Assignment_07

November 25, 2017

1 Assignment 07

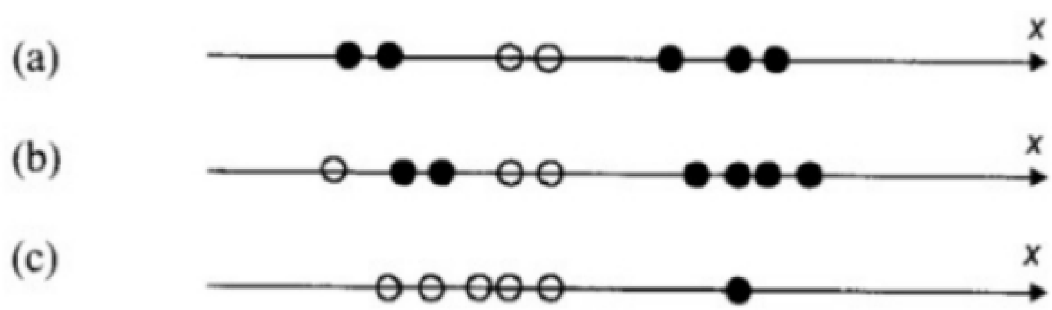
1.0.1 RaviKiran Bhat

1.0.2 Rubanraj RaviChandran

1.0.3 Ramesh Kumar

2 Exercise 2

```
In [19]: import numpy as np
         import matplotlib.pyplot as plt
         import random
         from IPython.display import Image
         from sklearn import svm
         %matplotlib inline

In [66]: Image(filename='fig1.png')

Out[66]:
```




```
In [41]: class Classifier:

             def svm_classifier(self, k,  x, y, d):

                 clf = svm.SVC(kernel = k, coef0=1, probability=True, degree=d, gamma=2)
```

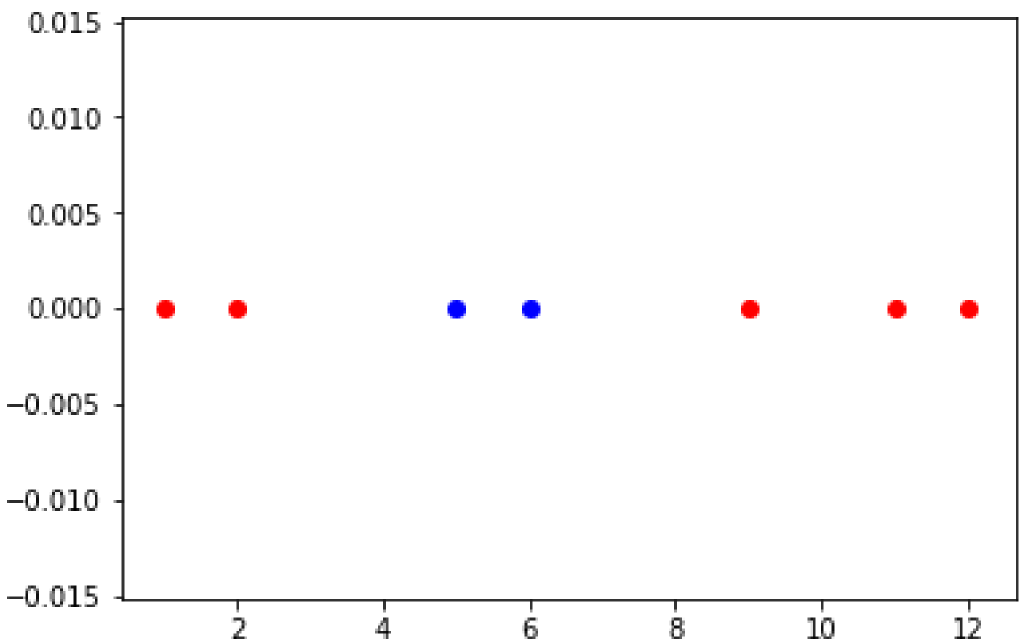
```python
        clf.fit(x,y)

        return clf

    #make mesh grid
    def make_meshgrid(self, xa, h=0.1):


        """Create a mesh of points to plot in
        Parameters
        ----------
        x: data to base x-axis meshgrid on
        y: data to base y-axis meshgrid on
        h: stepsize for meshgrid, optional

        Returns
        -------
        xx, yy : ndarray
        """

        x_min, x_max = xa[:,0].min() - 1, xa[:,0].max() + 1
        y_min, y_max = xa[:,1].min() - 1, xa[:,1].max() + 1
        h = 0.2
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))
        return xx, yy

    def plot_contours(self, ax, clf, xx, yy, **params):

        """Plot the decision boundaries for a classifier.

        Parameters
        ----------
        ax: matplotlib axes object
        clf: a classifier
        xx: meshgrid ndarray
        yy: meshgrid ndarray
        params: dictionary of params to pass to contourf, optional
        """

        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        out = ax.contourf(xx, yy, Z, **params)
        return out

    def plot(self, data,y):

        for i in range(len(data)):
```

```
            if y[i] == 1:
                plt.scatter(data[i][0],data[i][1],color='r')
            else:
                plt.scatter(data[i][0],data[i][1],color='b')

    def classifier_plot(self, models, titles, sub, x, y):

        xx, yy =  classifer.make_meshgrid(x)

        for clf, title, ax in zip(models, titles, sub.flatten()):
            ax.grid(True)
            classifer.plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
            ax.scatter(x[:,0], x[:,1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
            ax.set_title(title)
```

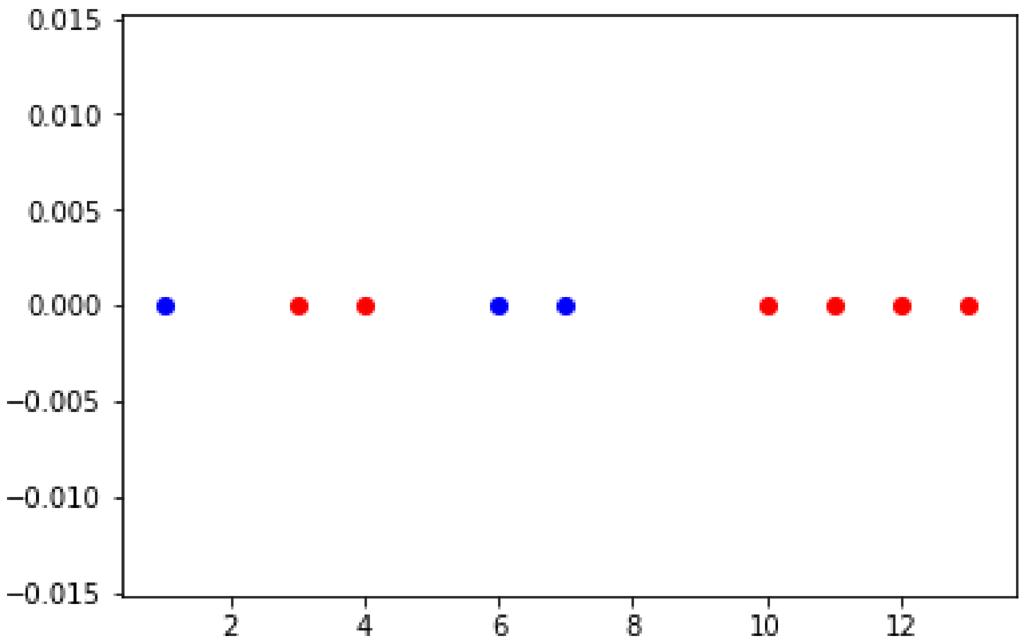
What is the lowest-order polynomial decision function that can correctly classify the given data? Black dots denote class 1 with target function value y1 = +1 and white dots depict class 2 with targets y2 = -1. What are the decision boundaries?

```
In [42]: classifer = Classifier()
```

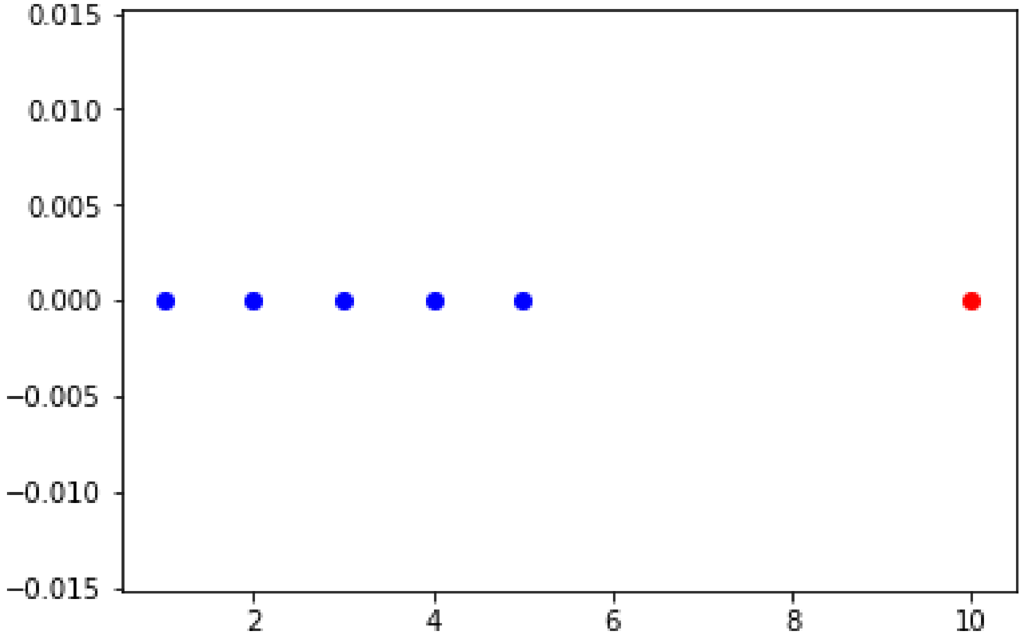
```
In [43]: #a
         xa = np.array([[1,0],[2,0],[5,0],[6,0],[9,0],[11,0],[12,0]])
         ya = np.array([1,1,-1,-1,1,1,1])
         classifer.plot(xa,ya)
```



In [44]:
```
#b
xb = np.array([[1,0],[3,0],[4,0],[6,0],[7,0],[10,0],[11,0],[12,0],[13,0]])
yb = np.array([-1,1,1,-1,-1,1,1,1,1])
classifer.plot(xb,yb)
```

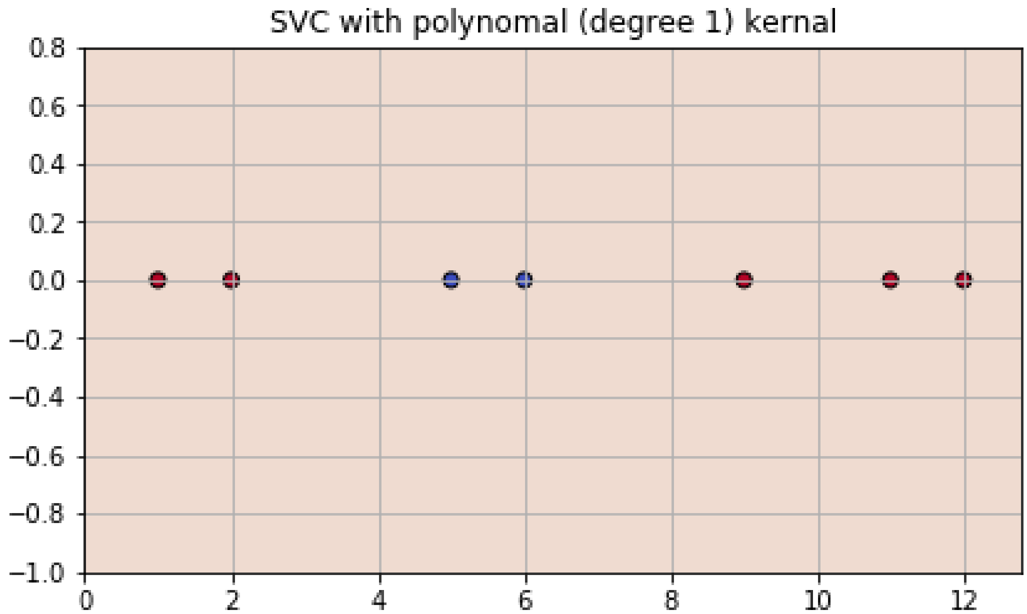


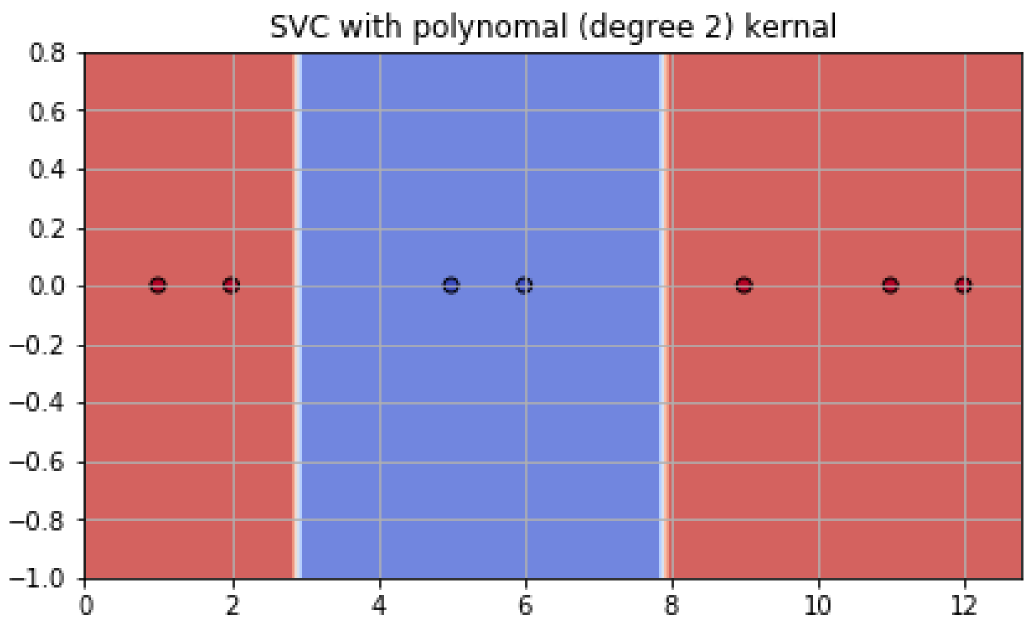
In [45]:
```
#c
xc = np.array([[1,0],[2,0],[3,0],[4,0],[5,0],[10,0]])
yc = np.array([-1,-1,-1,-1,-1,1])
classifer.plot(xc,yc)
```

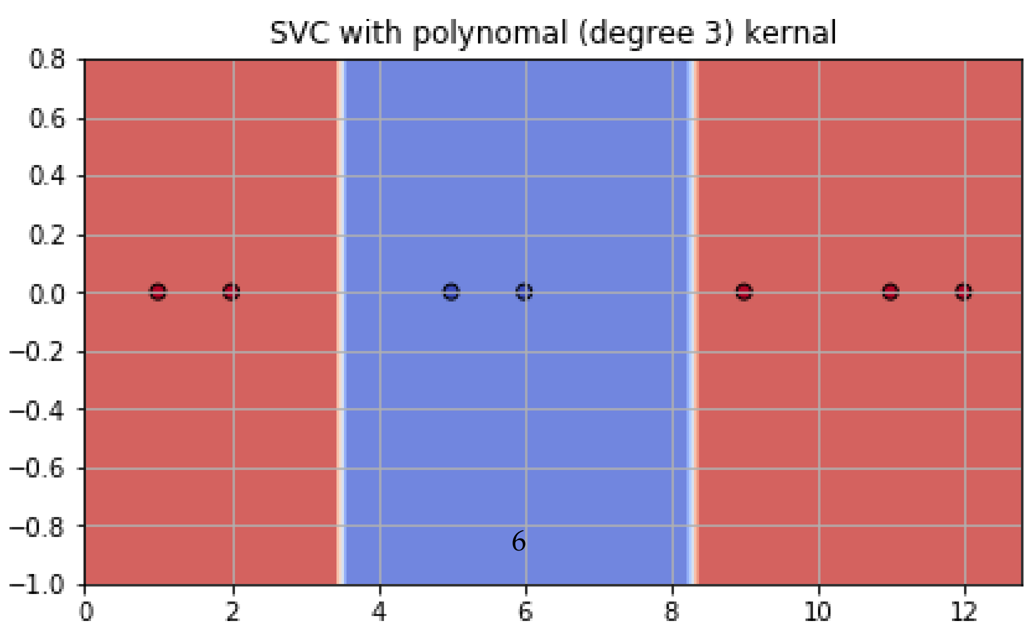


2.1 Polynomial Classifier using xa, ya

In [47]:
```
models = (classifer.svm_classifier('poly', xa, ya, 1),
          classifer.svm_classifier('poly', xa, ya, 2),
          classifer.svm_classifier('poly', xa, ya, 3))

# title for the plots
titles = ('SVC with polynomal (degree 1) kernal',
          'SVC with polynomal (degree 2) kernal',
          'SVC with polynomal (degree 3) kernal',)

# Set-up 2x1 grid for plotting.
fig, sub = plt.subplots(3, 1)
# plt.subplots_adjust(wspace=2, hspace=2)
plt.subplots_adjust(bottom=1.0, right=1.0, top=3.5)
classifer.classifier_plot(models, titles, sub, xa, ya)
plt.show()
```

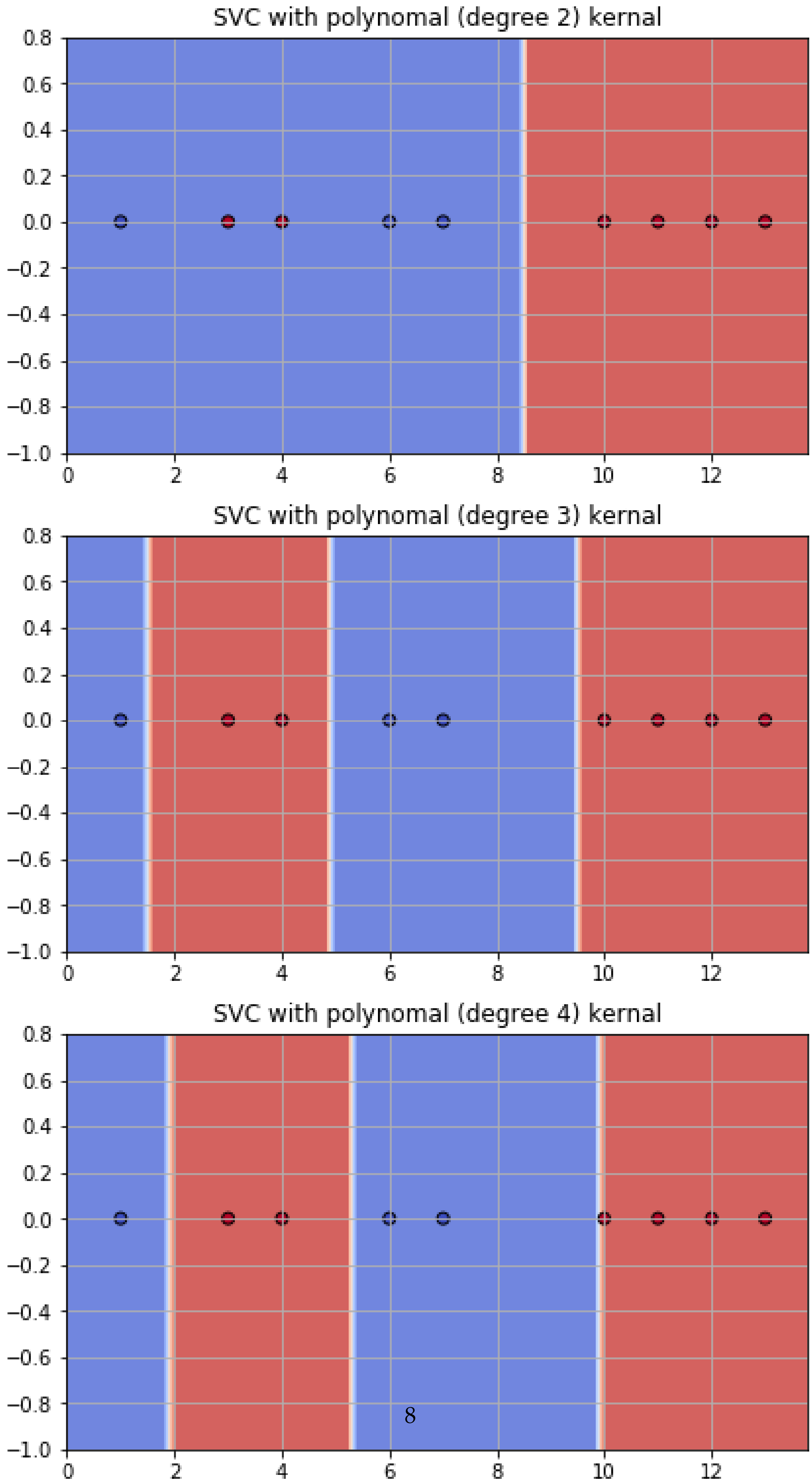

SVC with polynomal (degree 1) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10
12


SVC with polynomal (degree 2) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10
12


SVC with polynomal (degree 3) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10
12

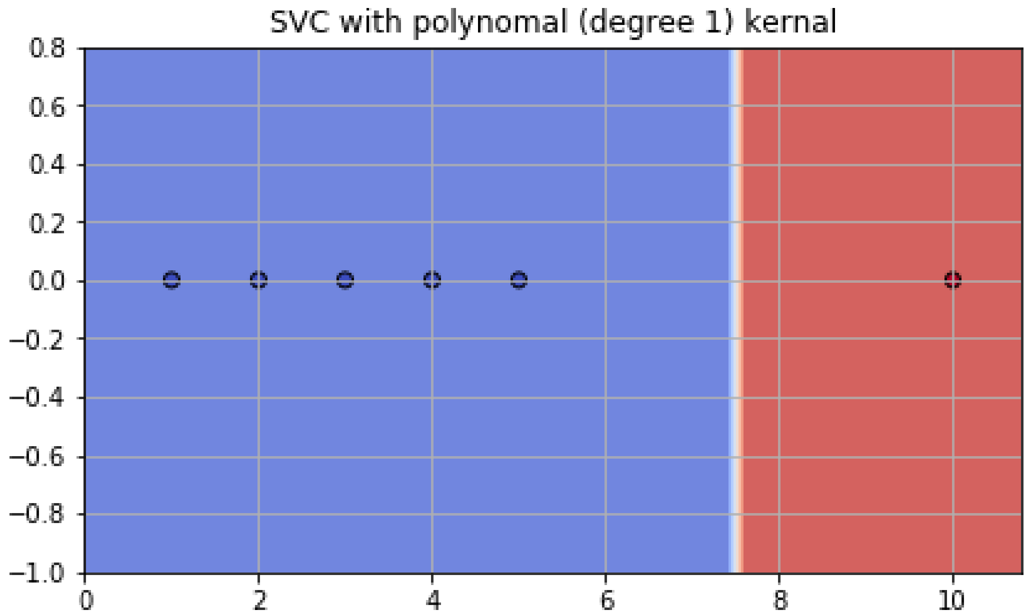
2.2 Polynomial Classifier using xb, yb

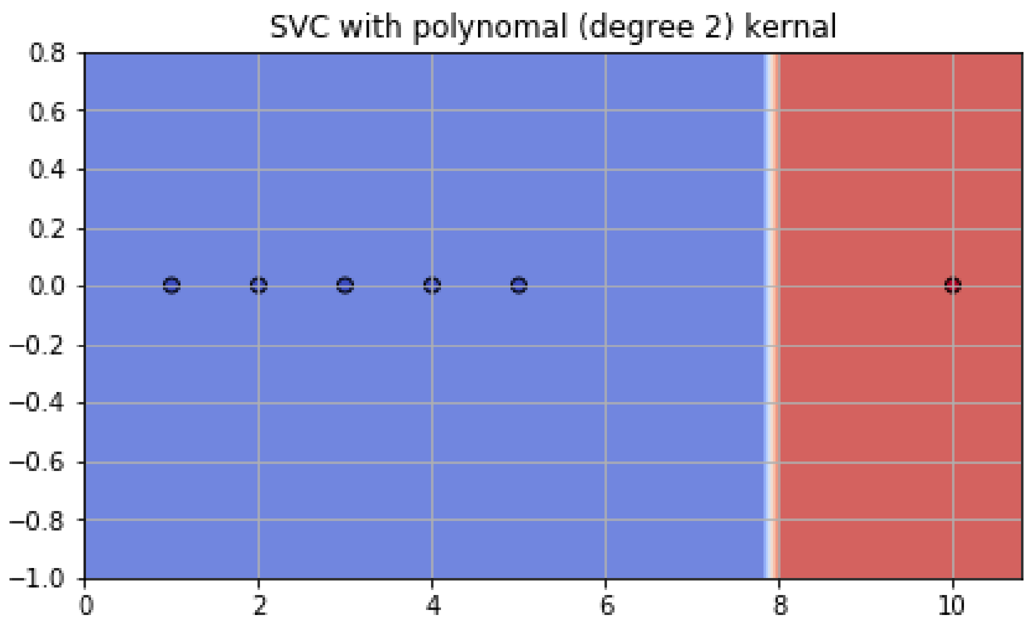
```
In [53]: models = (classifer.svm_classifier('poly', xb, yb, 2),
                   classifer.svm_classifier('poly', xb, yb, 3),
                   classifer.svm_classifier('poly', xb, yb, 4))

         # title for the plots
         titles = ('SVC with polynomal (degree 2) kernal',
                   'SVC with polynomal (degree 3) kernal',
                   'SVC with polynomal (degree 4) kernal',)

         # Set-up 2x1 grid for plotting.
         fig, sub = plt.subplots(3, 1)
         # plt.subplots_adjust(wspace=2, hspace=2)
         plt.subplots_adjust(bottom=1.0, right=1.0, top=3.5)
         classifer.classifier_plot(models, titles, sub, xb, yb)
         plt.show()
```

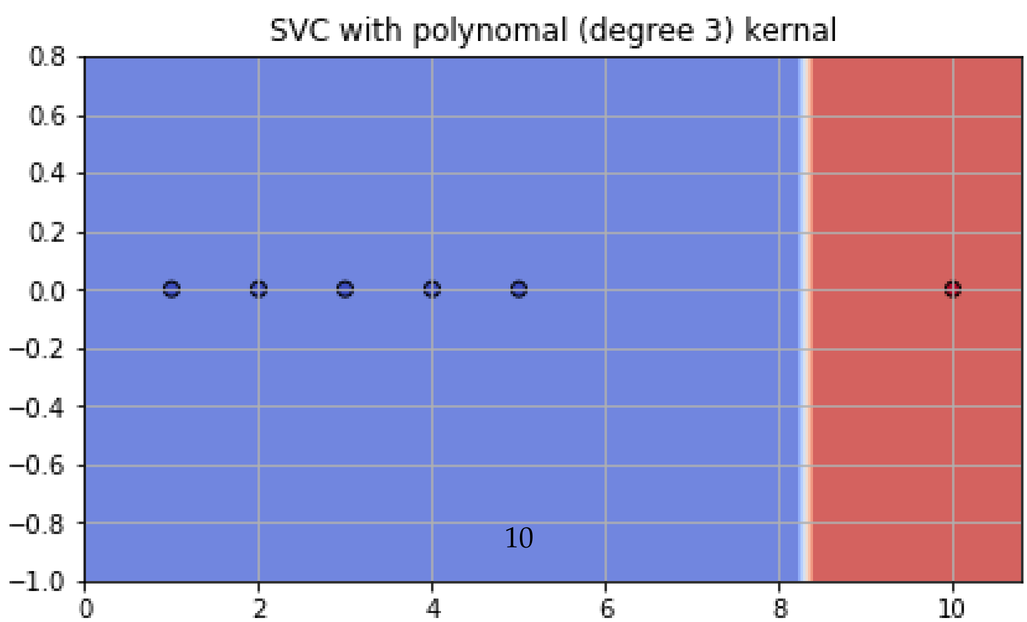

SVC with polynomal (degree 2) kernal
SVC with polynomal (degree 3) kernal
SVC with polynomal (degree 4) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10
12

2.3 Polynomial Classifier with xc, yc

```
In [54]: models = (classifer.svm_classifier('poly', xc, yc, 1),
                   classifer.svm_classifier('poly', xc, yc, 2),
                   classifer.svm_classifier('poly', xc, yc, 3))

         # title for the plots
         titles = ('SVC with polynomal (degree 1) kernal',
                   'SVC with polynomal (degree 2) kernal',
                   'SVC with polynomal (degree 3) kernal',)

         # Set-up 2x1 grid for plotting.
         fig, sub = plt.subplots(3, 1)
         # plt.subplots_adjust(wspace=2, hspace=2)
         plt.subplots_adjust(bottom=1.0, right=1.0, top=3.5)
         classifer.classifier_plot(models, titles, sub , xc,yc)
         plt.show()
```


SVC with polynomal (degree 1) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10


SVC with polynomal (degree 2) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10

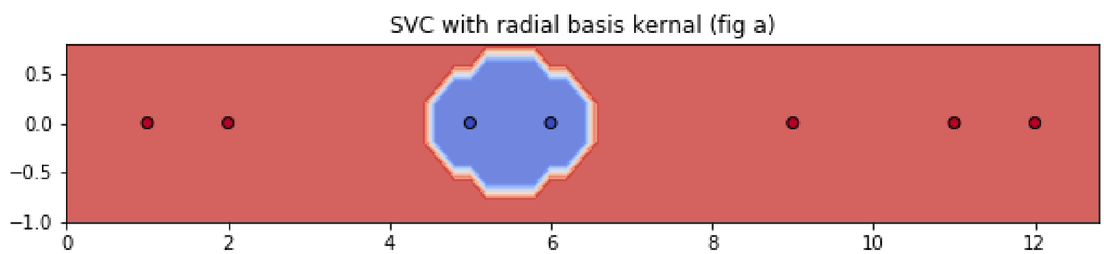

SVC with polynomal (degree 3) kernal
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10

2.3.1 Conclusion

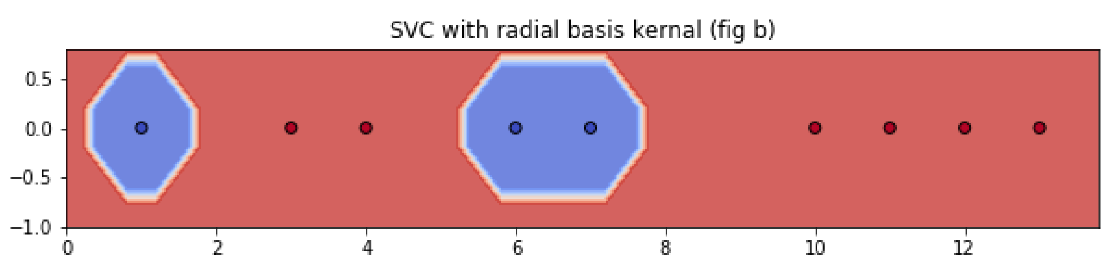
Above examples show that, for diagram a; when we use polynomial degree 2 and 3, it draws the decision boundaries. Therefore, lowest order of polynomial function is 2, that can separate two classes linearly. While for diagram b and c; lowest order of polynomial function is 2 and 1 respectively

2.4 Gaussian classifier

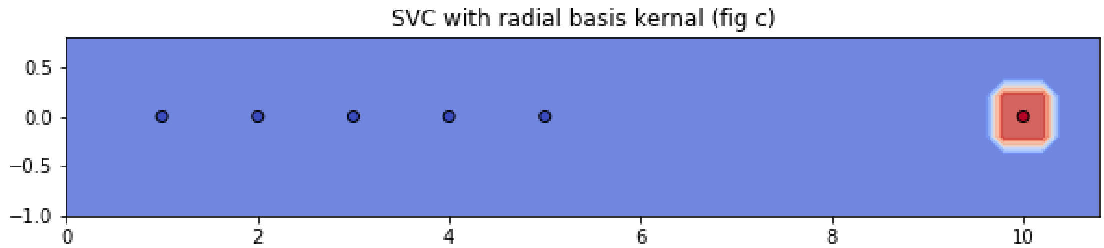
```
In [56]: def radial_basis_plot(title,clf,x,y):
             xx, yy =  classifer.make_meshgrid(x)
             fig = plt.figure(figsize=(10,6))
             ax = fig.add_subplot(3,1,1)
             plt.title(title)
             classifer.plot_contours(ax,clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
             ax.scatter(x[:,0], x[:,1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
             plt.show()

         clf1 = classifer.svm_classifier('rbf', xa, ya, 2)
         clf2 = classifer.svm_classifier('rbf', xb, yb, 2)
         clf3 = classifer.svm_classifier('rbf', xc, yc, 2)

         title = 'SVC with radial basis kernal (fig a)'
         radial_basis_plot(title,clf1,xa, ya,)
```




```
In [58]: title = 'SVC with radial basis kernal (fig b)'
         radial_basis_plot(title,clf2,xb, yb)
```



In [59]:
```
title = 'SVC with radial basis kernal (fig c)'
radial_basis_plot(title,clf3,xc, yc)
```



3 Exercise 3

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 moon's 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 the SVM-labrary (change parameters). Comment on your findings.

In [60]:
```
Image(filename='fig2.png')
```

Out[60]:

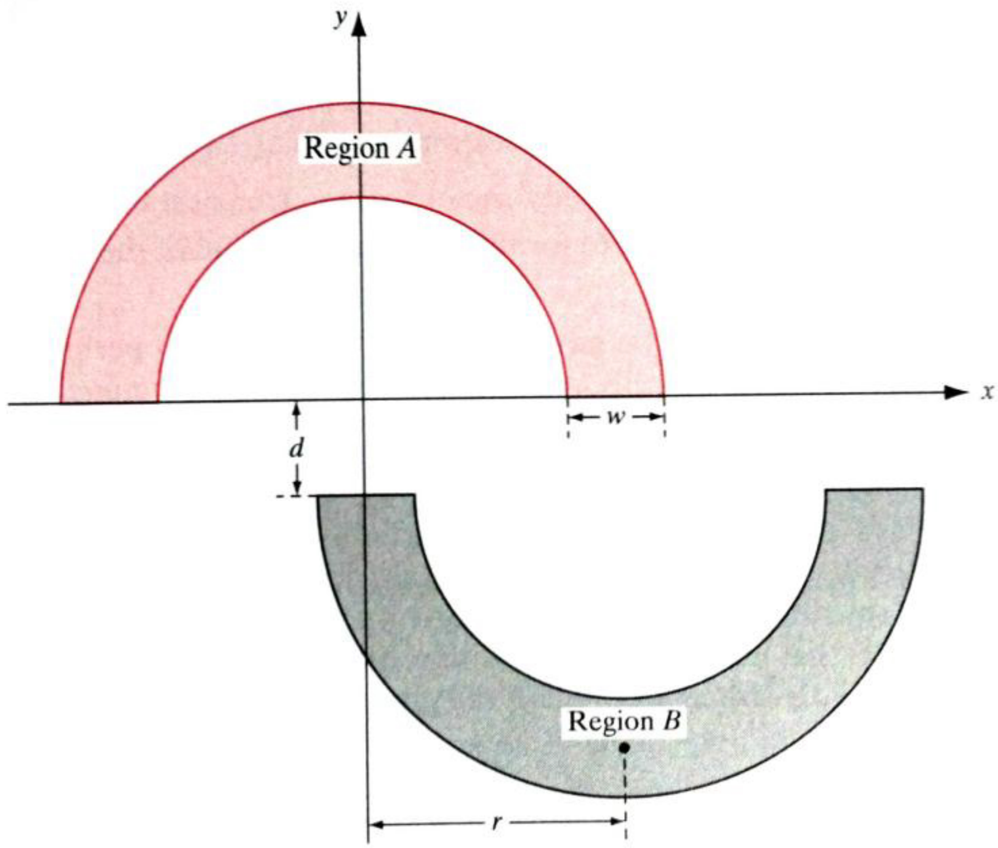


FIGURE 1.8 The double-moon classification problem.

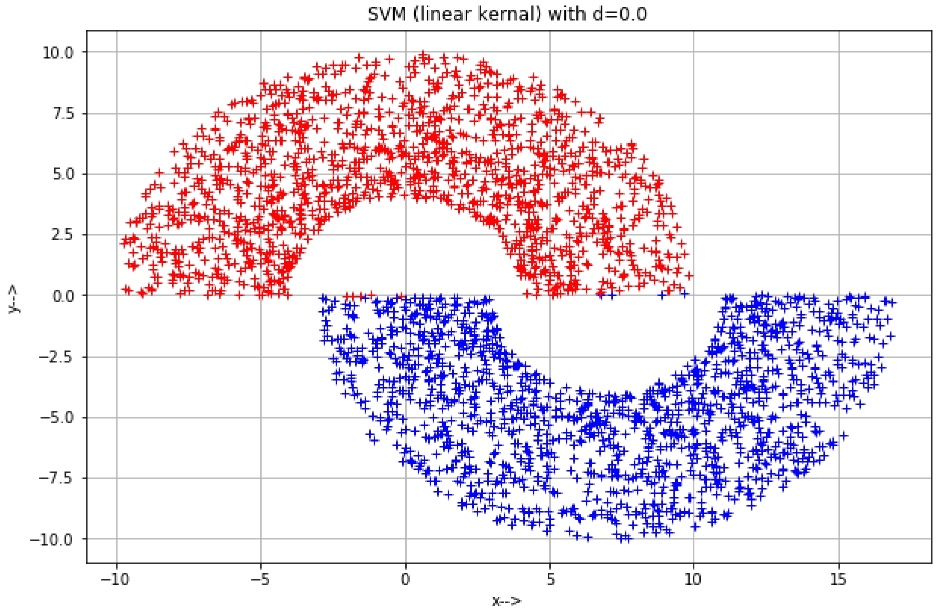
```
In [61]: class StateVectorMachine:
             def __init__(self,_radius,_width,_distance,_num_of_training_set,
                          _num_of_testing_set):
                 self.radius = _radius
                 self.width = _width
                 self.distance = _distance
                 self.num_of_training_set = _num_of_training_set
                 self.num_of_testing_set = _num_of_testing_set

             def generate_sample(self,_class):
                 random_theta = np.pi *  random.random()
                 random_r = (self.width*random.random())+(self.radius-self.width)
                 #Region one
                 if _class is 1:
                     x = random_r*np.cos(random_theta)
                     y = random_r*np.sin(random_theta)
                     return [x,y,1]
                 else:
```

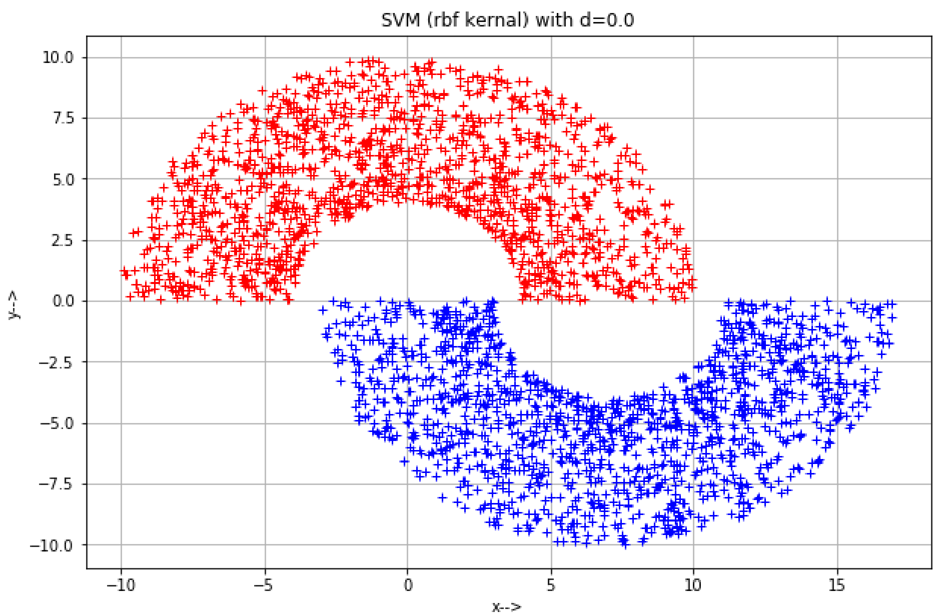
```
        #Region two
        random_theta += np.pi
        x = random_r*np.cos(random_theta)+(self.radius-(self.width/2.0))
        y = random_r*np.sin(random_theta)-self.distance
        return [x,y,2]

def get_samples(self,_flag):
    samples = np.empty((0,3))

    if _flag is "train":
        _no_of_samples = self.num_of_training_set
    else:
        _no_of_samples = self.num_of_testing_set

    """
    - generating number of samples
    - half samples belongs to region A and
      remaining half samples belongs to region B
    """
    for i in range(_no_of_samples):
        sample = self.generate_sample(1 if (i<_no_of_samples/2) else 2)
        samples = np.vstack([samples,sample])

    #returning samples and desired output
    return samples[:,0:2],samples[:,2:3]

def plot(self,points,output,title):
    plt.grid(True)
    plt.title(title)
    plt.xlabel("x-->")
    plt.ylabel("y-->")
    for index,point in enumerate(points):
        if (output[index] == 1.0):
            plt.plot(point[0],point[1],'r+',label='region a')
        else:
            plt.plot(point[0],point[1],'b+',label='region b')

def train(self,training_input,desired_output,_kernal):
    #trainig using support vector classifier
    classifier = svm.SVC(kernel=_kernal)
    classifier.fit(training_input, desired_output)
    return classifier
```

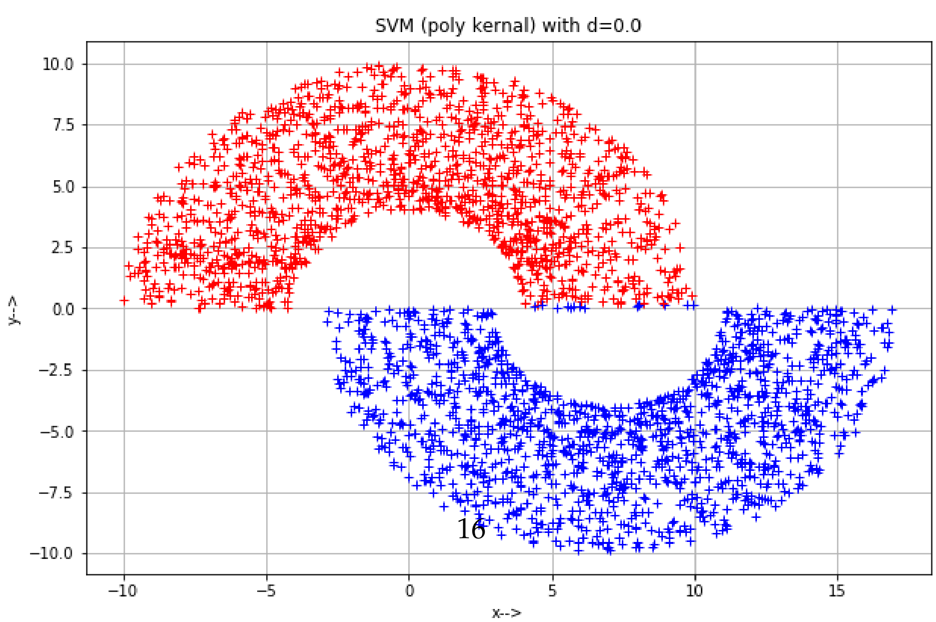
4 Case 1: d = 0.0

In [62]:
```
radius = 10.0
width = 6.0
distance = 0.0
num_of_training_samples = 1000
num_of_testing_samples = 3000

# initializing state vector machine with initial parameteres
state_vector_machine = StateVectorMachine(radius,
                                          width,
                                          distance,
                                          num_of_training_samples,
                                          num_of_testing_samples)



fig, ax = plt.subplots(nrows=3, ncols=1)
fig.set_figwidth(10)
fig.set_figheight(22)
kernals = ["linear","rbf","poly"]

for i in range(len(kernals)):
    # generating random training samples and desired output
    training_input,desired_output = state_vector_machine.get_samples("train")

    # training the state vector machine
    classifier = state_vector_machine.train(training_input,desired_output,kernals[i])

    # generating random testing samples
    test_samples,test_desired_output = state_vector_machine.get_samples("test")

    # predicting the output for test samples and ploting the result
    predicted_output = classifier.predict(test_samples)

    plt.subplot(3, 1, i+1)

    state_vector_machine.plot(test_samples,predicted_output,"SVM ("+kernals[i]+" kernal
```

```
/home/ramesh/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:578: DataConversi
  y = column_or_1d(y, warn=True)
```


SVM (linear kernal) with d=0.0
y-->
x-->
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10
15

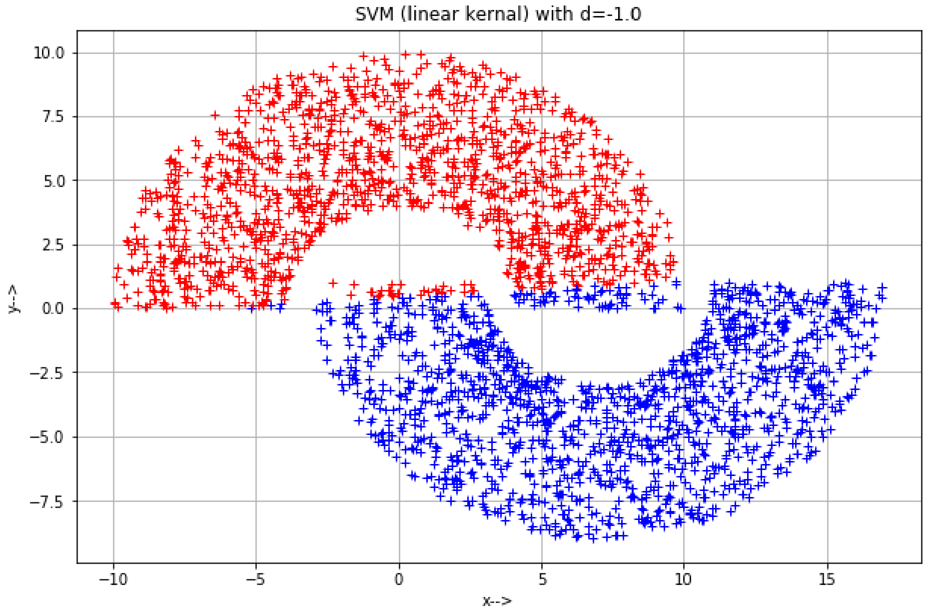

SVM (rbf kernal) with d=0.0
y-->
x-->
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10
15

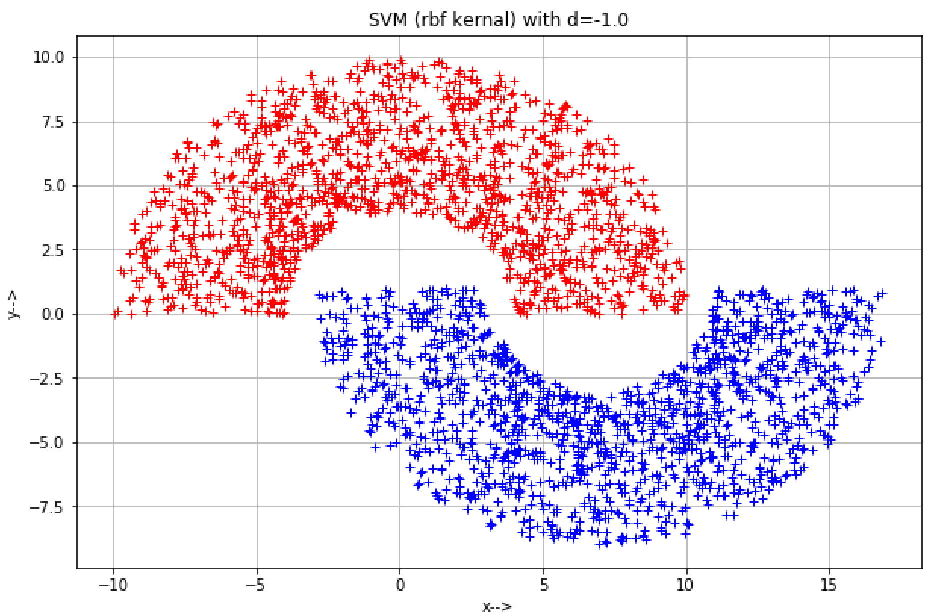

SVM (poly kernal) with d=0.0
y-->
x-->
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10
15

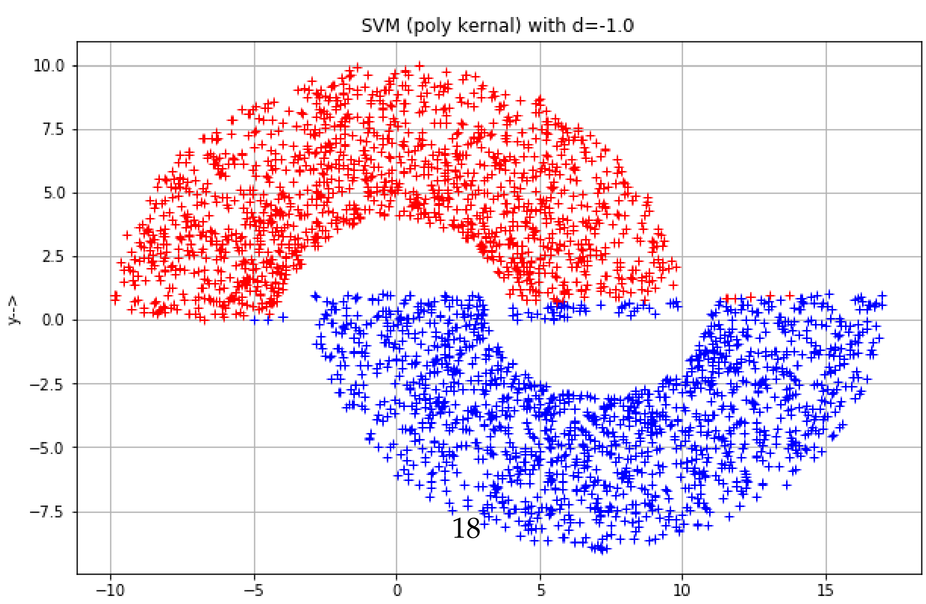
In case of d = 0.0, radial basis function kernal classifies the two region without any wrongly classified points. But linear and polynomial kernals classifies the regions with very few misclassified points.

5 Case 2: d = -1.0

In [63]:
```
radius = 10.0
width = 6.0
distance = -1.0
num_of_training_samples = 1000
num_of_testing_samples = 3000

# initializing state vector machine with initial parameteres
state_vector_machine = StateVectorMachine(radius,
                                          width,
                                          distance,
                                          num_of_training_samples,
                                          num_of_testing_samples)
fig, ax = plt.subplots(nrows=3, ncols=1)
fig.set_figwidth(10)
fig.set_figheight(22)
kernals = ["linear","rbf","poly"]

for i in range(len(kernals)):

    # generating random training samples and desired output
    training_input,desired_output = state_vector_machine.get_samples("train")

    # training the state vector machine
    classifier = state_vector_machine.train(training_input,desired_output,kernals[i])

    # generating random testing samples
    test_samples,test_desired_output = state_vector_machine.get_samples("test")

    # predicting the output for test samples and ploting the result
    predicted_output = classifier.predict(test_samples)

    plt.subplot(3, 1, i+1)

    state_vector_machine.plot(test_samples,predicted_output,"SVM ("+kernals[i]+" kernal
```


SVM (linear kernal) with d=-1.0
y-->
x-->
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10
15

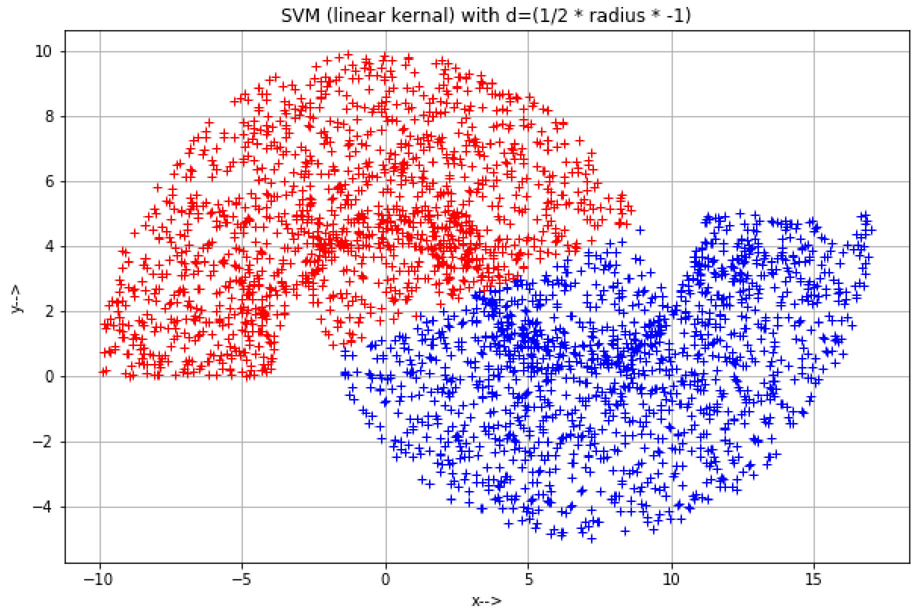

SVM (rbf kernal) with d=-1.0
y-->
x-->
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10
15

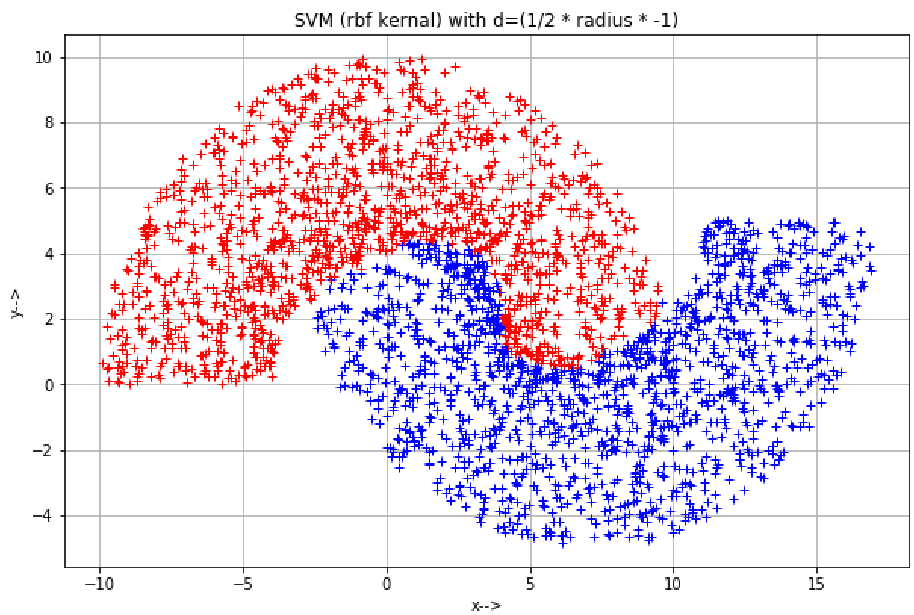

SVM (poly kernal) with d=-1.0
y-->
x-->
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10
−5
0
5
10
15

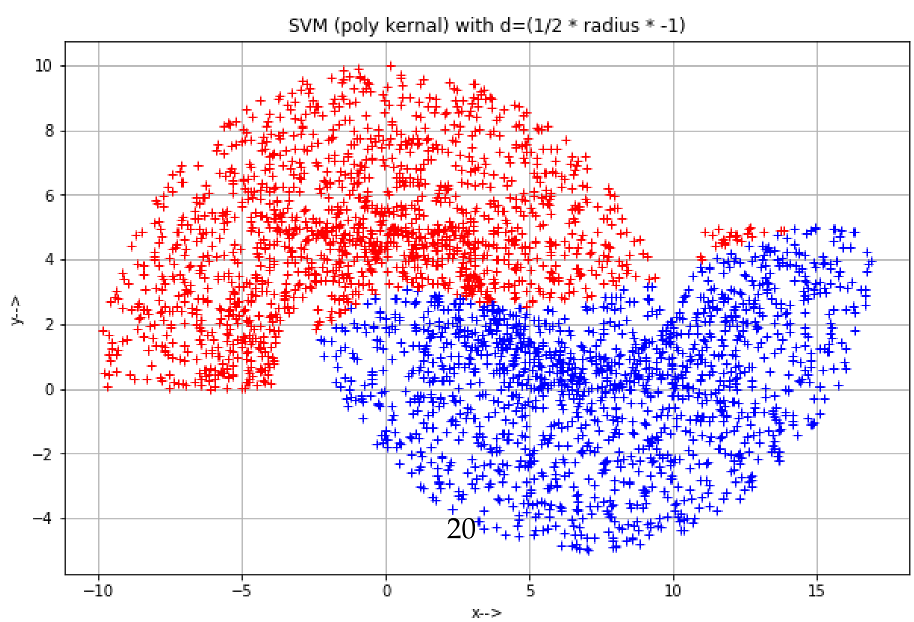
In case of d = -1.0, again radial basis function kernal classifies the two region without any misclassified points. But linear classifies the regions with more wrongly classified points compared to previous scenario (d=0). Polynomial kernal classified region B (blue color) correctly, but there are more wrongly classified points in region A (red color)

6 Case 3: $d = (1/2 * radius * (-1))$

```
In [64]: radius = 10.0
         width = 6.0
         distance = 0.5 * (radius) * -1.0
         num_of_training_samples = 1000
         num_of_testing_samples = 3000

         # initializing state vector machine with initial parameteres
         state_vector_machine = StateVectorMachine(radius,
                                                   width,
                                                   distance,
                                                   num_of_training_samples,
                                                   num_of_testing_samples)

         fig, ax = plt.subplots(nrows=3, ncols=1)
         fig.set_figwidth(10)
         fig.set_figheight(22)
         kernals = ["linear","rbf","poly"]

         for i in range(len(kernals)):

             # generating random training samples and desired output
             training_input,desired_output = state_vector_machine.get_samples("train")

             # training the state vector machine
             classifier = state_vector_machine.train(training_input,desired_output,kernals[i])

             # generating random testing samples
             test_samples,test_desired_output = state_vector_machine.get_samples("test")

             # predicting the output for test samples and ploting the result
             predicted_output = classifier.predict(test_samples)

             plt.subplot(3, 1, i+1)

             state_vector_machine.plot(test_samples,predicted_output,"SVM ("+kernals[i]+" kernal
```


SVM (linear kernal) with d=(1/2 * radius * -1)
y-->
x-->

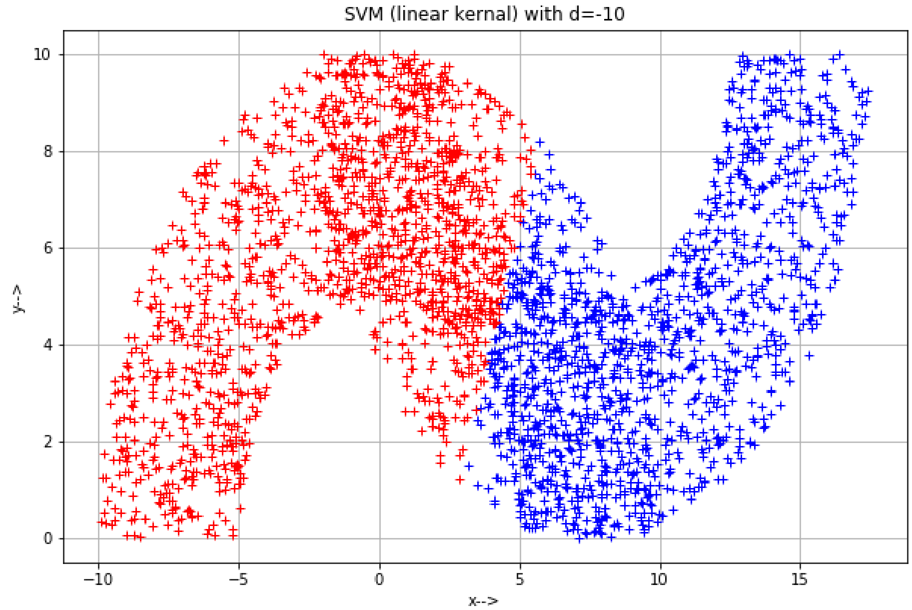

SVM (rbf kernal) with d=(1/2 * radius * -1)
y-->
x-->

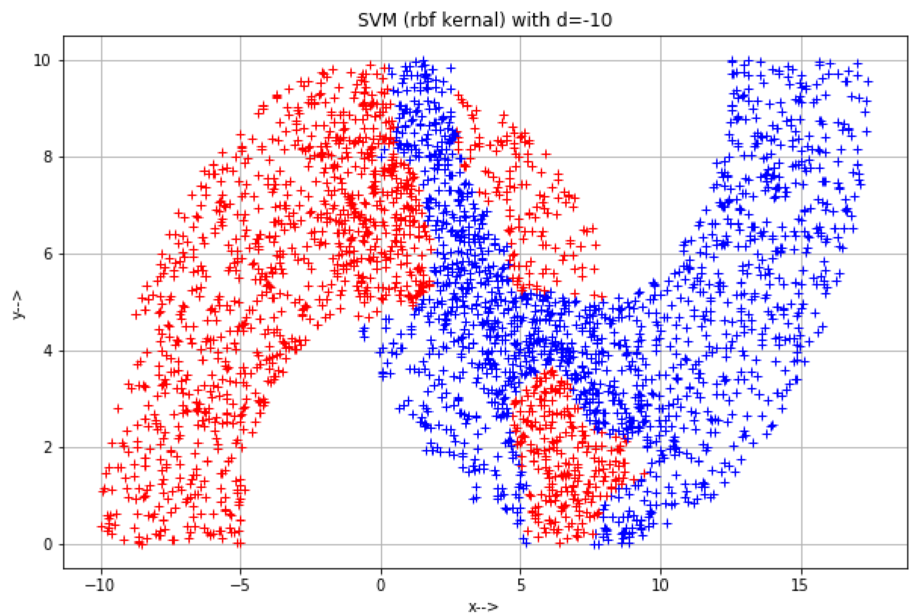

SVM (poly kernal) with d=(1/2 * radius * -1)
y-->
x-->

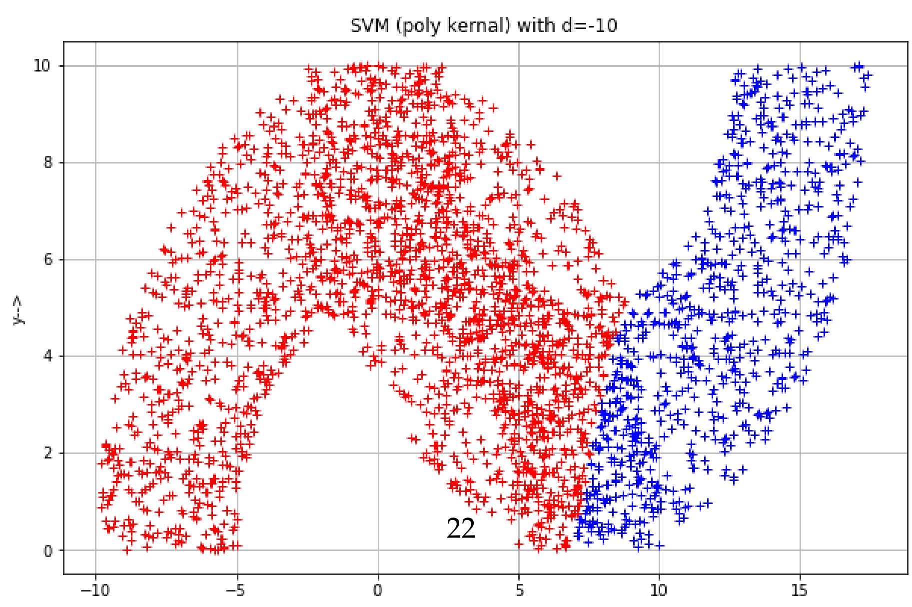
In case of $d = (1/2 * radius * (-1))$, radial basis function kernal classifies the two region with very few wrongly classified points. But linear and polynomial kernals classifies the regions with more number of misclassified points compared to previous scenarios.

7 Case 4: d=-10

In [65]:
```
radius = 10.0
width = 5.0
distance = -10.0
num_of_training_samples = 1000
num_of_testing_samples = 3000

# initializing state vector machine with initial parameteres
state_vector_machine = StateVectorMachine(radius,
                                          width,
                                          distance,
                                          num_of_training_samples,
                                          num_of_testing_samples)


fig, ax = plt.subplots(nrows=3, ncols=1)
fig.set_figwidth(10)
fig.set_figheight(22)
kernals = ["linear","rbf","poly"]

for i in range(len(kernals)):

    # generating random training samples and desired output
    training_input,desired_output = state_vector_machine.get_samples("train")

    # training the state vector machine
    classifier = state_vector_machine.train(training_input,desired_output,kernals[i])

    # generating random testing samples
    test_samples,test_desired_output = state_vector_machine.get_samples("test")

    # predicting the output for test samples and ploting the result
    predicted_output = classifier.predict(test_samples)

    plt.subplot(3, 1, i+1)

    state_vector_machine.plot(test_samples,predicted_output,"SVM ("+kernals[i]+" kernal
```


SVM (linear kernal) with d=-10
y-->
x-->
10
8
6
4
2
0
−10
−5
0
5
10
15


SVM (rbf kernal) with d=-10
y-->
x-->
10
8
6
4
2
0
−10
−5
0
5
10
15


SVM (poly kernal) with d=-10
y-->
x-->
10
8
6
4
2
0
−10
−5
0
5
10
15

In case of d = -10, all kernals classified the two regions with wrongly classified points and wrong decision boundaries.