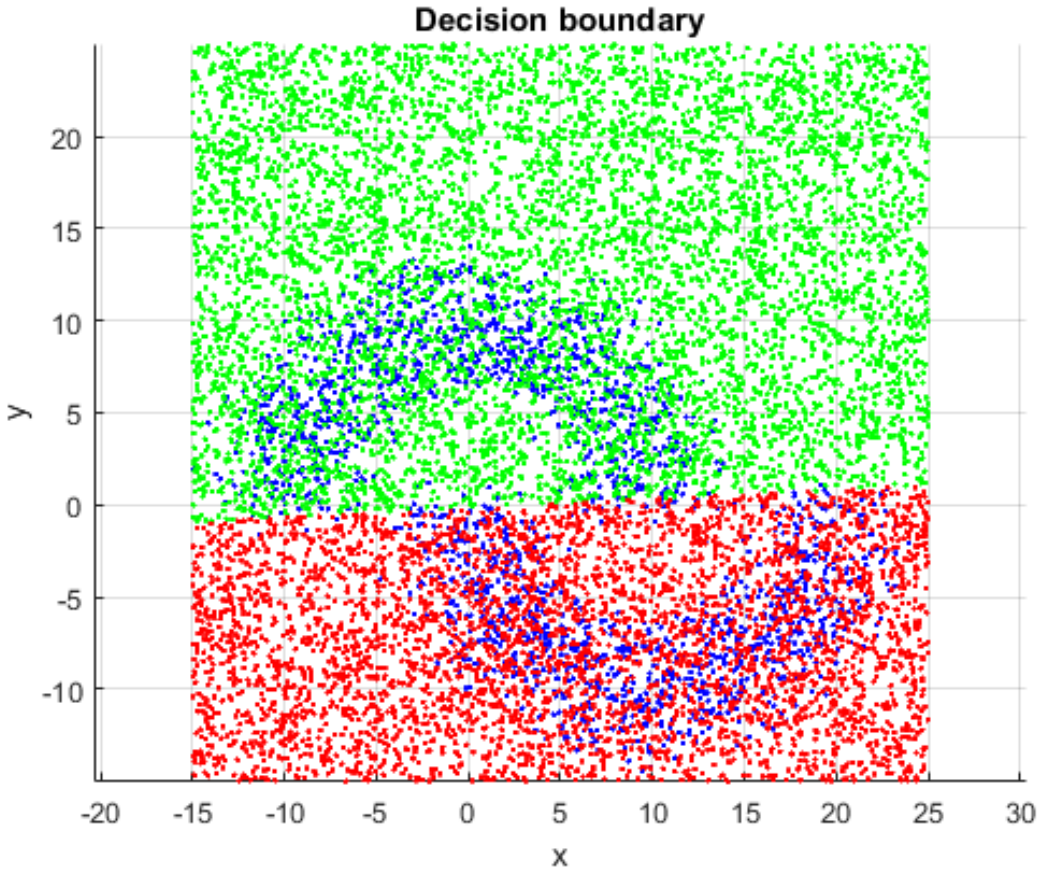


2.2.5 Case noise

Noise can spoil even the easiest example. With $d = 0$, linear kernel already has troubles




Data classification
15
10
5
0
-5
-10
-15
y
-15
-10
-5
0
5
10
15
20
x


Decision boundary
20
15
10
5
0
-5
-10
y
-20
-15
-10
-5
0
5
10
15
20
25
30
x

2.2.6 The hardest case

Overlapping moons and noise. RBF kernel SVM still shown good results.

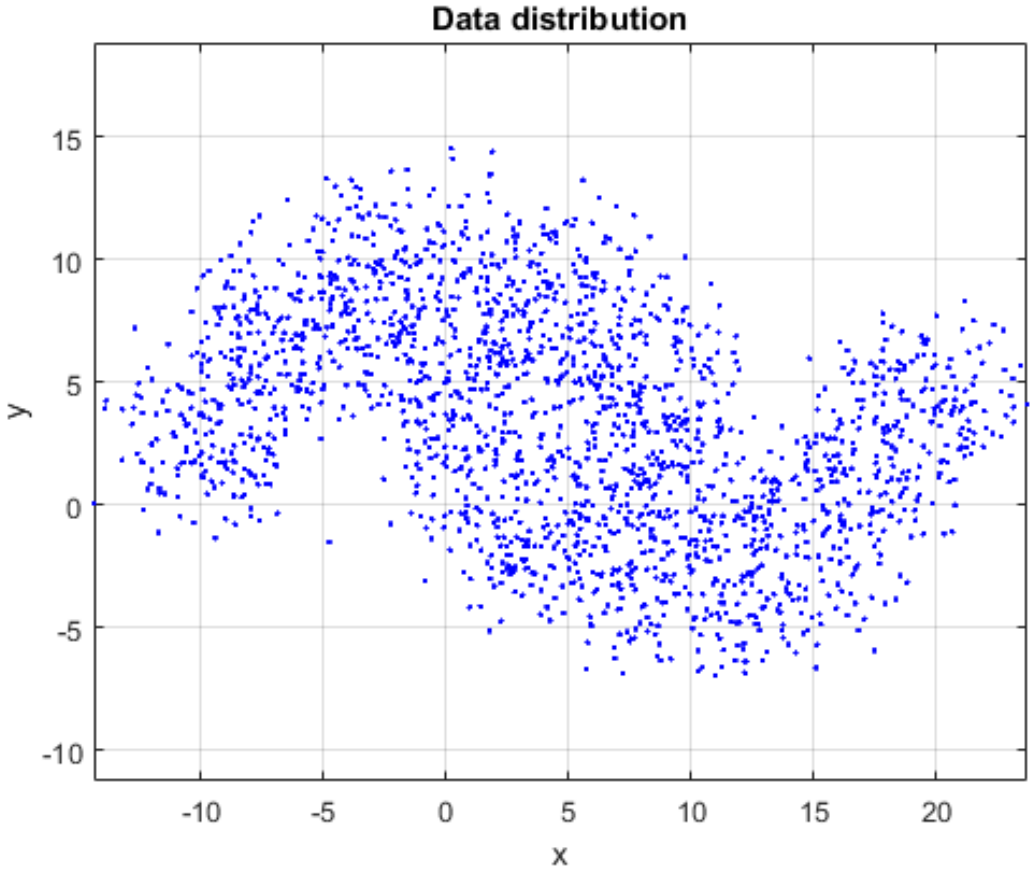


Figure 25: Data distribution

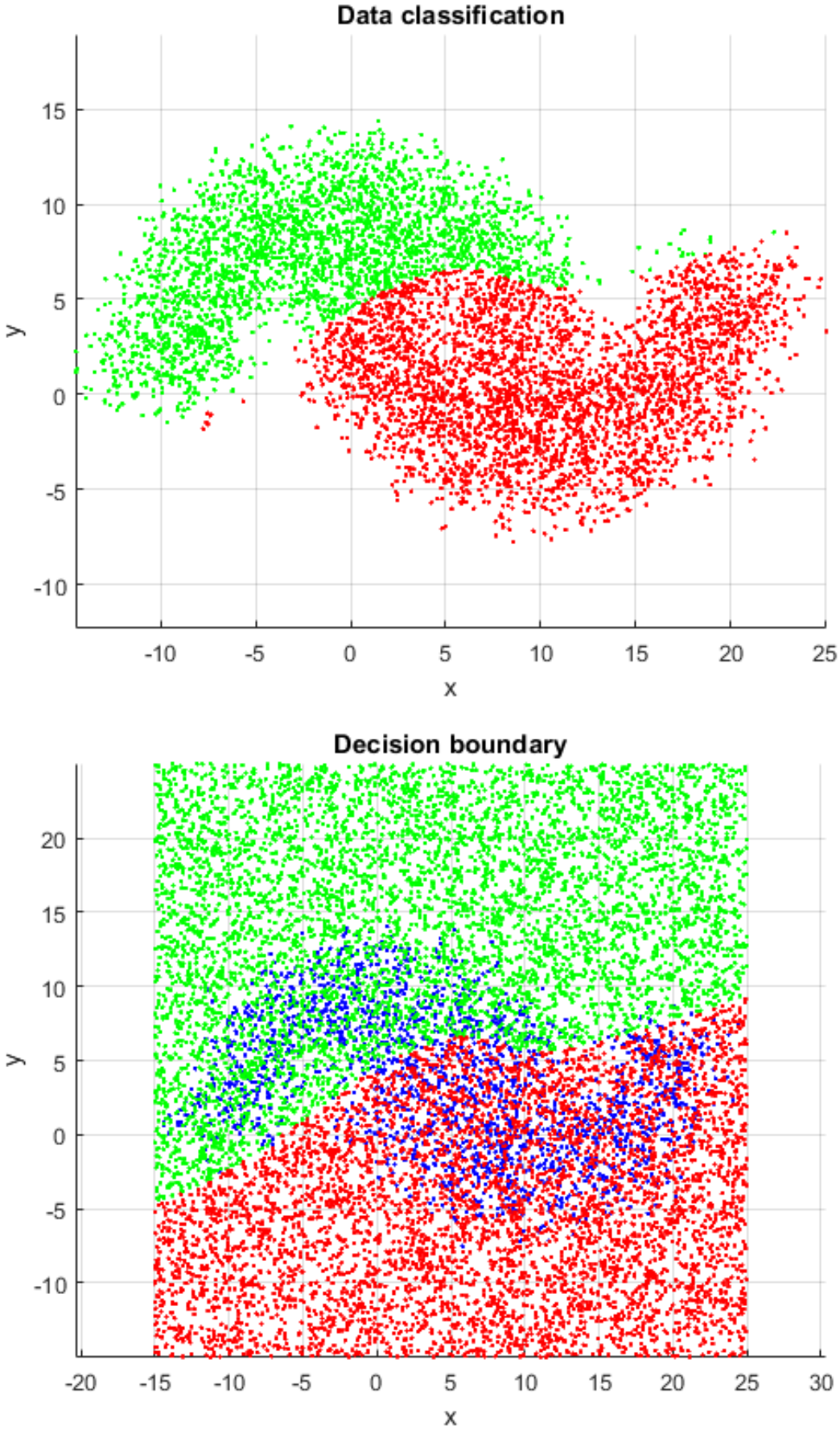


Figure 26: 3-rd order polynomial kernel

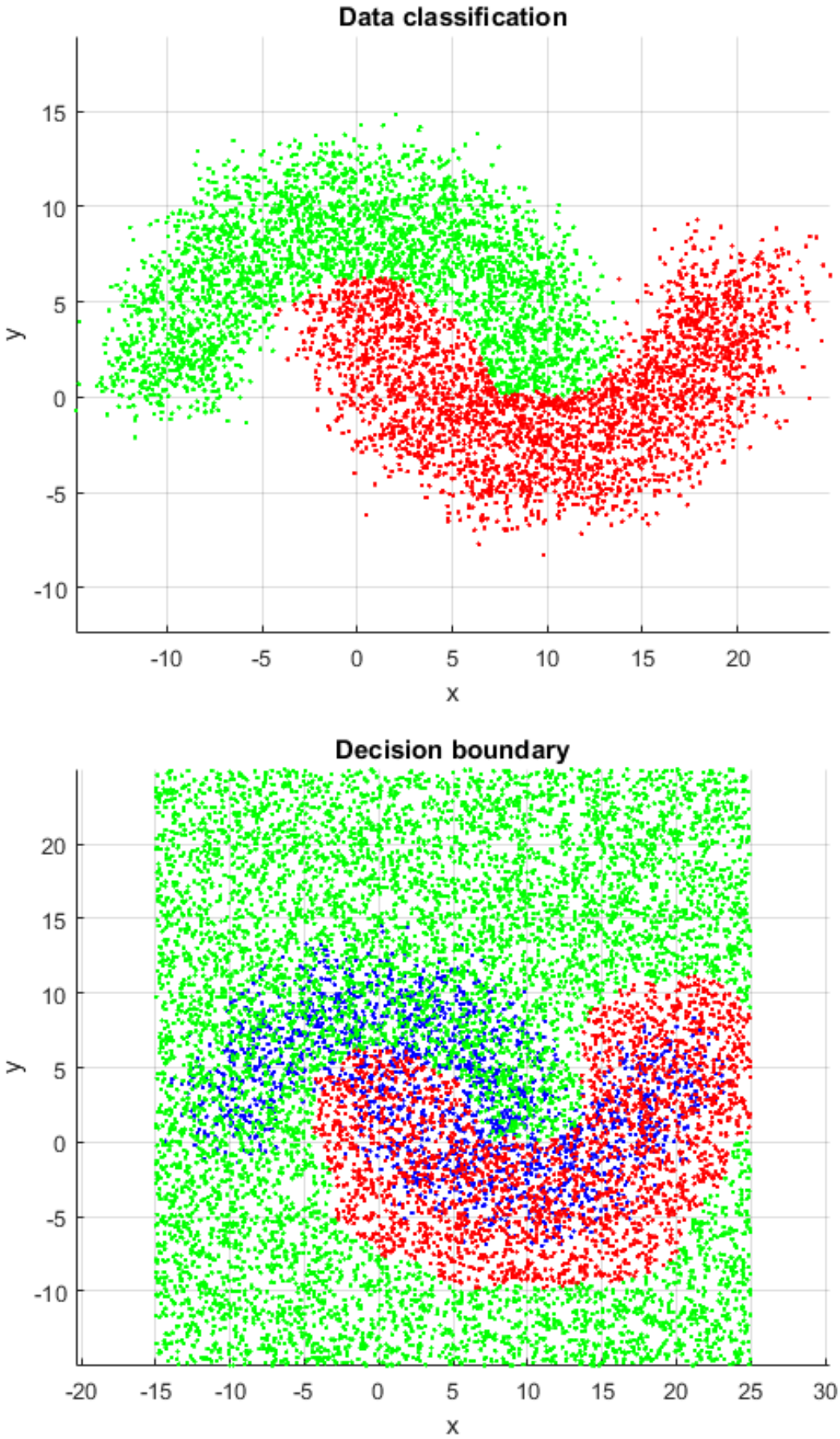


Figure 27: Gauss kernel

2.2.7 Results

- In case with two moons RBF kernel has the best classification abilities.
- It also shown good computational efficiency.
- Linear classification is really cheap but useless.
- Polynomial kernel are really expensive and usually worse that RBF.