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 problem:

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

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

2. Derivatives of Lagrangian
   $\frac{\partial L}{\partial w_1} = 16w_1^3 + \alpha = 0 --- > w_1* = -\sqrt[3]{\frac{\alpha}{16}}$
   $\frac{\partial L}{\partial w_2} = 16w_2^3 - \alpha = 0 --- > w_2* = \sqrt[3]{\frac{\alpha}{16}}$

3. Solving by using KKT conditions
   $\alpha h(w_1*, w_2*) = 0, \text{so } \alpha = 0 \text{ or } h(w_1*, w_2*) = 0$

   $h(w_1*, w_2*) = 4 - 2\sqrt[3]{\frac{\alpha}{16}} = 0$

   $\alpha = 2^7 = 128$

   $w_1* = -2$

$w_2* = 2$

4. Solving without KKT conditions
   We have to solve the dual problem: Maximize:
   $\frac{\partial L}{\partial \alpha} L(w_1, w_2, \alpha) = -4 - w_1 + w_2 = 0; w_1* = -\sqrt[3]{\frac{\alpha}{16}}; w_2* = \sqrt[3]{\frac{\alpha}{16}}$

   $\frac{\partial L}{\partial \alpha} L(w_1*, w_2*, \alpha) = -4 + 2(\frac{\alpha}{16})^{\frac{1}{3}} = 0$

   $\alpha = 2^3 * 16 = 2^7 = 128$

   $w_1* = -2$

   $w_2* = 2$

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:

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

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?
   $Solution Here$

2. Lagrangian function
   $Solution Here$

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 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 [22]:
```python
#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 [23]:
```python
#Nothing to do here
"""Function allows ro load datapoint from csv
@returns: tuple (X,y)"""
```

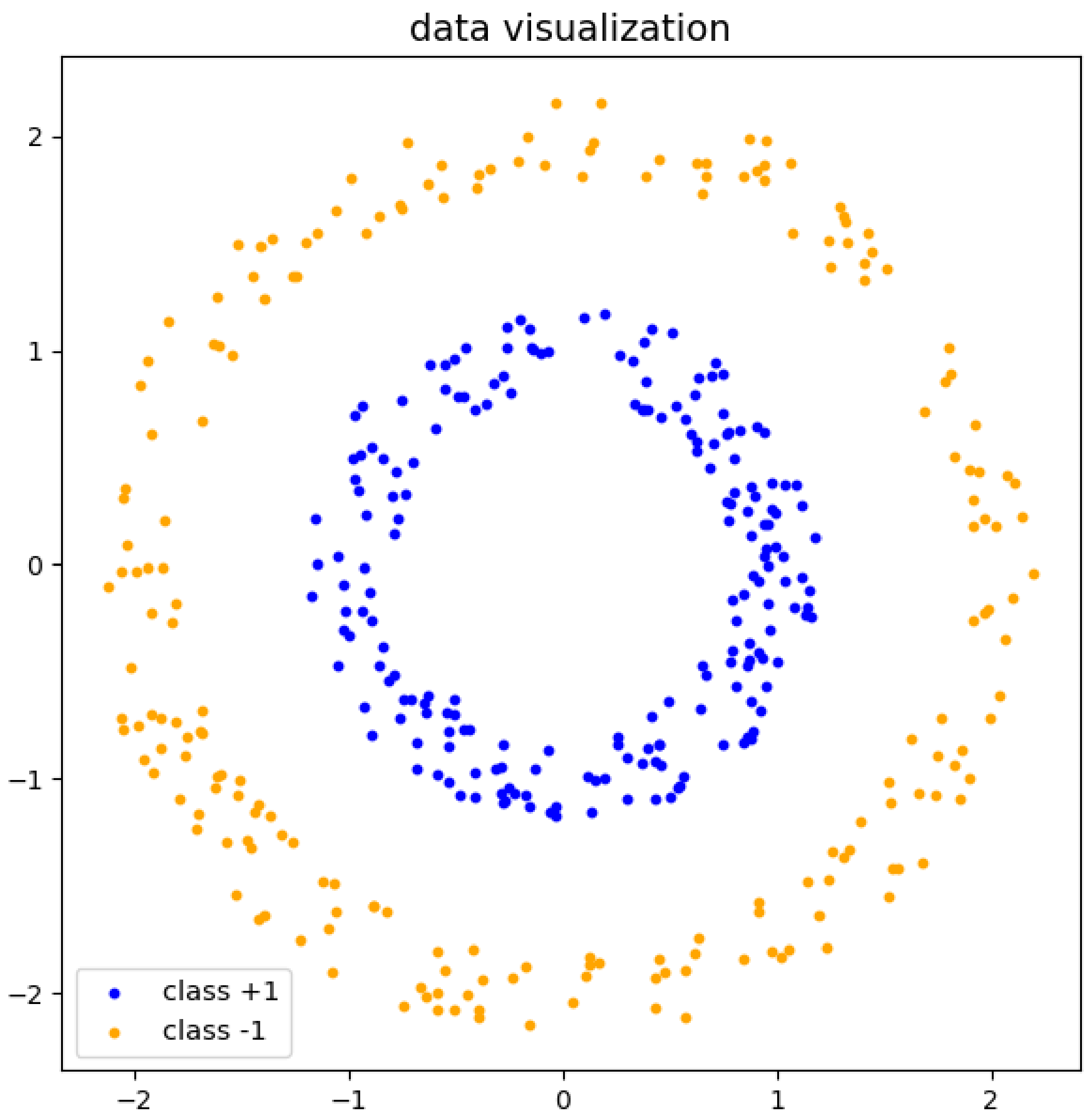
```
def load_data(id_data=1):
    if id_data ==1:
        Z = np.genfromtxt('radial_data.csv', delimiter=',')
        return Z[:,:-1], Z[:,-1]

"""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)
    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
                ax.scatter(grid[mask_neg,0], grid[mask_neg,1], c=color[1], alpha=0
                ax.scatter(X[mask_sv,0], X[mask_sv,1], c='g',label= str(np.sum(mod
            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
                ax.scatter(grid[mask_neg,0], grid[mask_neg,1], c=color[1], alpha=0
                classification = model.predict(X)
                mask_wrong = classification != y
                ax.scatter(X[mask_wrong,0], X[mask_wrong,1], c='magenta',label=str
            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 [24]:

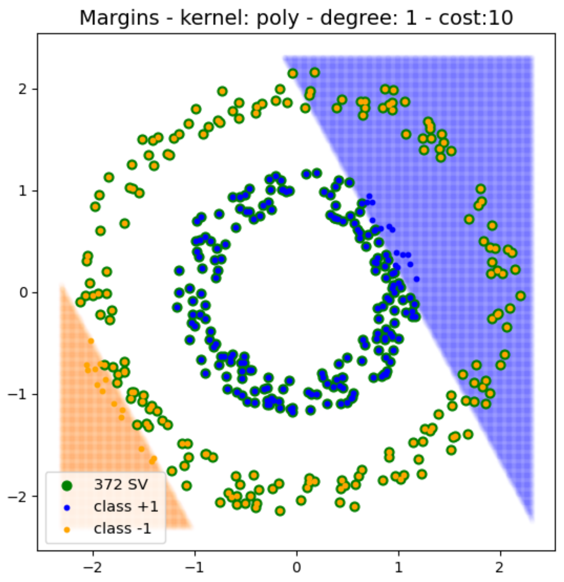
```
#load data
#leave as it is
X,y = load_data(1)
```

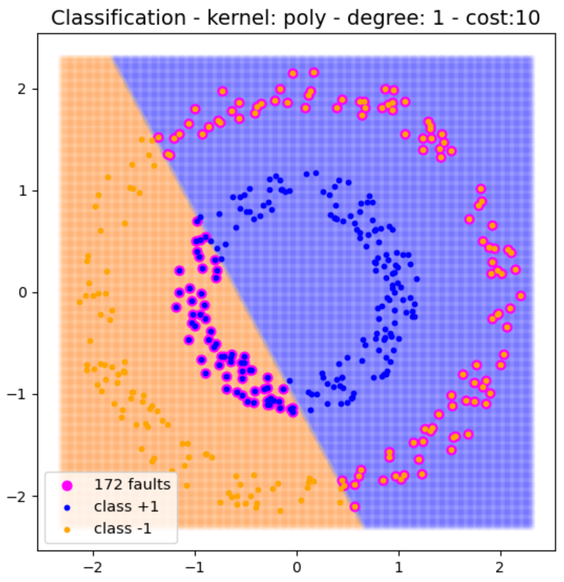
In [25]:

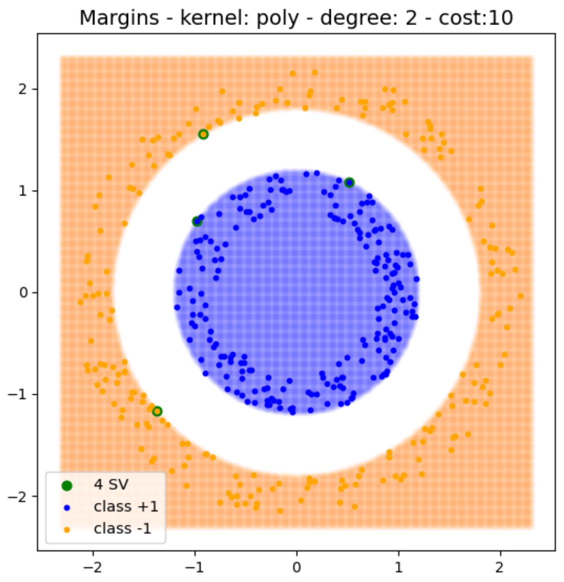
```
#plot your data ↓↓↓

plot_data(X, y)
```

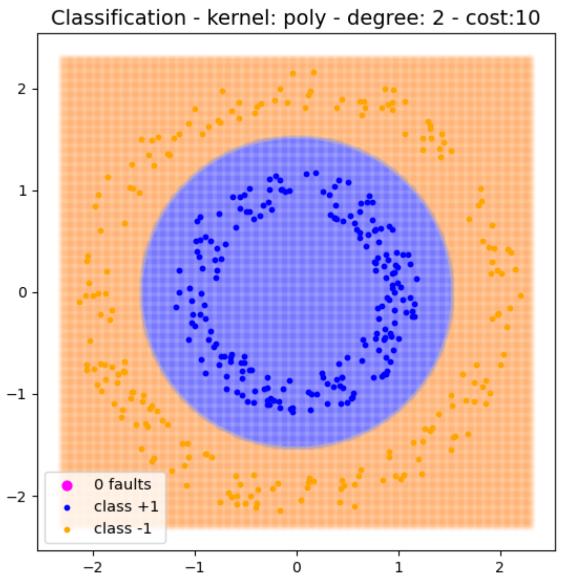


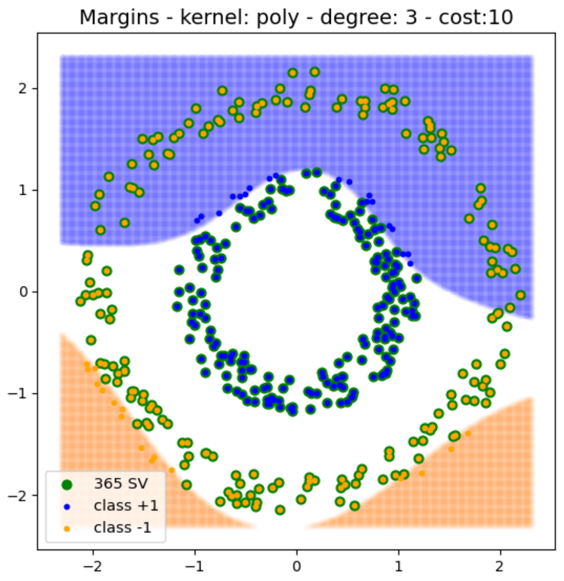
In [26]:
```
"""
Function iter_Degree fits SVM using defined range of degrees and plots every variat
@ degree_range: range of integer degrees
@ returns list of dictionaries, lenght of list = length of degree_range
"""
def iter_Degree(degree_range, X, y):
    list_of_models = []
    #your code ↓↓↓
    #code ends here
    for i in range (1,6):
        clf = svm.SVC(C = 10, kernel = 'poly', degree = i)
        clf.fit(X,y)
        list_of_models.append(clf.get_params())
    return list_of_models

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

degree_range = None #change this variable
iter_Degree(degree_range,X,y)

for i in list_of_models:

    clf = svm.SVC()
    clf.set_params(**i)
    clf.fit(X,y)
    plot_data(X,y, clf)
```

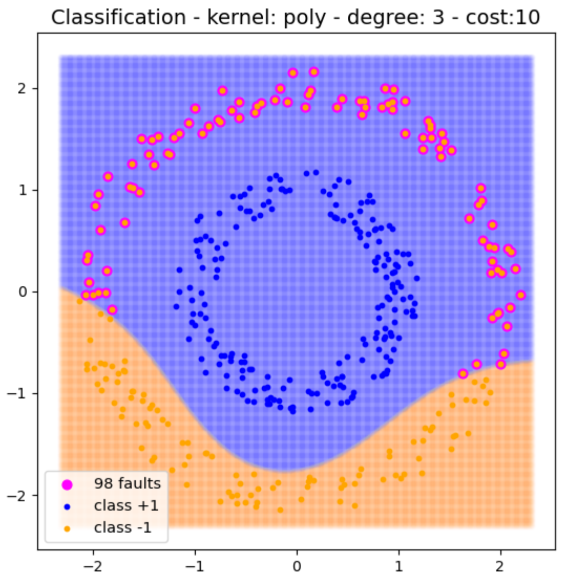


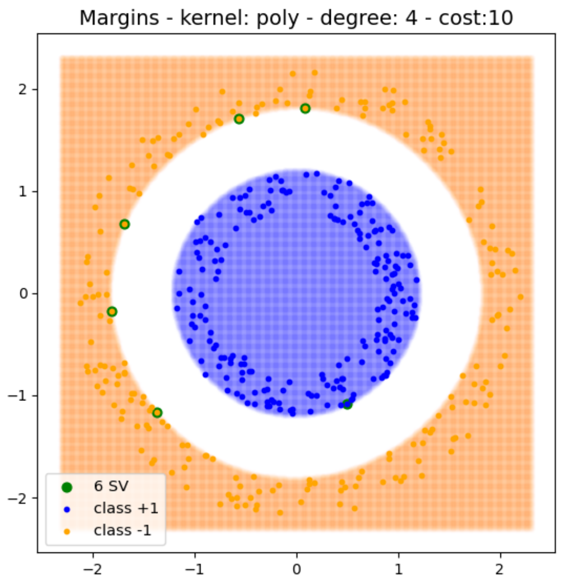


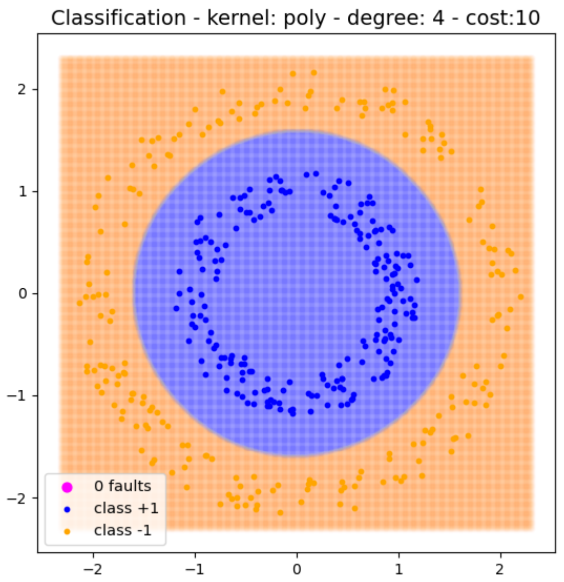


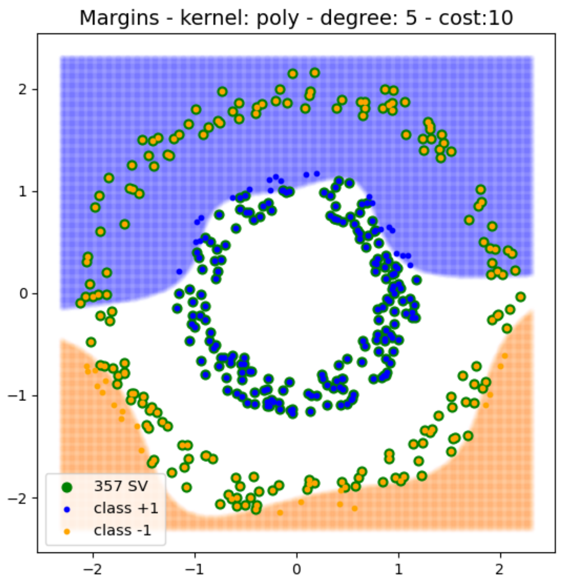


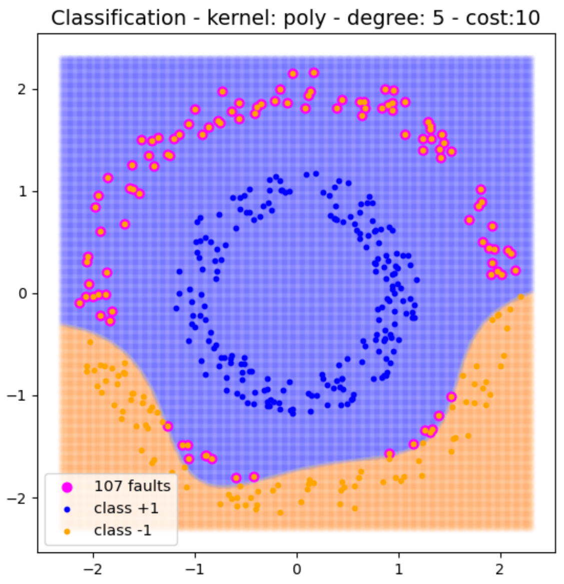












Question 1 (5 points):

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. </u>
_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. </u>

_d) There is hardly any difference between the pictures that were produced by polynomial kernels of even degree.</u>
_e) For kernels with an odd degree the number of misclassified samples decreases with an increasing degree.

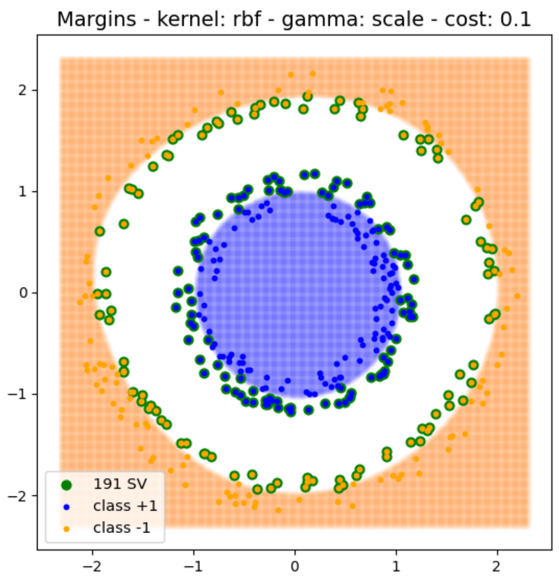
To answer the question, assign "True" or "False" boolean values to variables in the next cell. A non-correctly answered question yields negative points and no answer (i.e. answer "None") gives 0 points for a question. More details on grading can be found in the FAQ sheet.

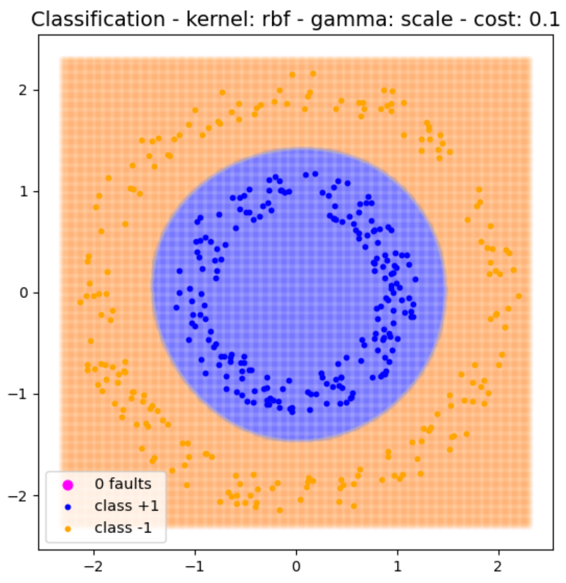
Note: Do not reuse these variable names. They are used for testing.

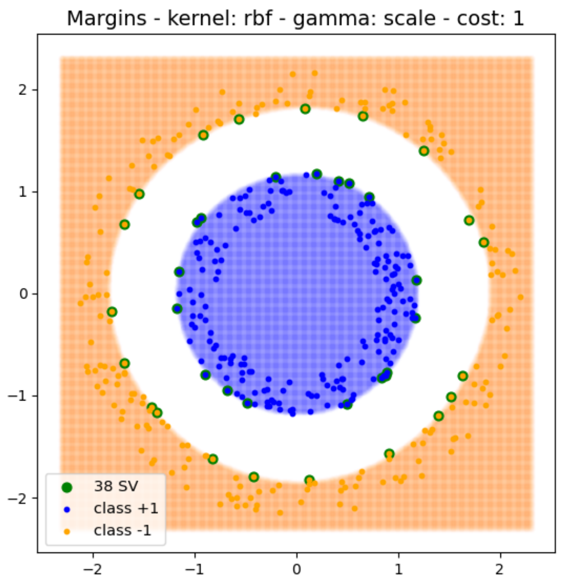
In [27]:
```
example_True = True
example_False = False

a_ = True
b_ = False
c_ = True

d_ = True
e_ = False
```

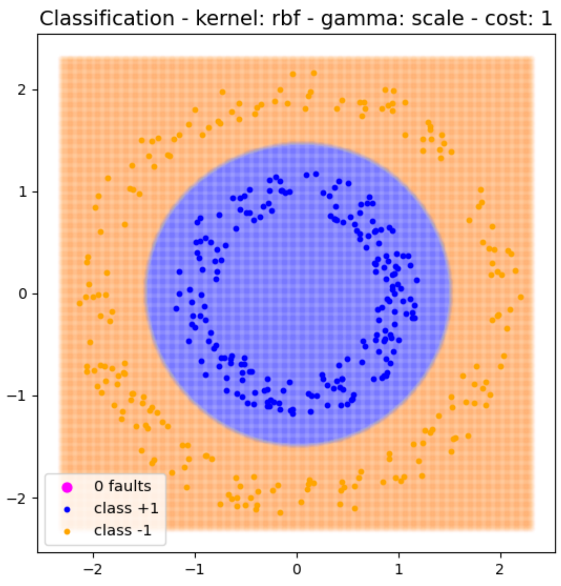
Now apply an RBF-kernel and vary the C-parameter from close to 0 to a very high value. After playing around a little bit, try to report on your observations in the subsequent question.

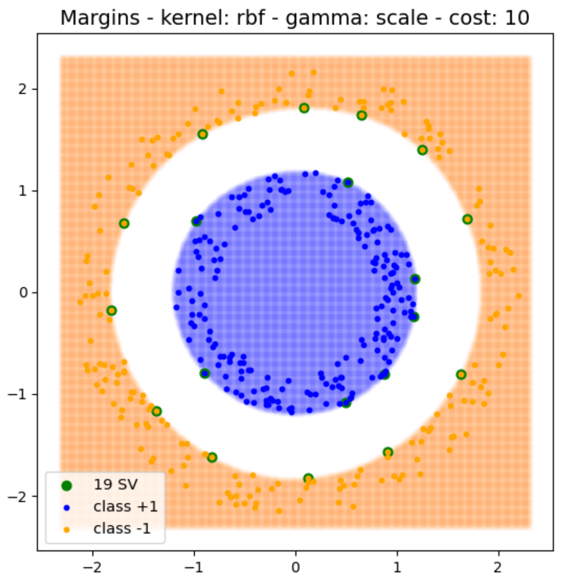
In [28]:
```
#Here you can play around with the code a little bit. Cell will not be used for gr

#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)
```

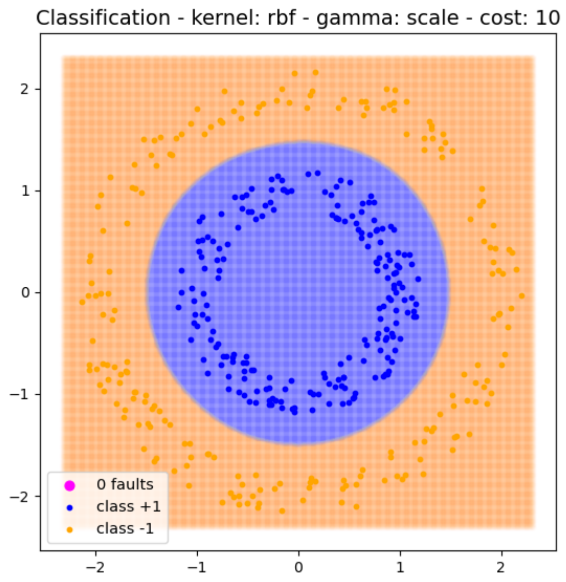


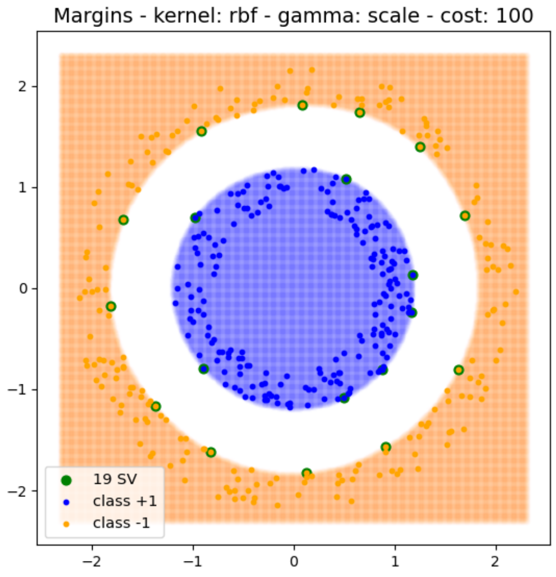


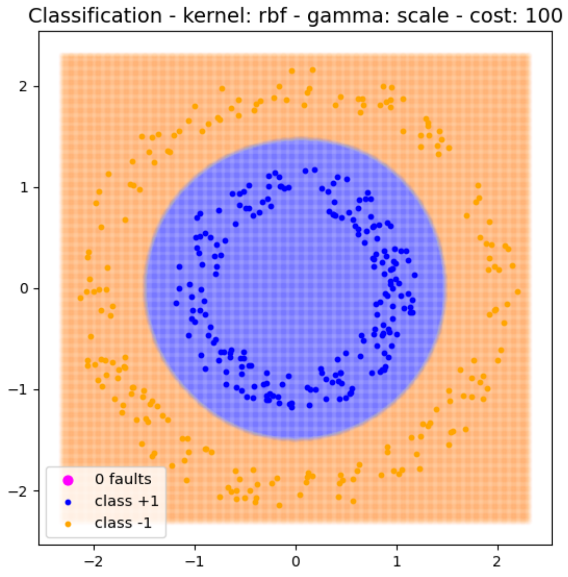


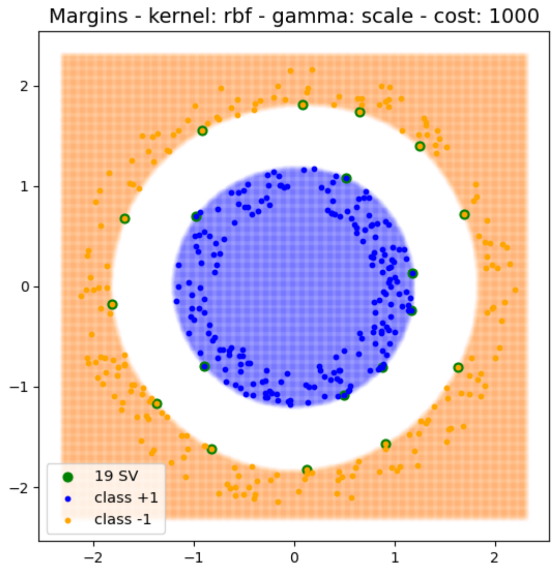


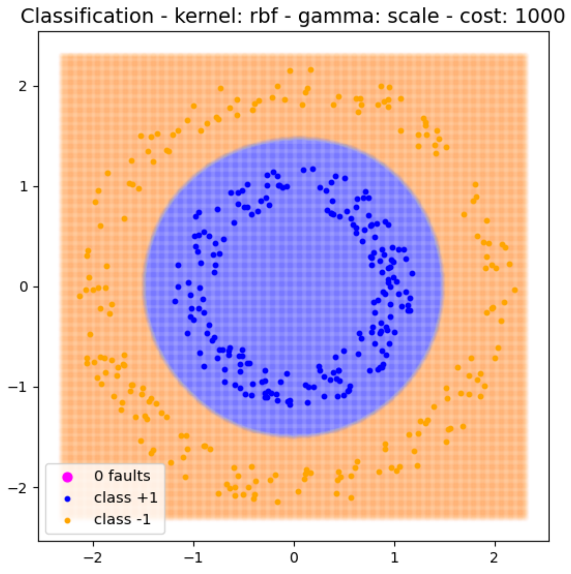












Question 2 (5 points):

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.</u>

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 [29]:
```
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 [30]:
```
#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 [31]:
```
#define new data ↓↓↓
X_new, y_new = None, None
```

In [32]:
```
#plot new data ↓↓↓
```

In [33]:
```
"""
Function iter_Gamma fits SVM using defined gamma range and plots every variation.
@ 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

    return list_of_models

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

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

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12096\760600618.py in <module>
     18 
     19 for cost in cost_values:
---> 20     iter_Gamma(gamma_range,10**cost,X_new,y_new)

NameError: name 'gamma_range' is not defined
```

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. </u>
_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 [ ]:
```
example_True = True
example_False = False

h_ = None
i_ = None
j_ = None
```

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 [ ]:
```
# update X_new and y_new by adding point as described in the task↓↓↓
X_new = None
y_new = None
```

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

#code here ↓↓↓
```

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. </u>
_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.</u>

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

k_ = None
l_ = None
m_ = None
```

In [ ]:
```
#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 [ ]:
```
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 [ ]:
```
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_]))
```