

Programming assignment 7: SVM

In [21]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import make_blobs

from cvxopt import matrix, solvers
```

Your task

In this sheet we will implement a simple binary SVM classifier.

We will use **CVXOPT** <http://cvxopt.org/> (<http://cvxopt.org/>) - a Python library for convex optimization. If you use Anaconda, you can install it using

```
conda install cvxopt
```

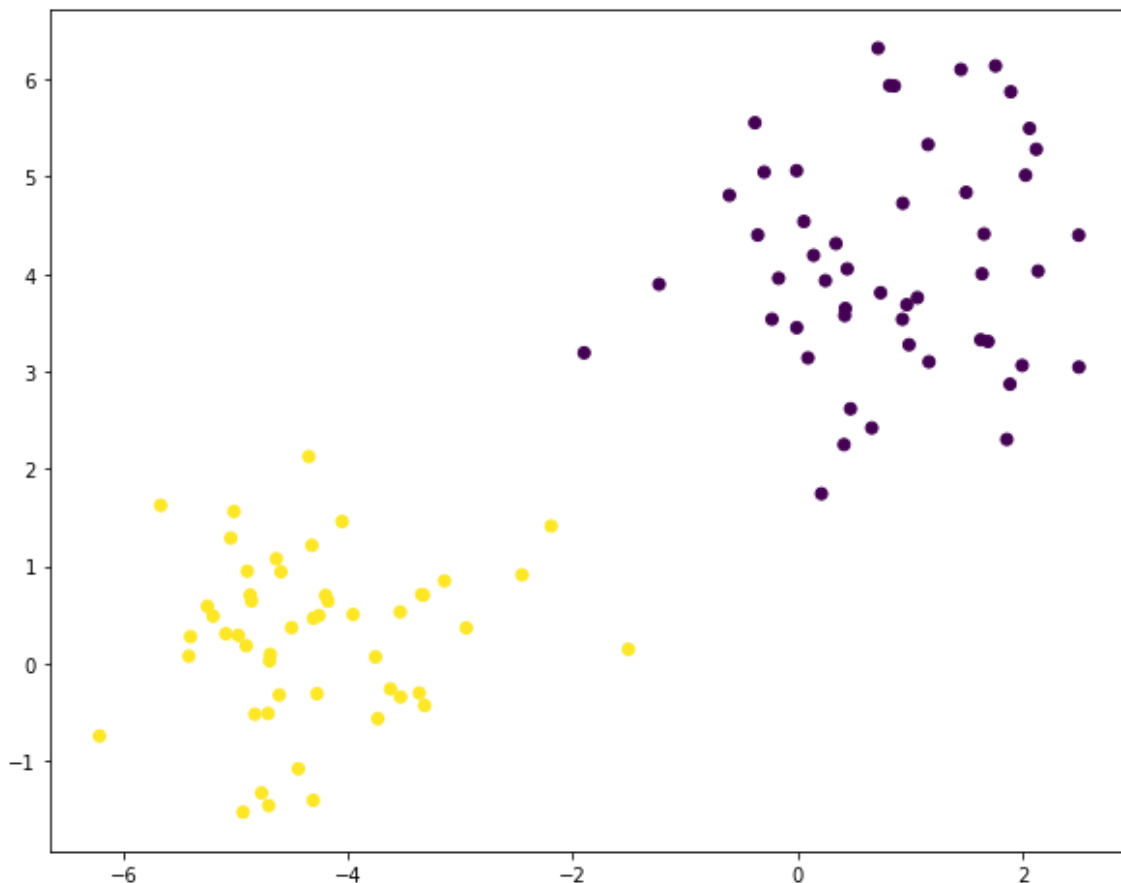
As usual, your task is to fill out the missing code, run the notebook, convert it to PDF and attach it to your HW solution.

Generate and visualize the data

In [22]:

```
N = 100 # number of samples
D = 2 # number of dimensions
C = 2 # number of classes
seed = 3 # for reproducible experiments

X, y = make_blobs(n_samples=N, n_features=D, centers=2, 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()
```



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.

The general form of a QP is

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2} \mathbf{x}^T \mathbf{P} \mathbf{x} - \mathbf{q}^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{G} \mathbf{x} \preceq \mathbf{h} \\ & \text{and } \mathbf{A} \mathbf{x} = \mathbf{b} \end{aligned}$$

where \preceq denotes "elementwise less than or equal to".

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

In [23]:

```
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

    N = X.shape[0]
    DIM = X.shape[1]

    K = y[:, None] * X
    K = np.dot(K, K.T)
    P = matrix(K)
    q = matrix(-np.ones((N, 1)))
    G = matrix(-np.eye(N))
    h = matrix(np.zeros(N))
    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
```

Task 2: Recovering the weights and the bias

In [24]:

```
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.  
    """  
  
    # get weights  
    w = np.sum(alpha * y[:, None] * X, axis = 0)  
  
    # get bias  
    cond = (alpha > 1e-4).reshape(-1)  
    b = y[cond] - np.dot(X[cond], w)  
  
    return w, b[0]
```

Visualize the result (nothing to do here)

In [25]:

```
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')
    # Plot all the datapoints
    plt.scatter(X[:, 0], X[:, 1], c=y)
    # Mark the support vectors
    support_vecs = (alpha > 1e-4).reshape(-1)
    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([[ -0.69192638],
           [-1.00973312]])
```

```
b = 0.907667782
```

Indices of the support vectors are

```
[38, 47, 92]
```

In [26]:

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

