# Assignment 7
# Akshay Aradhya

December 10, 2017

## 1 Problem 1

The Langrangian is given by: $L(x_1, x_2, \alpha) = -(x_1 + x_2) + \alpha(x_1^2 + x_2^2 - 1)$
The partial derivatives of L w.r.t. $x_1, x_2$ are:

$$\frac{\partial L}{\partial x_1} = -1 + 2\alpha x_1 \quad \text{and} \quad \frac{\partial L}{\partial x_2} = -1 + 2\alpha x_2$$

Hence, the optimal values of $x_1, x_2$ is at $\frac{1}{2\alpha}$ . Replacing this value in the Lagrangian, and then finding the maximum.

$$L(x_1^*, x_2^*, \alpha) = -\Big[\frac{1}{2\alpha} + \frac{1}{2\alpha}\Big] + \alpha\Big[\frac{1}{4\alpha^2} + \frac{1}{4\alpha^2} - 1\Big] \quad \Rightarrow L(\alpha) = \alpha - \frac{1}{2\alpha}$$

Now the derivative of L w.r.t. $\alpha$ is $\nabla_\alpha L = -1 + \frac{1}{2\alpha^2}$
Equating $\nabla_\alpha L$ to 0, we have $\alpha^* = \frac{1}{\sqrt{2}}$
As $x_1, x_2 = \frac{1}{2\alpha}$. Therefore, $f_0$ reaches optimal at $x_1^* = \frac{1}{\sqrt{2}}$ and $x_2^* = \frac{1}{\sqrt{2}}$
And the optimal value of $f_0$ is $-\sqrt{2}$

## 2 Problem 2

The comparison of SVM and Perceptron method is:

- Both Perceptron and SVM try to find a hyperplane that will linearly separate the data points. Perceptron method is a genralized method of SVM.
- Perceptron finds the hyperplane with minimal misclassified points that separates the data points without considering the distance between them. SVM tries to find a hyperplane that maximizes the distance (margin) between the data points or the support vectors.
- Perceptron assumes that the data set is linearly separable. However, SVM uses a kernel function for the same.

## 3 Problem 3

To show that the duality gap is 0 in an SVM, we have to show that Slater's condition holds.
Now, the SVM function is:
**minimize** $f_0(w, b) = \frac{1}{2}w^T w$
**subject to** $f_1(w, b) = y_i(w^T x_i + b) - 1 \geq 0$ for i = 1, ... , N

We can rewrite $f_1$ as

$$-y_i(w^T x_i + b) + 1 \leq 0 \quad \Rightarrow -y_i w^T x_i - y_i b + 1 \leq 0$$

This is an affine function in two variables w,b and a constant 1.
Hence, Slater's 2nd condition is satisfied. Therefore Duality gap is 0.

## 4 Problem 4

a)

The dual function is: $g(\alpha) = \sum\limits_{i=1}^{N} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j$

Here,each $x_i$ is also a vector. To write it in the form of a quadratic function, we can write Q as:

$$Q = -\begin{bmatrix} y_1y_1x_1^Tx_1 & y_1y_2x_1^Tx_2 & \cdots & y_1y_Nx_1^Tx_N \\ y_2y_1x_2^Tx_1 & y_2y_2x_2^Tx_2 & \cdots & y_2y_Nx_2^Tx_N \\ \vdots & \vdots & \ddots & \vdots \\ y_Ny_1x_N^Tx_1 & y_Ny_2x_N^Tx_2 & \cdots & y_Ny_Nx_N^Tx_N \end{bmatrix}$$

$$Q = -\begin{bmatrix} y_1y_1 & y_1y_2 & \cdots & y_1y_N \\ y_2y_1 & y_2y_2 & \cdots & y_2y_N \\ \vdots & \vdots & \ddots & \vdots \\ y_Ny_1 & y_Ny_2 & \cdots & y_Ny_N \end{bmatrix} \odot \begin{bmatrix} x_1^Tx_1 & x_1^Tx_2 & \cdots & x_1^Tx_N \\ x_2^Tx_1 & x_2^Tx_2 & \cdots & x_2^Tx_N \\ \vdots & \vdots & \ddots & \vdots \\ x_N^Tx_1 & x_N^Tx_2 & \cdots & x_N^Tx_N \end{bmatrix}$$

We can take Y to be a N $\times$ 1 vector of $y_1, y_2, \cdots, y_N$ values.
Similarly, we can take X to be 1 $\times$ N vector of vectors with $x_1, x_2, \cdots, x_N$ as values.

Then, Q can be written as: $Q = -(YY^T) \odot (X^TX)$

b)
Now, Q is negative semi-definite as $Q = -(YY^T) \odot (X^TX)$. Clearly, $(YY^T)$ and $(X^TX)$ are squared positive values. Hence, the negative sign in front of it makes Q a symmetric matrix with negative values which means it is negative semi-definite.

c)
This is required as the objective is to maximize the dual function $g(\alpha)$ whose solution is equal to the minimum of the primal problem. Being negative semi-definite, it ensures that there is only one global maximum of the dual. Hence, one global minimum of the primal.

# Programming assignment 7: SVM

In [1]:
```
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

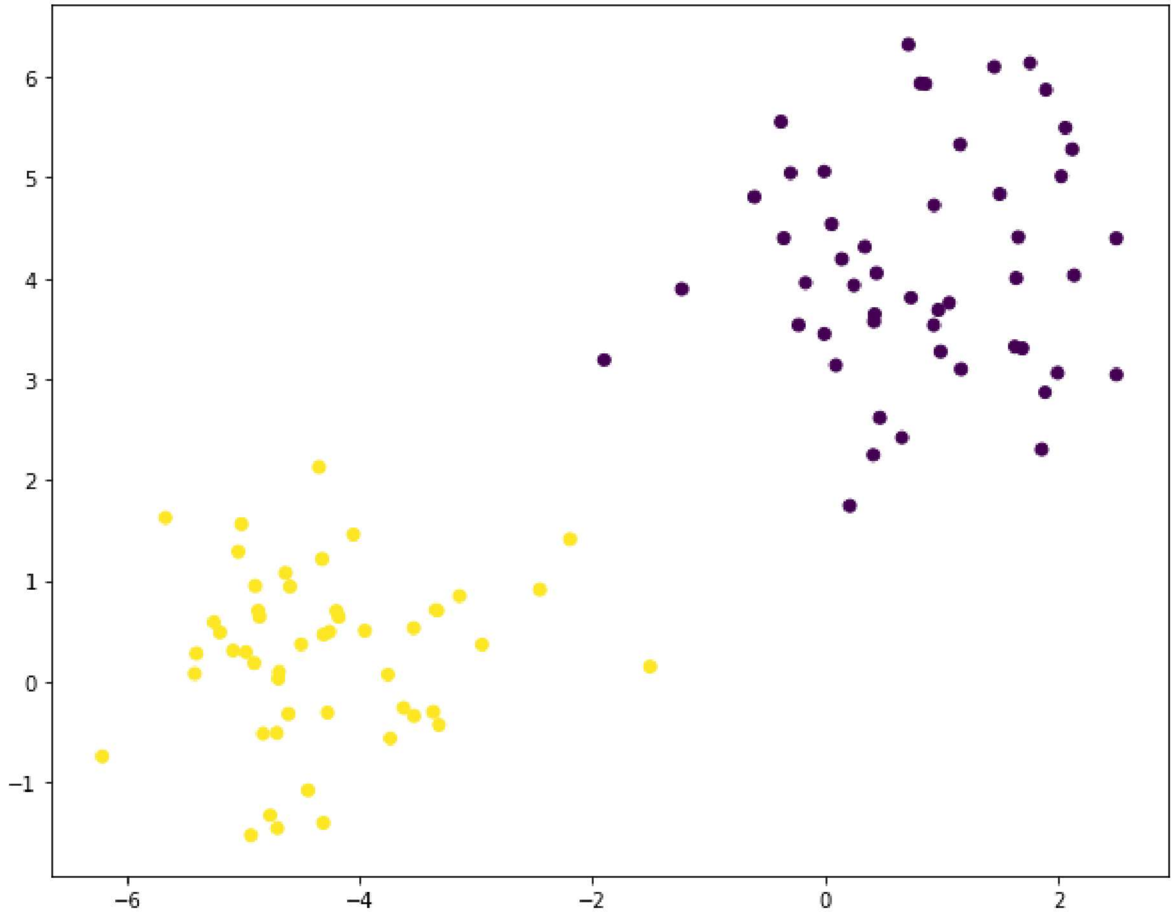
```
conda install cvxopt
```

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

## Generate and visualize the data

In [2]:
```
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 (inste
ad 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

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}$$

$$\text{subject to} \quad \mathbf{G}\mathbf{x} \preceq \mathbf{h}$$

$$\text{and} \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

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 [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

    # These variables have to be of type cvxopt.matrix
    N, D = X.shape

    # Obtaining the kernel

    K = y[:, None] * X

    # Solving the dual problem for SVM

    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
    sol = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(sol['x'])

    return alphas
```

# Task 2: Recovering the weights and the bias

In [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.
    """


    # Fit svm classifier
    alphas = solve_dual_svm(X, y)

    # Computation of weights
    w = np.sum(alphas * y[:, None] * X, axis = 0)

    # Computation of bias
    cond = (alphas > 1e-4).reshape(-1)
    b = y[cond] - np.dot(X[cond], w)
    b = b[1]

    return w, b
```

## Visualize the result (nothing to do here)

In [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 hyperplan
e.

    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=2
50, 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 [6]:
```
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()
```