BRSU

Neural Networks
Assignment 9

Bastian Lang

December 5, 2015

1 EXERCISE 2

1.1 TASK

The graphs below represent three different one-dimensional classification (dichotomization) tasks (along a sketched x-axis, dash means "no data point")(figure 1.1. What is the lowest-order polynomial decision function that can correctly classify the given data? Black dots denote class 1 with target function value y1 = +1 and white dots depict class 2 with targets y2 = -1. What are the decision boundaries?

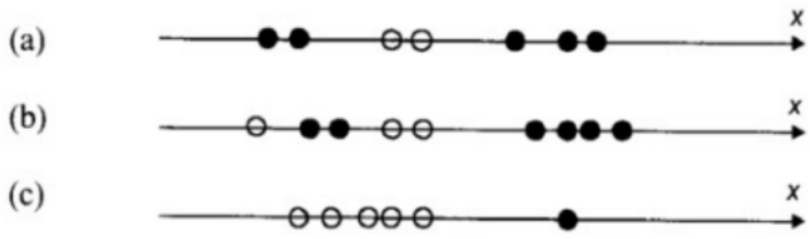

Figure 1.1:

If you wanted to classify the data sets (a), (b), (c) using SVM's with Gaussian basis functions, how many hidden layer neurons would you need for each problem?

1.2 SOLUTION

The order depends on the number of turning points. For one turning point a polynomial of order 2 is needed.
(a) The lowest order polynomial would be 2 (parabola).
(b) The lowest order polynomial would be 3.
(c) The lowest order polynomial would be 1 (line).

The number of hidden layer neurons equals the order of the polynomial needed -1.

2 EXERCISE 3

2.1 RESULTS

For the linear kernel the decision boundary obviously is a line, which works good if the separation is 0, but the worse the more the two figures are intermixed.

RBF works well for all figures but the one were the figures are overlapping.

Using polynomial kernel with default degree of 3 I would have expected a better fit, but even low separation results in some errors, the overleaping figures do look similar to the linear ker-

nel version.
Looking at the data increasing the degree should not help because degree 3 should be sufficient. But my laptop is not able to come up with a result within 10 minutes for higher degree. Maybe just repeating with degree 3 several times will yield better results.

2.2 Output

2.2.1 Sampled Pairs

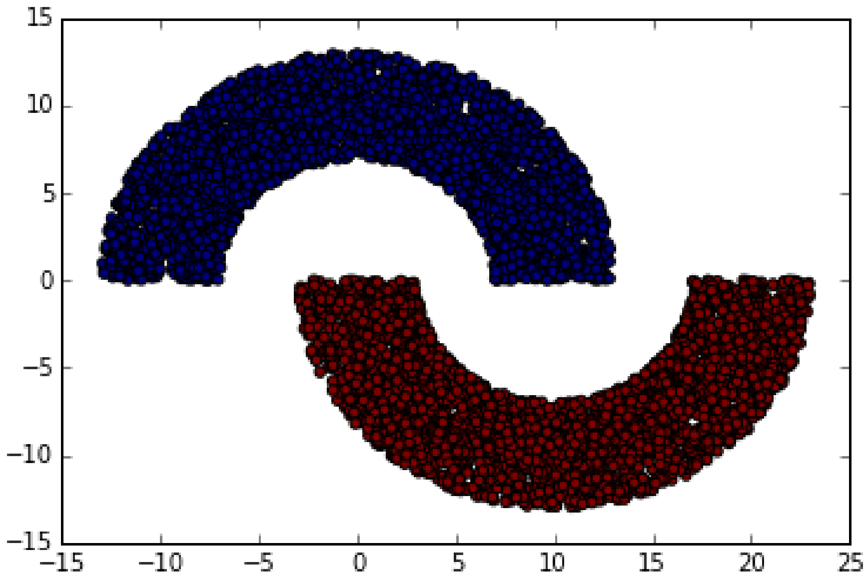

Figure 2.1: Test data with vertical separation = 0

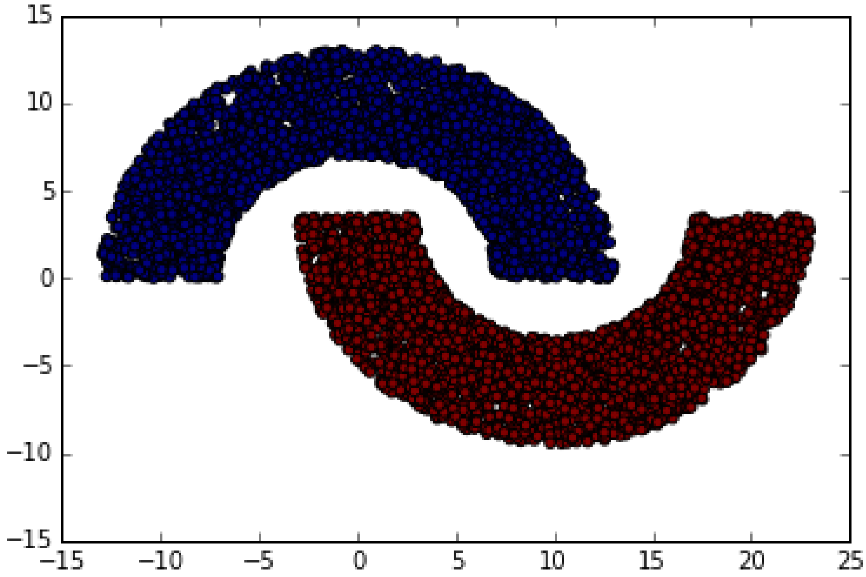

Figure 2.2: Test data with vertical separation = -1/2 * (radius - 1/2*width)

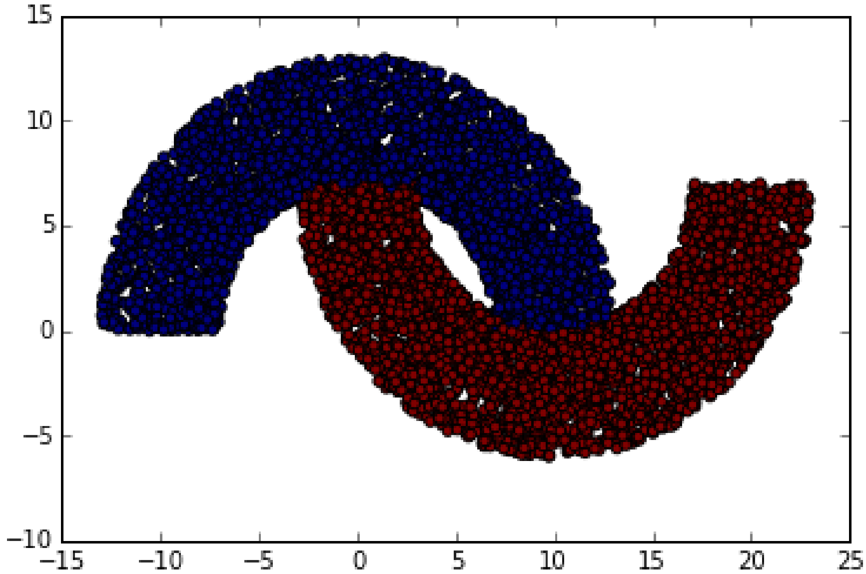

Figure 2.3: Test data with vertical separation = - radius of inner half circle

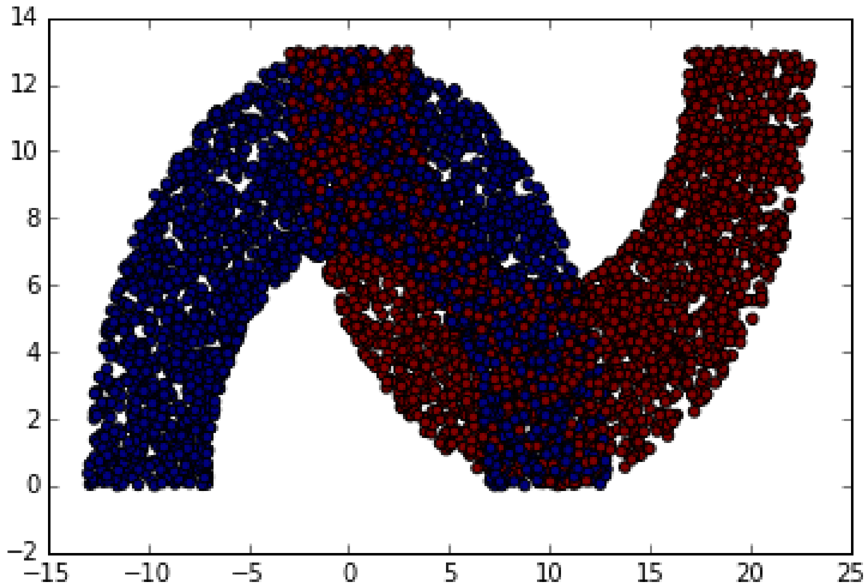

Figure 2.4: Test data with vertical separation = - radius of outer half circle

2.2.2 Linear Kernel

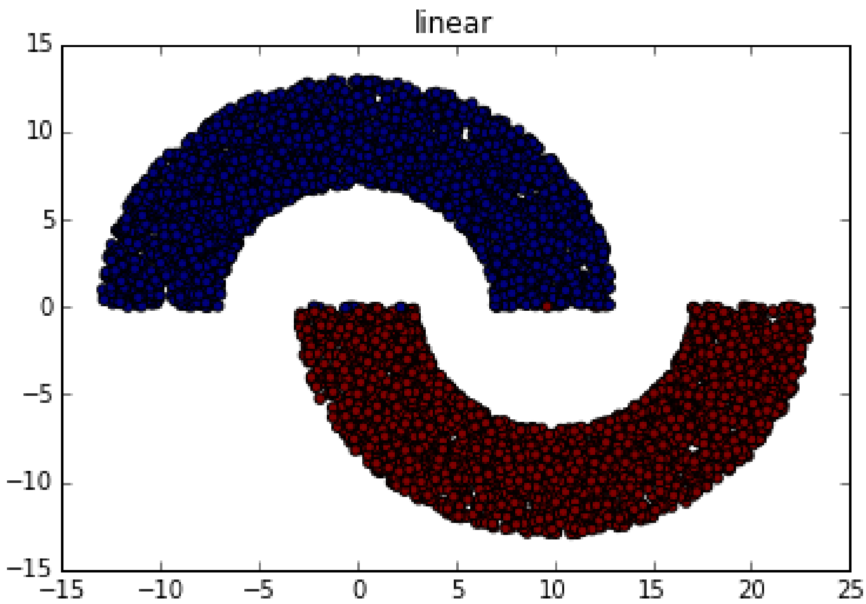

Figure 2.5: Classified test pairs with vertical separation = 0 using linear kernel

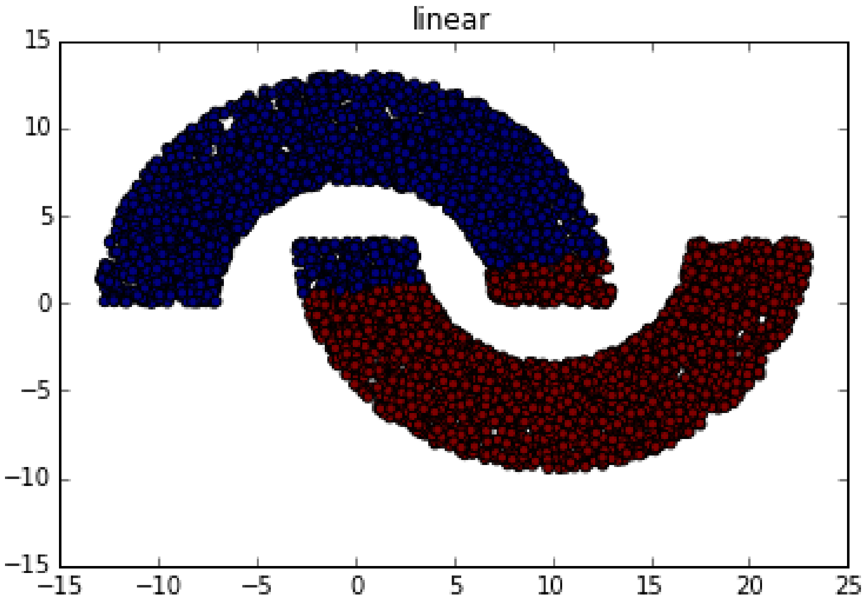

Figure 2.6: Classified test pairs with vertical separation = -1/2 * (radius - 1/2*width) using linear kernel

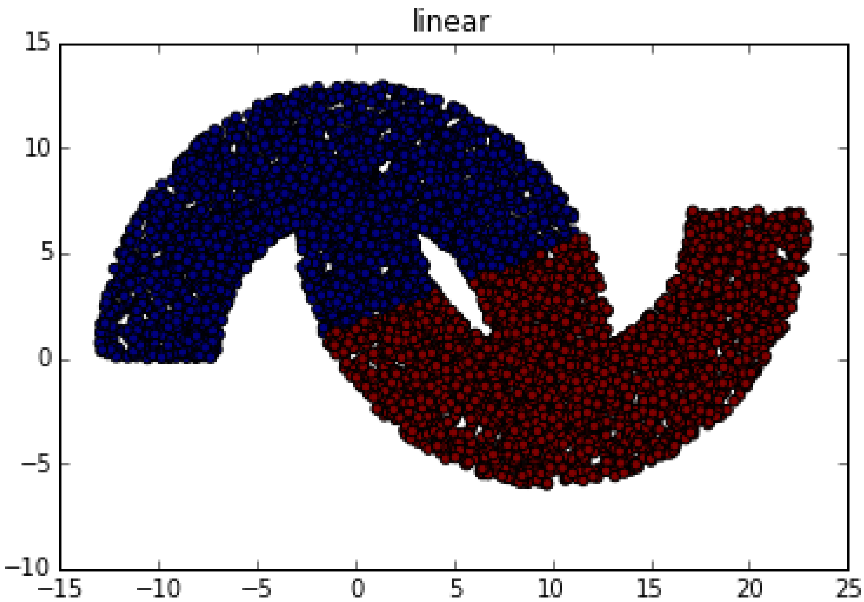

Figure 2.7: Classified test pairs with vertical separation = - radius of inner half circle using linear kernel

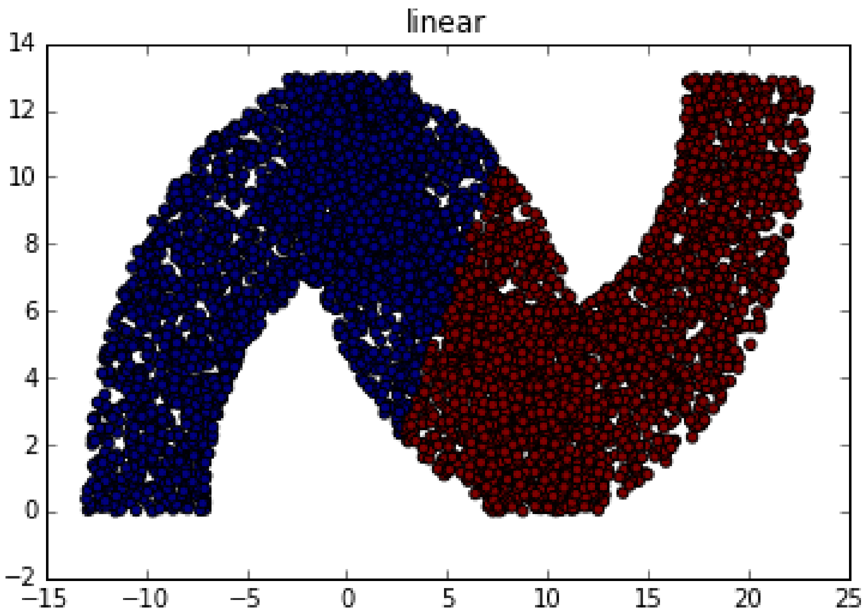

Figure 2.8: Classified test pairs with vertical separation = - radius of outer half circle using linear kernel

2.2.3 RBF

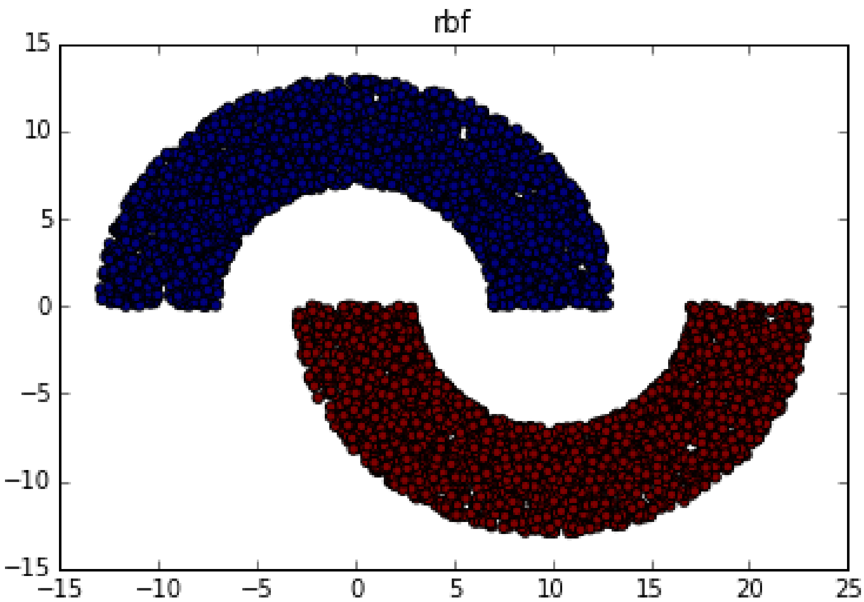

Figure 2.9: Classified test pairs with vertical separation = 0 using rbf kernel

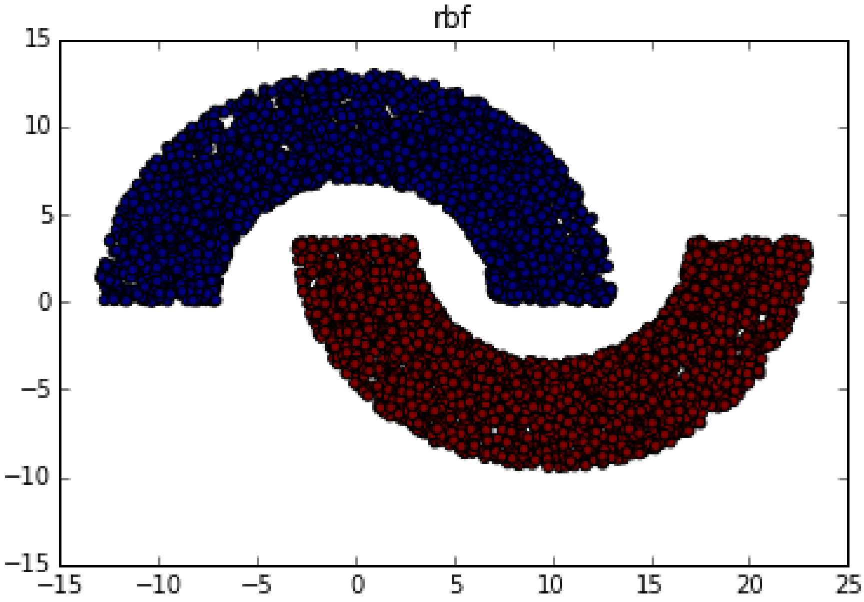

Figure 2.10: Classified test pairs with vertical separation = -1/2 * (radius - 1/2*width) using rbf kernel

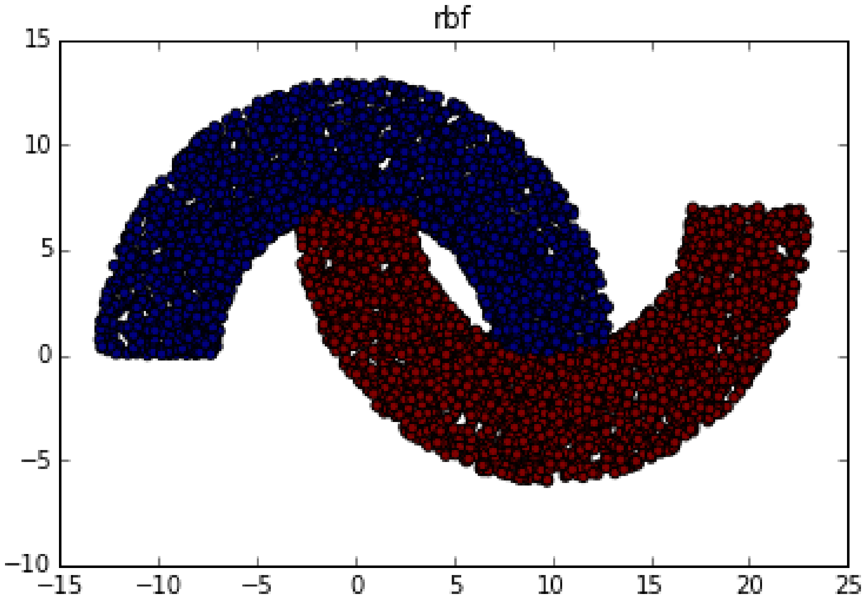

Figure 2.11: Classified test pairs with vertical separation = - radius of inner half circle using rbf kernel

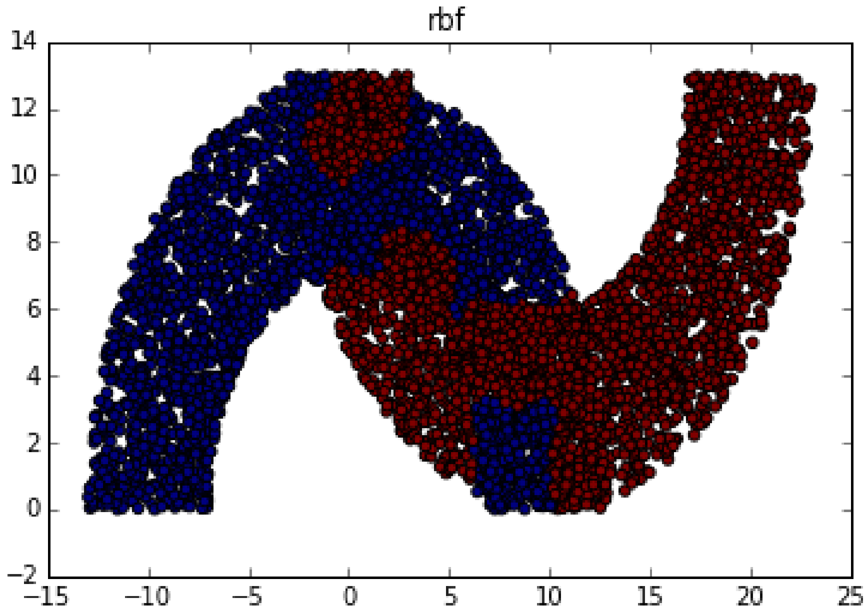

Figure 2.12: Classified test pairs with vertical separation = - radius of outer half circle using rbf kernel

2.2.4 Polynomial

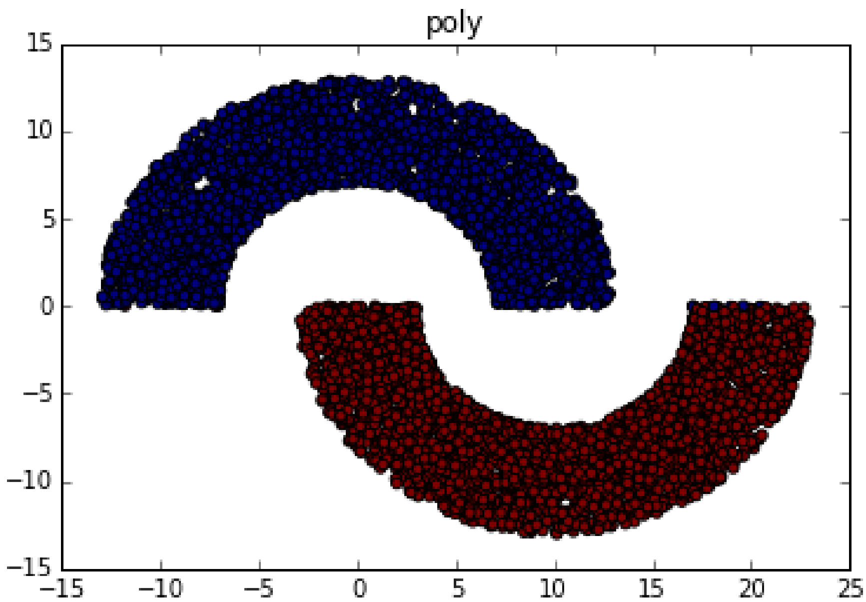

Figure 2.13: Classified test pairs with vertical separation = 0 using polynomial kernel

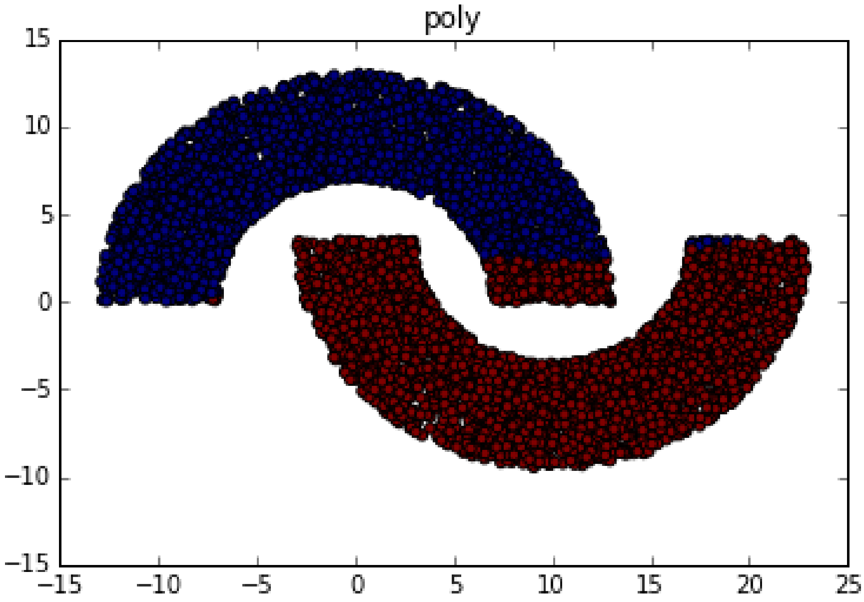

Figure 2.14: Classified test pairs with vertical separation = -1/2 * (radius - 1/2*width) using polynomial kernel

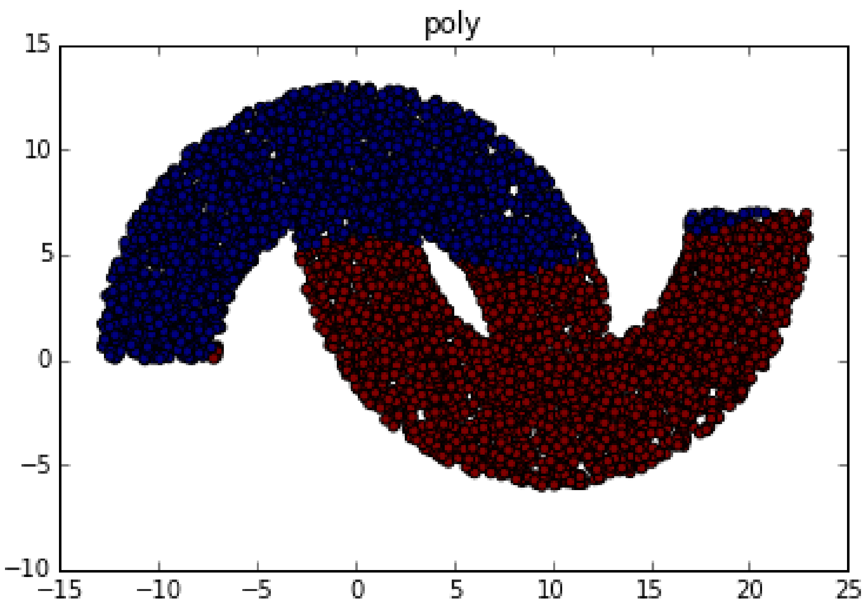

Figure 2.15: Classified test pairs with vertical separation = - radius of inner half circle using polynomial kernel

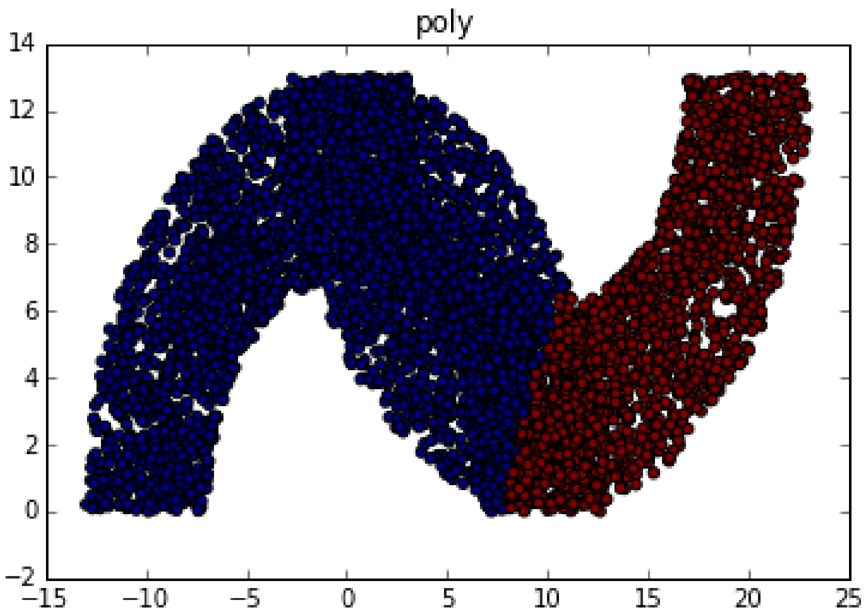

Figure 2.16: Classified test pairs with vertical separation = - radius of outer half circle using polynomial kernel

2.2.5 Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec  3 22:36:41 2015

@author: bastian
"""


import random
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from sklearn.svm import *

RADIUS = 10
WIDTH = 6
VERTICAL_SEPARATION = 0

# TODO: SVM is supervised -> assign labels according to part of upper or lower half moon

def sample_upper_halfmoon(radius, width):
```

```
    distance = random.random() * width + radius - 0.5 * width
    angle = random.random() * 180
    angle = np.radians(angle)
    x = np.cos(angle) * distance
    y = np.sin(angle) * distance
    return [x,y]

def sample_lower_halfmoon(radius, width, vertical_separation):
    distance = random.random() * width + radius - 0.5 * width
    angle = random.random() * 180 + 180
    angle = np.radians(angle)
    x = np.cos(angle) * distance + radius
    y = np.sin(angle) * distance - vertical_separation
    return [x,y]


def sample_pair(radius, width, vertical_separation):
    a = sample_upper_halfmoon(radius, width)
    b = sample_lower_halfmoon(radius, width, vertical_separation)
    return a,b

def plot_points_with_specified_separaration(vertical_separation):
    training_samples_x = []
    training_samples_y = []
    training_class = []
    for i in range(1000):
        a,b = sample_pair(RADIUS,WIDTH, vertical_separation)
        training_samples_x.append(a[0])
        training_samples_x.append(b[0])
        training_samples_y.append(a[1])
        training_samples_y.append(b[1])
        training_class.append(-1)
        training_class.append(1)

    test_samples_x = []
    test_samples_y = []
    test_class = []
    for i in range(3000):
        a,b = sample_pair(RADIUS,WIDTH, vertical_separation)
        test_samples_x.append(a[0])
        test_samples_x.append(b[0])
        test_samples_y.append(a[1])
        test_samples_y.append(b[1])
        test_class.append(-1)
```

```
            test_class.append(1)

        figure = plt.figure()
        ax = figure.add_subplot(111)
        ax.scatter(test_samples_x, test_samples_y, c=np.array(test_class).astype(np.float))
        return  training_samples_x, training_samples_y, training_class, test_samples_x, test_



    def fit_and_predict(samples, kernel_type):
        training = np.array([samples[0], samples[1]], dtype='float64').T
        clf = SVC(kernel=kernel_type).fit(training, np.array(samples[2], dtype='float64'))
        prediction = clf.predict(np.array([samples[3],samples[4]], dtype='float64').T)
        fig1 = plt.figure()
        plt.title(kernel_type)
        ax1 = fig1.add_subplot(111)
        ax1.scatter(samples[3],samples[4], c=prediction.astype(np.float))

    set1=plot_points_with_specified_separaration(0)
    set2=plot_points_with_specified_separaration((-0.5) * (RADIUS-0.5*WIDTH))
    set3=plot_points_with_specified_separaration((-1) * (RADIUS-0.5*WIDTH))
    set4=plot_points_with_specified_separaration((-1) * (RADIUS+0.5*WIDTH))

    fit_and_predict(set1, 'linear')
    fit_and_predict(set2, 'linear')
    fit_and_predict(set3, 'linear')
    fit_and_predict(set4, 'linear')

    fit_and_predict(set1, 'rbf')
    fit_and_predict(set2, 'rbf')
    fit_and_predict(set3, 'rbf')
    fit_and_predict(set4, 'rbf')

    fit_and_predict(set1, 'poly')
    fit_and_predict(set2, 'poly')
    fit_and_predict(set3, 'poly')
    fit_and_predict(set4, 'poly')

    fit_and_predict(set1, 'sigmoid')
    fit_and_predict(set2, 'sigmoid')
    fit_and_predict(set3, 'sigmoid')
    fit_and_predict(set4, 'sigmoid')
```