1 Exercise 1:

We consider the function space

$$\mathcal{H} = \{f : [0,1] \to \mathbb{R},\ \text{absolutely continuous}, f' \in L^2([0,1]), f(0) = 0\},$$

endowed with the inner product

$$\langle f, g\rangle_{\mathcal{H}} = \int_0^1 (f(u)g(u) + f'(u)g'(u))\, du.$$

We aim to show that $\mathcal{H}$ is a Hilbert space and then verify that it is a Reproducing Kernel Hilbert Space (RKHS).
$\mathcal{H}$ as a Hilbert space
To establish that $\mathcal{H}$ is a Hilbert space, we must prove that it is complete with respect to the norm induced by the inner product:

$$\|f\|^2_{\mathcal{H}} = \int_0^1 \left(f(u)^2 + f'(u)^2\right) du.$$

Consider a Cauchy sequence $\{f_n\}$ in $\mathcal{H}$. The norm involves both f_n and its derivative f'_n, so we conclude that: f_n is Cauchy in $L^2([0,1])$, meaning there exists $f \in L^2([0,1])$ such that $f_n \to f$ in L^2. f'_n is Cauchy in $L^2([0,1])$, ensuring the existence of a function $g \in L^2([0,1])$ such that $f'_n \to g$ in L^2. Since each f_n is absolutely continuous and satisfies $f_n(0) = 0$, we know that for all $x \in [0,1]$,

$$f_n(x) = \int_0^x f'_n(u)du.$$

Taking the limit as $n \to \infty$, and using the dominated convergence theorem, we conclude that:

$$f(x) = \int_0^x g(u)du.$$

This implies that f is absolutely continuous with $f' = g \in L^2([0,1])$, meaning $f \in \mathcal{H}$.
Since $\mathcal{H}$ contains the limit of every Cauchy sequence, it is complete, and thus a Hilbert space.
For $\mathcal{H}$ as an RKHS
To show that $\mathcal{H}$ is an RKHS, we must prove the existence of a reproducing kernel, meaning that for each $x \in [0,1]$, there exists a function $K_x \in \mathcal{H}$ such that:

$$f(x) = \langle f, K_x\rangle_{\mathcal{H}}, \quad \forall f \in \mathcal{H}.$$

By the Riesz representation theorem, for every x, there exists K_x such that for any $f \in \mathcal{H}$,

$$f(x) = \int_0^1 (f(u)K_x(u) + f'(u)K'_x(u))du.$$

This property ensures that $\mathcal{H}$ is an RKHS.
For the differential equation for the reproducing kernel
The reproducing kernel $K(x, y)$ is the function satisfying:

$$\langle K_y, f\rangle_{\mathcal{H}} = f(y), \quad \forall f \in \mathcal{H}.$$

Expanding the inner product, we obtain:

$$\int_0^1 \left(K(x,u)f(u) + \frac{\partial}{\partial u}K(x,u)f'(u)\right) du = f(x).$$

Since this must hold for all f, applying integration by parts to the second term (assuming $f(0) = 0$) gives:

$$\int_0^1 K(x,u)f(u)\, du - \int_0^1 \frac{d^2}{du^2}K(x,u)f(u)\, du = f(x).$$

From the fundamental lemma of calculus of variations, we deduce that $K(x,u)$ satisfies the differential equation:

$$K(x,y) - \frac{d^2}{du^2}K(x,y) = \delta(y-x),$$

with the boundary condition $K(0,y) = 0$ to ensure that $K(x,y) \in \mathcal{H}$.
This equation characterizes the reproducing kernel, which can be explicitly computed by solving the corresponding boundary value problem.

2 Exercise 2

1. Given $x, y \in \mathbb{R}^n$, we can expand the squared norm difference as:

$$||x-y||^2 = ||x||^2 - 2 < x, y > + ||y||^2.$$

We define the function $K(x,y)$ as:

$$K(x,y) = \phi(||x||^2)\phi(-2 < x, y >)\phi(||y||^2),$$

where ϕ represents the Gaussian density. The terms involving the norms remain positive, and since $\phi(-2 < x, y >) = \exp(2 < x, y > /2\sigma^2)$ forms a positive definite kernel—being a composition of the linear kernel with the exponential function—we can apply the Aronszajn representation theorem. This ensures the existence of a feature map ϕ satisfying:

$$K(x,y) =< \phi(x), \phi(y) > .$$

Utilizing the bilinearity of the inner product, we obtain:

$$K(x,y) =< \phi(x)\exp(-||x||^2/2\sigma^2), \phi(y)\exp(-||y||^2/2\sigma^2) > .$$

By applying Aronszajn's theorem once more, we conclude that the Gaussian kernel is indeed positive definite.
Consider the case where $0 < \sigma < \tau$ and let $f \in \mathcal{H}_\tau$. For $x, x' \in \mathbb{R}^n$, we analyze the integral:

$$\int_{-\infty}^{+\infty} \frac{\exp(-\frac{||x-t||^2}{\sigma})}{(\sqrt{\pi}\sigma)^d} \frac{\exp(-\frac{||x'-t||^2}{\sigma})}{(\sqrt{\pi}\sigma)^d} dt.$$

Expanding the exponent, we get:

$$\int_{-\infty}^{+\infty} \exp\left(-\frac{||x-t||^2}{\sigma} - \frac{||x'-t||^2}{\sigma}\right) dt.$$

A key identity follows:

$$-||x-t||^2 - ||x'-t||^2 = -2||t - \frac{x+x'}{2}||^2 - \frac{||x-x'||^2}{2}.$$

Evaluating the first term:

$$\int_{-\infty}^{+\infty} \frac{\exp(-\frac{2||t-\frac{x+x'}{2}||^2}{\sigma})}{(\sqrt{\pi}\sigma)^d} dt = \left(\frac{1}{\sqrt{2}}\right)^d.$$

For the second term, we directly obtain:

$$\frac{\exp(-\frac{||x-x'||^2}{2\sigma})}{(\sqrt{\pi}\sigma)^d}.$$

Combining these, we derive the Aronszajn representation for the Gaussian kernel:

$$\int_{-\infty}^{+\infty} \phi_x(t)\phi_{x'}(t)dt = \exp\left(-\frac{||x-x'||^2}{\sqrt{2\pi}\sigma}\right),$$

with:

$$\phi_x(t) = \frac{\exp(-\frac{||x-t||^2}{\sigma})}{(\sqrt{\pi}\sigma)^d}.$$

This implies that the Hilbert space is a subset of L_2 and inherits its inner product.
By the reproducing kernel property:

$$f(x) =< f, K_x >_{L_2} .$$

Consequently, the function $t \mapsto f(t)\exp(-\frac{||t-x||^2}{\sigma})$ is in L_2.
Furthermore, for $x \in \mathbb{R}$, we establish:

$$f(x) =< f, K_x >_{\mathcal{H}_\tau} .$$

The Gaussian kernel exhibits shift invariance. By applying Fourier transform techniques, we deduce that the corresponding Hilbert space forms a subset of L_2, consisting of functions satisfying:

$$f(x) = \int_{-\infty}^{+\infty} |\hat{f}(\omega)|^2 e^{\frac{\sigma^2\omega^2}{2}} d\omega.$$

Since the integral is bounded, we immediately conclude:

$$\mathcal{H}_\tau \subset \mathcal{H}_\sigma \subset L_2, \quad \text{for } \sigma < \tau,$$

and also:

$$||f||_{\mathcal{H}_\tau} \leq ||f||_{\mathcal{H}_\sigma}, \quad \text{for } \sigma < \tau.$$

Moreover, for any σ and $f \in \mathcal{H}_\sigma$, we observe:

$$||f||_{\mathcal{H}_\sigma} \leq ||f||_{L_2},$$

since the Fourier transform is an isometry, leading to:

$$\int_{-\infty}^{+\infty} |\hat{f}(\omega)|^2 d\omega = ||f||_{L_2}.$$

When $\sigma < \tau$, we analyze:

$$e^{\frac{\sigma^2\omega^2}{2}} - 1 = \sum_{n=1}^{+\infty} \frac{(\sigma^2\omega^2)^n}{2^n n!} \leq \frac{\sigma^2}{\tau^2} \sum_{n=1}^{+\infty} \frac{(\tau^2\omega^2)^n}{2^n n!}.$$

This gives:

$$||f||^2_{\mathcal{H}_\sigma} - ||f||^2_{L_2} \leq \frac{\sigma^2}{\tau^2}(||f||^2_{\mathcal{H}_\tau} - ||f||^2_{L_2}).$$

The positivity follows from the fact that the exponential function is always greater than 1 for positive inputs.
4. The final result is derived via the dominated convergence theorem. Since the L_2 norm is always well-defined and:

$$\lim_{\sigma \to 0} e^{\frac{\sigma^2\omega^2}{2}} = 1 \quad \text{for all } \omega,$$

and given that the exponential function is monotonically increasing in σ, we conclude:

$$\lim_{\sigma \to 0} ||f||_{\mathcal{H}_\sigma} = ||f||_{L_2}.$$

3 Exercice 3

1.(a) Le Lagrangien associé à notre problème s'écrit sous la forme suivante :

$$L(f, b, \xi, \alpha, \mu) = \frac{1}{2}||f||^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left(y_i(f(x_i) - b) - 1 + \xi_i\right) - \sum_{i=1}^{n} \mu_i \xi_i \tag{1}$$

avec $\alpha_i \geq 0$ et $\mu_i \geq 0$. D'après le théorème du representer, la fonction f peut s'écrire sous la forme :

$$f(x) = \sum_{i=1}^{n} \nu_i K(x_i, x). \tag{2}$$

Ainsi, nous pouvons reformuler le Lagrangien en remplaçant cette expression :

$$L(\alpha, b, \xi, \alpha, \mu) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \nu_i\nu_j K(x_i, x_j) + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\sum_{j=1}^{n} \nu_j K(x_j, x_i) - b \right) - 1 + \xi_i \right) - \sum_{i=1}^{n} \mu_i \xi_i. \tag{3}$$

Sous une forme matricielle, nous obtenons :

$$L(\alpha, b, \xi, \alpha, \mu, \nu) = \frac{1}{2}\nu^T K \nu + C\xi^T \nVdash - (diag(Y)\alpha)^T (K\nu - b) + \alpha^T 1 - (\mu + \alpha)^T \xi. \tag{4}$$

Minimiser ce Lagrangien par rapport à ν, ξ et b conduit aux conditions optimales suivantes :

$$\nu^{(*)} = diag(Y)\alpha, \tag{5}$$

$$\nu + \alpha = C, \tag{6}$$

$$\nVdash^T diag(Y)\alpha = 0. \tag{7}$$

Ainsi, la formulation duale du problème devient :

$$q(\alpha, \mu) = -\frac{1}{2}\alpha^T diag(Y) K diag(Y)\alpha + \alpha^T 1. \tag{8}$$

Finalement, nous obtenons le problème d'optimisation suivant :

$$\max_{0\leq\alpha\leq C,\sum_{i=1}^{n} y_i\nu_i=0} q(\alpha) = \max_{0\leq\alpha\leq C,\sum_{i=1}^{n} y_i\nu_i=0} -\frac{1}{2}\alpha^T diag(Y)K diag(Y)\alpha + \alpha^T 1. \tag{9}$$

1.(c) Les conditions de complémentarité, qui caractérisent la solution optimale, sont :

$$\alpha_i \left(y_i(f(x_i) - b) - 1 + \xi_i\right) = 0, \tag{10}$$

$$(\alpha_i - C)\xi_i = 0. \tag{11}$$

Les vecteurs supports sont les points pour lesquels $\alpha_i > 0$, ce qui implique la contrainte :

$$y_i(f(x_i) - b) = 1 - \xi_i \leq 1. \tag{12}$$

Support Vector Machines

```
import numpy as np
import pickle as pkl
from scipy import optimize
from scipy.linalg import cho_factor, cho_solve
import matplotlib.pyplot as plt
from utils import plot_multiple_images, generateRings,
scatter_label_points, loadMNIST
%matplotlib inline
from utils import plotClassification
```

Loading the data

The file 'classification_datasets' contains 3 small classification datasets:

```
- dataset_1: mixture of two well separated gaussians
- dataset_2: mixture of two gaussians that are not separeted
- dataset_3: XOR dataset that is non-linearly separable.
```

Each dataset is a hierarchical dictionary with the following structure:

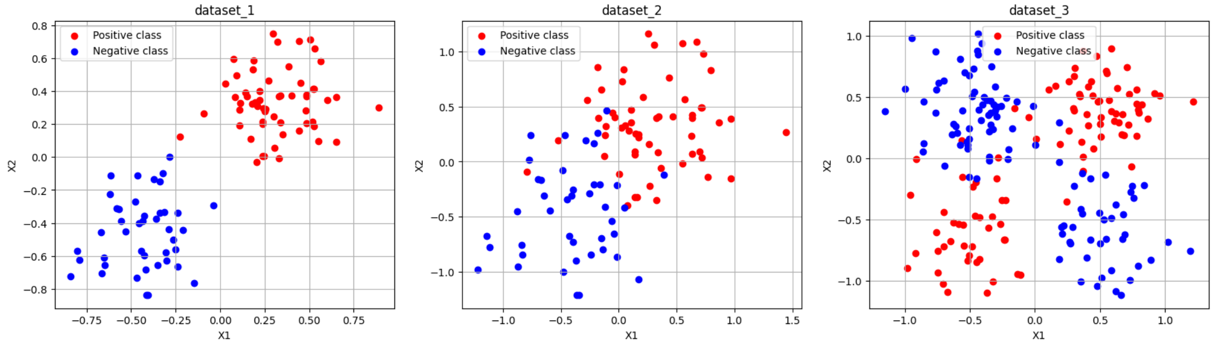
```
    dataset = {'train': {'x': data, 'y':label}
                'test': {'x': data, 'y':label}
              }
```

The data x is an N by 2 matrix, while the label y is a vector of size N.

Only the third dateset is used.

```
file = open('datasets/classification_datasets', 'rb')
datasets = pkl.load(file)
file.close()
fig, ax = plt.subplots(1,3, figsize=(20, 5))
for i, (name, dataset) in enumerate(datasets.items()):

    plotClassification(dataset['train']['x'], dataset['train']['y'],
ax=ax[i])
    ax[i].set_title(name)
```



III- Kernel SVC

1- Implementing the Gaussian Kernel

Implement the method 'kernel' of the class RBF and linear below, which takes as input two data matrices X and Y of size $N \times d$ and $M \times d$ and returns a gramm matrix G of shape $N \times M$ whose components are $k\left(x_i, y_j\right) = \exp\left(-\|x_i - y_i\|^2 / \left(2\sigma^2\right)\right)$ for the RBF kernel and $k\left(x_i, y_j\right) = x_i^\top y_j$ for the linear kernel. (The fastest solution does not use any for loop!)

```
class RBF:
    def __init__(self, sigma=1.):
        self.sigma = sigma  ## the variance of the kernel
    def kernel(self,X,Y):
        ## Input vectors X and Y of shape Nxd and Mxd
        return np.exp(-np.sum((X[:,None]-Y[None])**2, axis=-
1)/2/self.sigma**2)

class Linear:
    def kernel(self,X,Y):
        ## Input vectors X and Y of shape Nxd and Mxd
        return np.dot(X,Y.T)
```

2- Implementing the classifier

Implement the methods 'fit' and 'separating_function' of the class KernelSVC below to learn the Kernel Support Vector Classifier.

```
import numpy as np
from scipy import optimize

class KernelSVC:

    def __init__(self, C, kernel, epsilon=1e-3):
        self.type = 'non-linear'
        self.C = C  # Regularization parameter
        self.kernel = kernel  # Kernel function
        self.alpha = None  # Lagrange multipliers
```

```
        self.support_vectors = None  # Support vectors
        self.epsilon = epsilon  # Tolerance threshold
        self.norm_factor = None  # Normalization factor
        self.bias = None  # Bias term

    def fit(self, X, y):
        """ Train the Kernel SVM using quadratic optimization """
        num_samples = len(y)

        # Compute kernel matrix
        kernel_matrix = self.kernel(X, X)

        # Compute y * y.T * K
        yyK = np.outer(y, y) * kernel_matrix

        # Lagrange dual problem
        def loss_function(alpha):
            return 0.5 * alpha.dot(yyK).dot(alpha) - np.sum(alpha)

        # Gradient of the loss function
        def gradient_loss(alpha):
            return np.dot(yyK, alpha) - np.ones(num_samples)

        # Constraints on alpha:
        # - Equality constraint: sum(alpha_i * y_i) = 0
        equality_constraint = lambda alpha: np.dot(alpha, y)
        jacobian_equality = lambda alpha: y

        # - Inequality constraints: 0 <= alpha <= C
        inequality_constraint = lambda alpha: np.concatenate((alpha,
self.C - alpha))
        jacobian_inequality = lambda alpha:
np.concatenate((np.eye(num_samples), -np.eye(num_samples)))

        constraints = [
            {'type': 'eq', 'fun': equality_constraint, 'jac':
jacobian_equality},
            {'type': 'ineq', 'fun': inequality_constraint, 'jac':
jacobian_inequality}
        ]

        # Solve the optimization problem
        optimization_result = optimize.minimize(
            fun=loss_function,
            x0=np.ones(num_samples),
            method='SLSQP',
            jac=gradient_loss,
            constraints=constraints
        )
```

```
        # Store optimized alpha values
        self.alpha = optimization_result.x

        # Identify support vectors
        support_mask = self.alpha > self.epsilon
        self.support_multipliers = self.alpha[support_mask] *
y[support_mask]
        self.support_vectors = X[support_mask]
        support_indices = np.where(support_mask)[0]

        # Compute bias term b
        self.bias = np.mean(
            y[support_indices] - np.dot(kernel_matrix[support_indices]
[:, support_indices], self.alpha[support_indices] *
y[support_indices])
        )

        # Compute norm factor
        self.norm_factor = (self.alpha *
y).dot(kernel_matrix).dot(self.alpha * y)

    def separating_function(self, x):
        """ Compute the decision function values """
        # Compute kernel values between new points and support vectors
        kernel_values = self.kernel(x, self.support_vectors)
        return kernel_values.dot(self.support_multipliers)

    def predict(self, X):
        """ Predict class labels {-1, 1} """
        decision_values = self.separating_function(X)
        return 2 * (decision_values + self.bias > 0) - 1
```

2 b- Implementing the visualization function

Implement the function plotClassification that takes new data as input and the model, then displays separating function and margins along with misclassified points.

```
import matplotlib.pyplot as plt
import matplotlib.colors as pltcolors
import seaborn as sns

def plotHyperSurface(ax, xRange, model, intercept, label,
color='grey', linestyle='-', alpha=1.):
    #xx = np.linspace(-1, 1, 100)
    if model.type=='linear':
        xRange = np.array(xRange)
        yy = -(model.w[0] / model.w[1]) * xRange  -
intercept/model.w[1]
```

```
        ax.plot(xRange, yy, color=color, label=label,
linestyle=linestyle)
    else:
        xRange = np.linspace(xRange[0], xRange[1], 100)
        X0, X1 = np.meshgrid(xRange, xRange)
        xy = np.vstack([X0.ravel(), X1.ravel()]).T
        Y30 = model.separating_function(xy).reshape(X0.shape) +
intercept
        ax.contour(X0, X1, Y30, colors=color, levels=[0.],
alpha=alpha, linestyles=[linestyle]);


def plotClassification(X, y, model=None, label='',
separatorLabel='Separator',
            ax=None, bound=[[-1., 1.], [-1., 1.]]):
    """ Plot the SVM separation, and margin """
    colors = ['blue','red']
    labels = [1,-1]
    cmap = pltcolors.ListedColormap(colors)
    if ax is None:
        fig, ax = plt.subplots(1, figsize=(11, 7))
    for k, label in enumerate(labels):
        im = ax.scatter(X[y==label,0], X[y==label,1],
alpha=0.5,label='class '+str(label))

    if model is not None:
        # Plot the seprating function
        plotHyperSurface(ax, bound[0], model, model.bias,
separatorLabel)
        if model.support_vectors is not None:
            ax.scatter(model.support_vectors[:,0],
model.support_vectors[:,1], label='Support', s=80, facecolors='none',
edgecolors='r', color='r')
            print("Number of support vectors = %d" %
(len(model.support_vectors)))

        # Plot the margins
        intercept_neg = model.bias - 1
        intercept_pos = model.bias + 1
        xx = np.array(bound[0])
        plotHyperSurface(ax, xx, model, intercept_neg , 'Margin -',
linestyle='-.', alpha=0.8)
        plotHyperSurface(ax, xx, model, intercept_pos , 'Margin +',
linestyle='--', alpha=0.8)

        # Plot points on the wrong side of the margin
        wrong_side_points = X[np.abs(model.separating_function(X) +
model.bias) > 1]
        ax.scatter(wrong_side_points[:,0], wrong_side_points[:,1],
label='Beyond the margin', s=80, facecolors='none',
```

```
                edgecolors='grey', color='grey')

    ax.legend(loc='upper left')
    ax.grid()
    ax.set_xlim(bound[0])
    ax.set_ylim(bound[1])
```

3- Fitting the classifier

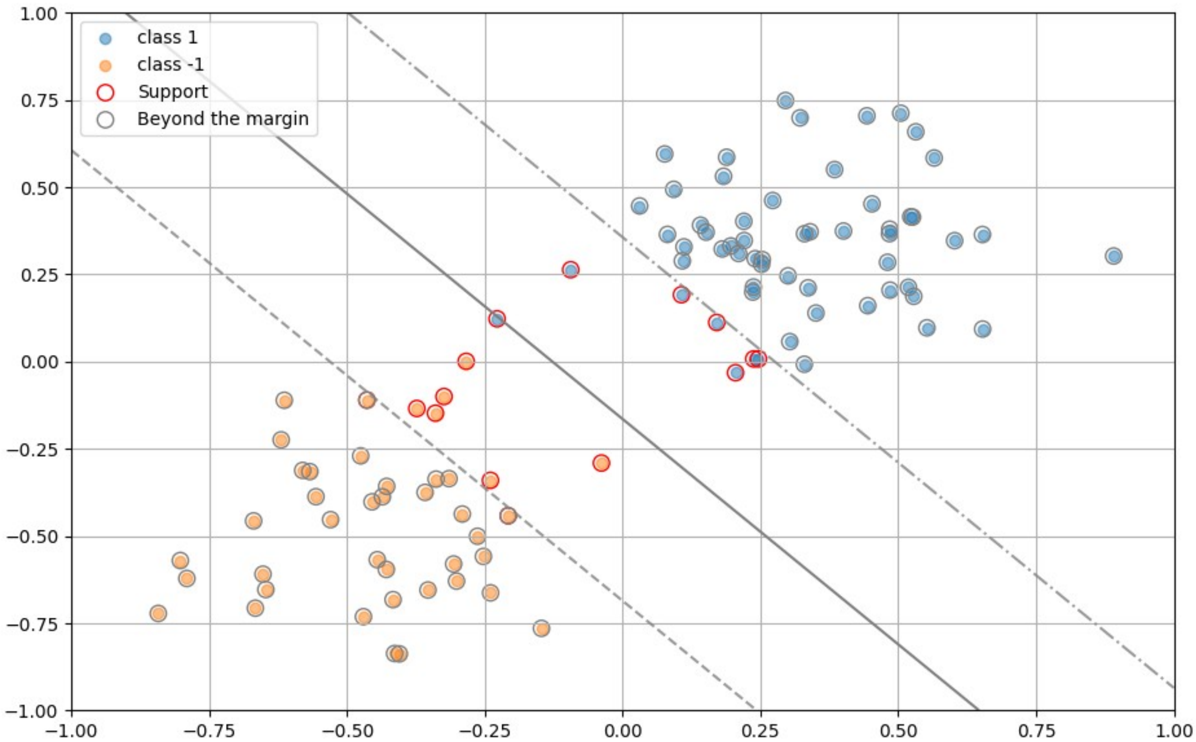
Run the code block below to fit the classifier and report its output.

Dataset 1

Linear classifier

```
C=1
kernel = Linear().kernel
model = KernelSVC(C=C, kernel=kernel, epsilon=1e-14)
train_dataset = datasets['dataset_1']['train']
model.fit(train_dataset['x'], train_dataset['y'])
plotClassification(train_dataset['x'], train_dataset['y'], model,
label='Training')
```

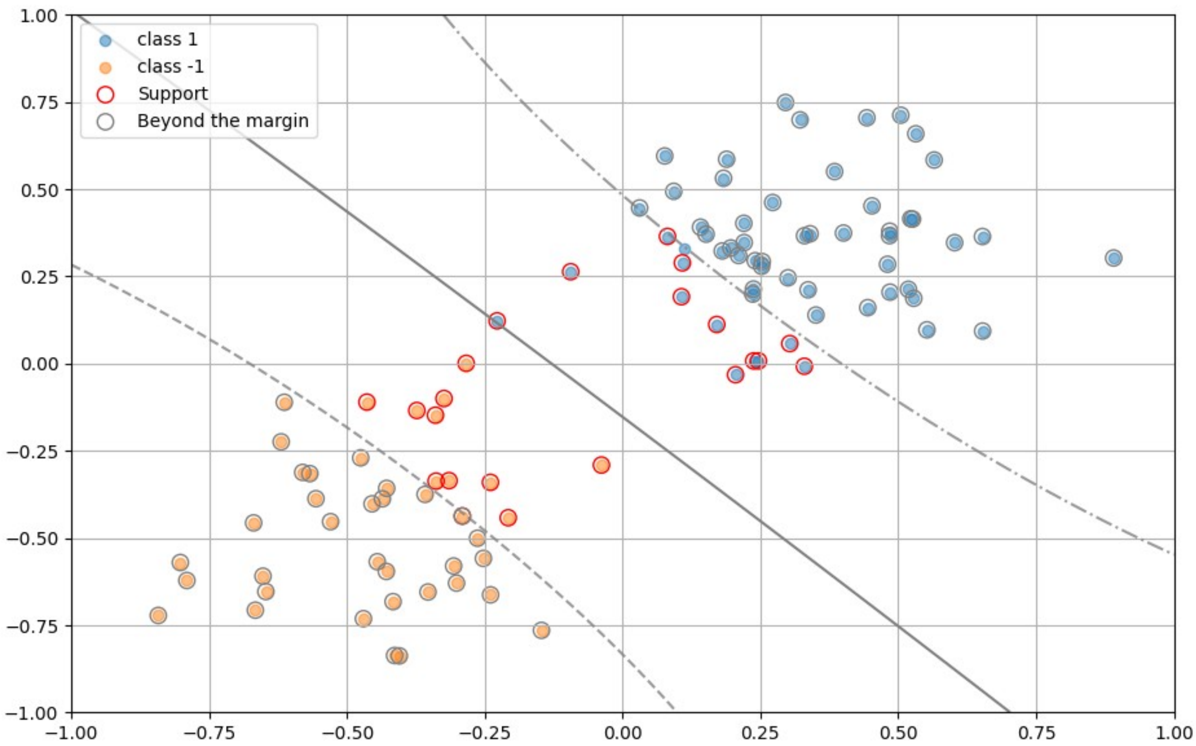
```
Number of support vectors = 15
```



Gaussian classifier

```
sigma = 1.5
C=1.
kernel = RBF(sigma).kernel
model = KernelSVC(C=C, kernel=kernel, epsilon=1e-14)
train_dataset = datasets['dataset_1']['train']
model.fit(train_dataset['x'], train_dataset['y'])
plotClassification(train_dataset['x'], train_dataset['y'], model,
label='Training')
```

```
Number of support vectors = 22
```

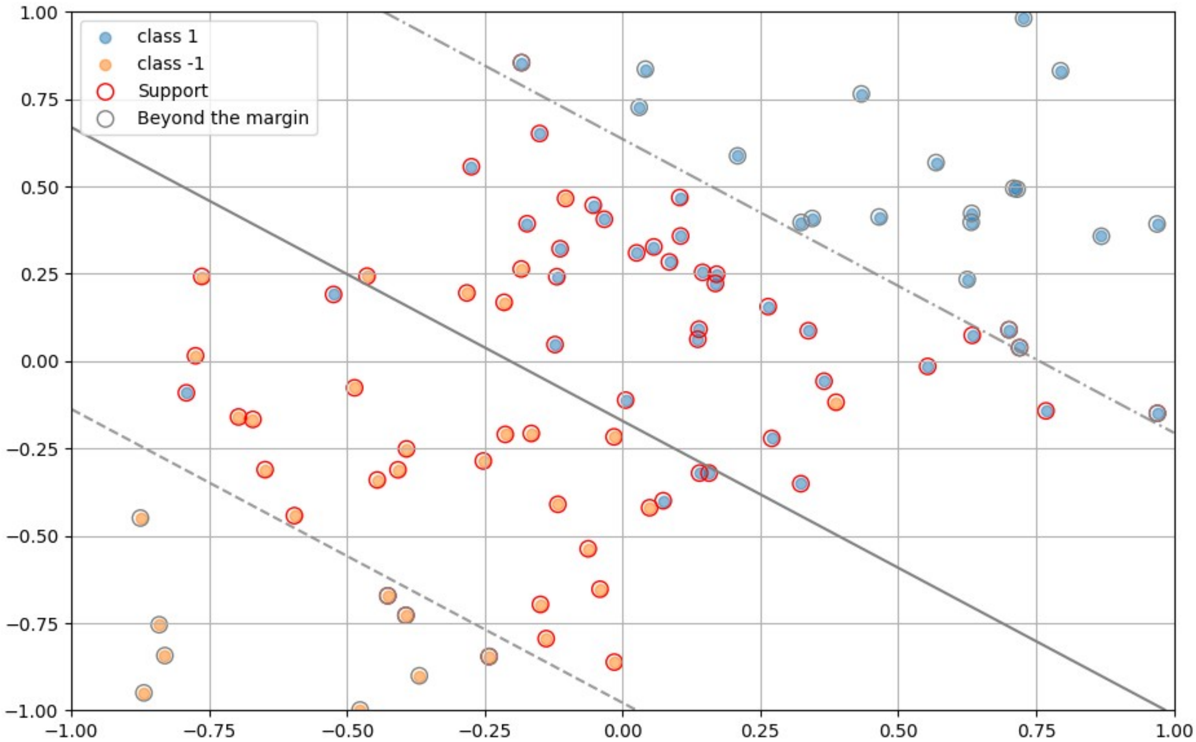


Dataset 2

Linear SVM

```
C=.1
kernel = Linear().kernel
model = KernelSVC(C=C, kernel=kernel, epsilon=1e-14)
train_dataset = datasets['dataset_2']['train']
model.fit(train_dataset['x'], train_dataset['y'])
plotClassification(train_dataset['x'], train_dataset['y'], model,
label='Training')
```

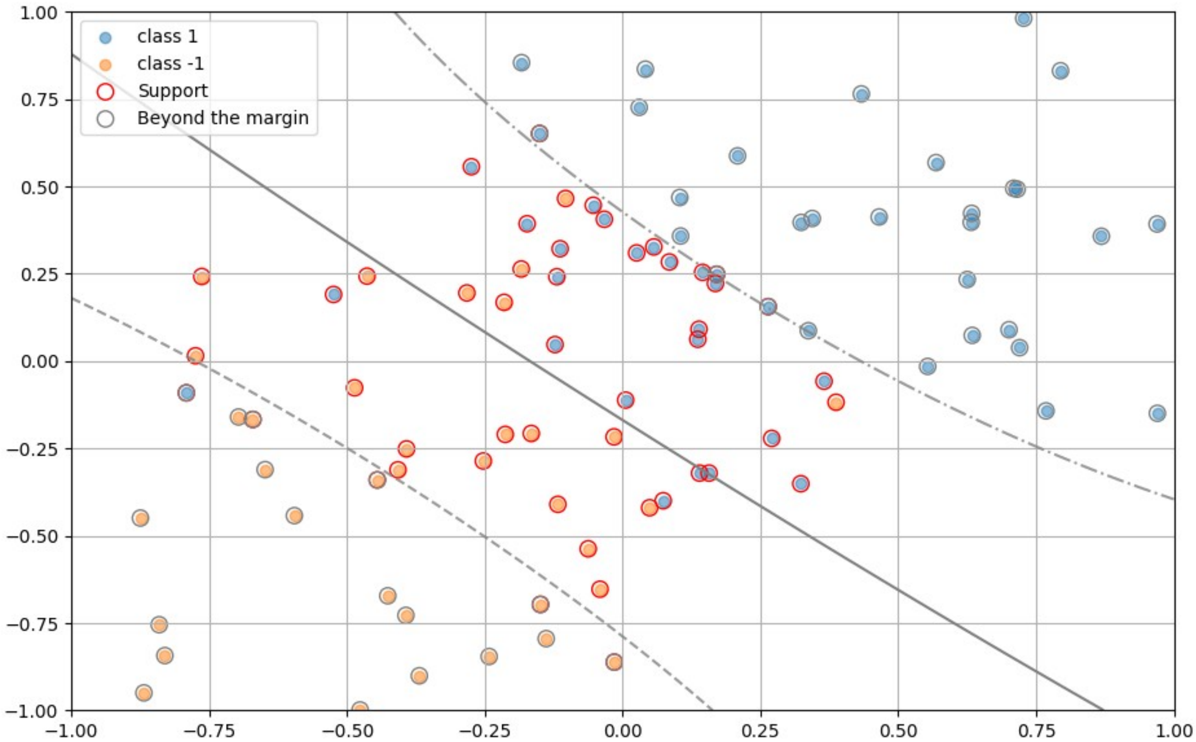
```
Number of support vectors = 67
```



Gaussian SVM

```
sigma = 1.5
C=1.
kernel = RBF(sigma).kernel
model = KernelSVC(C=C, kernel=kernel, epsilon=1e-14)
train_dataset = datasets['dataset_2']['train']
model.fit(train_dataset['x'], train_dataset['y'])
plotClassification(train_dataset['x'], train_dataset['y'], model,
label='Training')

Number of support vectors = 50
```

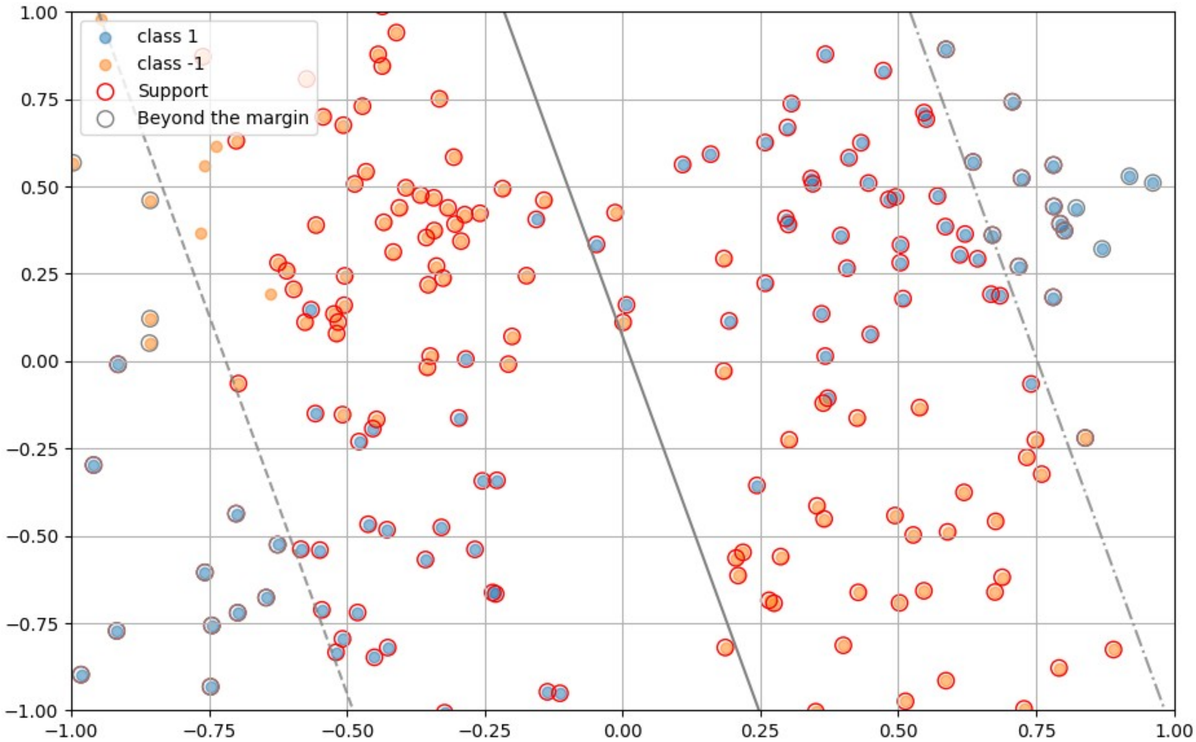


Dataset 3

Linear SVM

```
C=1.
kernel = Linear().kernel
model = KernelSVC(C=C, kernel=kernel, epsilon=1e-14)
train_dataset = datasets['dataset_3']['train']
model.fit(train_dataset['x'], train_dataset['y'])
plotClassification(train_dataset['x'], train_dataset['y'], model,
label='Training')
```

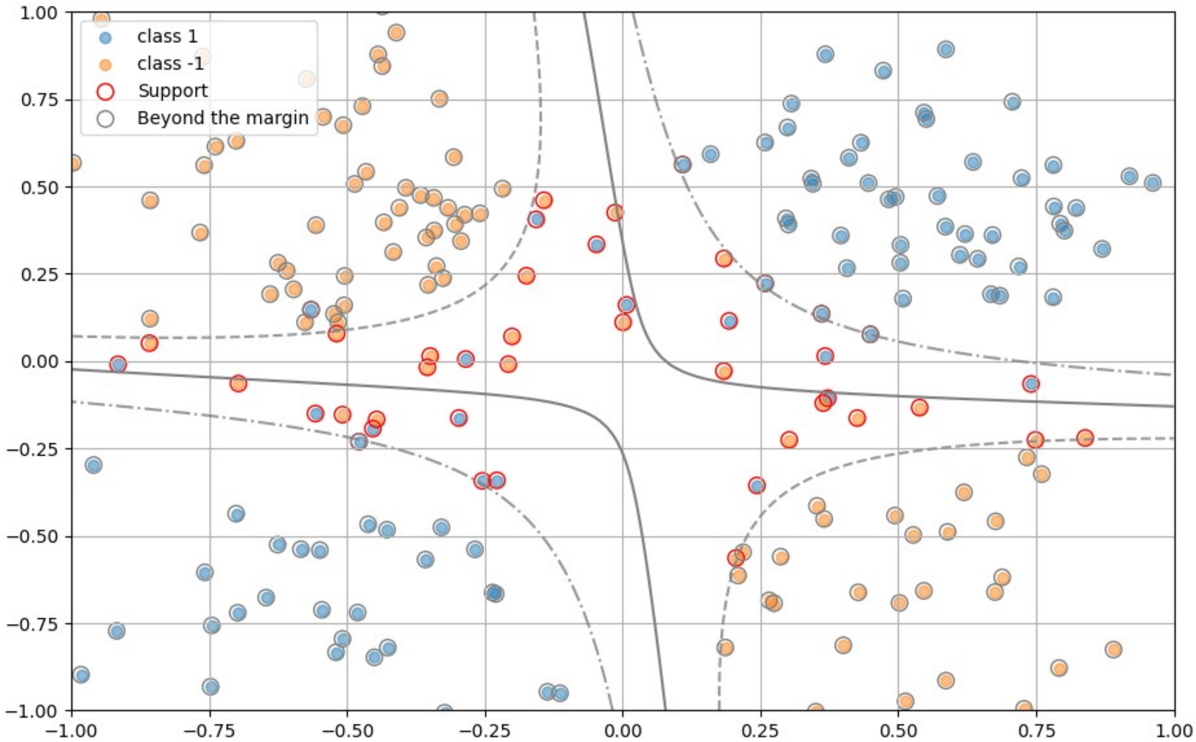
```
Number of support vectors = 185
```



Gaussian SVM

```
sigma = 1.5
C=100.
kernel = RBF(sigma).kernel
model = KernelSVC(C=C, kernel=kernel)
train_dataset = datasets['dataset_3']['train']
model.fit(train_dataset['x'], train_dataset['y'])
plotClassification(train_dataset['x'], train_dataset['y'], model,
label='Training')

Number of support vectors = 43
```


class 1
class -1
Support
Beyond the margin
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00