6.

Similarities:

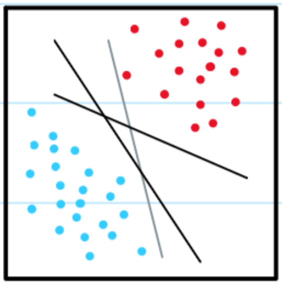
- Both algorithms are used to model linear and non-linear data sets.

- SVM does not support multiclass classification natively therefore we need to implement one of two indirect approaches: One-vs-one (1 model for each class pair) or One-vs-rest (where we consider one class and the remaining on a per model basis). Regarding the perceptron, we also require indirect modeling of this problem with the methods, in order to do so, being the same as in the SVM case.

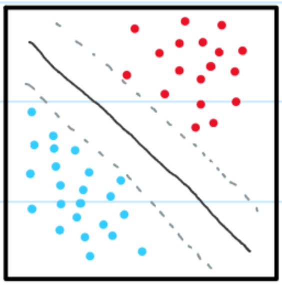
Differences

- SVM is based on a margin parameter, thus the algorithm stops after finding the best margin, whereas the perceptron algorithm stop when the data is classified correctly (converges on a set of weights).

- Main goal of the SVM is to separate data rigorously since a good margin avoids new samples falling on the wrong side of the boundary. The perceptron cannot compute this evaluation as precisely.



Perceptron, with multiple correct data divisions



Only possible SVM

- In non linear separable data, the Perceptron makes use of basis functions which extend linear regression models by allowing combinations of non-linear functions to better fit the data. Meanwhile, SVM introduces the kernel trick which simplifies working with basis functions. Some kernels are equivalent to using infinite-dimensional basis functions. While computing these feature transformations would be impossible, directly evaluating the kernels is often easy, making it more efficient. Kernels can also be used to encode similarity between arbitrary non-numerical data, from strings to graphs.

- Regarding optimization, the perceptron can be trained using back propagation, while SVM is solved by Sequential minimal optimization (SMO) for example, or any particular quadratic solver of the user's choice.

7.

$$g(\alpha) = \frac{1}{2}\alpha^T Q \alpha + \alpha^T \mathbb{1}_N$$

$$= \underbrace{-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j}_{①} + \underbrace{\sum_{i=1}^{N}\alpha_i}_{②}$$

① $-\frac{1}{2}\underbrace{\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j}\cdot(x_i^T x_j)$

$$= -\frac{1}{2}\cdot\sum_{i=1}^{N} y_i \alpha_i x_i (y_1\alpha_1 x_1 + y_2\alpha_2 x_2 + \dots + y_N \alpha_N x_N)$$

$$= -\frac{1}{2}\cdot(y_1\alpha_1 x_1 + y_2\alpha_2 x_2 + \dots + y_N\alpha_N x_N)^2 \in \mathbb{R}^{1\times 1}$$

Therefore, given that matrix product is commutative and the dual function SVM format:

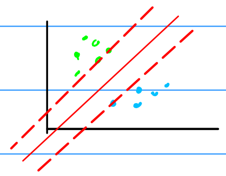
$Q = xx^T yy^T$, with $y \in \mathbb{R}^{N\times 1}$, $X \in \mathbb{R}^{N\times D}$

b) As seen in the previous exercise (a)), $\alpha^T Q \alpha$ results in a quadratic function (dependent on α). Given the concavity of such function, the local maximizer/minimizer is always a global maximum/minimum.

In order to determine the correct concavity (which must be negative) we need to determine the correct signal of Q, since $\alpha \geq 0$, for $i=1,\dots,N$

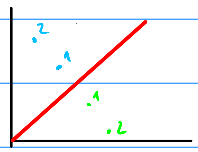
Given $Q = \underbrace{(-1)}_{\leq 0}\cdot\underbrace{\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j x_i x_j}_{\geq 0}$, so $Q \leq 0$ $\downarrow$ negative semidefine matrix

③



· an instance can only be missclassified if it forms a sv and it is removed from the training set, such that the next closest instance of the class holds $y_k(w^Tx+b) > m$ with m = margin size
(assuming the vector between the removed sv and the new one is perpendicular to the hyperplane on the full set, if this is not the case a similar argument can be made)

which implies that in the case where the next closest instance of each class holds $y_k(w^Tx+b) > m$ we obtain $\epsilon = \frac{S}{N}$ (all sv are missclassified in LOOCV)



· by removing the sv 1 the new sv would be 2 but since the distance between 1 and 2 is greater than m the new hyperplane would be located above 1, missclassifying it as blue. with the symmetric argument for the sv 1, we can see that in this case all sv would be misclassified

if this is not the case and the next closest points are situated in a margin m around the sv then, no point would be missclassified as the shift in the hyperplane wouldn't be big enough. In this case $\epsilon < \frac{S}{N}$

$\Rightarrow \epsilon \leq S/N$

//

(10) $k(x_1, x_2) = \sum_{i=1}^{N} a_i (x_1^T x_2)^i + a_0$ with $x_1, x_2 \in \mathbb{R}^d$

from the lecture:

- $x_1^T x_2 = x_1^T I x_2$ ⟶ since I is a symmetric, PSD matrix $\Rightarrow k(x_1, x_2) = x_1^T x_2$ is a kernel
- $\sum_{i=1}^{N} k(x_1, x_2)$ is a kernel (adding kernels)
- $\sum_{i=1}^{N} k(x_1, x_2)^i$ is a kernel (multiplying kernels)
- $\sum_{i=1}^{N} a_i\, k(x_1, x_2)^i$ is a kernel (multiplying by non-negative constant)
- $\sum_{i=1}^{N} a_i\, k(x_1, x_2)^i + a_0$ is a kernel

(11) $k(x_1, x_2) = \frac{1}{1 - x_1 x_2}$ with $x_1, x_2 \in (0,1)$

$k(x_1, x_2) := \phi(x_1)^T \phi(x_2)$

$\Rightarrow$ consider $\phi(x) = (1, x, x^2, \dots)$

$$k(x_1, x_2) = (1, x_1, x_1^2, \dots)\begin{pmatrix} 1 \\ x_2 \\ x_2^2 \\ \vdots \end{pmatrix} = 1 + x_1 x_2 + x_1^2 x_2^2 + \dots = \sum_{i=0}^{\infty} (x_1 x_2)^i \overset{x_1 x_2 < 1}{=} 1/(1 - x_1 x_2) \quad //$$

code

December 22, 2021

1 Programming assignment 4: SVM

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

from sklearn.datasets import make_blobs

from cvxopt import matrix, spmatrix, 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.

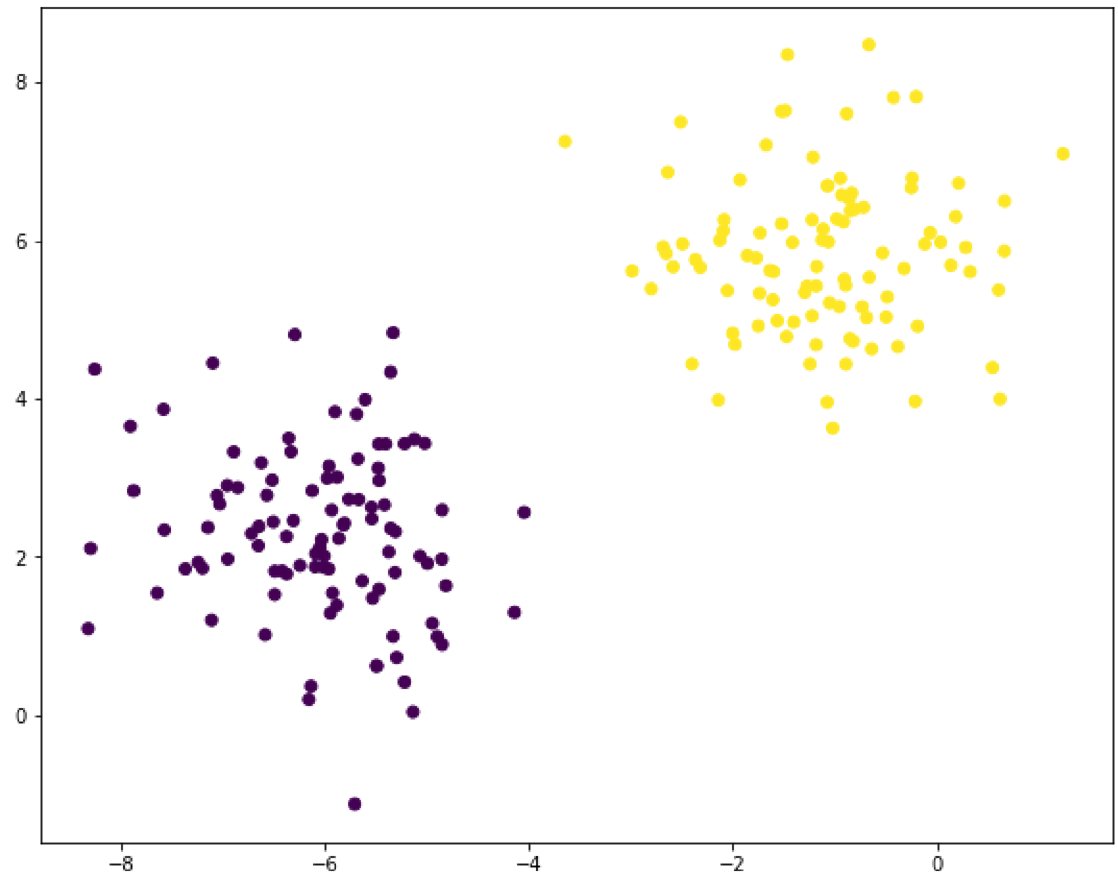
1.3 Generate and visualize the data

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

[ ]:
```
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.
    """
    n = y.shape[0]
    # -p and -q because its maximize in the slides

    y = y.reshape(-1,1) * 1
    P = matrix(X @ X.T * y * y.T)

    #P = matrix(temp)
    q = matrix(-np.ones(n,None))
    G = matrix(-np.identity(n))
    h = matrix(np.zeros(n,None))
    A = matrix(y.T)
    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

[ ]:
```
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 ###

    sv = np.where(alpha > 1e-4)
    xi, yi = X[sv], y[sv]

    w = np.sum((alpha * y)[:,None] * X, axis=0).reshape(-1,1)
    b = np.mean(yi.reshape(-1,1) - xi@w)

    return w, b
```

1.6 Visualize the result (nothing to do here)

[ ]:
```
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
```

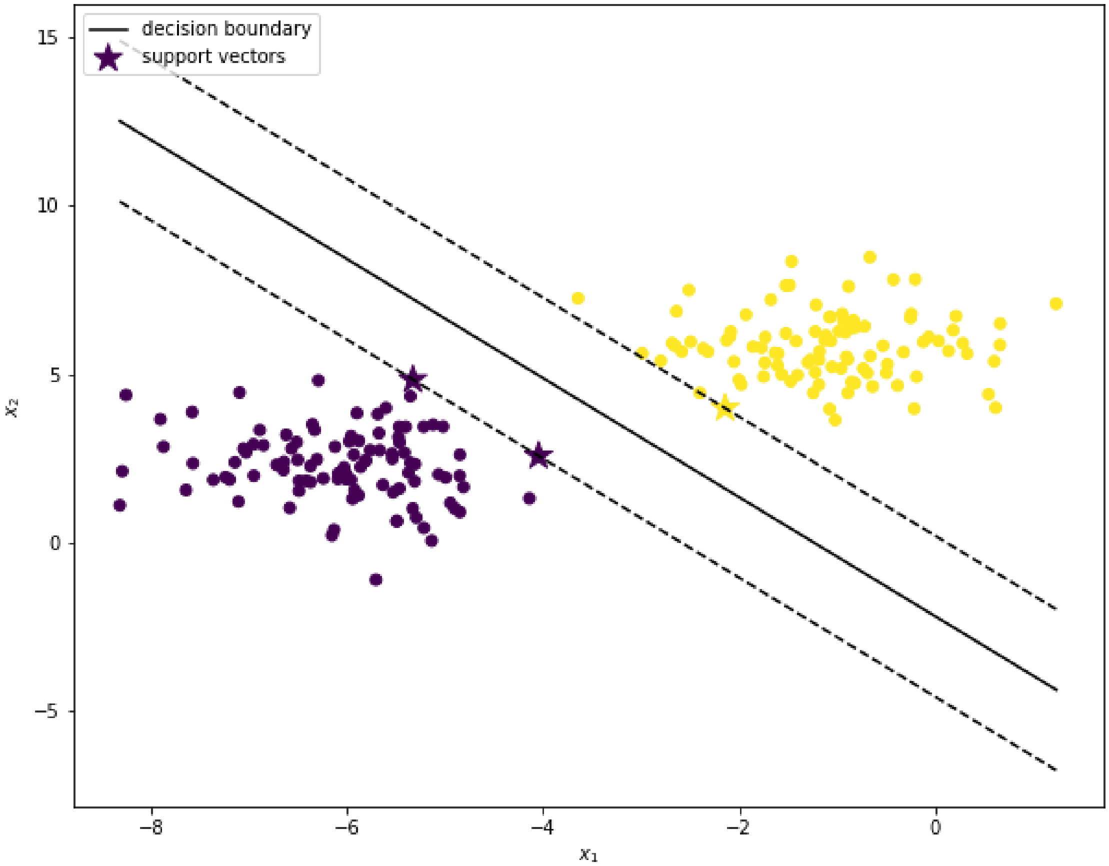
Indices of the support vectors are

```
[ 78 134 158]
```

[ ]:
```
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 = 0.9199371454171258
support vectors: [ 78 134 158]
```

[ ]:
```
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