Kernel Support Vector Machines

In this exercise sheet, we will implement a kernel SVM. Our implementation will be based on a generic quadratic programming optimizer provided in CVXOPT (`python-cvxopt` package, or directly from the website `www.cvxopt.org`). The SVM will then be tested on the UCI breast cancer dataset, a simple binary classification dataset accessible via the `scikit-learn` library.

1. Building the Gaussian Kernel (5 P)

As a starting point, we would like to implement the Gaussian kernel, which we will make use of in our kernel SVM implementation. It is defined as:

$$k(x,x') = \exp\Big(-\frac{\|x-x'\|^2}{2\sigma^2}\Big)$$

- **Implement a function `getGaussianKernel` that returns for a Gaussian kernel of scale σ, the Gram matrix of the two data sets given as argument.**

In [1]:

```
import numpy,scipy,scipy.spatial

def getGaussianKernel(X1,X2,scale):
    ### TODO: REPLACE BY YOUR OWN CODE
    '''
    scipy.spatial.distance.cdist(X1, X2, 'sqeuclidean')
    是 SciPy 库中计算两个数据集之间成对距离的函数，用于计算每对样本之间的“平方欧几里得距离”（squared Euclidean distance）。

    参数解释：
    X1：第一个数据集，形状为 (m,n)(m,n)，表示有 mm 个样本，每个样本有 nn 个特征。
    X2：第二个数据集，形状为 (p,n)(p,n)，表示有 pp 个样本，每个样本也有 nn 个特征（两组样本的特征维度需要相同）。
    'sqeuclidean'：距离度量方法，表示计算平方欧几里得距离（squared Euclidean distance）。
    '''
    D=scipy.spatial.distance.cdist(X1,X2,'sqeuclidean')

    K = numpy.exp(-D/(2*scale**2))
    return K
    ###
```

2. Building the Matrices for the CVXOPT Quadratic Solver (20 P)

We would like to learn a nonlinear SVM by optimizing its dual. An advantage of the dual SVM compared to the primal SVM is that it allows to use nonlinear kernels such as the Gaussian kernel. The dual SVM consists of solving the following quadratic program:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \qquad \text{subject to:} \qquad 0 \leq \alpha_i \leq C \qquad \text{and} \qquad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

We would like to rely on a CVXOPT solver to obtain a solution to our SVM dual. The function `cvxopt.solvers.qp` solves an optimization problem of the type:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2} \boldsymbol{x}^\top P \boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} \\ \text{subject to} \quad & G\boldsymbol{x} \preceq \boldsymbol{h} \\ \text{and} \quad & A\boldsymbol{x} = \boldsymbol{b}. \end{aligned}$$

which is of similar form to our dual SVM (note that $\boldsymbol{x}$ will correspond to the parameters $(\alpha_i)_i$ of the SVM). We need to build the data structures (vectors and matrices) that makes solving this quadratic problem equivalent to solving our dual SVM.

- **Implement a function `getQPMatrices` that builds the matrices `P`, `q`, `G`, `h`, `A`, `b` (of type `cvxopt.matrix`) that need to be passed as argument to the optimizer `cvxopt.solvers.qp`.**

In [2]:

```
import cvxopt,cvxopt.solvers
cvxopt.solvers.options['show_progress'] = False

def getQPMatrices(K,T,C):
    ### TODO: REPLACE BY YOUR CODE
    N = T.shape[0]
    P = numpy.outer(T, T) * K
    q = -numpy.ones(N)
    G_std = -numpy.eye(N)
    G_slack = numpy.eye(N)
    G = numpy.vstack((G_std, G_slack))
    h = numpy.hstack((numpy.zeros(N), numpy.ones(N) * C))

    A = T.reshape(1, -1)
    b = numpy.array([0])
    P = cvxopt.matrix(P)
    q = cvxopt.matrix(q)
    G = cvxopt.matrix(G)
    h = cvxopt.matrix(h)
    A = cvxopt.matrix(A)
    b = cvxopt.matrix(b.astype(numpy.double))

    return P,q,G,h,A,b
    ###
```

3. Computing the Bias Parameters (10 P)

Given the parameters $(\alpha_i)_i$ the optimization procedure has found, the prediction of the SVM is given by:

$$f(x) = \text{sign}\Big(\sum_{i=1}^{N} \alpha_i y_i k(x, x_i) + \theta\Big)$$

Note that the parameter θ has not been computed yet. It can be obtained from any support vector that lies exactly on the margin, or equivalently, whose associated parameter α is not equal to 0 or C. Calling one such vector "x_M", the parameter θ can be computed as:

$$\theta = y_M - \sum_{j=1}^{N} \alpha_j y_j k(x_M, x_j)$$

- **Implement a function `getTheta` that takes as input the Gram Matrix used for training, the label vector, the solution of our quadratic program, and the hyperparameter C. The function should return the parameter θ.**

In [3]:

```
def getTheta(K,T,alpha,C):
    ### TODO: REPLACE BY YOUR CODE
    '''
    sv = (alpha > 1e-5) & (alpha < C - 1e-5)
    sv_index = numpy.where(sv)[0][0]
    theta = T[sv_index] - numpy.sum(alpha * T * K[sv_index, :])
    return theta
    '''
    sv=numpy.argmin(numpy.abs(alpha-C/2))
    theta = T[sv] - (K[sv] * alpha * T).sum()

    return theta
    ###
```

4. Implementing a class `GaussianSVM` (15 P)

All functions that are needed to learn the SVM have now been built. We would like to implement a SVM class that connects them and make the SVM easily usable. The class structure is given below and contains two functions, one for training the model, and one for applying it to test data.

- **Implement the function `fit` that makes use of the functions `getGaussianKernel`, `getQPMatrices`, `getTheta` you have already implemented. The function should learn the SVM model and store the support vectors, their label, $(\alpha_i)_i$ and θ into the object (`self`).**
- **Implement the function `predict` that makes use of the stored information to compute the SVM output for any new collection of data points**

In [4]:
```
class GaussianSVM:

    def __init__(self,C=1.0,scale=1.0):

        self.C, self.scale = C, scale

    def fit(self,X,T):

        ### TODO: REPLACE BY YOUR CODE
        self.X = X

        K = getGaussianKernel(X, X, self.scale)

        P,q,G,h,A,b = getQPMatrices(K,T,self.C)
        '''
        numpy.ravel 是一个将数组展平（flatten）为一维数组的函数。
        它返回的是原始数组视图（view）上的一维数组，因此通常不占用新的内存空间，除非数组不连续。
        '''
        solution = cvxopt.solvers.qp(P,q,G,h,A,b)
        alpha = numpy.ravel(solution['x'])


        self.alpha = alpha
        self.support_vectors = X[(alpha > 1e-5)]
        self.support_labels = T[(alpha > 1e-5)]


        self.theta = getTheta(K, T, alpha, self.C)
        ###

    def predict(self,X):

        ### TODO: REPLACE BY YOUR CODE
        K = getGaussianKernel(X, self.support_vectors, self.scale)

        Y = numpy.sign(K @ (self.alpha[(self.alpha > 1e-5)] * self.support_labels) + self.theta)
        return Y
        ###
```

5. Analysis

The following code tests the SVM on some breast cancer binary classification dataset for a range of scale and soft-margin parameters. For each combination of parameters, we output the number of support vectors as well as the train and test accuracy averaged over a number of random train/test splits. Running the code below should take approximately 1-2 minutes.

In [5]:

```
import numpy,sklearn,sklearn.datasets,numpy

D = sklearn.datasets.load_breast_cancer()
X = D['data']
T = D['target']
```

```
T = (D['target']==1)*2.0-1.0

for scale in [30,100,300,1000,3000]:
    for C in [10,100,1000,10000]:

        acctrain,acctest,nbsvs = [],[],[]

        svm = GaussianSVM(C=C,scale=scale)

        for i in range(10):

            # Split the data
            R = numpy.random.mtrand.RandomState(i).permutation(len(X))
            Xtrain,Xtest = X[R[:len(R)//2]]*1,X[R[len(R)//2:]]*1
            Ttrain,Ttest = T[R[:len(R)//2]]*1,T[R[len(R)//2:]]*1

            # Train and test the SVM
            svm.fit(Xtrain,Ttrain)
            acctrain += [(svm.predict(Xtrain)==Ttrain).mean()]
            acctest  += [(svm.predict(Xtest)==Ttest).mean()]
            nbsvs += [len(svm.X)*1.0]

        print('scale=%9.1f  C=%9.1f  nSV: %4d  train: %.3f  test: %.3f'%(
            scale,C,numpy.mean(nbsvs),numpy.mean(acctrain),numpy.mean(acctest)))
    print('')
```

```
scale=    30.0  C=     10.0  nSV:  284  train: 0.997  test: 0.921
scale=    30.0  C=    100.0  nSV:  284  train: 1.000  test: 0.918
scale=    30.0  C=   1000.0  nSV:  284  train: 1.000  test: 0.918
scale=    30.0  C=  10000.0  nSV:  284  train: 1.000  test: 0.918

scale=   100.0  C=     10.0  nSV:  284  train: 0.965  test: 0.935
scale=   100.0  C=    100.0  nSV:  284  train: 0.987  test: 0.940
scale=   100.0  C=   1000.0  nSV:  284  train: 0.998  test: 0.932
scale=   100.0  C=  10000.0  nSV:  284  train: 1.000  test: 0.926

scale=   300.0  C=     10.0  nSV:  284  train: 0.939  test: 0.924
scale=   300.0  C=    100.0  nSV:  284  train: 0.963  test: 0.943
scale=   300.0  C=   1000.0  nSV:  284  train: 0.978  test: 0.946
scale=   300.0  C=  10000.0  nSV:  284  train: 0.991  test: 0.941

scale=  1000.0  C=     10.0  nSV:  284  train: 0.926  test: 0.916
scale=  1000.0  C=    100.0  nSV:  284  train: 0.935  test: 0.929
scale=  1000.0  C=   1000.0  nSV:  284  train: 0.956  test: 0.946
scale=  1000.0  C=  10000.0  nSV:  284  train: 0.971  test: 0.951

scale=  3000.0  C=     10.0  nSV:  284  train: 0.912  test: 0.903
scale=  3000.0  C=    100.0  nSV:  284  train: 0.926  test: 0.919
scale=  3000.0  C=   1000.0  nSV:  284  train: 0.933  test: 0.929
scale=  3000.0  C=  10000.0  nSV:  284  train: 0.953  test: 0.944
```

We observe that the highest accuracy is obtained with a scale parameter that is neither too small nor too large. Best parameters are also often associated to a low number of support vectors.

Exercises for the course

Machine Learning 1

Winter semester 2024/25

Fachgebiet Maschinelles Lernen
Institut für Softwaretechnik und theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Exercise Sheet 6

Exercise 1: Dual formulation of the Soft-Margin SVM $(5 + 20 + 10 + 5$ P$)$

The primal program for the linear soft-margin SVM is

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

subject to

$$\forall_{i=1}^{N}: \; y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

where $\|.\|$ denotes the Euclidean norm, ϕ is a feature map, $\boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}$ are the parameter to optimize, and $\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$ are the labeled data points regarded as fixed constants. Once the hard-margin SVM has been learned, prediction for any data point $\boldsymbol{x} \in \mathbb{R}^d$ is given by the function

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b).$$

(a) *State* the conditions on the data under which a solution to this program can be found from the Lagrange dual formulation *(Hint: verify the Slater's conditions).*

(b) *Derive* the Lagrange dual and show that it reduces to a constrained quadratic optimization problem. State both the objective function and the constraints of this optimization problem.

(c) *Describe* how the solution $(\boldsymbol{w}, b)$ of the primal program can be obtained from a solution of the dual program.

(d) *Write* a kernelized version of the dual program and of the learned decision function.

Exercise 2: SVMs and Quadratic Programming (10 P)

We consider the CVXOPT Python software for convex optimization. The method `cvxopt.solvers.qp` solves quadratic optimization problems given in the matrix form:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^\top P\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} \\ \text{subject to} \quad & G\boldsymbol{x} \preceq \boldsymbol{h} \\ \text{and} \quad & A\boldsymbol{x} = \boldsymbol{b}. \end{aligned}$$

Here, $\preceq$ denotes the element-wise inequality: $(\boldsymbol{h} \preceq \boldsymbol{h}') \Leftrightarrow (\forall_i : h_i \leq h_i')$. Note that the meaning of the variables $\boldsymbol{x}$ and $\boldsymbol{b}$ is different from that of the same variables in the previous exercise.

(a) *Express* the matrices and vectors $P, \boldsymbol{q}, G, \boldsymbol{h}, A, \boldsymbol{b}$ in terms of the variables of Exercise 1, such that this quadratic minimization problem corresponds to the kernel dual SVM derived above.

Exercise 3: Programming (50 P)

Download the programming files on ISIS and follow the instructions.

Exercise 1: Dual formulation of the Soft-Margin SVM (5 + 20 + 10 + 5 P)

The primal program for the linear soft-margin SVM is

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

subject to

$$\forall_{i=1}^{N}: \; y_i \cdot (\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

where $\|.\|$ denotes the Euclidean norm, ϕ is a feature map, $\boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}$ are the parameter to optimize, and $\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$ are the labeled data points regarded as fixed constants. Once the hard-margin SVM has been learned, prediction for any data point $\boldsymbol{x} \in \mathbb{R}^d$ is given by the function

$$f(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^\top \phi(\boldsymbol{x}) + b).$$

(a) *State* the conditions on the data under which a solution to this program can be found from the Lagrange dual formulation *(Hint: verify the Slater's conditions)*.

Solution: ? ? ?

To verify the Slater's conditions, for any w and ρ, we can always increase ξ_i until constraints become strict inequalities.

$$\forall_{i=1}^{N}: y_i(w^T\phi(x_i)+b) > 1 - \xi_i \quad \text{and} \quad \xi_i > 0$$

So we can always find a feasible solution for the primal problem.

always increase ξ_i until constraints are satifisfied with strict inequalities

Then the dual problem can be written as:

$$\min_{\alpha} \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j k(x_i, x_j) - \sum_i \alpha_i$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \;, \forall i$$

$$\sum_i \alpha_i y_i = 0$$

so if we set $\alpha_i = \frac{1}{N}$, then we have.

$$0 < \alpha_i < C \;, \quad \sum_{i=1}^{N} \alpha_i y_i = 1$$

∴ there is also a feasible solution for the dual problem

∴ Then the strong duality holds. ✓

(b) *Derive* the Lagrange dual and show that it reduces to a constrained quadratic optimization problem. State both the objective function and the constraints of this optimization problem.

Solution:

Use the Lagrange method to derive the Lagrange dual.

the problem can be reformulated as

$$\max_{\alpha,\beta} \min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i + \sum_i \alpha_i \left[1 - \xi_i - y_i(w^T\phi(x_i)+b)\right] + \sum_i \beta_i(-\xi_i)$$

✓

$$\text{s.t.} \quad \alpha_i \geq 0 \qquad \forall i$$
$$\beta_i \geq 0$$

Then we compute the partial derivatives w.r.t (w, b, ξ_i), and set them to 0

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_i \alpha_i y_i \phi(x_i) = 0$$ ✓

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_i \alpha_i y_i = 0$$ ✓

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \beta_i = 0$$ ✓

→

$$w^* = \sum_i \alpha_i y_i \phi(x_i)$$

$$\sum_i \alpha_i y_i = 0$$

$$\beta_i = C - \alpha_i \Rightarrow \alpha_i \leq C$$

Then the Lagrange dual can be written as:

$$\max_{\alpha} \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \phi^T(x_i)\phi(x_j) + \sum_i \alpha_i - \sum_i \alpha_i y_i\left(\sum_j \alpha_j y_j \phi^T(x_j)\right)\phi(x_i)$$

$$= \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \phi^T(x_i)\phi(x_j)$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \ \forall i, \quad \sum_i \alpha_i y_i = 0$$ ✓

(c) *Describe* how the solution $(\boldsymbol{w}, b)$ of the primal program can be obtained from a solution of the dual program.

Solution.

We need to derive (w, ξ, b)

① w

$$w = \sum_i \alpha_i y_i \phi(x_i)$$ ✓

② ξ_i

if $\alpha_i < C$, then $\beta_i > 0$

$$\beta_i(-\xi_i) = 0$$

$$\xi_i = 0$$ ✓

③ b

$$\alpha_i(1 - \xi_i - y_i(w^T\phi(x_i) + b)) = 0$$ ✓

if $\alpha_i > 0$, then

$$1 - \xi_i - y_i(w^T\phi(x_i) + b) = 0$$

$$y_i(w^T\phi(x_i) + b) = 1 - \xi_i$$

since $y_i = \pm 1 \rightarrow \frac{1}{y_i} = y_i$

$\therefore$ $$b = y_i(1 - \xi_i) - w^T\phi(x_i)$$

since $\xi_i = 0$

$\therefore$ $$b = y_i - w^T\phi(x_i)$$ ✓ $0 < \alpha < C$

b is the soft margin factor. Therefore, we can conclude that the points in the margin have penalty factor $\alpha_i = C$, the support vectors have penalty factor $0 < \alpha_i < C$

then the other points has 0 penalty factor.

(d) *Write* a kernelized version of the dual program and of the learned decision function.

Solution:

define a kernel function as $k(x_i, x_j) = \phi^T(x_i)\,\phi(x_j)$ ↗ feature map

$$\boxed{\begin{aligned} &\max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j\, k(x_i, x_j) \\ &\text{s.t.} \quad 0 \leqslant \alpha_i \leqslant C, \ \forall i \\ &\qquad\quad \sum_i \alpha_i y_i = 0 \end{aligned}}$$

✓

$$\boxed{\begin{aligned} &\min_{\alpha} \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j\, k(x_i, x_j) - \sum_i \alpha_i \\ &\text{s.t.} \quad 0 \leqslant \alpha_i \leqslant C, \ \forall i \\ &\qquad\quad \sum_i \alpha_i y_i = 0 \end{aligned}}$$

The learned decision function can be written as

$$\boxed{\begin{aligned} f(x) &= \operatorname{sign}(w^T x + b) \\ &= \operatorname{sign}\Big(\sum_i \alpha_i y_i \phi^T(x_i)\,\phi(x) + b\Big) \\ &= \operatorname{sign}\Big(\sum_i \alpha_i y_i\, k(x_i, x) + b\Big) \end{aligned}}$$

✓

Exercise 2: SVMs and Quadratic Programming (10 P)

We consider the CVXOPT Python software for convex optimization. The method `cvxopt.solvers.qp` solves quadratic optimization problems given in the matrix form:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^\top P\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} \\ \text{subject to} \quad & G\boldsymbol{x} \preceq \boldsymbol{h} \\ \text{and} \quad & A\boldsymbol{x} = \boldsymbol{b}. \end{aligned}$$

Here, $\preceq$ denotes the element-wise inequality: $(\boldsymbol{h} \preceq \boldsymbol{h}') \Leftrightarrow (\forall_i : h_i \leq h'_i)$. Note that the meaning of the variables $\boldsymbol{x}$ and $\boldsymbol{b}$ is different from that of the same variables in the previous exercise.

(a) *Express* the matrices and vectors $P, \boldsymbol{q}, G, \boldsymbol{h}, A, \boldsymbol{b}$ in terms of the variables of Exercise 1, such that this quadratic minimization problem corresponds to the kernel dual SVM derived above.

Solution:

$$\boxed{\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x^T P x + q^T x \\ \text{s.t.} \quad & \boxed{G}\,x \preceq h \\ & \boxed{A}\,x = \boxed{b} \end{aligned}} \Rightarrow \boxed{\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^T y^T K y\, \alpha - \mathbb{1}^T\alpha \\ \text{s.t.} \quad & \boxed{\begin{bmatrix} -\alpha \\ \alpha \end{bmatrix}} \preceq \begin{bmatrix} 0 \\ C \end{bmatrix} \end{aligned}}$$

Index der Kommentare

10.1 don't skip steps