Machine Learning Exercise sheet 08

SVM and Kernels

Stamatouli Anastasia : 03710902

08/12/19

• SVM

Problem (1) :

Similarities :

1. Both are linear classifiers
2. linear seperable data
3. Both can apply the context of a hard-decision based classifier
4. Both can be adjusted in order to perform multi-class classification (one-versus-one)
5. Both can use Kernels in order to transform data in a dimension that is linear seperable.

Differences :

1. SVM seperates the classes with the maximum margin while perceptron looks only for linear separation
2. SVM is optimizing a constrained problem while perceptron is an iterative method
3. Perceptron can be trained online, while SVM not.

Problem 2 :

a) $w = \sum_{i=1}^{N} a_i y_i x_i$ (1)

$\sum_{i=1}^{N} a_i y_i = 0$ (2)

We are plugging back into $L(w, b, a) = \frac{1}{2} w^T w - \sum_{i=1}^{N} a_i [y_i (w^T x_i + b) - 1]$

(3)

$$(3) \underset{2}{\overset{1}{\Longrightarrow}} \frac{1}{2} \left\| \sum_{i=1}^{N} a_i y_i x_i \right\|_2^2 + \sum_{i=1}^{N} a_i \left[1 - y_i \left[\left(\sum_{i=1}^{N} a_i y_i x_i \right)^T x_i + b \right] \right]$$

$$= \frac{1}{2} a^T Q a + \sum_{i=1}^{N} a_i - a^T Q a - b \sum_{i=1}^{N} a_i y_i =$$

$$-\frac{1}{2} a^T Q a + 1^T a \quad (4)$$

where $Q = \begin{bmatrix} y_1 & 0 & 0 \\ 0 & \vdots & \vdots \\ 0 & 0 & y_i \end{bmatrix} \begin{bmatrix} - x_1^T - \\ \vdots \\ - x_i^T - \end{bmatrix} \begin{bmatrix} | & & | \\ x_1 & \cdots & x_i \\ | & & | \end{bmatrix} \begin{bmatrix} y_1 & 0 & 0 \\ 0 & \vdots & \\ 0 & 0 & y_i \end{bmatrix}$

$$= Y^T (X^T X) Y = Y^T K Y$$

K: Kernel matrix

b) We need to show that $v Y^T X^T X Y v^T \geq 0$ for every $v \in \mathbb{R}^N$.

(5)

$(5) \Rightarrow (X Y v^T)^T (X Y v^T) = \| X Y v^T \|_2^2 \geq 0$

Therefore Q positive semidefinite

But from (4) we need to incorporate (-) sign.

Therefore, Q negative semi-definite and we can find global maximum.

Problem ③ :

We define misclassification rate as the number of misclassified elements over the amount of experiments. In Leave-one-out cross validation the amount of experiments equals the number of data N.

$$\varepsilon = \frac{t}{N} \overset{n=N}{=} \frac{t}{N}$$

In our case, there can be misclassification error only when we remove support vector, otherwise there is no misclassification.

So in case we remove support vector, the only elements which can be misclassified are the sv. So at most we will have s out of N misclassified elements,

which means that $0 \leqslant \varepsilon \leqslant \frac{s}{N}$

- Kernels

Problem ⑤ :

In order to show that $K(X_1, X_2)$ is a valid Kernel we need to show that it can be derived from Kernel preserving operations.

$K_1(X_1, X_2) = (X_1^T X_2)^i \rightarrow$ polynomial kernel

$K_2(X_1, X_2) = a \cdot K_1(X_1, X_2) \rightarrow$ multiplication of valid Kernel with $a \geqslant 0$

$\sum_{i=1}^{N} K_2(X_1, X_2) = K_3(X_1, X_2) \rightarrow$ sum of valid Kernels

$K_4(X_1, X_2) = K_3(X_1, X_2) + a_0 \rightarrow$ adding constant >0 to valid Kernel

Therefore $K_4(X_1, X_2) = K(X_1, X_2) =$ valid Kernel

Problem 6 :

We will re-write $K(X_1, X_2) = \dfrac{1}{1 - X_1 X_2} = \sum_{n=0}^{\infty} (X_1 X_2)^n$

using Maclaurin Series.

Therefore, $\varphi(x) = (x^0, x^1, x^2, \dots)$

exercise_08_notebook

December 8, 2019

1 Programming assignment 4: SVM

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import make_blobs

from cvxopt import matrix, solvers
```

1.1 Your task

In this sheet we will implement a simple binary SVM classifier. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

To solve optimization tasks we will use `CVXOPT` http://cvxopt.org/ - a Python library for convex optimization. If you use `Anaconda`, you can install it using

```
conda install cvxopt
```

1.2 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

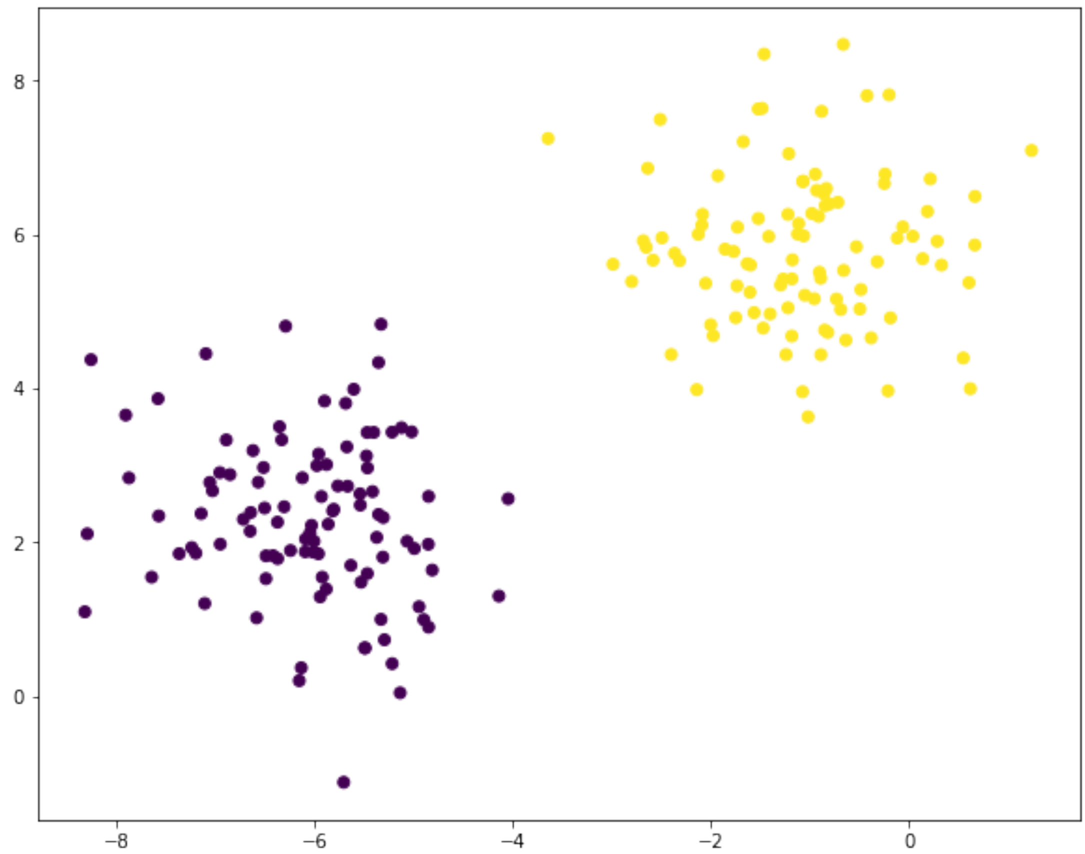
Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.3 Generate and visualize the data

[3]:
```
N = 200  # number of samples
D = 2  # number of dimensions
C = 2  # number of classes
seed = 1234  # for reproducible experiments

alpha_tol = 1e-4 # threshold for choosing support vectors

X, y = make_blobs(n_samples=N, n_features=D, centers=C, random_state=seed)
y[y == 0] = -1  # it is more convenient to have {-1, 1} as class labels (instead␣
 ↪of {0, 1})
y = y.astype(np.float)
plt.figure(figsize=[10, 8])
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```



1.4 Task 1: Solving the SVM dual problem

Remember, that the SVM dual problem can be formulated as a Quadratic programming (QP) problem. We will solve it using a QP solver from the `CVXOPT` library.

We use the following form of a QP problem:

$$\text{minimize}_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}\text{subject to} \quad \mathbf{G}\mathbf{x} \leq \mathbf{h} \text{ and } \mathbf{A}\mathbf{x} = \mathbf{b}\,.$$

Your task is to formulate the SVM dual problems as a QP of this form and solve it using `CVXOPT`, i.e. specify the matrices $\mathbf{P}, \mathbf{G}, \mathbf{A}$ and vectors $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

[4]:
```
def solve_dual_svm(X, y):
    """Solve the dual formulation of the SVM problem.

    Parameters
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    alphas : array, shape [N]
        Solution of the dual problem.
    """
    # TODO
    # These variables have to be of type cvxopt.matrix
    P = y[:, None] * X
    P = matrix(P @ P.T)
    q = matrix(-np.ones((len(y), 1)))
    G = matrix(-np.eye(len(y)))
    h = matrix(np.zeros((len(y), 1)))
    A = matrix(y.reshape(1, -1))
    b = matrix(np.zeros(1))
    solvers.options['show_progress'] = False
    solution = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solution['x'])
    return alphas.reshape(-1)
```

1.5 Task 2: Recovering the weights and the bias

[8]:
```
def compute_weights_and_bias(alpha, X, y):
    """Recover the weights w and the bias b using the dual solution alpha.

    Parameters
    ----------
```

```
    alpha : array, shape [N]
        Solution of the dual problem.
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
    """
    w = (alpha.reshape(1, -1) * y) @ X
    b = np.mean((y - w @ X.T))
    return w.T, b
```

1.6 Visualize the result (nothing to do here)

[9]:
```
def plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b):
    """Plot the data as a scatter plot together with the separating hyperplane.

    Parameters
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).
    alpha : array, shape [N]
        Solution of the dual problem.
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
    """
    plt.figure(figsize=[10, 8])
    # Plot the hyperplane
    slope = -w[0] / w[1]
    intercept = -b / w[1]
    x = np.linspace(X[:, 0].min(), X[:, 0].max())
    plt.plot(x, x * slope + intercept, 'k-', label='decision boundary')
    plt.plot(x, x * slope + intercept - 1/w[1], 'k--')
    plt.plot(x, x * slope + intercept + 1/w[1], 'k--')
    # Plot all the datapoints
    plt.scatter(X[:, 0], X[:, 1], c=y)
```

```
    # Mark the support vectors
    support_vecs = (alpha > alpha_tol)
    plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs],␣
↪s=250, marker='*', label='support vectors')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.legend(loc='upper left')
```

The reference solution is

`w = array([0.73935606 0.41780426])`

`b = 0.919937145`

Indices of the support vectors are

`[ 78 134 158]`

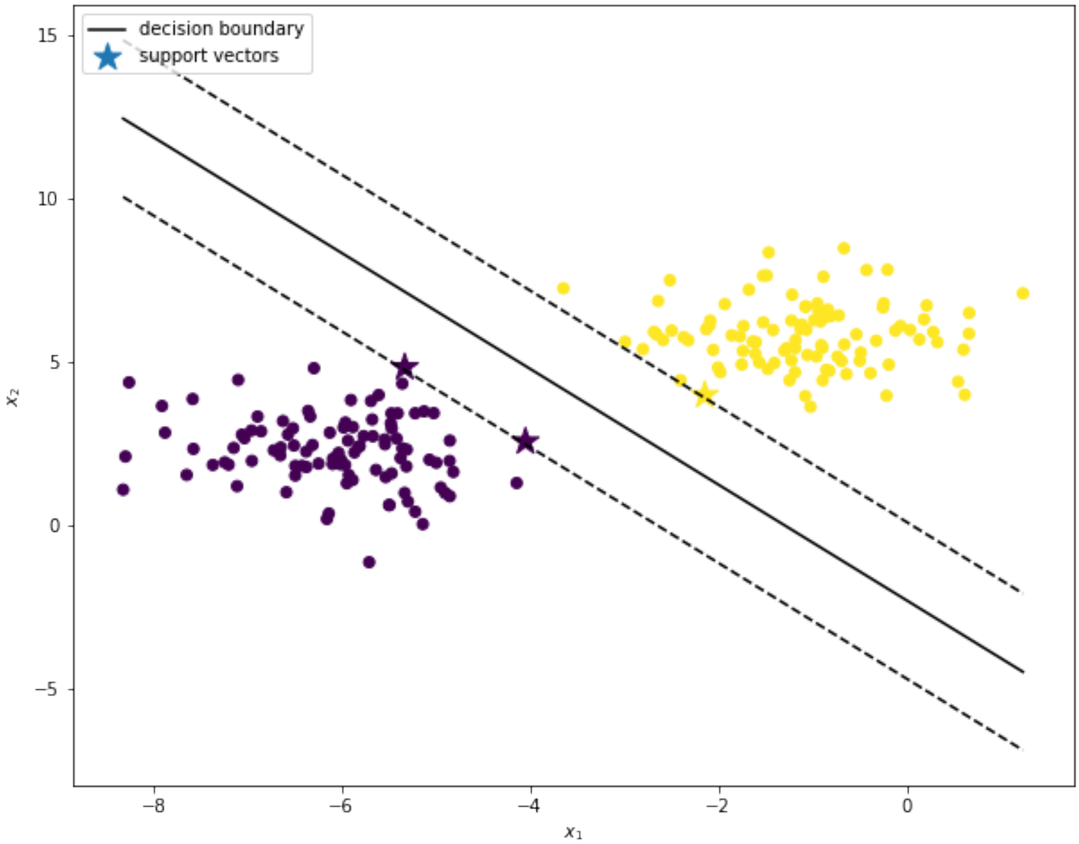
[10]:

```
alpha = solve_dual_svm(X, y)
w, b = compute_weights_and_bias(alpha, X, y)
print("w =", w)
print("b =", b)
print("support vectors:", np.arange(len(alpha))[alpha > alpha_tol])
```

```
w = [[0.73935606]
 [0.41780426]]
b = 0.9545661359845496
support vectors: [ 78 134 158]
```

[11]:

```
plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
plt.show()
```


decision boundary
support vectors
15
10
5
0
−5
−8
−6
−4
−2
0
x_1
x_2

[ ]: