Exercise Sheet 09

Problem 6

- Both are binary classification problems using classes labeled with $\{-1, 1\}$. Both make use of a hyperplane to classify datapoints. The soft margin SVM can be formalated with the hinge loss (not just hinge-loss). The perceptron algorithm also uses hinge-loss.
- The perceptron algorithm uses all datapoints to estimate weight w. But SVM just use the support vectors. Additionally, SVM tries to maximize margin by minimizing $\frac{1}{2} w^T w$. This term is not present in the perceptron algorithm.

Problem 7

(a) $\alpha^T \cdot 1_N = \sum_{i=1}^{N} \alpha_i$

$$\left.\begin{aligned} &\frac{1}{2} \alpha^T \cdot v = \frac{1}{2} \left(\sum_{i=1}^{N} \alpha_i \cdot v_i \right) \\ &Q \cdot \alpha = \begin{matrix} \sum_{j=1}^{N} q_{1j} \cdot \alpha_j \\ \vdots \\ \sum_{j=1}^{N} q_{Nj} \cdot \alpha_j \end{matrix} \end{aligned}\right\} \quad q_{ij} = -y_i \cdot y_j \cdot x_i^T \cdot x_j$$

$$y \cdot y^T = \begin{pmatrix} y_1 y_1 & y_1 y_2 & \cdots & y_1 y_N \\ y_2 y_1 & y_2 y_2 & \cdots & y_2 y_N \\ \vdots & \vdots & \ddots & \vdots \\ y_N y_1 & y_N y_2 & \cdots & y_N \cdot y_N \end{pmatrix} \quad X \cdot X^T = \begin{pmatrix} x_1^T x_1 & x_1^T x_2 & \cdots & x_1^T \cdot x_N \\ x_2^T \cdot x_1 & x_2^T \cdot x_2 & \cdots & x_2^T \cdot x_N \\ \vdots & \vdots & \ddots & \vdots \\ x_N^T \cdot x_1 & x_N^T \cdot x_2 & \cdots & x_N^T x_N \end{pmatrix}$$

$$Q = -(Y \cdot Y^T) \odot (X \cdot X^T)$$

b) First we show that Q is symmetric negative (semi-) definite

Symmetric

- $(-y \cdot y^T)^T = y^{TT} \cdot -y^T = -y \cdot y^T \Rightarrow -y \cdot y^T$ is symmetric
- $(X \cdot X^T)^T = X^{TT} \cdot X^T = X \cdot X^T \Rightarrow (X \cdot X^T)$ is symmetric

Since Hadamard product is elementwise multiplication Q is also symmetric

$y \cdot y^T$ is positive (semi-) definite

$$a^T \cdot (y \cdot y^T) \cdot a = (a^T \cdot y) \cdot (y^T a) = (\Sigma a_i \cdot y_i)^2 \geq 0$$

$\Rightarrow y \cdot y^T$ is positive (semi-) definit

$-y \cdot y^T$ is negative (semi-) definite

$y \cdot y^T$ positive (semi-) definite $\Rightarrow -y \cdot y^T$ is negativ semi-) definite

$X \cdot X^T$ is positive (semi-) definite

$$a^T (X \cdot X^T) \cdot a = (a^T \cdot X)(X^T \cdot a) = (X^T \cdot a)^T (X^T \cdot a) \geq 0$$

$\Rightarrow X \cdot X^T$ positive (semi-) definite with Schur theorem, we know that

$-Q = (y \cdot y^T) \odot (x \cdot x^T)$ is positive (semi-)definite

$\Rightarrow Q$ is negative (semi-)definite

The Hessian of g is Q. Since Q is negative (semi-)definite the function is concave and therefore the local maxima is the global maxima

Problem 8

When making LOOCV, we differentiate between 2 case at each iteration:

- Case 1: We leave a vector that is not a support vector

Since the data is linear differentiable and SVM is trained with all support vectors, we know that there won't exist missclassifications.

There is a total of $N-S$ such cases

- Case 2: We leave a support vector γ out.

When leaving γ out the SVM might use other vector that make that γ is inside margin and missclassified. It could still find a solution where γ is correctly classified.

This case occurs s times. Therefore at most we have s misclassifications

All ~~the~~ in all, follows: $\varepsilon \leq \frac{s}{N}$

Problem 10

$$K(x_1, x_2) = \sum_{i=1}^{N} a_i (x_1^T \cdot x_2)^i + a_0 = \sum_{i=0}^{N} a_i (x_1^T \cdot x_2)^i$$

- $(x_1^T \cdot x_2)^i$ are polynomial Kernels
- $a_i (x_1^T \cdot x_2)^i$ are valid Kernels ~~rule~~ for $a_i \geq 0$; rules of slide 39
- The sum is also a Kernel preserving operation
- In the cases where $a_i = 0$, we ignore them. Then we still have sum of Kernels
- In the case where all $a_i = 0$ then we can use $\phi(x) = 0$ and $K(x_1, x_2) = \phi(x_1)^T \cdot \phi(x_2) = 0$

Problem 11

We use geometric series

$$K(x_1, x_2) = \frac{1}{1 - x_1 x_2} = 1 + x_1 \cdot x_2 + (x_1 \cdot x_2)^2 + \ldots = \sum_{k=0}^{\infty} (x_1 \cdot x_2)^k$$

$$= \sum_{k=0}^{\infty} x_1^k \cdot x_2^k = (x_1^0 \;\; x_1^1 \;\; x_1^2 \ldots) \begin{pmatrix} x_2^0 \\ x_2^1 \\ x_2^2 \\ \vdots \end{pmatrix} = \phi(x_1)^T \cdot \phi(x_2)$$

with $\phi: \mathbb{R} \to \mathbb{R}^{\infty}$

$$\phi(x) \mapsto \begin{pmatrix} x^0 \\ x^1 \\ x^2 \\ \vdots \end{pmatrix}$$

exercise_09_notebook

December 22, 2021

1 Programming assignment 4: SVM

[1]:
```
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.

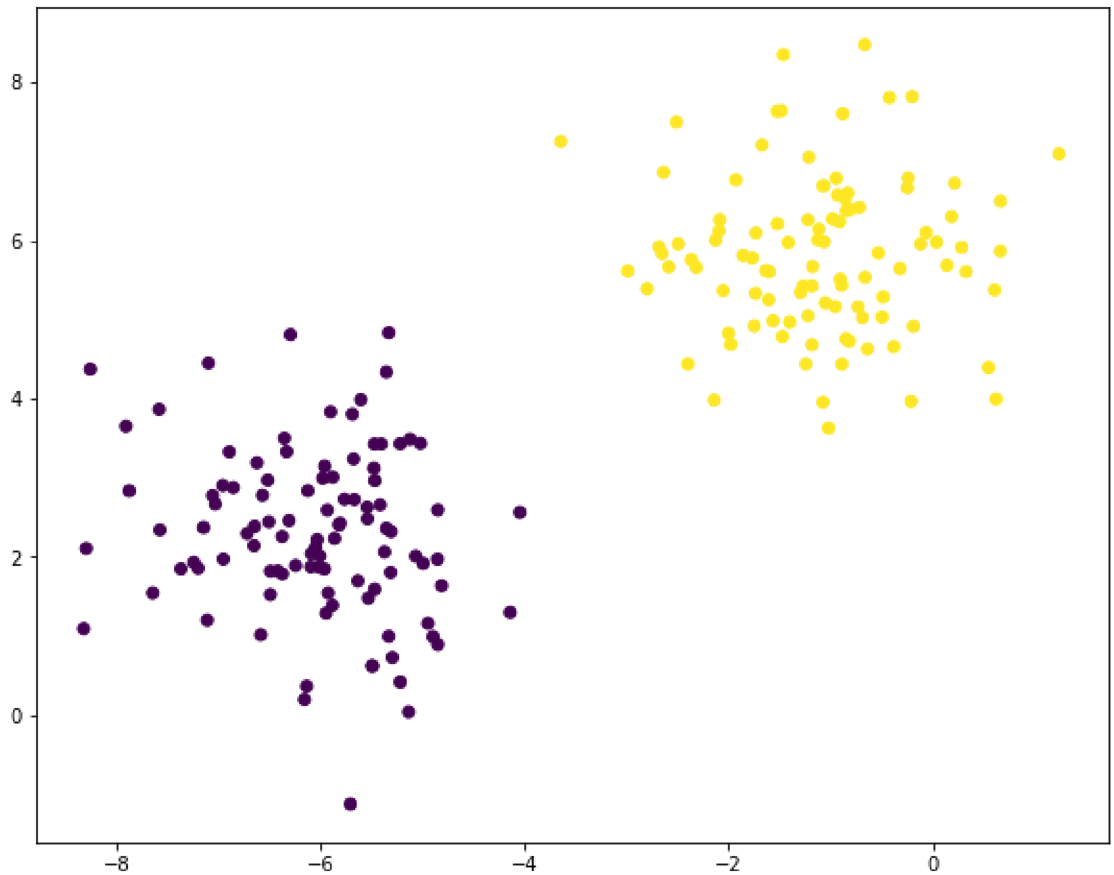
1.3 Generate and visualize the data

[2]:
```
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 $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

[3]:
```
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: Your code below ###
    # These variables have to be of type cvxopt.matrix
    N, D = X.shape
    Q = -np.multiply(np.outer(y, np.transpose(y)), np.matmul(X, np.
 ↪transpose(X)))

    P = matrix(-Q)
    q = matrix(-np.ones(N))
    G = matrix(-1.0 * np.identity(N))
    h = matrix(np.zeros(N))
    A = matrix(y, (1,N))
    b = matrix(0.)
    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

[4]:
```
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.
    """
    ### TODO: Your code below ###
    support_indices = np.argwhere(alpha > alpha_tol).flatten()

    N, D = X.shape
    #   w = np.
↪sum(alpha[support_indices]*y[support_indices]*X[support_indices], axis=0)
    vals = alpha*y
    vals = np.repeat(vals[:, np.newaxis], D, axis=1)
    mult = np.multiply(vals[support_indices], X[support_indices])
    w = np.sum(mult, axis=0)
    ind = support_indices[0]
    b = y[ind] - np.transpose(w).dot(X[ind])
    return w, b
```

1.6 Visualize the result (nothing to do here)

[5]:
```
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
```

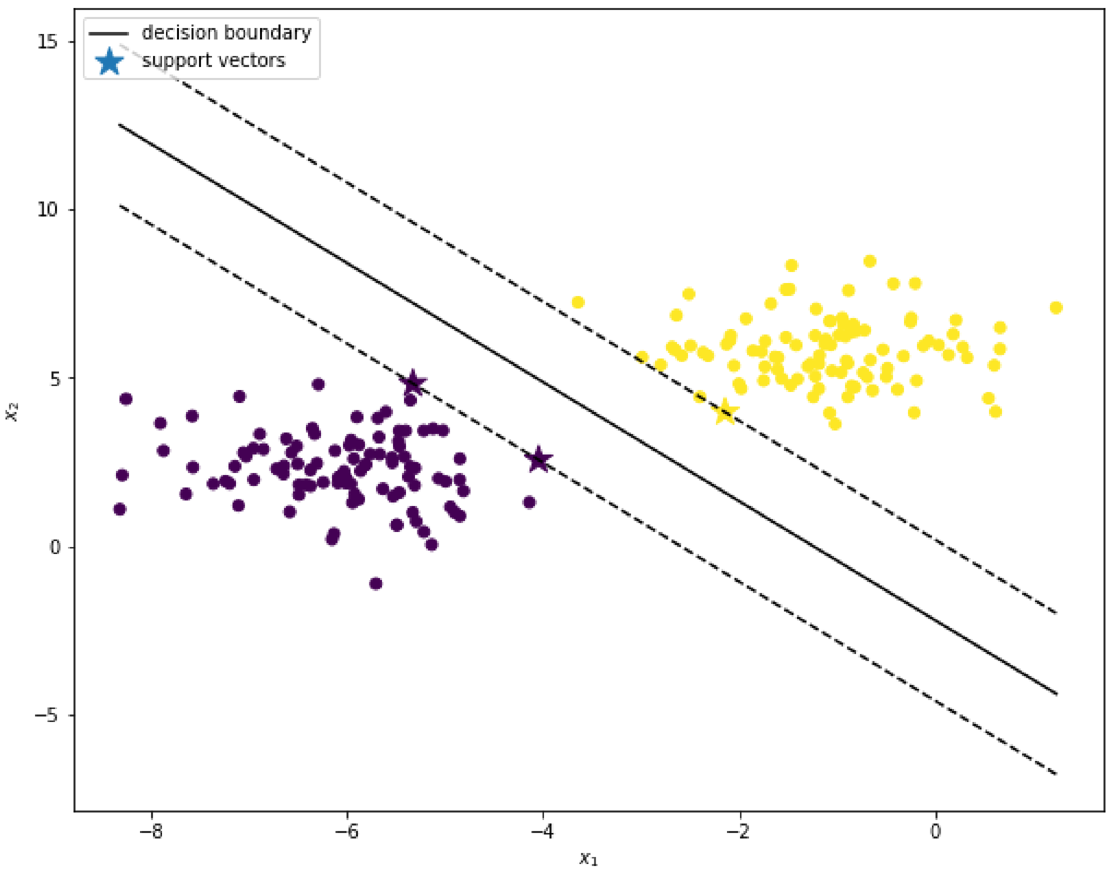
Indices of the support vectors are

```
[ 78 134 158]
```

[6]:
```
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 = [0.73940861 0.41770617]
b = 0.9206912669307183
support vectors: [ 78 134 158]
```

[7]:
```
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