Assignment 3: Constrained Optimization, SVMs

Copyrighting and Fare Use

Automatic Testing Guidelines

Automatic unittesting requires you, as a student, to submit a notebook which contains strictly defined objects. Strictness of definition consists of unified shapes, dtypes, variable names and more.

Within the notebook, we provide detailed instruction which you should follow in order to maximise your final grade.

Name your notebook properly, follow the pattern in template name:

Assignment_N_NameSurname_matrnumber

1. N - number of assignment
2. NameSurname - your full name where every part of the name starts with a capital letter, no spaces
3. matrnumber - your 8-digit student number on ID card (without k)

Example:
✅ Assignment_0_RenéDescartes_12345678
✅ Assignment_0_SørenAabyeKierkegaard_12345678
❌ Assignment0_Peter_Pan_k12345678

Don't add any cells but use the ones provided by us. You may notice that most cells are tagged such that the unittest routine can recognise them.

We highly recommend you to develop your code within the provided cells. You can implement helper functions where needed unless you put them in the same cell they are actually called. Always make sure that implemented functions have the correct output and given variables contain the correct data type. Don't import any other packages than those listed in the cell with the "imports" tag.

Note: Never use variables you defined in another cell in your functions directly; always pass them to the function as a parameter. In the unitest they won't be available either.

Good luck!

Calculation 1 (25 points):

Consider the following primal (constrained convex optimization) problem:

$$\begin{aligned} \text{Minimize} \quad & f(w_1, w_2) = 2\left(w_1^4 + w_2^4\right) \\ \text{subject to} \quad & h(w_1, w_2) = 4 - w_1 + w_2 \leq 0 \end{aligned}$$

We try to solve it via two ways. Perform the following tasks:

- Compute the Lagrangian $L(w_1, w_2, \alpha)$.
- Calculate its derivatives with respect to w_1 and w_2 and calculate the zeros of these derivatives (w_1^*, w_2^*).

First way:

- Solve the problem using the KKT conditions.

Second way:

- Write down the dual problem and solve it directly without(!) the help of the KKT-conditions. Hint: You need the derivative of $\mathcal{L} = L(w_1^*, w_2^*, \alpha)$ with respect to α.

For your calculation use the given notation.

1. Lagrangian function

$$L(w_1, w_2, \alpha) = f(w_1, w_2) + \alpha h(w_1, w_2)$$

$$L(w_1, w_2, \alpha) = 2w_1^4 + 2w_2^4 + \alpha(4 - w_1 + w_2)$$

$$L(w_1, w_2, \alpha) = 2w_1^4 + 2w_2^4 + 4\alpha - \alpha w_1 + \alpha w_2$$

2. Derivatives of Lagrangian and zeros of these derivatives (w_1^*, w_2^*)

$$\frac{\partial L}{\partial w_1} = 8w_1^3 - \alpha => w_1^* = \sqrt[3]{\frac{\alpha}{8}} = \frac{\sqrt[3]{\alpha}}{2}$$

$$\frac{\partial L}{\partial w_2} = 8w_2^3 - \alpha => w_1^* = \sqrt[3]{\frac{-\alpha}{8}} = \frac{\sqrt[3]{-\alpha}}{2}$$

3. Solving by using KKT conditions

   $1)\ \alpha > 0, h(w_1^*, w_2^*) = 0$

$$h(w_1^*, w_2^*) = 4 - \frac{\sqrt[3]{\alpha}}{2} + \frac{\sqrt[3]{-\alpha}}{2} = 0$$

$$-\frac{\sqrt[3]{\alpha}}{2} + \frac{\sqrt[3]{-\alpha}}{2} = -4$$

$$-\sqrt[3]{\alpha} + \sqrt[3]{-\alpha} = -8$$

$$-\sqrt[3]{\alpha} + \sqrt[3]{-1}\sqrt[3]{\alpha} = -8$$

$$-2\sqrt[3]{\alpha} = -8$$

$$\sqrt[3]{\alpha} = 4$$

$$\alpha = 64$$

$$w_1^* = \frac{\sqrt[3]{64}}{2} = 2$$

$$w_1^* = \frac{\sqrt[3]{-64}}{2} = -2$$

4. Solving without KKT conditions

$$\mathcal{L} = L(w_1, w_2, \alpha) = 2w_1^4 + 2w_2^4 + \alpha(4 - w_1 + w_2)$$

$$= 2(\frac{\sqrt[3]{\alpha}}{2})^4 + 2(\frac{\sqrt[3]{-\alpha}}{2})^4 + \alpha(4 - \frac{\sqrt[3]{\alpha}}{2} + \frac{\sqrt[3]{-\alpha}}{2})$$

$$= \frac{\alpha^{\frac{4}{3}}}{8} + \frac{(-\alpha)^{\frac{4}{3}}}{8} + 4\alpha - \alpha * \alpha^{\frac{1}{3}}$$

$$= \frac{\alpha^{\frac{4}{3}}}{8} + \frac{\sqrt[3]{(-1^4)}\alpha^{\frac{4}{3}}}{8} - 4\alpha + \alpha^{\frac{4}{3}}$$

$$= \frac{2\alpha^{\frac{4}{3}}}{8} + 4\alpha - \alpha^{\frac{4}{3}}$$

$$= \frac{\alpha^{\frac{4}{3}}}{4} + 4\alpha - \alpha^{\frac{4}{3}} = \frac{-3\alpha^{\frac{4}{3}}}{4} + 4\alpha$$

$$\boxed{\text{\newline}}$$

$$\frac{\partial L}{\partial \alpha} = \frac{4 * (-3)\sqrt[3]{\alpha}}{3 * 4} + 4 = 0$$

$$-\sqrt[3]{\alpha} + 4 = 0$$

$$\sqrt[3]{\alpha} = 4 => \alpha = 64$$

Calculation 2 (25 points):

Suppose we replace in the primal optimization problem of C-SVMs the penalty term $\|\xi\|_1 = \sum_{i=1}^{l} \xi_i$ with $\|\xi\|_2^2 = \sum_{i=1}^{l} \xi_i^2$ (that is we use the quadratic hinge loss instead). Thus, the primal problem we are considering is given as follows (with $C > 0$):

$$\begin{aligned} \text{Minimize} \quad & \frac{1}{2}\|\mathrm{w}\|^2 + C\sum_{i=1}^{l} \xi_i^2 \\ \text{subject to} \quad & -\left[y_i\left(\mathrm{w}\cdot \mathrm{x}_i - b\right) - 1 + \xi_i\right] \leq 0 \\ \text{and} \quad & -\xi_i \leq 0 \ \text{ for } \ i = 1, \ldots, l \end{aligned}$$

for all $i = 1, \ldots, l$. Give the associated dual optimization problem by making use of the KKT-Theorem and perform the following tasks:

- Why can the KKT-Theorem be applied?
- Calculate the Lagrange function L.
- Calculate its derivatives (with respect to w_i, b and ξ_i) and compute their zeros.
- Write down the dual function (named $\mathcal{L}$ in the slides) and the corresponding dual problem, depending on $2l$ Lagrange multipliers. (You don't need to solve the dual problem.)
- Simplify the dual problem so that it only depends on l Lagrange variables. Hint: The maximization with respect to λ_i can be done quite easily. (Again: You don't need to solve the dual problem. Just find the simplified form.)

Please provide reasoning and explanations in full sentences.

1. Why can the KKT-Theorem be applied?
   Because the problem can be cast into a convex quadratic optimization again. Also $\alpha_i * h_i = 0$ holds.

2. Lagrangian function
   $SolutionHere$

3. Derivatives of Lagrangian
   $SolutionHere$

4. The dual function and the dual problem
   $SolutionHere$

5. Simplify
   $SolutionHere$

Code 1 (10 points):

The aim of the following task is to equip you with some intuition concerning the application of different SVMs to an easy data set.
You should also observe how different versions of the SVMs with different hyperparameters react to additional noise. To this end we provided you with a function

`plot_data` which is intended to create proper visualizations of important characteristics of linear and nonlinear SVMs, like the decision border and support vectors. The usual routine for applying SVMs in Python, which is also used here, is given by the following sklearn-package: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

Your first task is to get familiar with the `plot_data` function by applying it to the easy data set `radial_data.csv`. Iterate over different kernels and hyperparameters (consider the documentation for details), i.e. perform the following tasks:

- Plot the dataset using the provided functions. (Just execute the existing code.)
- Using the sklearn-package mentioned above, write a function `iter_Degree` that applies an SVM with a polynomial kernel and $C = 10$ to the data and iterates over the degrees (i.e. hyperparameter) from 1 to 5. Function `iter_Degree` must return a list of model parameters for each model you initiated and fitted on our data. The output of `get_params()` is a dictionary; see sklearn docs. Apply `plot_data` to each hyperparameter setting. Then return.

In [1]:
```
#Nothing to do here
import numpy as np
import matplotlib.pyplot as plt
import random
import csv
from typing import Sequence, List, Tuple

np.random.seed(1234)

import warnings
warnings.filterwarnings("ignore")

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

from sklearn import svm
```

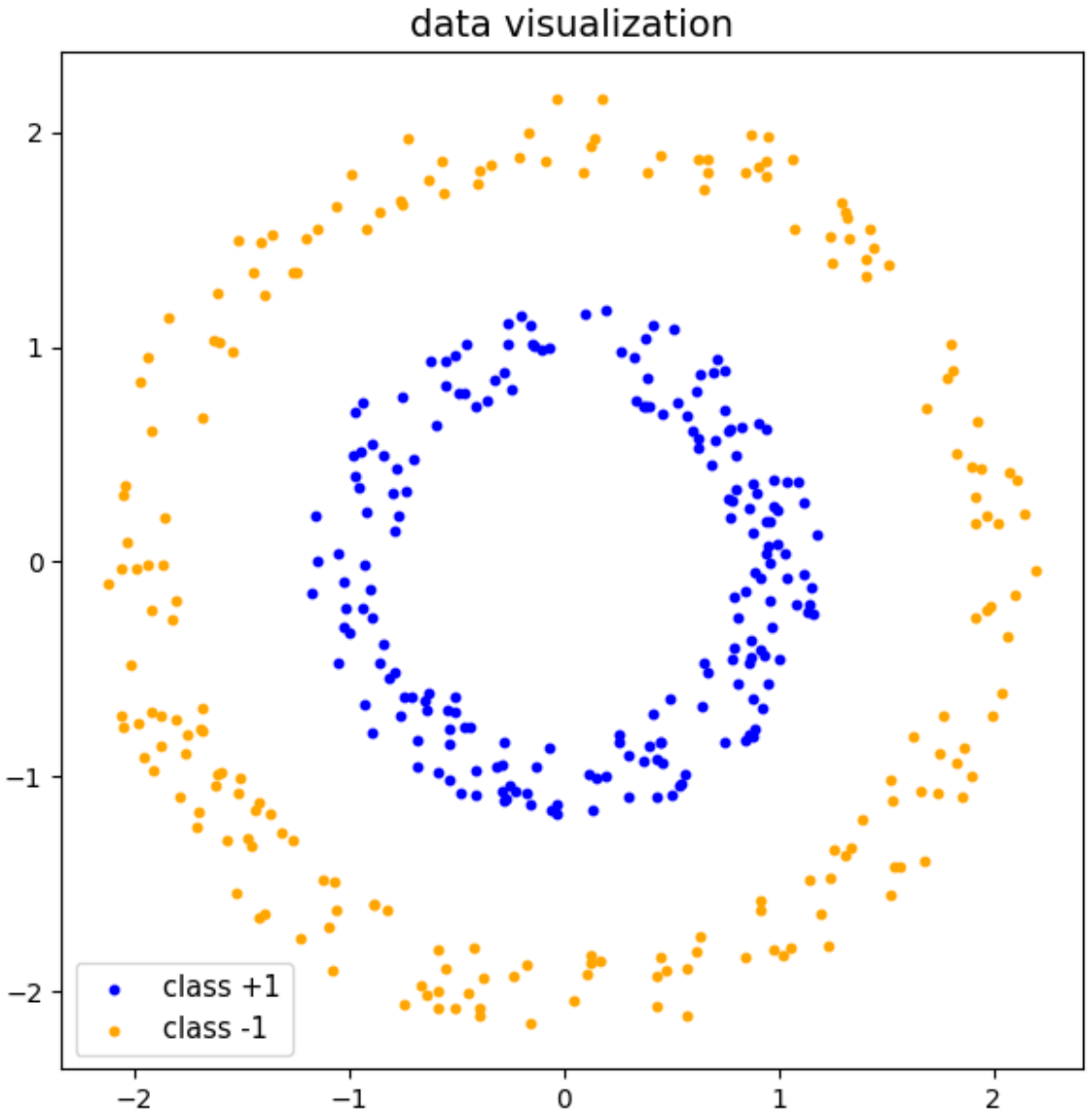
In [2]:
```
#Nothing to do here
def load_data(id_data: int=1):
    """Function allows to load data from csv
    @returns: tuple (X,y)"""
    if id_data ==1:
        Z = np.genfromtxt('radial_data.csv', delimiter=',')
        return Z[:,:-1], Z[:,-1]

def get_meshgrid(X,resolution):
    """Function creates space/grid. Mostly used for plotting"""
    s = np.max(np.abs(X))*1.05
    ls = np.linspace(-s, s, resolution)
    X1,X2 = np.meshgrid(ls, ls, sparse=False)
    return np.c_[X1.ravel(), X2.ravel()]

def plot_data(X, y,
              model=None,
              plot_boarders=True,
              plot_classification=True,
```

```
            plot_support_vectors = True,
            plot_size=7,
            resolution=500,
            title='data visualization',
            color = ['blue','orange']):
    """Plotting your data
    @param model already trained SVM model, is None if you want to plot data onl
    all other parameters must be intuitively clear for you"""

    if model is not None:#if you want to plot model
        if plot_classification and plot_boarders:
            col=2  #if you want to plot model and boarders
        else :
            col=1  #if you want to plot model only

        fig,axs = plt.subplots(1,col,figsize=(plot_size*col,plot_size))

        grid = get_meshgrid(X,resolution)
        V = model.support_vectors_
        mask_sv = model.support_ #np.where(np.isin(X[:,0],V[:,0]))[0]

        kernel = model.kernel
        if kernel == 'poly':
            title = f"kernel: {kernel} - degree: {model.degree} - cost:{model.C}
        elif kernel == 'rbf':
            title = f"kernel: {kernel}"
            if model.gamma != "auto_deprecated" :
                title+= f" - gamma: {model.gamma}"
            title += f" - cost: {model.C}"

        for i in range(col):
            if col>1:
                ax = axs[i]
            else:
                ax = axs
            ax.set_aspect('equal')
            if i==0 and plot_boarders:
                ax.set_title('Margins - ' + title,fontsize=plot_size*2)
                boarders = model.decision_function(grid)
                mask_pos = boarders >= 1
                mask_neg = boarders <= -1
                ax.scatter(grid[mask_pos,0], grid[mask_pos,1], c=color[0], alpha
                ax.scatter(grid[mask_neg,0], grid[mask_neg,1], c=color[1], alpha
                ax.scatter(X[mask_sv,0], X[mask_sv,1], c='g',label= str(np.sum(m
            if plot_classification and (i==1 or not plot_boarders):
                ax.set_title('Classification - ' + title,fontsize=plot_size*2)
                classification = model.predict(grid)
                mask_pos = classification > 0
                mask_neg = classification < 0
                ax.scatter(grid[mask_pos,0], grid[mask_pos,1], c=color[0], alpha
                ax.scatter(grid[mask_neg,0], grid[mask_neg,1], c=color[1], alpha
                classification = model.predict(X)
                mask_wrong = classification != y
                ax.scatter(X[mask_wrong,0], X[mask_wrong,1], c='magenta',label=s
            m = y > 0
            ax.scatter(X[m,0], X[m,1], c=color[0],label='class +1',s=10)
            m = np.logical_not(m)
            ax.scatter(X[m,0], X[m,1], c=color[1],label='class -1',s=10)
            ax.legend(loc='lower left', fontsize=plot_size*1.5)
```

```
    else:
        fig,axs = plt.subplots(1,1,figsize=(plot_size,plot_size))
        axs.set_aspect('equal')
        m = y > 0
        axs.scatter(X[m,0], X[m,1], c=color[0],label='class +1',s=10)
        m = np.logical_not(m)
        axs.scatter(X[m,0], X[m,1], c=color[1],label='class -1',s=10)
        axs.legend(loc='lower left', fontsize=plot_size*1.5)
        plt.title(title, fontsize=plot_size*2)
    plt.show()
```

In [3]:
```
#load data
#leave as it is
X,y = load_data(1)
```

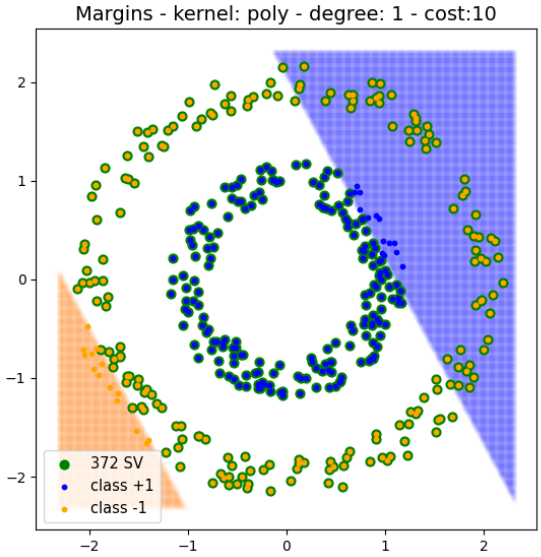
In [4]:
```
#plot your data ↓↓↓
plot_data(X, y)
```

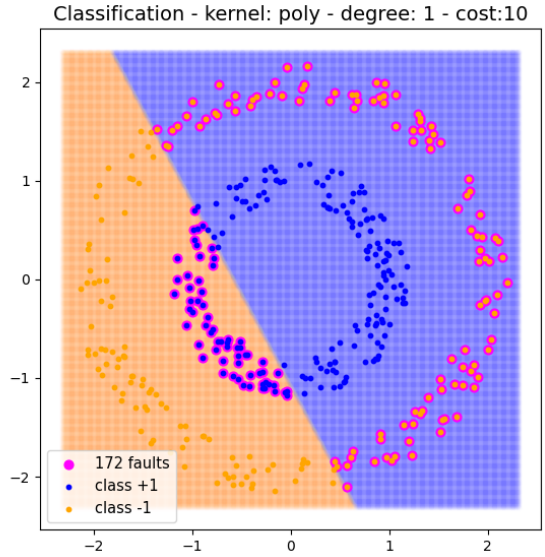


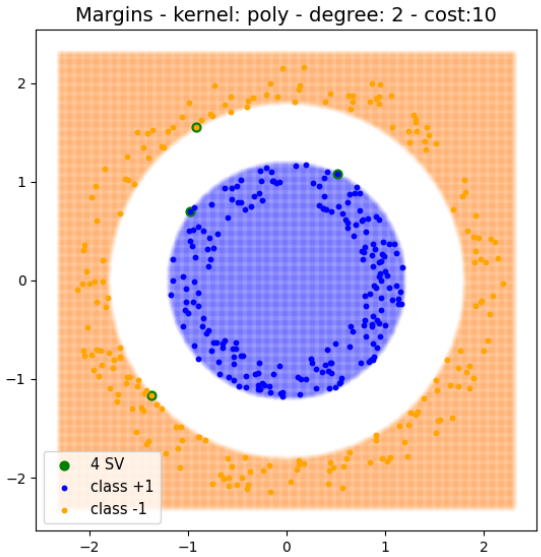
In [5]:
```
def iter_Degree(degree_range: Sequence[int], X: np.ndarray, y:np.ndarray) -> Lis
    """
    Function iter_Degree fits SVM using defined range of degrees and plots every
    @ degree_range: range of integer degrees
    @ returns list of dictionaries, length of list = length of degree_range
    """
```

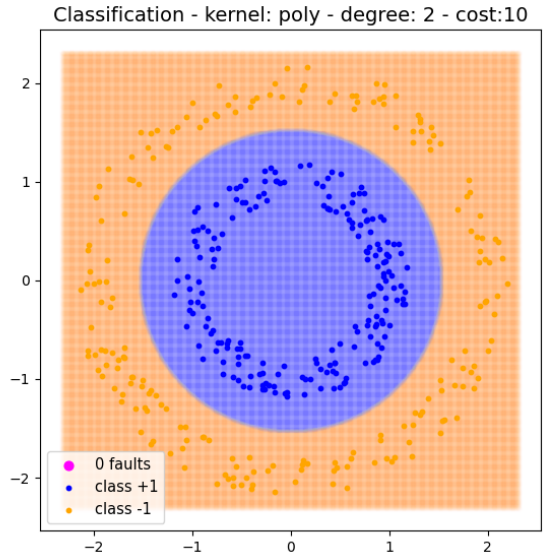
```
    list_of_model_params = []
    #your code ↓↓↓
    # hint: for plotting use a already defined function
    for degree in degree_range:
        model = svm.SVC(kernel="poly", degree=degree, C=10)
        model.fit(X, y)
        list_of_model_params.append(model.get_params())
        plot_data(X, y, model)

    return list_of_model_params
```

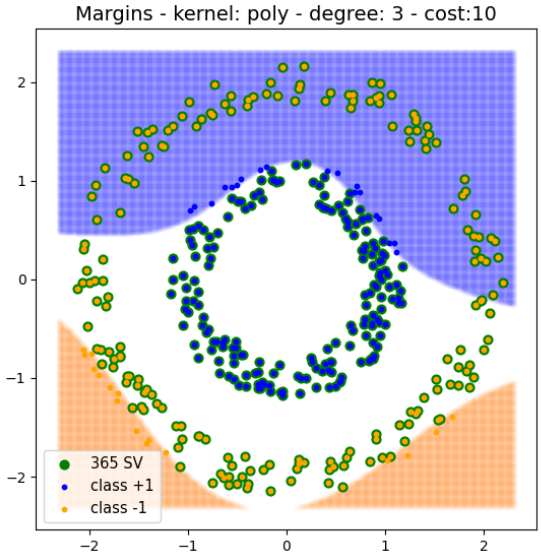
In [6]:
```
plots_list_2 = []
for degree in range(1,6):

    model = svm.SVC(kernel="poly",degree=degree,C=10)
    model.fit(X,y)

    axs = plot_data(X,y,model)
    plots_list_2.append(axs)
```

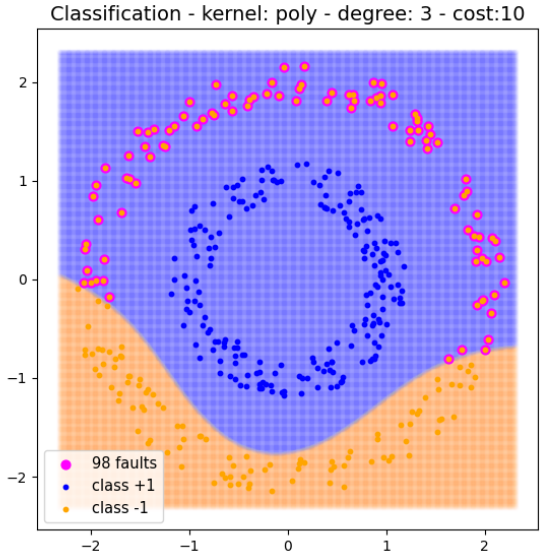


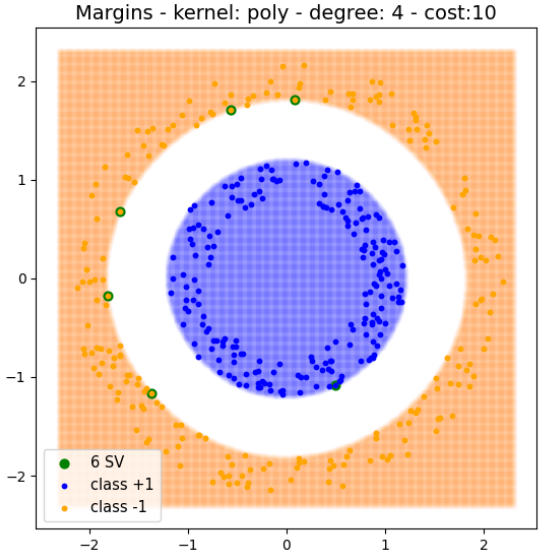


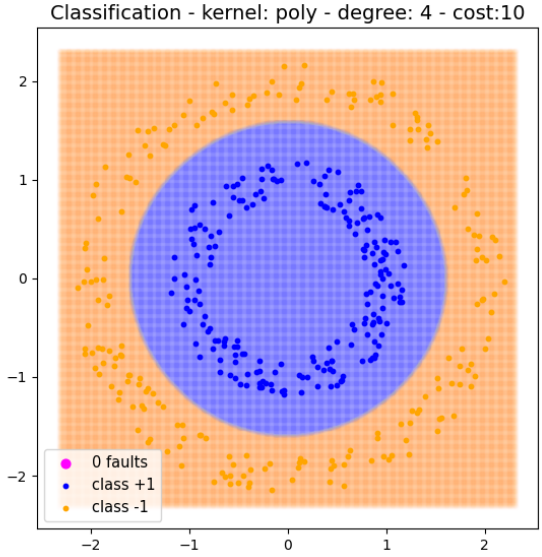


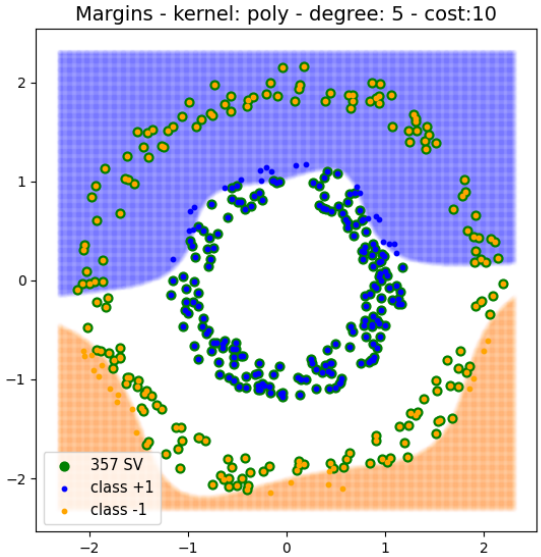


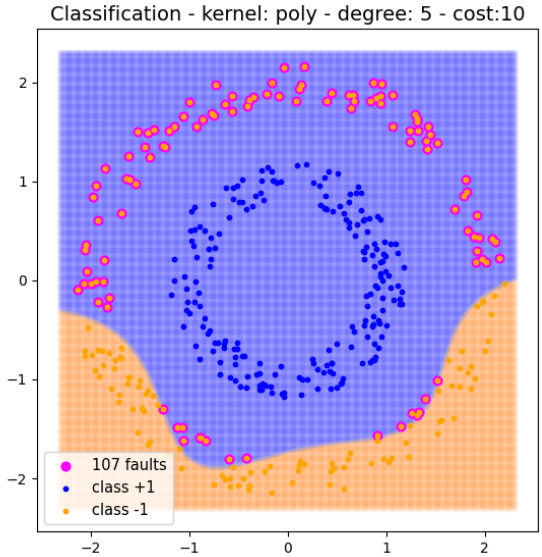












Question 1 (5 points):

What observations can you make from your plots? (several answers may be correct)

a1_) The SVM with polynomial degree 2 already seems to do quite well on the data set.
b1_) The higher the polynomial degree, the better the classifier.

c1_) A very high number of support vectors seems to be an indicator of a good choice of the kernel.

d1_) There is only small difference between the pictures that were produced by polynomial kernels of even degree.
e1_) For kernels with an odd degree, the number of misclassified samples decreases with an increasing degree.
f1_) For odd degrees, one can see that the model complexity increases with increasing degree.

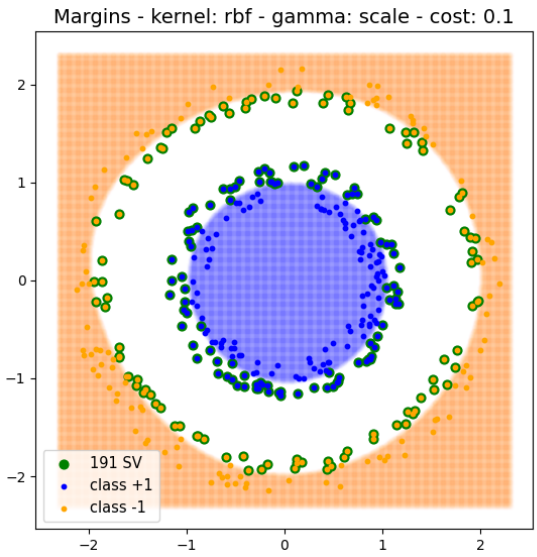
To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question.

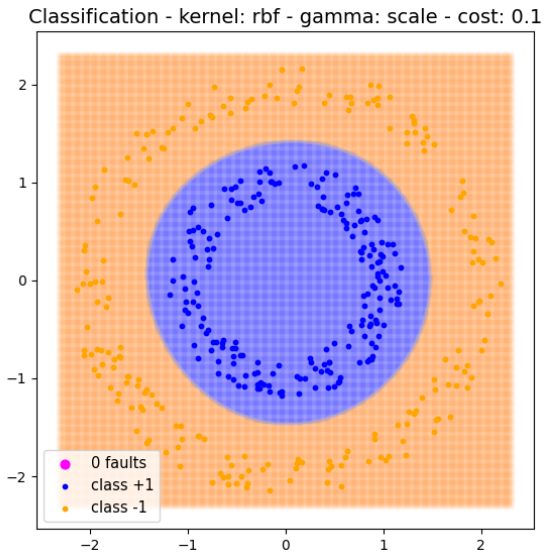
Note: Do not reuse these variable names. They are used for testing.

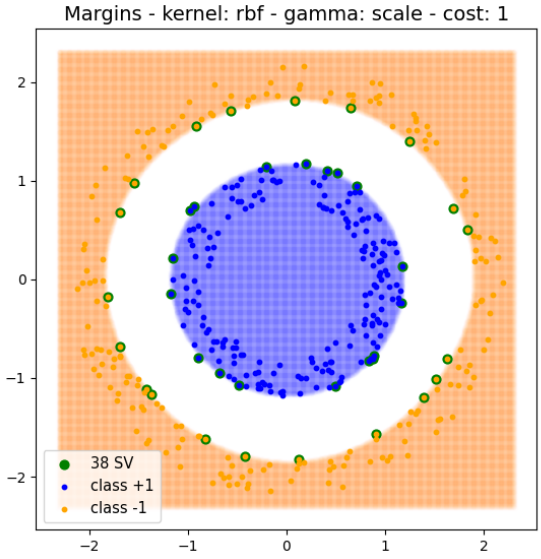
In [7]:
```
example_true = True
example_False = False

a1_ = True
b1_ = False # degree 3 is not working well
c1_ = False

d1_ = True
e1_ = False # degree 3 has less faults than degree 5
f1_ = True
```

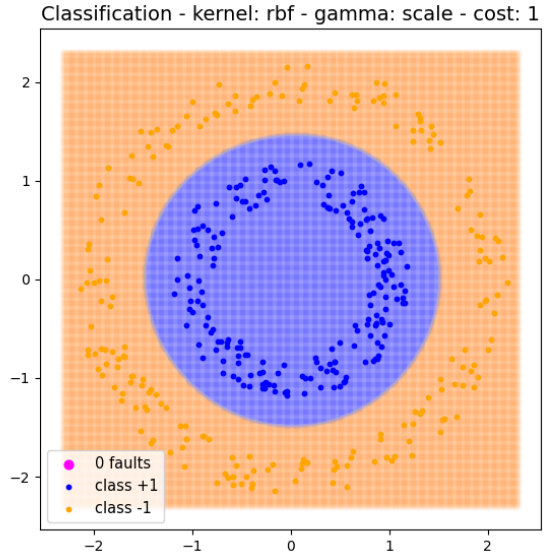
Now apply an rbf-kernel (with default gamma="scale") and vary the C-parameter from close 0 to very large values. After playing around a little bit, try to report on your observations in the subsequent question.

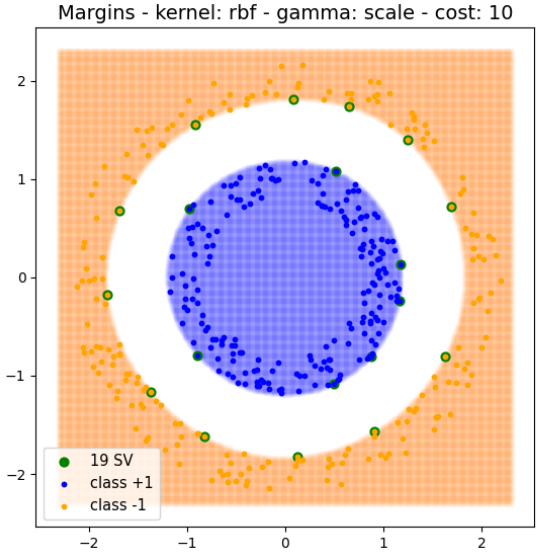
In [8]:
```
# Here you can play around with the code a little bit. Cell will not be used for

for C in (0.1,1,10,100,1000):
    model = svm.SVC(kernel="rbf",C=C)
    model.fit(X,y)
    plot_data(X,y,model)
```

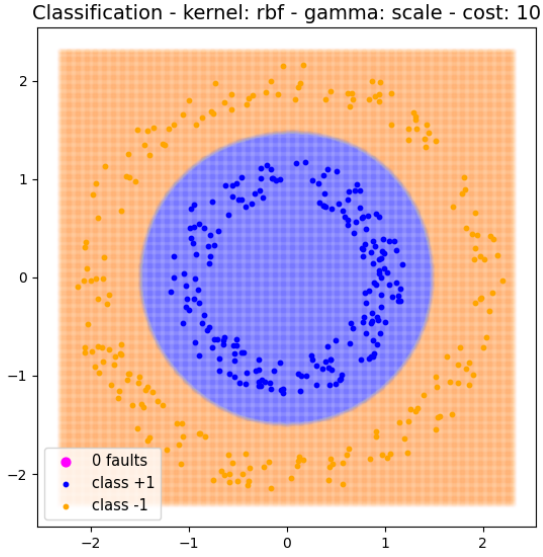


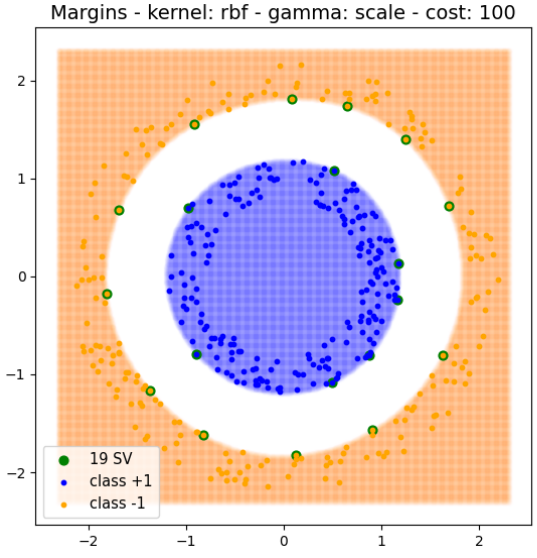


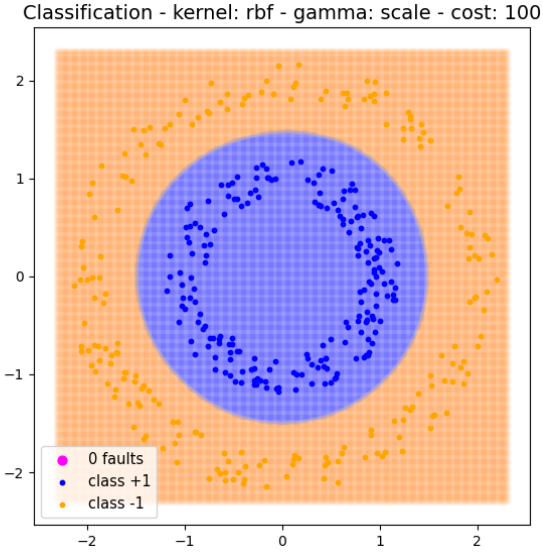

Margins - kernel: rbf - gamma: scale - cost: 1
38 SV
class +1
class -1

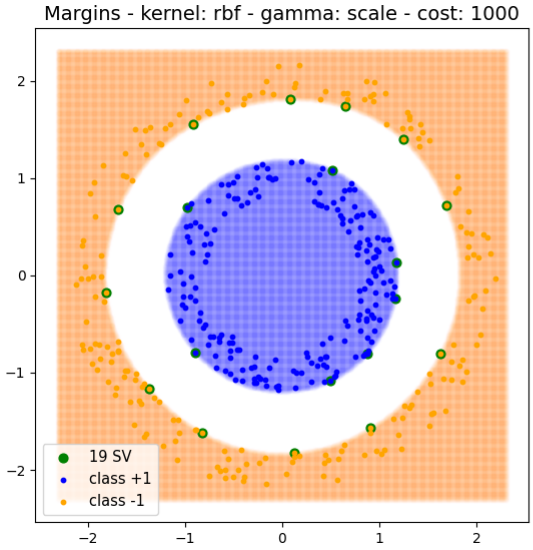

Classification - kernel: rbf - gamma: scale - cost: 1
0 faults
class +1
class -1

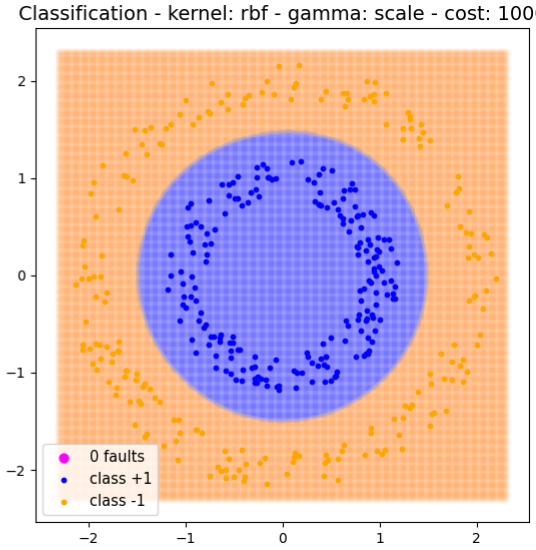

Margins - kernel: rbf - gamma: scale - cost: 10
19 SV
class +1
class -1


Classification - kernel: rbf - gamma: scale - cost: 10
0 faults
class +1
class -1


Margins - kernel: rbf - gamma: scale - cost: 100
19 SV
class +1
class -1


Classification - kernel: rbf - gamma: scale - cost: 100
0 faults
class +1
class -1





Question 2 (5 points):

What observations can you make from your plots? (several options may be correct):

a2_) The higher the cost, the more support vectors we have.
b2_) The decision boarders don't change drastically with increasing C, only the number of support vectors does.
c2_) The lower the cost, the larger the margin.

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer “None”) gives 0 points for a question.
Note: Do not reuse these variable names. They are used for testing.

In [9]:
```
example_true = True
example_False = False

a2_ = False # lower cost -> more SVs
b2_ = True
c2_ = True
```

Code 2 (10 points):

- Now use function (polar-to-cartesian) `pol2cart()` which returns **100 two-dimensional** points which are **uniformly** distributed within the circle of radius $r = 0.25$ to generate new data points. Label these points with -1 and add them to the main (imported) `X` feature matrix and `y` label vector. Your new variables will be `X_new` and `y_new`. Note: In polar coordinates, you may sample the angle uniformly, but be careful with the radius. If you sample the radius uniformly, you will not end up with uniformly sampled points within the circle.
- Now use an **rbf kernel** (which is the default option for `sklearn.svm.SVC`) and again play around with the parameters to explore the effects on the classification performance by appropriately using the `plot_data` function. Write a function `iter_Gamma`, analogous to `iter_Degree`. Try out small costs $C \sim 0.1$ and large

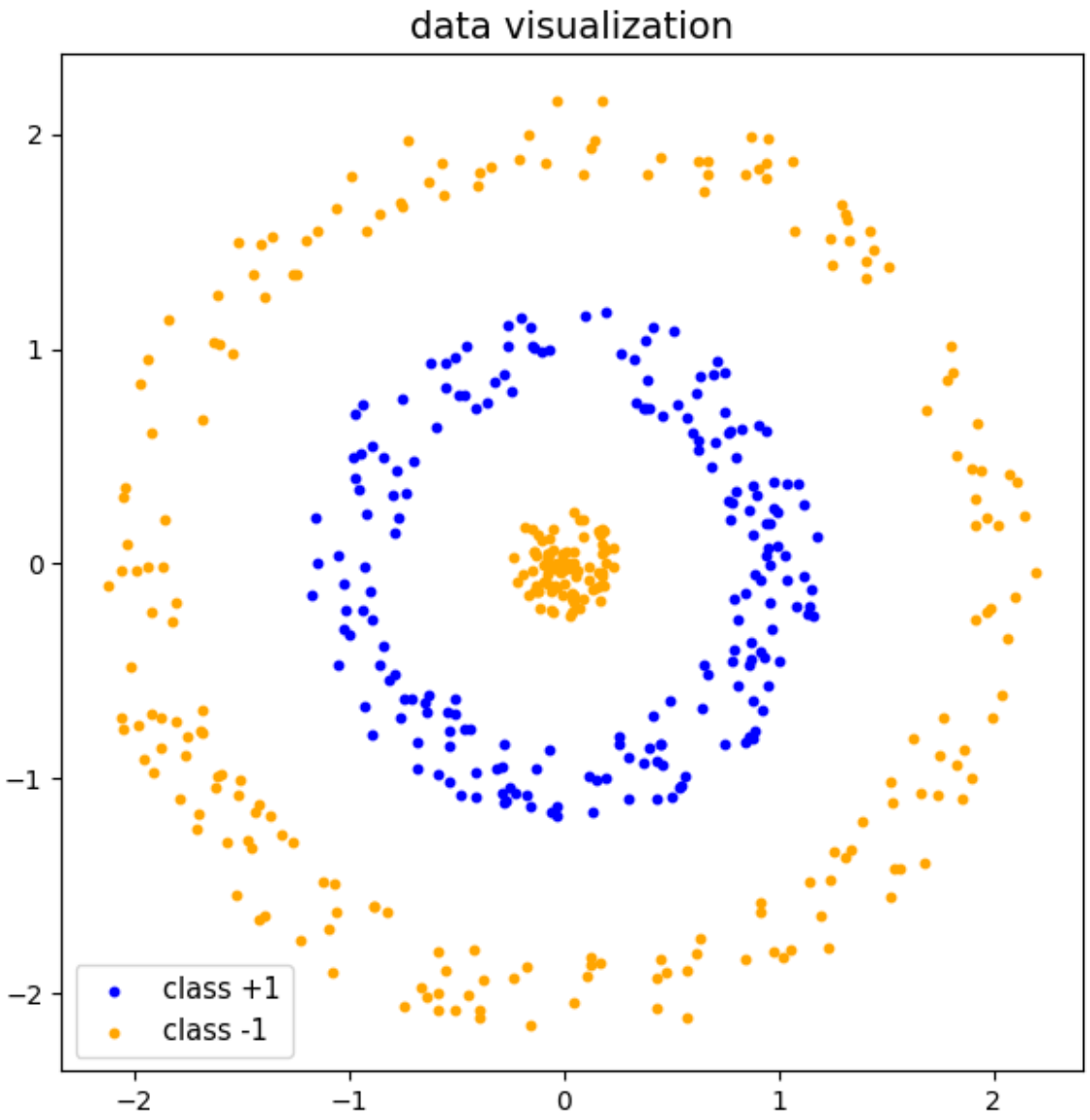
costs $C \sim 1000$ and iterate over different values of $\gamma := 1/(2\sigma^2)$ (compare with the RBF definition in the lecture slides), ranging from 0.1 to 1. Again report your observations in the subsequent question.

In [10]:
```
#code that should help you
#leave as it is
X,y=load_data(1)
def pol2cart(r, phi):
    x = r * np.cos(phi)
    y = r * np.sin(phi)
    return(x, y)
```

In [11]:
```
#define new data ↓↓↓
def create_new_data() -> Tuple[np.ndarray, np.ndarray]:
    """Function that creates the new data

    For random numbers use np.random module, not np.random.default_rng

    Hint
    ----
    use pol2cart

    Returns
    -------
    tuple
        tuple of 100 2d datapoints and labels (X_new, y_new)
    """
    # Your solution here:
    n = 100
    rad = 0.25
    # phi ranges from 0 to 2*pi
    # (full rotation is 2*pi)
    phi = np.random.uniform(0, 2 * np.pi, n)
    # distance from center is in [0, rad]
    r = np.random.uniform(0, rad, n)

    x1, x2 = pol2cart(r, phi)
    y_new = np.full(100, -1)

    X_new = np.stack((x1, x2), axis=1)
    return X_new, y_new
```

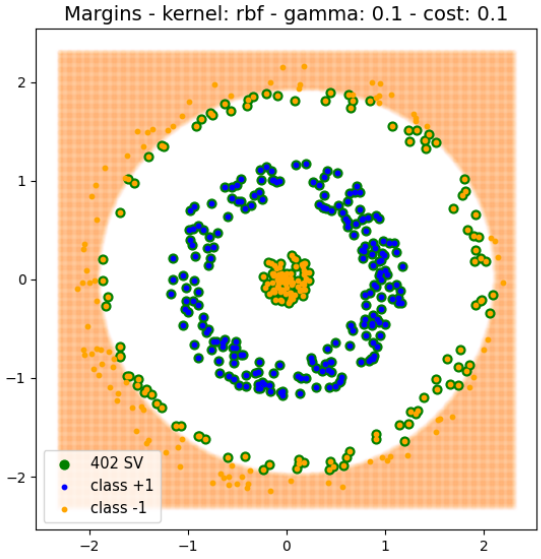
In [12]:
```
X_new, y_new = create_new_data()
```

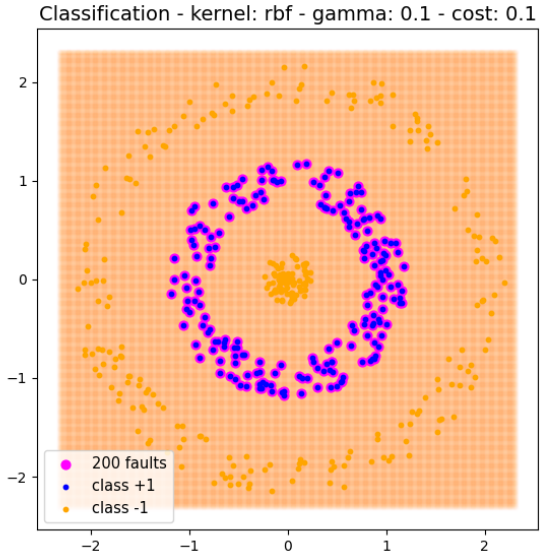
In [13]:
```
# plot new data ↓↓↓
# Hint: use already defined functions
X_new = np.concatenate((X, X_new), axis=0)
y_new = np.concatenate((y, y_new), axis=0)
plot_data(X_new, y_new)
```

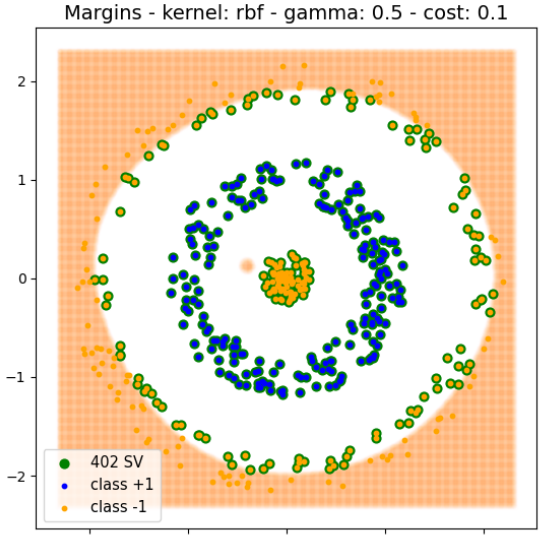


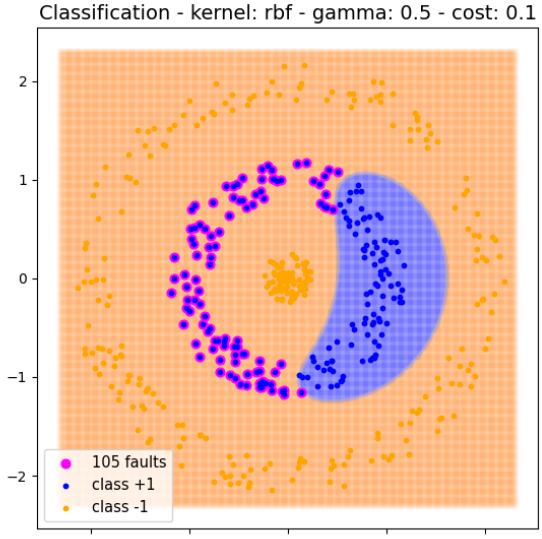
In [14]:
```
def iter_Gamma(gamma_range: Sequence[float], C: float, X_new: np.ndarray, y_new:
    """
    Function iter_Gamma fits SVM using defined gamma range and plots every varia
    @ gamma_range: list of gammas
    @ returns: list of dictionaries, length of list = length of gamma_range
    """
    list_of_model_parameters = []
    # your code ↓↓↓
    # code ends here
    for gamma in gamma_range:
        model = svm.SVC(gamma=gamma, C=C)
        model.fit(X_new, y_new)
        list_of_model_parameters.append(model.get_params())
        plot_data(X_new, y_new, model)

    return list_of_model_parameters
```

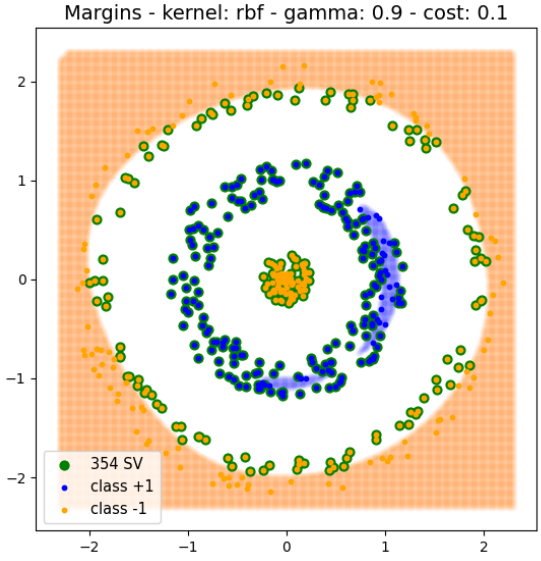
In [15]:
```
cost_values = range(-1,4)
gamma_values = [0.1,0.5,0.9]

for cost in cost_values:
    iter_Gamma(gamma_values,10**cost,X_new,y_new)
```

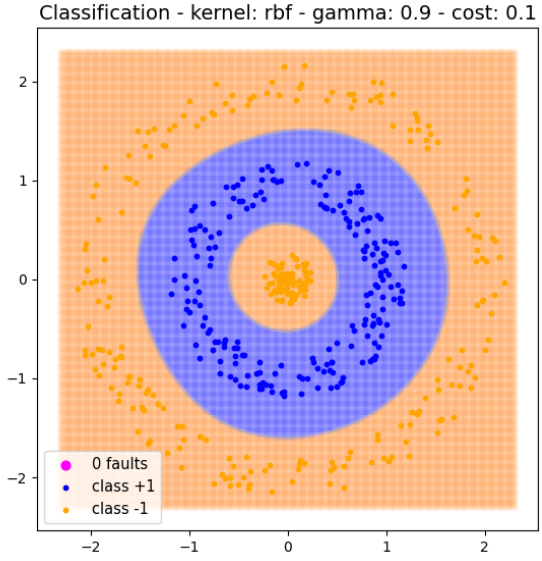

Margins - kernel: rbf - gamma: 0.1 - cost: 0.1
402 SV
class +1
class -1

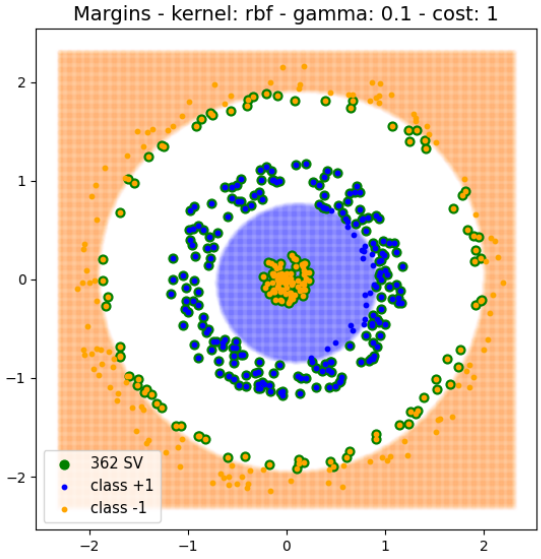

Classification - kernel: rbf - gamma: 0.1 - cost: 0.1
200 faults
class +1
class -1

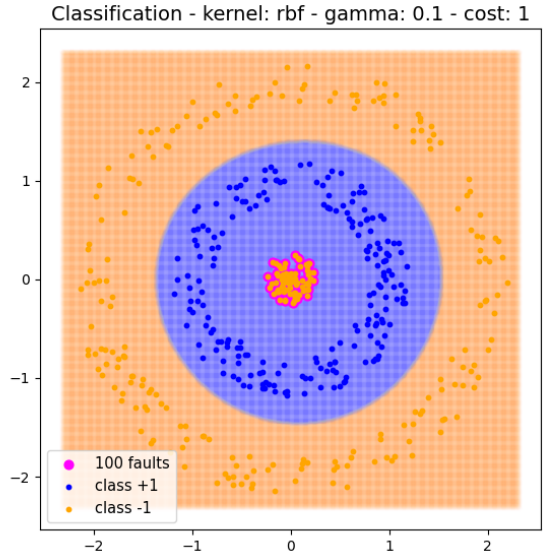

Margins - kernel: rbf - gamma: 0.5 - cost: 0.1
402 SV
class +1
class -1

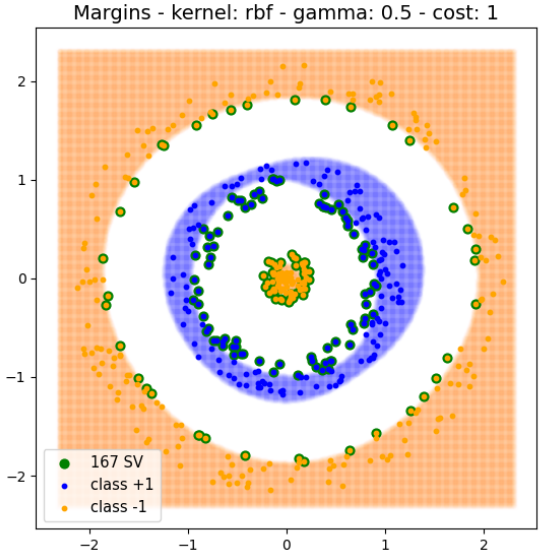

Classification - kernel: rbf - gamma: 0.5 - cost: 0.1
105 faults
class +1
class -1

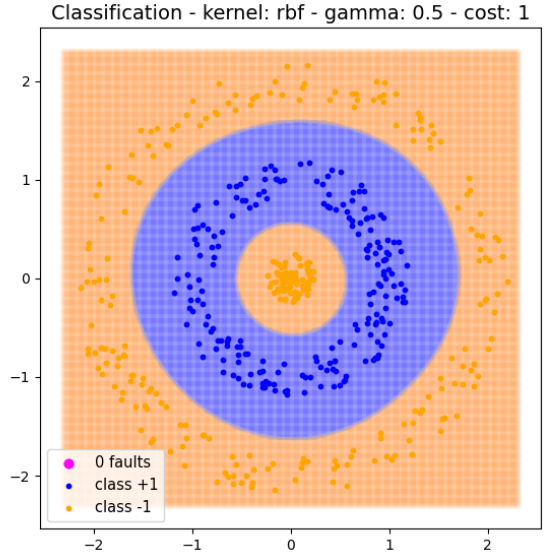

Margins - kernel: rbf - gamma: 0.9 - cost: 0.1
354 SV
class +1
class -1

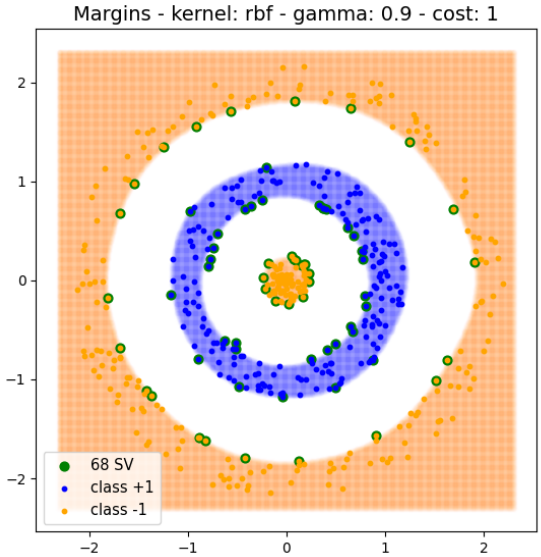

Classification - kernel: rbf - gamma: 0.9 - cost: 0.1
0 faults
class +1
class -1

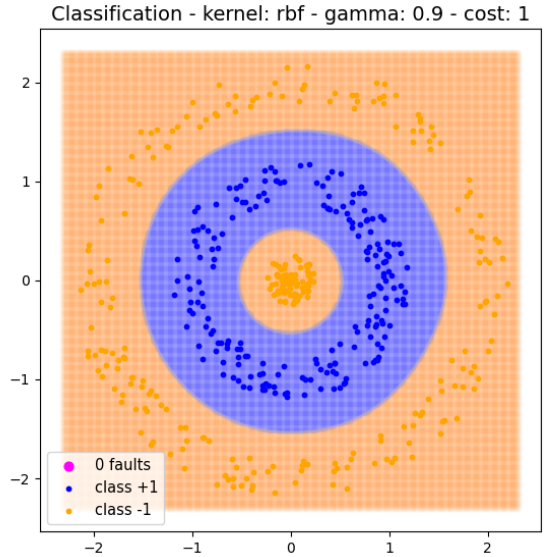

Margins - kernel: rbf - gamma: 0.1 - cost: 1
362 SV
class +1
class -1

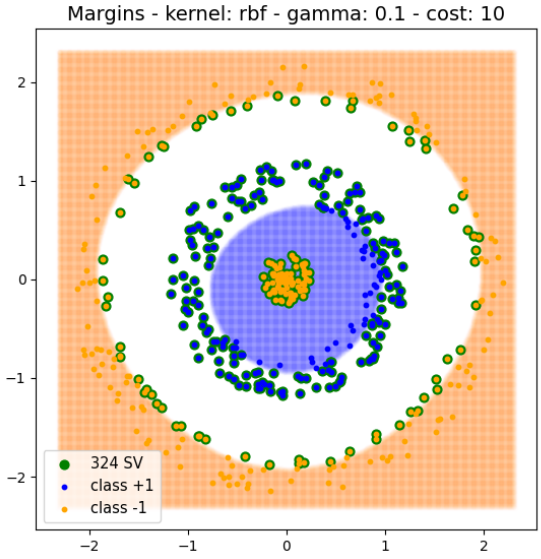

Classification - kernel: rbf - gamma: 0.1 - cost: 1
100 faults
class +1
class -1

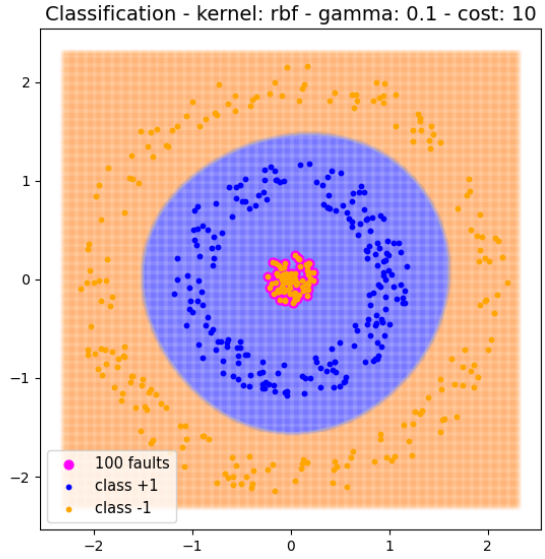

Margins - kernel: rbf - gamma: 0.5 - cost: 1
167 SV
class +1
class -1

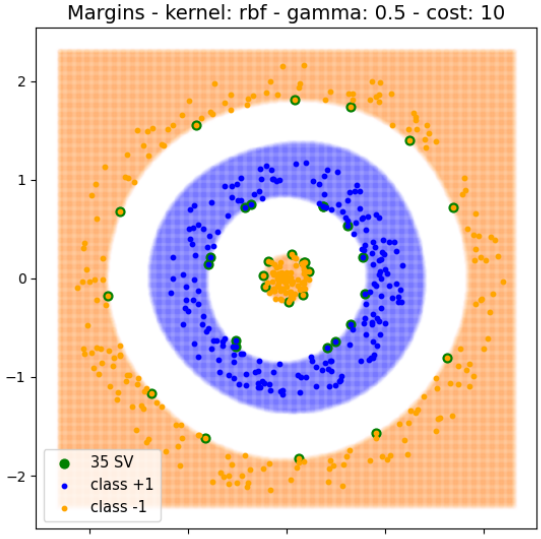

Classification - kernel: rbf - gamma: 0.5 - cost: 1
0 faults
class +1
class -1

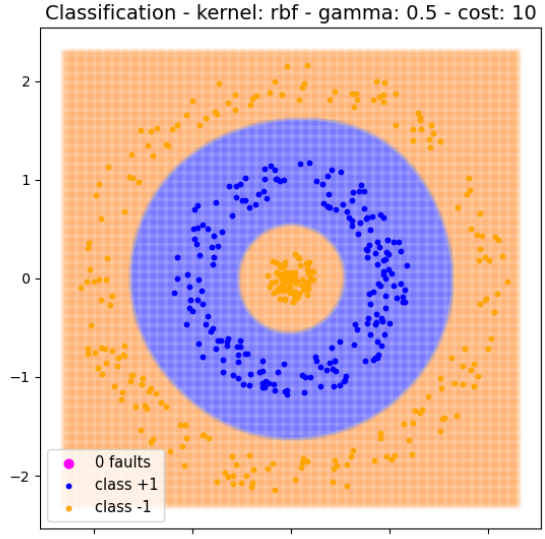

Margins - kernel: rbf - gamma: 0.9 - cost: 1
68 SV
class +1
class -1

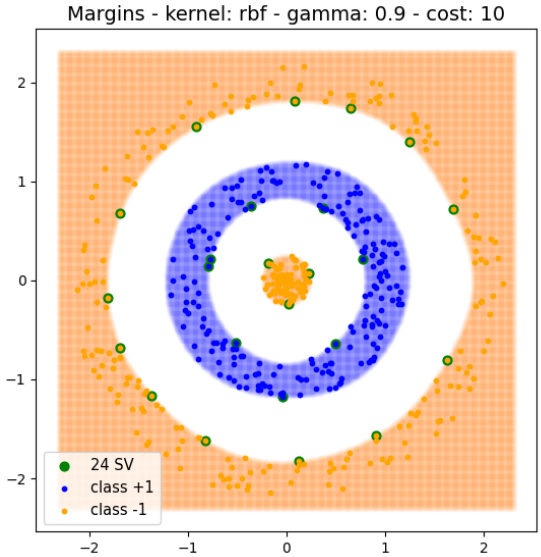

Classification - kernel: rbf - gamma: 0.9 - cost: 1
0 faults
class +1
class -1

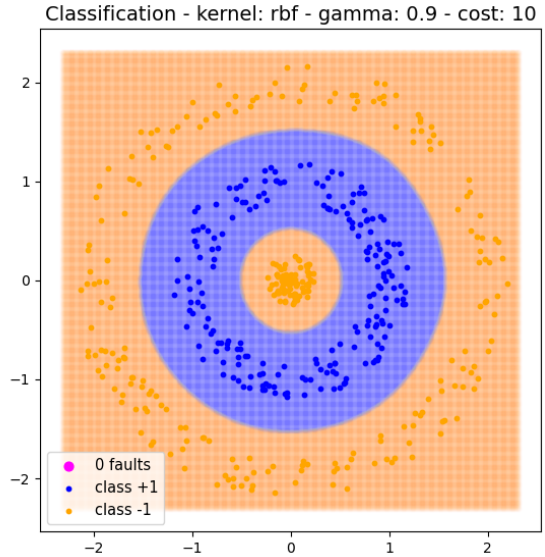

Margins - kernel: rbf - gamma: 0.1 - cost: 10
324 SV
class +1
class -1

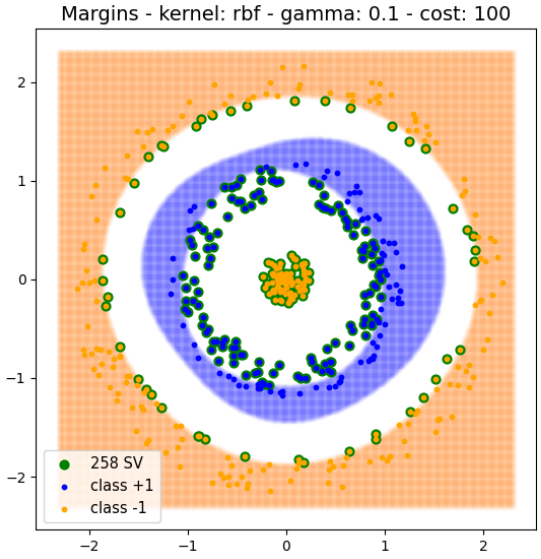

Classification - kernel: rbf - gamma: 0.1 - cost: 10
100 faults
class +1
class -1

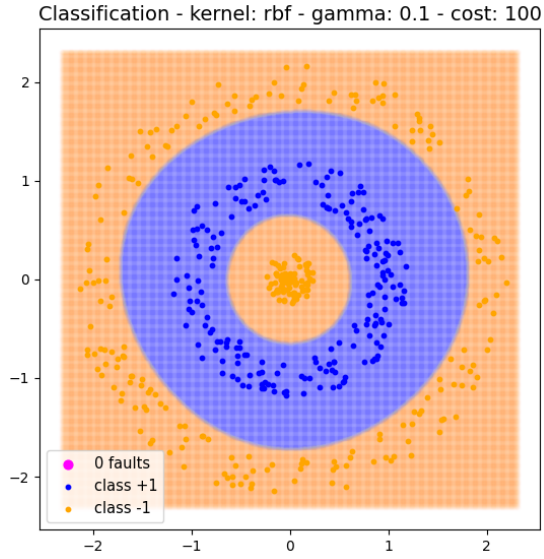

Margins - kernel: rbf - gamma: 0.5 - cost: 10
35 SV
class +1
class -1

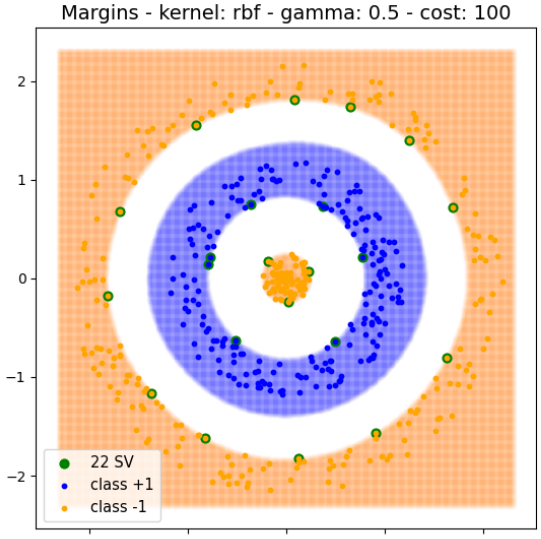

Classification - kernel: rbf - gamma: 0.5 - cost: 10
0 faults
class +1
class -1

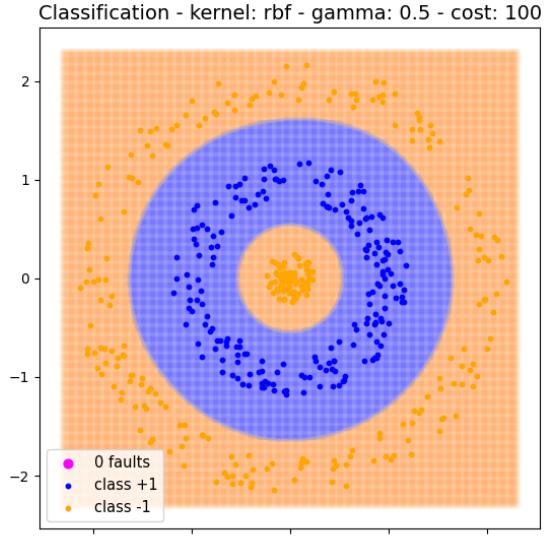

Margins - kernel: rbf - gamma: 0.9 - cost: 10
24 SV
class +1
class -1

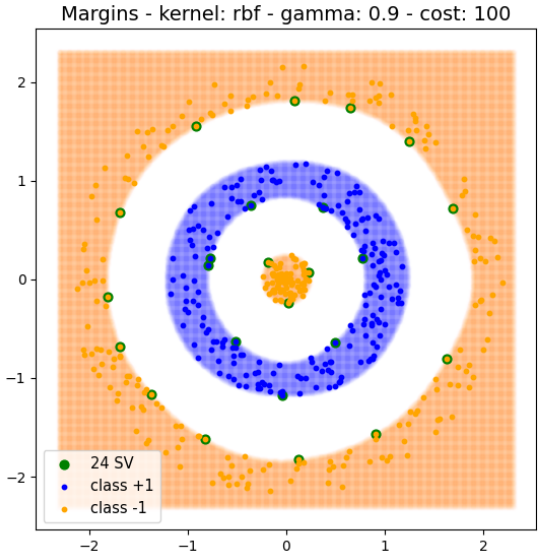

Classification - kernel: rbf - gamma: 0.9 - cost: 10
0 faults
class +1
class -1

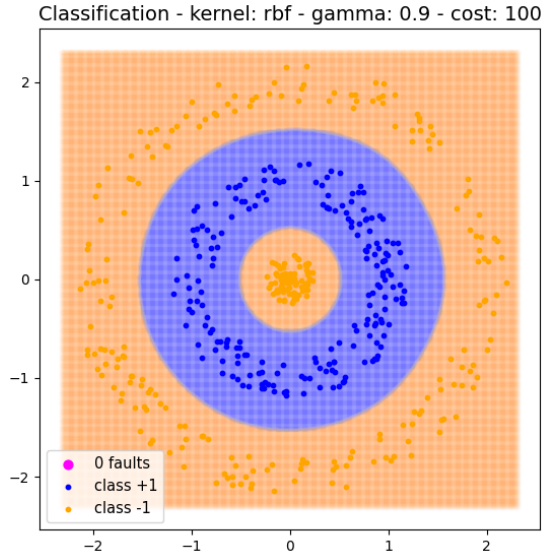

Margins - kernel: rbf - gamma: 0.1 - cost: 100
258 SV
class +1
class -1

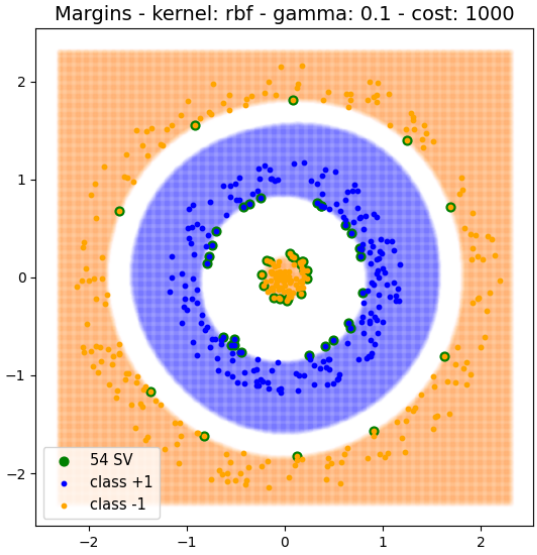

Classification - kernel: rbf - gamma: 0.1 - cost: 100
0 faults
class +1
class -1

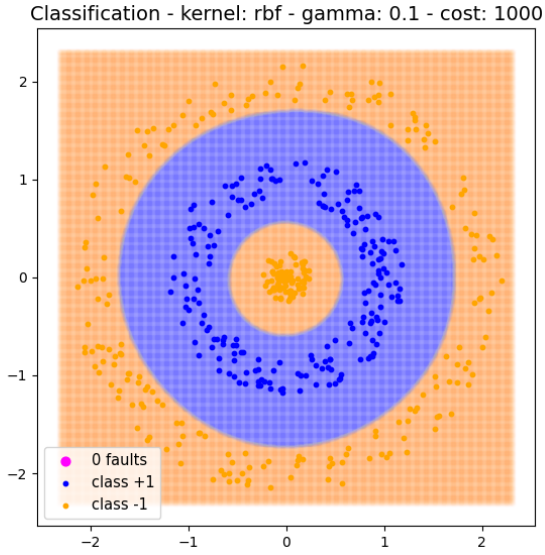

Margins - kernel: rbf - gamma: 0.5 - cost: 100
22 SV
class +1
class -1

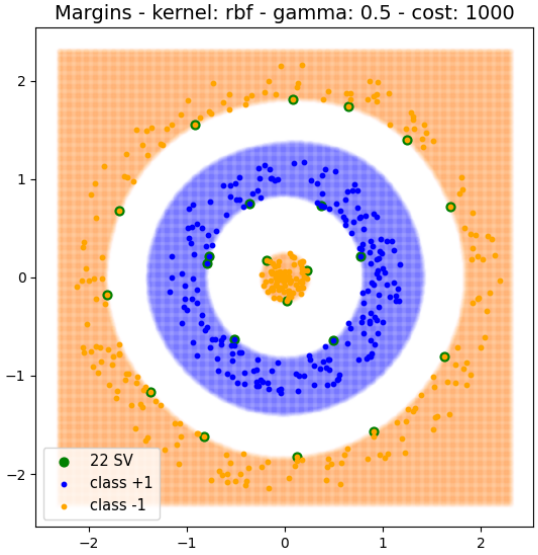

Classification - kernel: rbf - gamma: 0.5 - cost: 100
0 faults
class +1
class -1

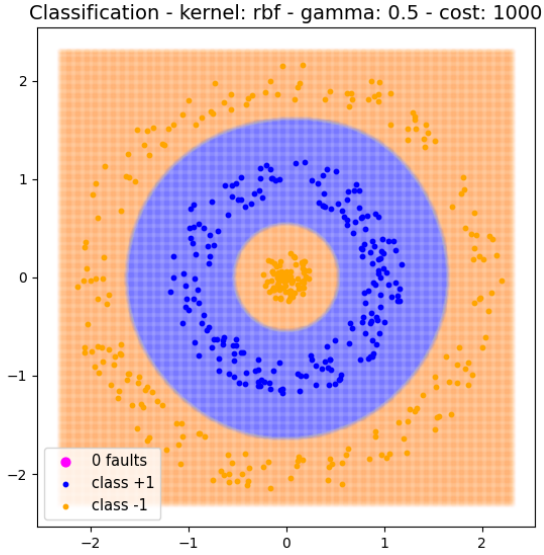

Margins - kernel: rbf - gamma: 0.9 - cost: 100
24 SV
class +1
class -1

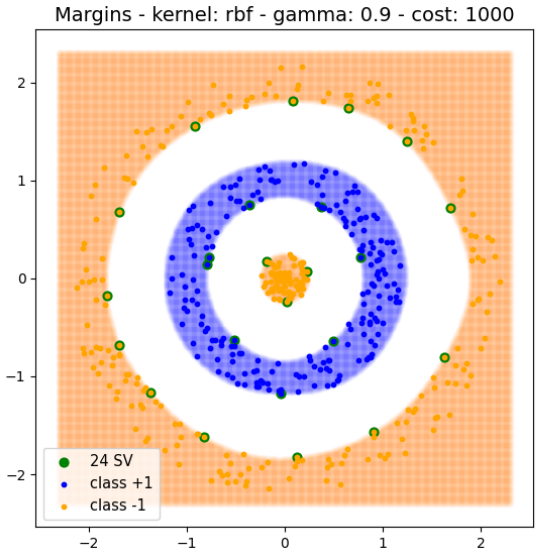

Classification - kernel: rbf - gamma: 0.9 - cost: 100
0 faults
class +1
class -1

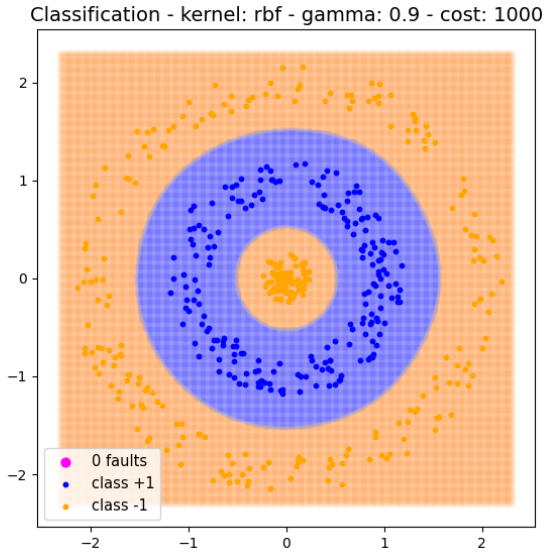












Question 3 (5 points):

What observations can you make from your plots? Tick the correct boxes:

a3_) For a fixed γ, the larger the cost C, the higher the number of support vectors.
b3_) For relatively large C and relatively large γ (say $C \geq 100$ and $\gamma > 0.5$), enlarging γ further doens't improve your performance significantly.

c3_) For fixed C, increasing γ (i.e. decreasing the width of the Gaussian) typically increases the model complexity of the SVM.
d3_) The higher the cost, the smaller the margin.

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question.
Note: Do not reuse these variable names. They are used for testing.

In [16]:
```
example_true = True
example_False = False

a3_ = False # for gamma=0.1: C=1 has fewer support vectors C=0.1
b3_ = True
c3_ = True
d3_ = True
```

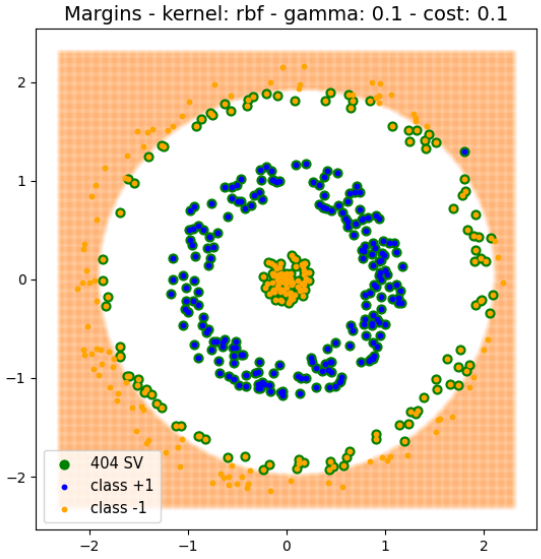
Code 3 (10 points):

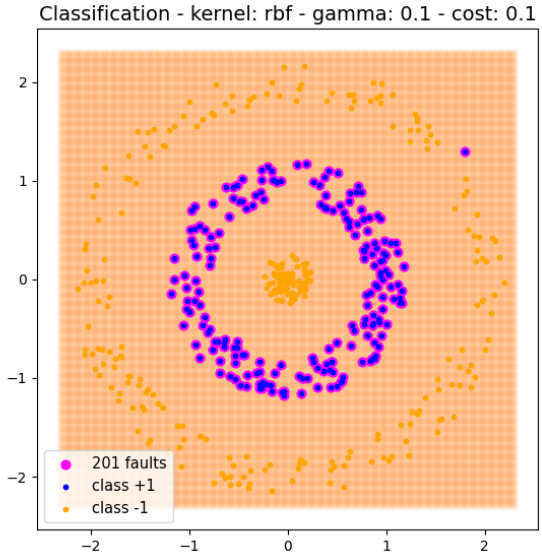
- Finally, we want to investigate, how the rbf kernel classifier reacts to outliers. To this end add a single (outlier) point $(1.8, 1.3)$ to your new data set `X_new`, `y_new` (i.e. the data with the additional disk with -1 labels) and label this point with $y = +1$. Plot the data set.
- Use an rbf kernel and play around with the parameter to explore the effects on the classification performance by again appropriately using the `plot_data` function. Try out small costs $C \sim 0.1$ and large costs $C \sim 1000$ and also iterate over different values of γ, ranging from 0.1 to 1. (Reuse iterative functions defined above). Report your observations in the subsequent question.

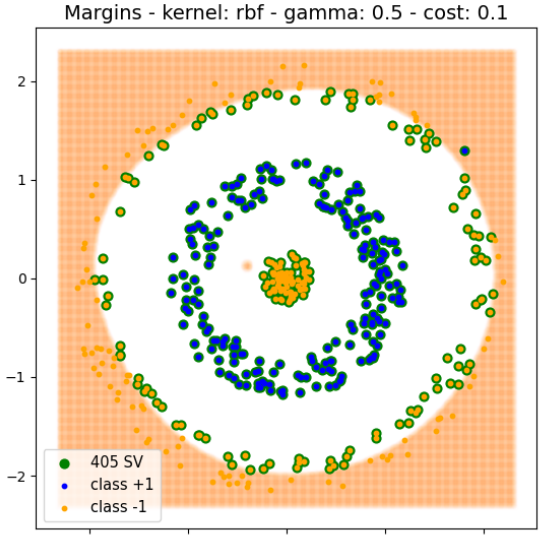
In [17]:
```
# update X_new and y_new by adding point as described in the task↓↓↓
def add_points(X_new: np.ndarray, y_new: np.ndarray) -> Tuple[np.ndarray, np.nda
    # your code here
    X_new_ = np.append(X_new, [[1.8, 1.3]], axis=0)
    y_new_ = np.append(y_new, 1)
    return X_new_, y_new_
```

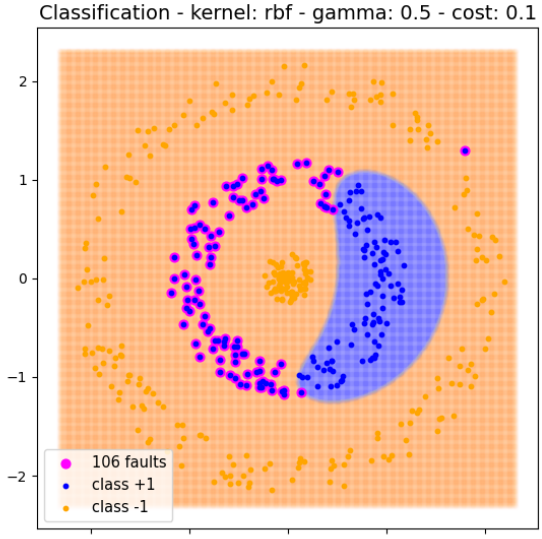
In [18]:
```
# nothing to do, just run the cell
X_new_plus1, y_new_plus1 = add_points(X_new, y_new)
```

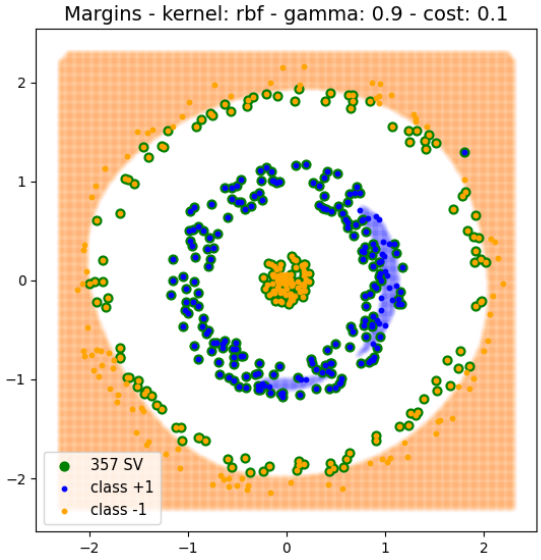
In [19]:
```
# This cell is just to discover the behaviour of SVM given the extra point, not

cost_values = range(-1,4)
gamma_values = [0.1,0.5,0.9]

for cost in cost_values:
    iter_Gamma(gamma_values,10**cost,X_new_plus1,y_new_plus1)
```

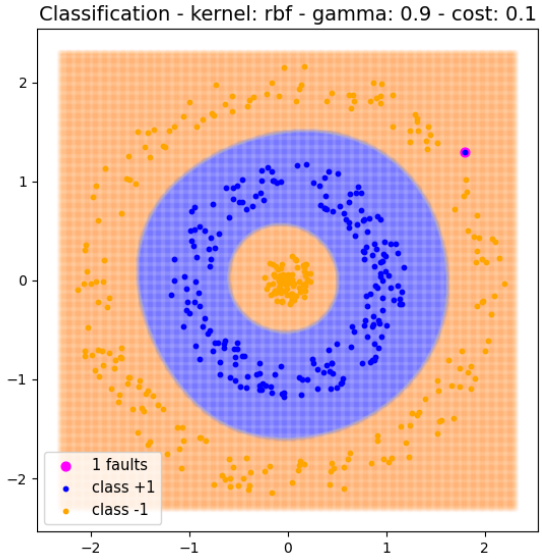

Margins - kernel: rbf - gamma: 0.1 - cost: 0.1
404 SV
class +1
class -1

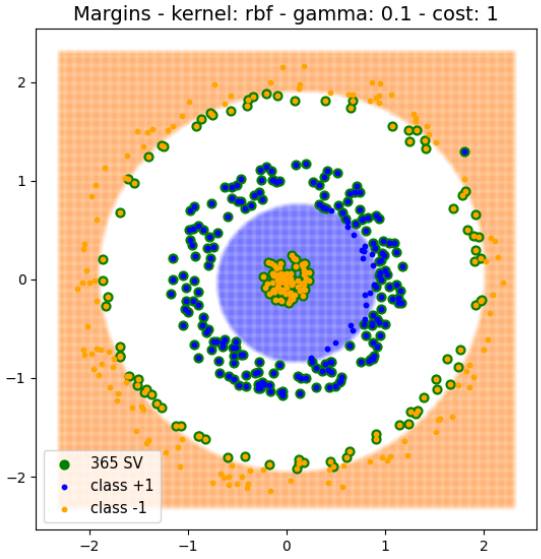

Classification - kernel: rbf - gamma: 0.1 - cost: 0.1
201 faults
class +1
class -1

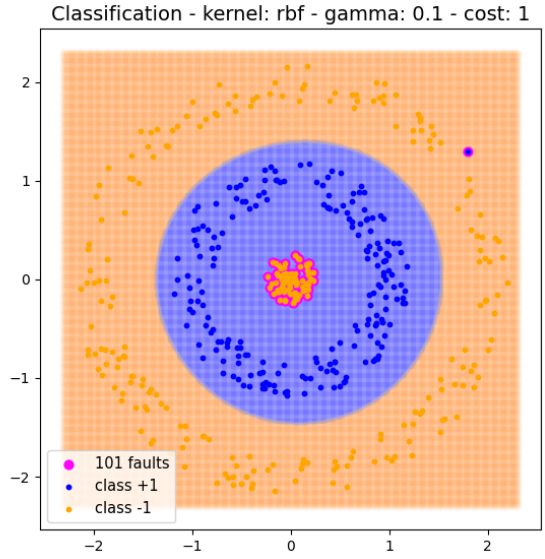

Margins - kernel: rbf - gamma: 0.5 - cost: 0.1
405 SV
class +1
class -1

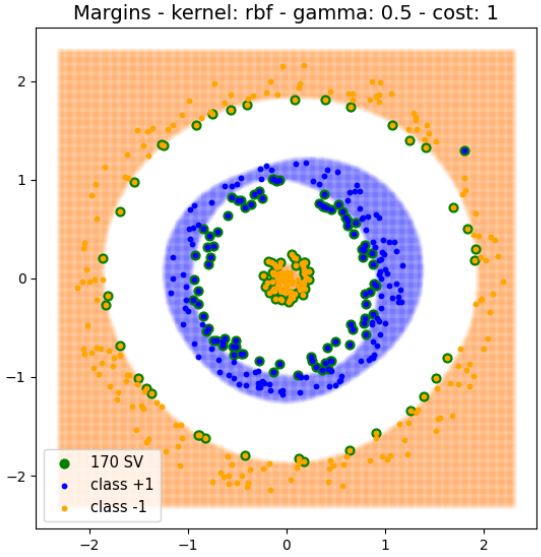

Classification - kernel: rbf - gamma: 0.5 - cost: 0.1
106 faults
class +1
class -1

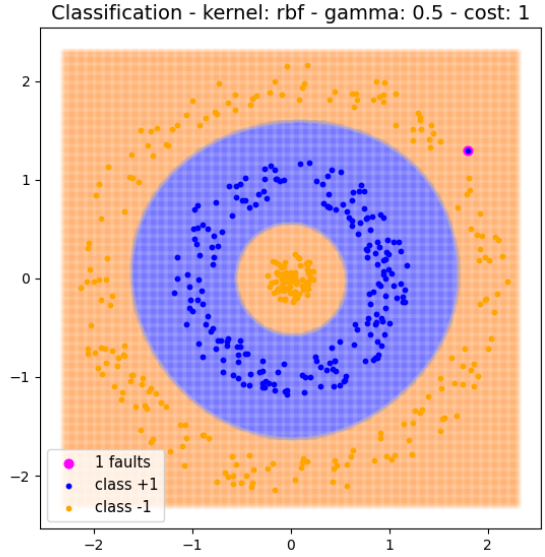

Margins - kernel: rbf - gamma: 0.9 - cost: 0.1
357 SV
class +1
class -1

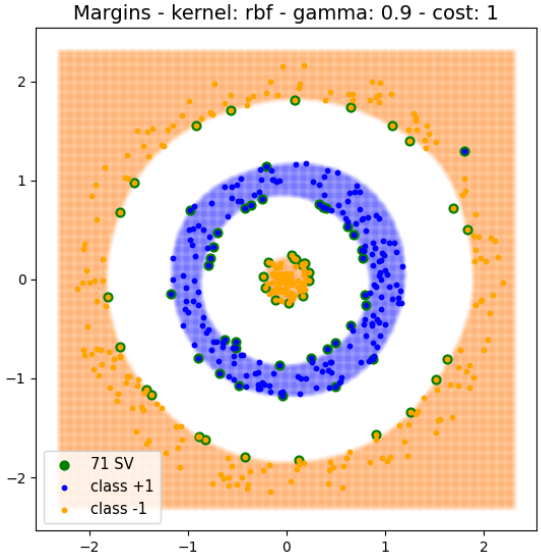

Classification - kernel: rbf - gamma: 0.9 - cost: 0.1
1 faults
class +1
class -1

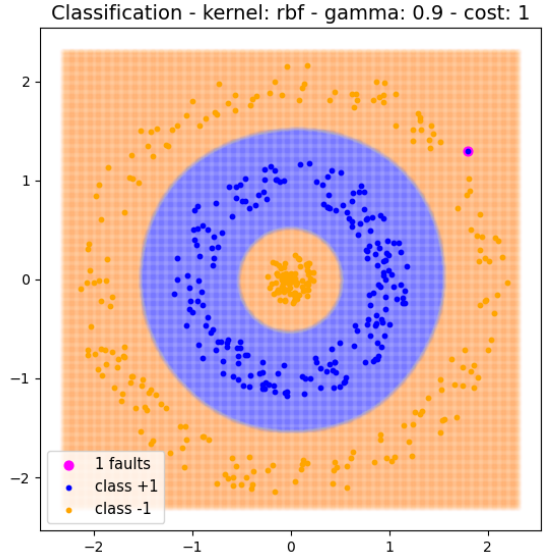

Margins - kernel: rbf - gamma: 0.1 - cost: 1
365 SV
class +1
class -1

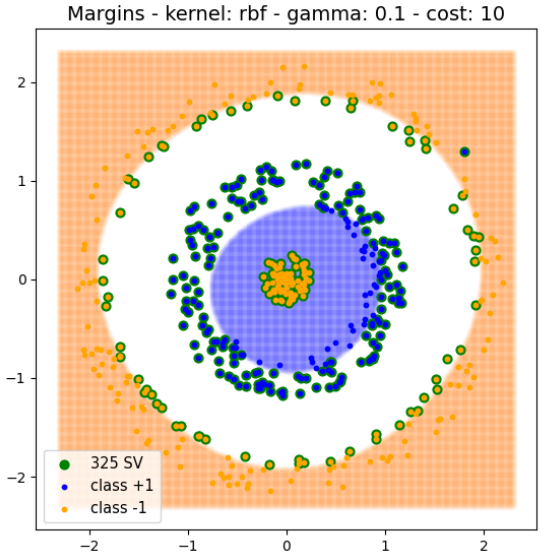

Classification - kernel: rbf - gamma: 0.1 - cost: 1
101 faults
class +1
class -1

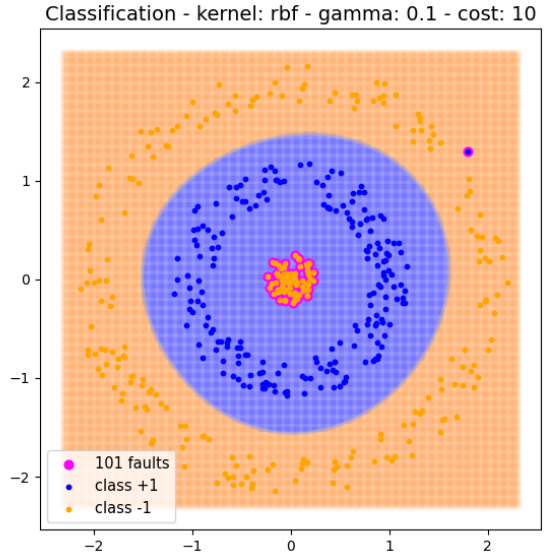

Margins - kernel: rbf - gamma: 0.5 - cost: 1
170 SV
class +1
class -1

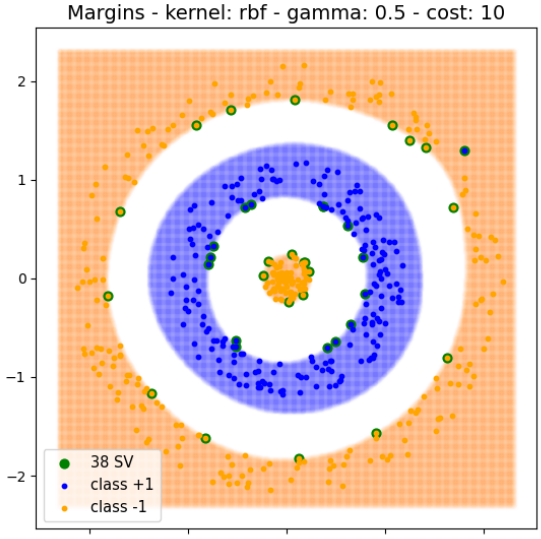

Classification - kernel: rbf - gamma: 0.5 - cost: 1
1 faults
class +1
class -1

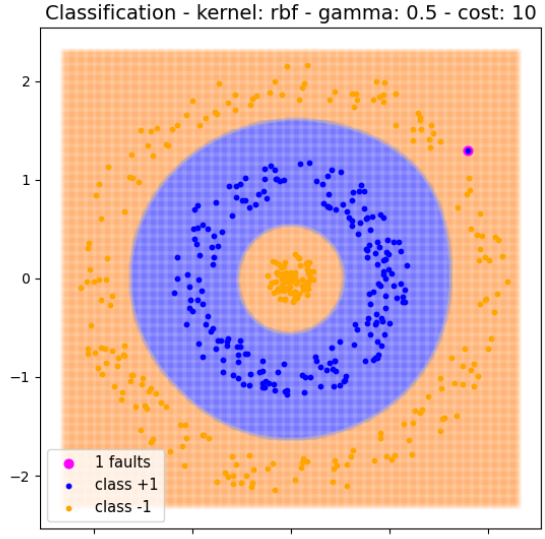

Margins - kernel: rbf - gamma: 0.9 - cost: 1
71 SV
class +1
class -1

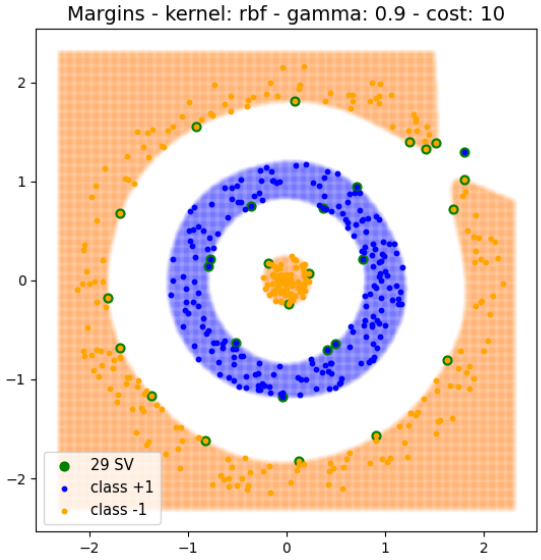

Classification - kernel: rbf - gamma: 0.9 - cost: 1
1 faults
class +1
class -1

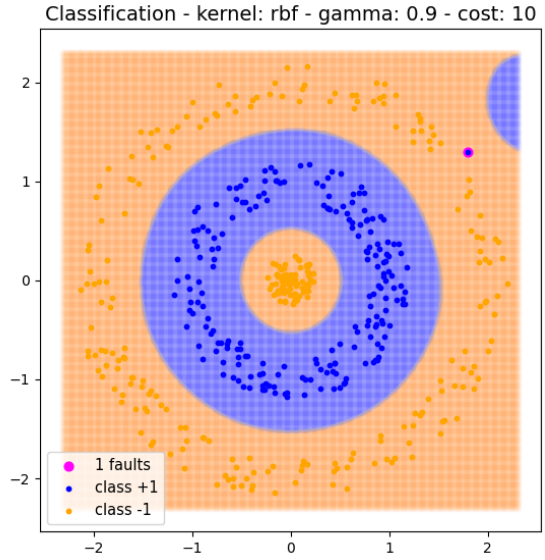

Margins - kernel: rbf - gamma: 0.1 - cost: 10
325 SV
class +1
class -1

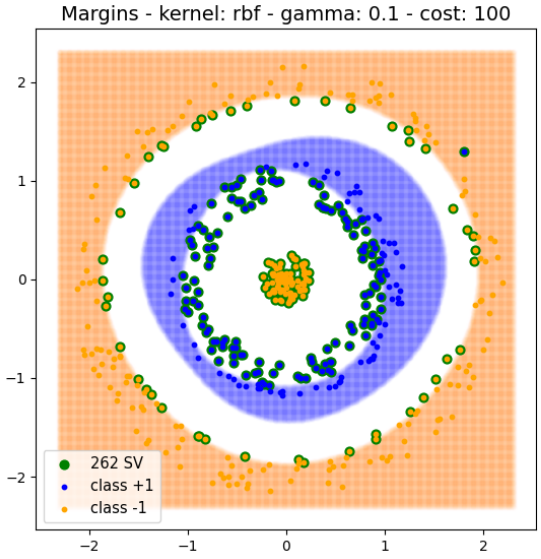

Classification - kernel: rbf - gamma: 0.1 - cost: 10
101 faults
class +1
class -1

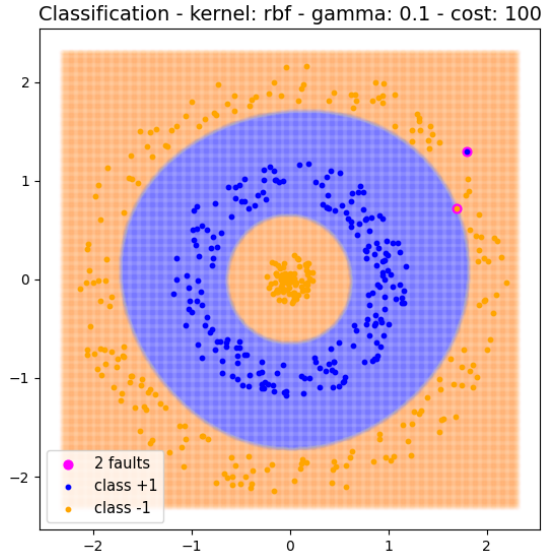

Margins - kernel: rbf - gamma: 0.5 - cost: 10
38 SV
class +1
class -1

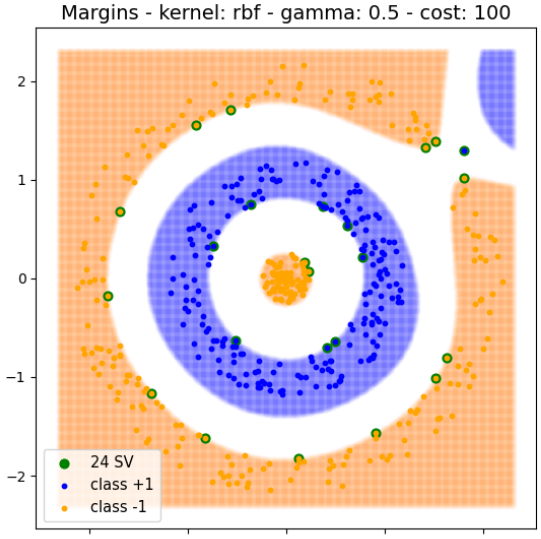

Classification - kernel: rbf - gamma: 0.5 - cost: 10
1 faults
class +1
class -1

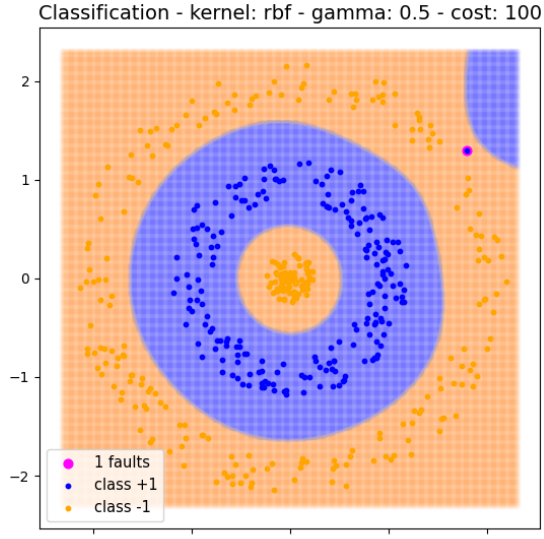

Margins - kernel: rbf - gamma: 0.9 - cost: 10
29 SV
class +1
class -1

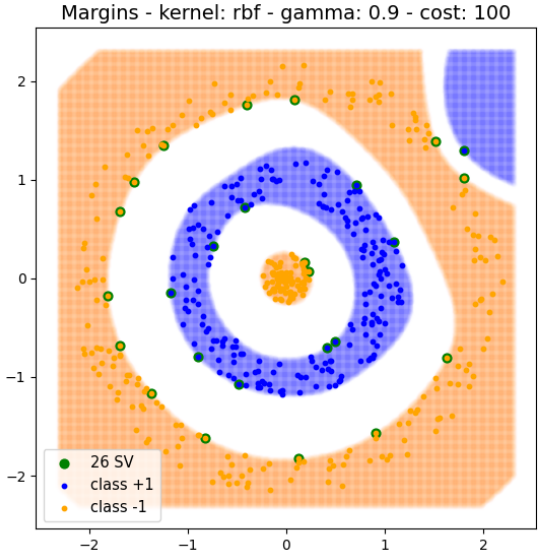

Classification - kernel: rbf - gamma: 0.9 - cost: 10
1 faults
class +1
class -1

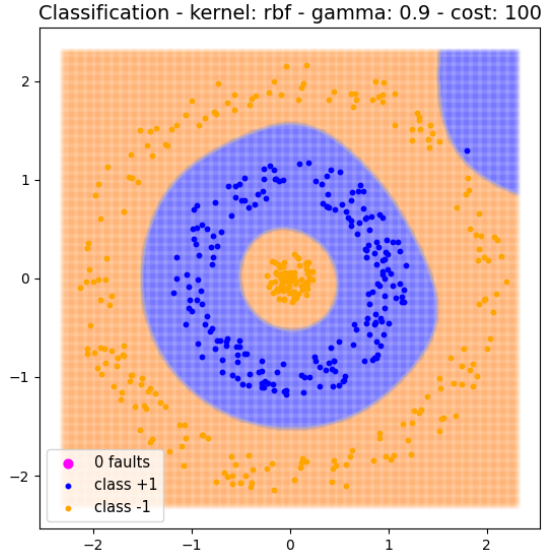

Margins - kernel: rbf - gamma: 0.1 - cost: 100
262 SV
class +1
class -1

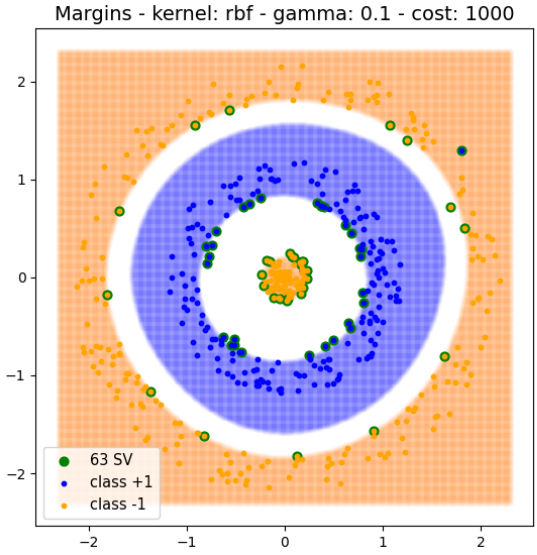

Classification - kernel: rbf - gamma: 0.1 - cost: 100
2 faults
class +1
class -1

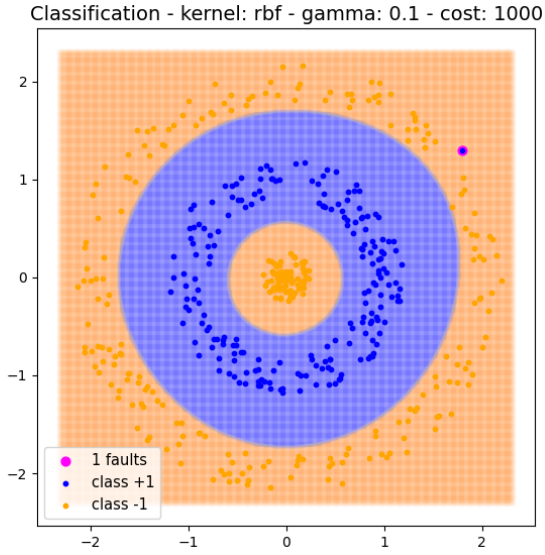

Margins - kernel: rbf - gamma: 0.5 - cost: 100
24 SV
class +1
class -1

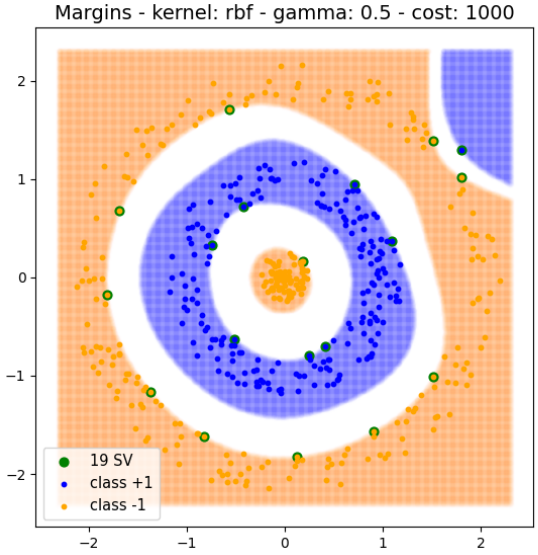

Classification - kernel: rbf - gamma: 0.5 - cost: 100
1 faults
class +1
class -1

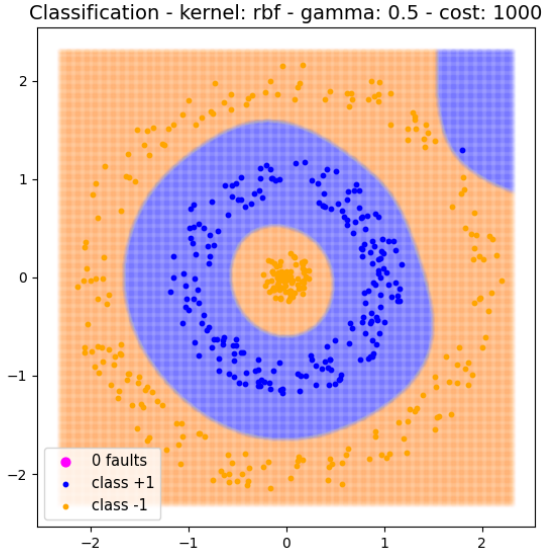

Margins - kernel: rbf - gamma: 0.9 - cost: 100
26 SV
class +1
class -1

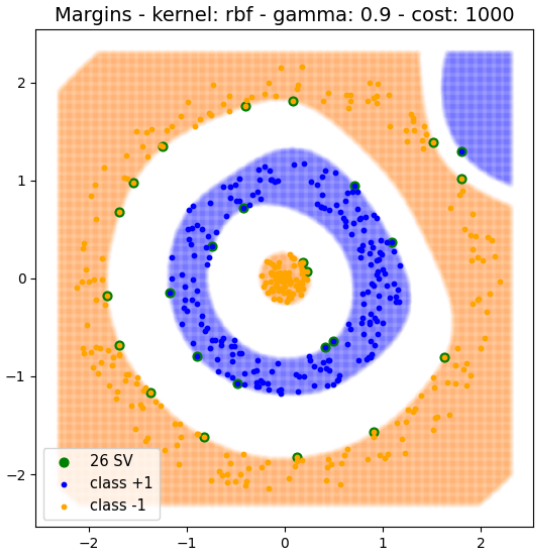

Classification - kernel: rbf - gamma: 0.9 - cost: 100
0 faults
class +1
class -1

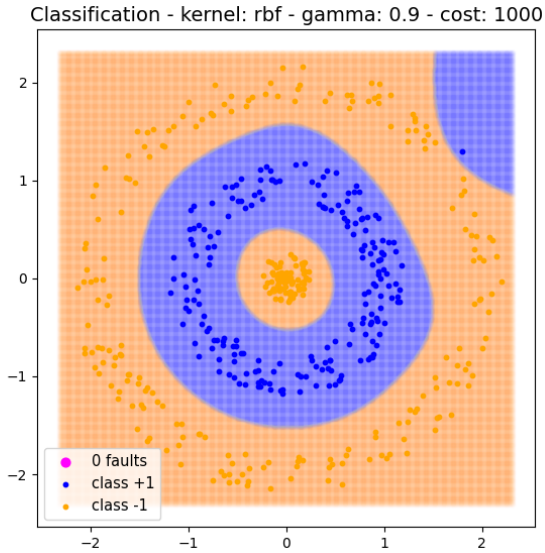












Question 4 (5 points):

What observations can you make from your plots? Tick the correct boxes:

a4_) For relatively low costs (e.g. $C \leq 1$) and an appropriately chosen γ (e.g. 0.9) the classifier correctly classifies all points except for the outlier.
b4_) For relatively high costs (e.g. $C \geq 100$) the classifier always, i.e. independent of γ,

shows a region of the positive class near the outlier.
c4_) Classifiers with high costs ($C \geq 100$ say) and high γ ($\gamma \geq 0.5$) are susceptible to overfitting.
d4_) For relatively high costs ($C \geq 100$) and small γ (0.1), the SVM correctly classifies except for the outlier.

To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question.
Note: Do not reuse these variable names. They are used for testing.

You are of course encouraged to further apply the given plot routine to further datasets and different hyperparameters to get further intuition!

In [20]:
```
example_true = True
example_False = False

a4_ = True
b4_ = False # gamme=0.1 and C=100 does not show the region of a possitive class
c4_ = True
d4_ = True
```

In [21]:
```
#here you can experiment, cell is not graded
```

Technical cells

The cells below are needed for efficient unittesting. Do not delete or change in order to receive proper evaluation.
Executability check might help you with datatypes, but does not guarantee your answers are 100% correct

In [22]:
```
import matplotlib.figure
import matplotlib.collections
import matplotlib.axes
def X_y_tolist(X,y):
    return X.tolist(), y.tolist()

def testoptions(options):
    for elem in options:
        if elem!=True and elem!=False and elem!=None:
            raise ValueError(f"Check answers for questions again")
    print("Test questions answers are ok")
```

In [23]:
```
try:
    X_y_tolist(X,y)
except:
    raise ValueError("Check your X and y variables")
try:
    X_y_tolist(X_new,y_new)
except:
    raise ValueError("Check your X_new and y_new variables")
testoptions(np.array([a1_,b1_,c1_,d1_,e1_,f1_,a2_,b2_,c2_,a3_,b3_,c3_,d3_,a4_,b4
```

```
Test questions answers are ok
```