Assignment 3: Constrained Optimization, SVMs

Copyrighting and Fare Use

This material, no matter whether in printed or electronic form, may be used for personal and non-commercial educational use only. Any reproduction of this material, no matter whether as a whole or in parts, no matter whether in printed or in electronic form, requires explicit prior acceptance of the authors.

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 problem:

$$\begin{aligned} \text{Minimize} \quad & 4(w_1^4 + w_2^4) \\ \text{subject to} \quad & 4 + w_1 - w_2 \leq 0 \end{aligned}$$

We try to solve it via two ways, thus do 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.

For your calculation use the proposed notation.

Calculation 1 (25 points):

1. Lagrangian function

$$f(\mathbf{w_1}, \mathbf{w_2}) = 4 \cdot (\mathbf{w_1}^4 + \mathbf{w_2}^4)$$
$$h(\mathbf{w_1}, \mathbf{w_2}) = 4 + \mathbf{w_1} - \mathbf{w_2}$$

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

2. Derivatives of Lagrangian

$$\frac{\partial L}{\partial \mathbf{w_1}}(\mathbf{w_1}, \mathbf{w_2}, L) = 16 \cdot \mathbf{w_1}^3 + \alpha$$

$$\frac{\partial L}{\partial \mathbf{w_2}}(\mathbf{w_1}, \mathbf{w_2}, L) = 16 \cdot \mathbf{w_2}^3 - \alpha$$

$$16 \cdot \mathbf{w_1}^3 + \alpha = 0 \implies \mathbf{w_1}^3 = -\frac{\alpha}{16} \implies \mathbf{w_1}^* = \sqrt[3]{-\frac{\alpha}{16}}$$

$$16 \cdot \mathbf{w_2}^3 - \alpha = 0 \implies \mathbf{w_2}^3 = \frac{\alpha}{16} \implies \mathbf{w_2}^* = \sqrt[3]{\frac{\alpha}{16}}$$

3. Solving by using KKT conditions
   from the first KKT condition:

$$\alpha \cdot (\mathbf{w_1}^*, \mathbf{w_2}^*) = 0$$
$$\text{firstly, if } \mathrm{h}(\mathbf{w_1}^*, \mathbf{w_2}^*) < 0 \implies \alpha = 0$$
$$4 - 2 \cdot \sqrt[3]{\frac{\alpha}{16}} < 0$$
$$\Leftrightarrow 4 < 2 \cdot \sqrt[3]{\frac{\alpha}{16}}$$
$$\Leftrightarrow 2 < \sqrt[3]{\frac{\alpha}{16}}$$
$$\Leftrightarrow \frac{\alpha}{16} > 8$$
$$\implies \alpha > 128$$
$$\alpha = 0 \implies \mathbf{w_1}^* = 0, \mathbf{w_2}^* = 0 \implies h(\mathbf{w_1}^*, \mathbf{w_2}^*) = 4 + 0 - 0 = 4 \leq 0(\mathit{false} \implies \mathit{contradiction})$$

$$\text{secondly, if } \mathrm{h}(\mathbf{w_1}^*, \mathbf{w_2}^*)) = 0$$
$$\Leftrightarrow 4 - 2 \cdot \sqrt[3]{\frac{\alpha}{16}} = 0$$
$$\implies \alpha = 128$$
$$\implies \mathbf{w_1}^* = \sqrt[3]{-\frac{128}{16}} = \sqrt[3]{-8} = -2$$
$$\mathbf{w_2}^* = \sqrt[3]{\frac{128}{16}} = \sqrt[3]{8} = 2$$
$$\implies f(\mathbf{w_1}^*, \mathbf{w_2}^*) = 4 \cdot ((-2)^4 + 2^4) = 4 \cdot 32 = 128$$
$$\alpha = 128 \implies \mathbf{w_1}^* = -2, \mathbf{w_2}^* = 2 \implies h(\mathbf{w_1}^*, \mathbf{w_2}^*) = 0 \leq 0(\mathit{true})$$

4. Solving without KKT conditions

$$L(\alpha) = L(\mathbf{w_1^*}, \mathbf{w_2^*}, \alpha) = 4((\sqrt[3]{-\frac{\alpha}{16}})^4 + (\sqrt[3]{\frac{\alpha}{16}})^4) + \alpha \cdot (4 + \sqrt[3]{-\frac{\alpha}{16}} - \sqrt[3]{\frac{\alpha}{16}})$$

$$= 4((\sqrt[3]{-\frac{\alpha}{16}})^4 + (\sqrt[3]{\frac{\alpha}{16}})^4) + \alpha \cdot (4 - \sqrt[3]{\frac{\alpha}{16}} - \sqrt[3]{\frac{\alpha}{16}})$$

$$= 4((\sqrt[3]{-\frac{\alpha}{16}})^4 + (\sqrt[3]{\frac{\alpha}{16}})^4) + \alpha \cdot (4 - 2 \cdot \sqrt[3]{\frac{\alpha}{16}})$$

$$(L(\mathbf{w_1^*}, \mathbf{w_2^*}, \alpha))' = (4((\sqrt[3]{-\frac{\alpha}{16}})^4 + (\sqrt[3]{\frac{\alpha}{16}})^4) + \alpha \cdot (4 - 2 \cdot \sqrt[3]{\frac{\alpha}{16}}))' =$$

$$= (\frac{\alpha^{\frac{4}{3}}}{2^{\frac{7}{3}}} + \alpha(4 - \frac{\sqrt[3]{\alpha}}{\sqrt[3]{2}}))'$$

$$= \frac{1}{2^{\frac{7}{3}}} \cdot \frac{4}{3} \cdot \alpha^{\frac{1}{3}} + (4 - \frac{\sqrt[3]{\alpha}}{\sqrt[3]{2}})' + (4 - \frac{\sqrt[3]{\alpha}}{\sqrt[3]{2}})$$

$$= \frac{\sqrt[3]{\alpha}}{3\sqrt[3]{2}} + \alpha \cdot (-\frac{1}{\sqrt[3]{2}} \cdot \frac{1}{3} \cdot \alpha^{\frac{1}{3}-1}) + (4 - \frac{\sqrt[3]{\alpha}}{\sqrt[3]{2}})$$

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

$$= -\frac{\sqrt[3]{2^2}\sqrt[3]{\alpha}}{3} - \frac{\sqrt[3]{2^2\alpha}}{6} + 4$$

$$= -\frac{\sqrt[3]{4}\alpha}{3} - \frac{\sqrt[3]{4}\alpha}{6} + 4$$

$$= -\frac{1}{2} \cdot \sqrt[3]{4\alpha} + 4 \implies -\frac{1}{2} \cdot \sqrt[3]{4\alpha} + 4 = 0$$

$$\Leftrightarrow -\frac{1}{2} \cdot \sqrt[3]{4\alpha} = -4$$

$$\Leftrightarrow \sqrt[3]{4\alpha} = 8$$

$$\Leftrightarrow 4 \cdot \alpha = 512 \implies \alpha = 128 \implies \mathbf{w_1^*} = \sqrt[3]{-\frac{128}{16}} = \sqrt[3]{-8} = -2$$

$$\mathbf{w_2^*} = \sqrt[3]{\frac{128}{16}} = \sqrt[3]{8} = 2$$

$$\alpha = 128 \implies \mathbf{w_1^*} = -2, \mathbf{w_2^*} = 2 \implies h(\mathbf{w_1^*}, \mathbf{w_2^*}) = 4 - 2 - 2 = 0 \leq 0(true)$$

$$\implies f(\mathbf{w_1^*}, \mathbf{w_2^*}) = 4 \cdot ((-2)^4 + 2^4) = 4 \cdot 32 = 128$$

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:

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

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.
- 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.
- Simplify the dual problem so that it only depends on l Lagrange variables. Argue why this can be done.

Please provide reasoning and explanations in full sentences. Grading of the task will heavily depend on it.

1. Why can the KKT-Theorem be applied?
   The KKT-Theorem can be applied because the task is about an optimiziation problem concerning non-linear separability.

2. Lagrangian function

$$f(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i^2$$

$$y_i\Big(\mathbf{w}\cdot\mathbf{x}_i + b\Big) + \xi_i - 1 \geq 0$$

$$L(\mathbf{w}, b, \xi_1, \ldots, \xi_l, \alpha_1, \ldots, \alpha_l, \lambda_1, \ldots, \lambda_l) = \frac{1}{2}\cdot\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i^2 - \sum_{i=1}^{l}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1 + \xi_i) - \sum_{i=1}^{l}\lambda_i\xi_i^2$$

3. Derivatives of Lagrangian

$$\frac{\partial L}{\partial \mathbf{w}}L(\mathbf{w}, b, \xi_1, \ldots, \xi_l, \alpha_1, \ldots, \alpha_l, \lambda_1, \ldots, \lambda_l) = \|w\| - \sum_{i=1}^{l}\alpha_i\cdot y_i\cdot\mathbf{w}$$

$$\frac{\partial L}{\partial \mathbf{w}}L(\mathbf{w}, b, \xi_1, \ldots, \xi_l, \alpha_1, \ldots, \alpha_l, \lambda_1, \ldots, \lambda_l) = \sum_{i=1}^{l}\alpha_i\cdot y_i$$

$$\frac{\partial L}{\partial \mathbf{w}}L(\mathbf{w}, b, \xi_1, \ldots, \xi_l, \alpha_1, \ldots, \alpha_l, \lambda_1, \ldots, \lambda_l) = 2\cdot C\xi_i - \alpha_i - 2\cdot\lambda_i$$

$$\implies \|w\| = \sum_{i=1}^{l}\alpha_i\cdot y_i\cdot\mathbf{w}$$

$$\implies \sum_{i=1}^{l}\alpha_i\cdot y_i = 0$$

$$\implies \xi_i = \frac{\alpha_i + 2\lambda_i}{2C}$$

4. The dual function and the dual problem.

$$L_{dual} = \frac{1}{2}\cdot\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i^2 - \sum_{i=1}^{l}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1 + \xi_i) - \sum_{i=1}^{l}\lambda_i\xi_i^2$$

$$= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\cdot\alpha_j\cdot y_i\cdot y_j\cdot\mathbf{x_i}\cdot\mathbf{x_j} - \sum_{i=1}^{l}\alpha_i\cdot\xi_i - \sum_{i=1}^{l}\lambda_i\cdot\xi_i + C\cdot\sum_{i=1}^{l}\Big(\frac{\lambda_i + 2\alpha_i}{2C}\Big)^2$$

$$= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\cdot\alpha_j\cdot y_i\cdot y_j\cdot\mathbf{x_i}\cdot\mathbf{x_j} - \sum_{i=1}^{l}\alpha_i\cdot\frac{\alpha_i + 2\lambda_i}{2C} - \sum_{i=1}^{l}\lambda_i\cdot\frac{\alpha_i + 2\lambda_i}{2C} + \frac{C}{4C^2}\sum_{i=1}^{l}\big(\alpha_i^2 + 4\alpha_i\lambda_i + 4\lambda_i^2\big)$$

$$= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\cdot\alpha_j\cdot y_i\cdot y_j\cdot\mathbf{x_i}\cdot\mathbf{x_j} - \frac{1}{2C}\sum_{i=1}^{l}(\alpha_i^2 + 2\cdot\alpha_i\cdot\lambda_i) - \frac{1}{2C}\sum_{i=1}^{l}\big(\lambda_i\cdot\alpha_i + 2\lambda_i^2\big) + \frac{1}{4C}\sum_{i=1}^{l}\big(\alpha_i^2 + 4\alpha_i\lambda_i + 4\lambda_i^2\big)$$

$$= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\cdot\alpha_j\cdot y_i\cdot y_j\cdot\mathbf{x_i}\cdot\mathbf{x_j} - \frac{1}{2C}\Big(\sum_{i=1}^{l}(\alpha_i^2 + 2\cdot\alpha_i\cdot\lambda_i) + \sum_{i=1}^{l}(\lambda_i\cdot\alpha_i + 2\lambda_i^2)\Big) + \frac{1}{4C}\sum_{i=1}^{l}\big(\alpha_i^2 + 4\alpha_i\lambda_i + 4\lambda_i^2\big)$$

5. Simplify

$$L_{dual} = \frac{1}{2}\cdot\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i^2 - \sum_{i=1}^{l}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1 + \xi_i) - \sum_{i=1}^{l}\lambda_i\xi_i^2$$

The $\xi_i \geq 0$ can be drooped as $\sum_{i=1}^{l}\xi_i^2$ is always positive $\implies$ any negative values of the slack will be replaced by 0 during optimization. The simplification is the following one:

$$L_{dual} = \frac{1}{2}\cdot\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i^2 - \sum_{i=1}^{l}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1 + \xi_i)$$

Also:

$$\frac{\partial L}{\partial \mathbf{w}} L(\mathbf{w}, b, \xi_1, \ldots, \xi_l, \alpha_1, \ldots, \alpha_l, \lambda_1, \ldots, \lambda_l) = 2 \cdot C\xi_i - \alpha_i$$

$$\implies \xi_i = \frac{\alpha_i}{2C}$$

$$L_{dual} = \frac{1}{2} \cdot \|\mathbf{w}\|^2 + C\sum_{i=1}^{l} \xi_i^2 - \sum_{i=1}^{l} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i - b) - 1 + \xi_i)$$

$$= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \mathbf{x_i} \cdot \mathbf{x_j} - \sum_{i=1}^{l} \alpha_i \cdot \xi_i$$

$$= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \mathbf{x_i} \cdot \mathbf{x_j} + C \cdot \sum_{i=1}^{l} (\frac{\alpha_i}{2C})^2 - \sum_{i=1}^{l} \alpha_i \cdot \frac{\alpha_i}{2C}$$

$$= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \mathbf{x_i} \cdot \mathbf{x_j} + C \cdot \sum_{i=1}^{l} \frac{\alpha_i^2}{4C^2} - \sum_{i=1}^{l} \frac{\alpha_i^2}{2C}$$

$$= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \mathbf{x_i} \cdot \mathbf{x_j} + \frac{1}{4C} \cdot \sum_{i=1}^{l} \alpha_i^2 - \frac{1}{2C} \cdot \sum_{i=1}^{l} \alpha_i^2$$

$$= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \mathbf{x_i} \cdot \mathbf{x_j} - \frac{1}{4C} \cdot \sum_{i=1}^{l} \alpha_i^2$$

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 a function `plot_data`, that 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):

- As a first task plot the dataset using the provided functions.
- 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 from 1 to 5. Function `iter_Degree` must return a list of models' parameters for each model you initiated and fitted on our data. Model parameters is a dictionary, consult the sklearn docs .
  Apply `plot_data` to each hyperparameter setting.

In [1]:
```
#Nothing to do here
import numpy as np
import matplotlib.pyplot as plt
import random
import csv

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
"""Function allows ro load datapoint from csv
@returns: tuple (X,y)"""
def load_data(id_data=1):
    if id_data ==1:
        Z = np.genfromtxt('radial_data.csv', delimiter=',')
        return Z[:,:-1], Z[:,-1]

"""Function creates space/grid. Mostly used for plotting"""
def get_meshgrid(X,resolution):
    s = np.max(np.abs(X))*1.05
    ls = np.linspace(-s, s, resolution)
```

```python
    X1,X2 = np.meshgrid(ls, ls, sparse=False)
    return np.c_[X1.ravel(), X2.ravel()]

"""Plotting your data
@param model already trained SVM model, is None if you want to plot data only
all other parameters must be intuitively clear for you"""
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']):

    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=0.01, s=10)
                ax.scatter(grid[mask_neg,0], grid[mask_neg,1], c=color[1], alpha=0.01, s=10)
                ax.scatter(X[mask_sv,0], X[mask_sv,1], c='g',label= str(np.sum(model.n_support_)) +' SV',s=40,ma
            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=0.01, s=10)
                ax.scatter(grid[mask_neg,0], grid[mask_neg,1], c=color[1], alpha=0.01, s=10)
                classification = model.predict(X)
                mask_wrong = classification != y
                ax.scatter(X[mask_wrong,0], X[mask_wrong,1], c='magenta',label=str(np.sum(mask_wrong)) + ' faul
            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()
    return
```

In [3]:

```python
#load data
#leave as it is
```

```
X,y = load_data(1)
```

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

In [5]:
```
"""
Function iter_Degree fits SVM using defined range of degrees and plots every variation.
@ degree_range: range of integer degrees
@ returns list of dictionaries, lenght of list = length of degree_range
"""
def iter_Degree(degree_range, X, y):
    list_of_models = []
    #your code ↓↓↓
    #code ends here

    for degree in degree_range:
        svclassifier=svm.SVC(C=10,kernel='poly',degree=degree)
        svclassifier.fit(X,y)
        list_of_models.append(svclassifier)

    return list_of_models

##########################################################################################
#using the written function

degree_range = list(range(1,6)) #change this variable
models_list=iter_Degree(degree_range,X,y)
for model in models_list:
    print(model)
for model in models_list:
    plot_data(X,y,model)
```

```
SVC(C=10, degree=1, kernel='poly')
SVC(C=10, degree=2, kernel='poly')
SVC(C=10, kernel='poly')
SVC(C=10, degree=4, kernel='poly')
SVC(C=10, degree=5, kernel='poly')
```

Margins - kernel: poly - degree: 1 - cost:10
372 SV
class +1
class -1
Classification - kernel: poly - degree: 1 - cost:10
172 faults
class +1
class -1
Margins - kernel: poly - degree: 2 - cost:10
4 SV
class +1
class -1
Classification - kernel: poly - degree: 2 - cost:10
0 faults
class +1
class -1

Margins - kernel: poly - degree: 3 - cost:10
365 SV
class +1
class -1
Classification - kernel: poly - degree: 3 - cost:10
98 faults
class +1
class -1
Margins - kernel: poly - degree: 4 - cost:10
6 SV
class +1
class -1
Classification - kernel: poly - degree: 4 - cost:10
0 faults
class +1
class -1

Question 1 (5 points):

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

_a) The SVM with polynomial degree=2 already seems to do quite well on the data set.
_b) The higher the polynomial degree, the better the classifier.
_c) A very high number of support vectors seems to be an indicator of a bad choice of the kernel.

_d) There is hardly any difference between the pictures that were produced by polynomial kernels of even degree.
_e) For kernels with an odd degree the number of misclassified samples decreases with an 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. More details on grading can be found in the FAQ sheet.
Note: Do not reuse these variable names. They are used for testing.

In [6]:
```
example_True = True
example_False = False

a_ = True
b_ = True
c_ = False

d_ = True
e_ = True
```

Now apply an RBF-kernel and vary the C-parameter from close to 0 to a very high value. After playing around a little bit, try to report on your observations in the subsequent question.

In [7]:
```
#Here you can play around with the code a little bit. Cell will not be used for grading.

#SOLUTION
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)

# model = svm.SVC(kernel="rbf",C=1)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=2)
# model.fit(X,y)
# plot_data(X,y,model)
```

```
# model = svm.SVC(kernel="rbf",C=10)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=15)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=34)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=50)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=534)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=701)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=856)
# model.fit(X,y)
# plot_data(X,y,model)

# model = svm.SVC(kernel="rbf",C=999)
# model.fit(X,y)
# plot_data(X,y,model)
```

Question 2 (5 points):

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

_f) The higher the cost the more support vectors we have.

_g) The decision boarders don't change drastically with increasing C, only the number of support vectors does.

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. More details on grading can be found in the FAQ sheet.
Note: Do not reuse these variable names. They are used for testing.

In [8]:
```
example_True = True
example_False = False

f_ = False
g_ = True
```

Code 2 (10 points):

- Now use function `pol2cart()` which returns **100 two-dimensional** points which are **uniformly** distributed within the circle of radius $r = 0.3$ to generate new data points. Label these points with -1 and add them to the main `X` feature matrix and `y` label vector. Your new variables will be `X_new` and `y_new`
- Now use an **rbf kernel** and again play around with the parameter to explore the effects on the classification performance by again 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 iterating over different values of $\gamma := 1/(2\sigma^2)$ (compare RBF definition in lecture slides), ranging from 0.1 to 1. Again report your observations in the subsequent question.

In [9]:
```
#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 [10]:
```
#define new data ↓↓↓
X_new, y_new = None, None

pi_value=np.pi
phi=np.random.uniform(0,2*pi_value,size=(100,))

X_new,y_new=pol2cart(0.3,phi)
X_new=np.stack((X_new,y_new),axis=1)

y_new=np.empty(100,)
y_new.fill(-1)

X_new=np.vstack((X,X_new))
y_new=np.append(y,y_new)
```

In [11]:
```
#plot new data ↓↓↓
plt.title("Extra 100 points added")
plt.scatter(X_new[:,0],X_new[:,1],c=y_new)
plt.show()
```

In [12]:
```
"""
Function iter_Gamma fits SVM using defined gamma range and plots every variation.
```

```python
@ gamma_range: list of gammas
@ returns: list of dictionaries, lenght of list = length of gamma_range
"""
def iter_Gamma(gamma_range,C,X_new,y_new):
    list_of_models = []
    #your code ↓↓↓
    #code ends here
    for gamma in gamma_range:
        svm_classifier=svm.SVC(C=C,kernel='rbf',gamma=gamma)
        svm_classifier.fit(X_new,y_new)
        plotting_data=plot_data(X_new,y_new,svm_classifier)
        list_of_models.append(plotting_data)

    return list_of_models

##########################################################################################
#using the written function

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

gamma_range=gamma_values

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

Margins - kernel: rbf - gamma: 0.9 - cost: 0.1
377 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
356 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
184 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
74 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
317 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
38 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
45 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
282 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
27 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
45 SV
class +1
class -1

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

Margins - kernel: rbf - gamma: 0.1 - cost: 1000
62 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.1 - cost: 1000
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.5 - cost: 1000
27 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.5 - cost: 1000
0 faults
class +1
class -1

Question 3 (5 points):

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

_h) For a fixed γ, the larger the cost, the higher the number of support vectors .
_i) 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.
_j) For fixed C, increasing γ usually reduces the model complexity of the SVM.

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. More details on grading can be found in the FAQ sheet.
Note: Do not reuse these variable names. They are used for testing.

In [13]:
```
example_True = True
example_False = False

h_ = False
i_ = True
j_ = False
```

Code 3 (10 points):

- Finally, we want to investigate, how the RBF kernel classifier reacts to outliers. To this end add a single point $(1.8, 1.2)$ to your new data set (with the additional circle around 0) and label it with $y = +1$. Again plot the data set.?
- Again 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 [14]:
```
# update X_new and y_new by adding point as described in the task↓↓↓
X_new = None
y_new = None

X_new=np.copy(X)
y_new=np.copy(y)

new_label=+1
tmp_array=np.array([1.8,1.2])
X_new=np.vstack((X_new,tmp_array))
y_new=np.append(y,new_label)

plt.title("One extra point added")
plt.scatter(X_new[:,0],X_new[:,1],c=y_new)
plt.show()
```

In [15]:
```
#This cell is just to discover the behaviour of SVM given extra point, not graded.

#code here ↓↓↓
def iter_Gamma(gamma_range,C,X_new,y_new):
    list_of_models = []
    #your code ↓↓↓
    #code ends here
    for gamma in gamma_range:
        svm_classifier=svm.SVC(C=C,kernel='rbf',gamma=gamma)
        svm_classifier.fit(X_new,y_new)
        plotting_data=plot_data(X_new,y_new,svm_classifier)
        list_of_models.append(plotting_data)

    return list_of_models

##########################################################################################
#using the written function

cost_values = [-1,0,1,2,3]
gamma_values = np.arange(0.1,1.0,0.1)
list_of_models_rbf=[]
models=[]

gamma_range=gamma_values

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

Margins - kernel: rbf - gamma: 0.2 - cost: 0.1
245 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.2 - cost: 0.1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.30000000000000004 - cost: 0.1
206 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.30000000000000004 - cost: 0.1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.4 - cost: 0.1
194 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.4 - cost: 0.1
1 faults
class +1
class -1

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

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

Margins - kernel: rbf - gamma: 0.6 - cost: 0.1
184 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.6 - cost: 0.1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.7000000000000001 - cost: 0.1
184 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.7000000000000001 - cost: 0.1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.8 - cost: 0.1
184 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.8 - cost: 0.1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.9 - cost: 0.1
185 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
92 SV
class +1
class -1

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

Margins - kernel: rbf - gamma: 0.2 - cost: 1
50 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.2 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.30000000000000004 - cost: 1
43 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.30000000000000004 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.4 - cost: 1
40 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.4 - cost: 1
1 faults
class +1
class -1

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

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

Margins - kernel: rbf - gamma: 0.6 - cost: 1
38 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.6 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.7000000000000001 - cost: 1
38 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.7000000000000001 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.8 - cost: 1
38 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.8 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.9 - cost: 1
42 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
19 SV
class +1
class -1

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

Margins - kernel: rbf - gamma: 0.2 - cost: 10
15 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.2 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.30000000000000004 - cost: 10
17 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.30000000000000004 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.4 - cost: 10
18 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.4 - cost: 10
1 faults
class +1
class -1

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

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

Margins - kernel: rbf - gamma: 0.6 - cost: 10
21 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.6 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.7000000000000001 - cost: 10
22 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.7000000000000001 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.8 - cost: 10
26 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.8 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.9 - cost: 10
27 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
11 SV
class +1
class -1

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

Margins - kernel: rbf - gamma: 0.2 - cost: 100
9 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.2 - cost: 100
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.30000000000000004 - cost: 100
15 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.30000000000000004 - cost: 100
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.4 - cost: 100
18 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.4 - cost: 100
1 faults
class +1
class -1

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

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

Margins - kernel: rbf - gamma: 0.6 - cost: 100
19 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.6 - cost: 100
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.7000000000000001 - cost: 100
22 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.7000000000000001 - cost: 100
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.8 - cost: 100
22 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.8 - cost: 100
0 faults
class +1
class -1

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

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

Margins - kernel: rbf - gamma: 0.1 - cost: 1000
11 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.1 - cost: 1000
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.2 - cost: 1000
11 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.2 - cost: 1000
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.30000000000000004 - cost: 1000
10 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.30000000000000004 - cost: 1000
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.4 - cost: 1000
12 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.4 - cost: 1000
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.5 - cost: 1000
15 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.5 - cost: 1000
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.6 - cost: 1000
18 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.6 - cost: 1000
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.7000000000000001 - cost: 1000
20 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.7000000000000001 - cost: 1000
0 faults
class +1
class -1

Question 4 (5 points):

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

_k) 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 the outlier.
_l) 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.
_m) Classifiers with high costs ($C \geq 100$ say) and high γ ($\gamma \geq 0.5$) are susceptible to overfitting.

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. More details on grading can be found in the FAQ sheet.
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 [16]:
```
example_True = True
example_False = False

k_ = True
l_ = False
m_ = True
```

In [17]:
```
#here you can experiment, cell is not graded
```

```python
def iter_Gamma(gamma_range,C,X_new,y_new):
    list_of_models = []
    #your code ↓↓↓
    #code ends here
    for gamma in gamma_range:
        svm_classifier=svm.SVC(C=C,kernel='rbf',gamma=gamma)
        svm_classifier.fit(X_new,y_new)
        plotting_data=plot_data(X_new,y_new,svm_classifier)
        list_of_models.append(plotting_data)

    return list_of_models

##########################################################################################
#using the written function

cost_values = range(-1,4)
gamma_values = [0.67,0.87,0.99]
list_of_models_rbf=[]
models=[]

gamma_range=gamma_values

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

Margins - kernel: rbf - gamma: 0.99 - cost: 0.1
184 SV
class +1
class -1
Classification - kernel: rbf - gamma: 0.99 - cost: 0.1
1 faults
class +1
class -1
Margins - kernel: rbf - gamma: 0.67 - cost: 1
38 SV
class +1
class -1
Classification - kernel: rbf - gamma: 0.67 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.87 - cost: 1
40 SV
class +1
class -1
Classification - kernel: rbf - gamma: 0.87 - cost: 1
1 faults
class +1
class -1
Margins - kernel: rbf - gamma: 0.99 - cost: 1
42 SV
class +1
class -1
Classification - kernel: rbf - gamma: 0.99 - cost: 1
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.67 - cost: 10
21 SV
class +1
class -1
Classification - kernel: rbf - gamma: 0.67 - cost: 10
1 faults
class +1
class -1
Margins - kernel: rbf - gamma: 0.87 - cost: 10
26 SV
class +1
class -1
Classification - kernel: rbf - gamma: 0.87 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.99 - cost: 10
27 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.99 - cost: 10
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.67 - cost: 100
21 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.67 - cost: 100
1 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.87 - cost: 100
22 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.87 - cost: 100
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.99 - cost: 100
23 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.99 - cost: 100
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.67 - cost: 1000
19 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.67 - cost: 1000
0 faults
class +1
class -1

Margins - kernel: rbf - gamma: 0.87 - cost: 1000
21 SV
class +1
class -1

Classification - kernel: rbf - gamma: 0.87 - cost: 1000
0 faults
class +1
class -1

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 [18]:
```
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 [19]:
```
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([a_,b_,c_,d_,e_,f_,g_,h_,i_,j_,k_,l_,m_]))
```

```
Test questions answers are ok
```

Processing math: 100%