Linear Regression and SVM

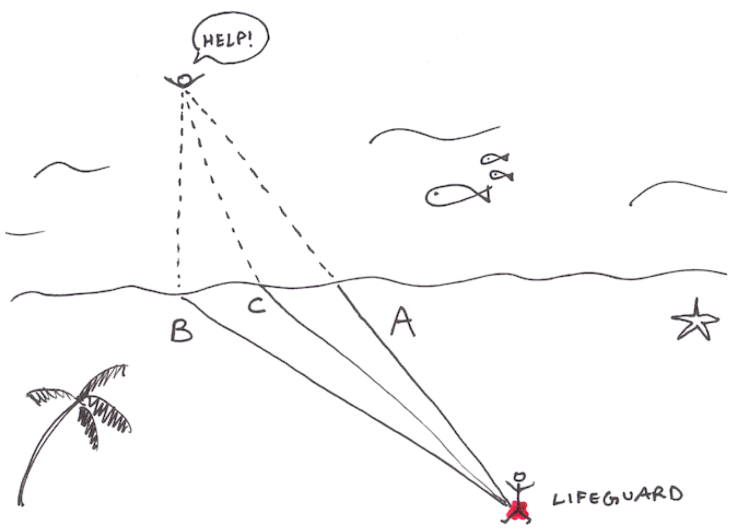
without Scikit Learn

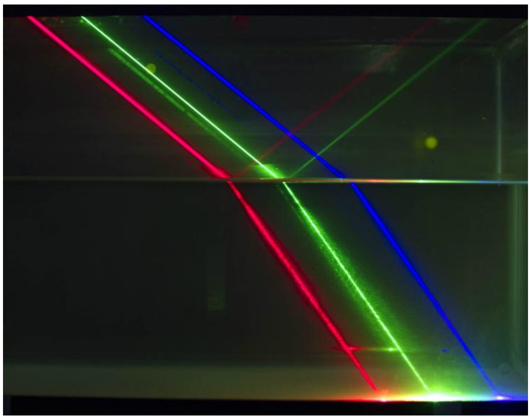
by Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

1. Optimization

- an important tool in 1) engineering problem solving and 2) decision science
- peolple optimize
- nature optimizes



(source: http://nautil.us/blog/to-save-drowning-people-ask-yourself-what-would-light-do (http://nautil.us/blog/to-save-drowning-people-ask-yourself-what-would-light-do))

3 key components

1. objective
2. decision variable or unknown
3. constraints

Procedures

1. The process of identifying objective, variables, and constraints for a given problem is known as "modeling"
2. Once the model has been formulated, optimization algorithm can be used to find its solutions.

In mathematical expression

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \qquad i = 1, \cdots, m \end{aligned}$$

Remarks) equivalent

$$\begin{aligned} \min_x f(x) \quad &\leftrightarrow \quad \max_x -f(x) \\ g_i(x) \leq 0 \quad &\leftrightarrow \quad -g_i(x) \geq 0 \\ h(x) = 0 \quad &\leftrightarrow \quad \begin{cases} h(x) \leq 0 & \text{and} \\ h(x) \geq 0 \end{cases} \end{aligned}$$

The good news: for many classes of optimization problems, people have already done all the "hardwork" of developing numerical algorithms

$$\begin{array}{rl} \max\limits_{x} & x_1 + x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 29 \\ & x_1 + 2x_2 \leq 25 \\ & x_1 \geq 2 \\ & x_2 \geq 5 \end{array} \implies \begin{array}{rl} \min_x & -\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 29 \\ 25 \end{bmatrix} \\ & \begin{bmatrix} 2 \\ 5 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} \ \end{bmatrix} \end{array}$$

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx

f = np.array([[1, 1]])
A = np.array([[2, 1], [1, 2]])
b = np.array([[29], [25]])
lb = np.array([[2], [5]])

x = cvx.Variable(2,1)
obj = cvx.Minimize(-f*x)
const = [A*x <= b, lb <= x]

prob = cvx.Problem(obj, const).solve()

print (x.value)
```

```
[[ 11.]
 [  7.]]
```

2. Linear Regression

Begin by considering linear regression (easy to extend to more comlex predictions later on)

$\text{Given} \begin{cases} x_i : \text{inputs} \\ y_i : \text{outputs} \end{cases},$ find θ_1 and θ_2

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \theta_1 x_i + \theta_2$$

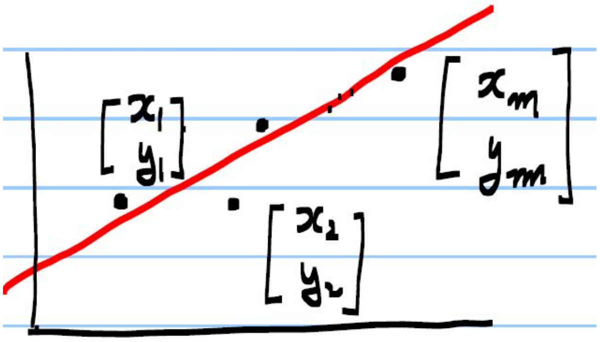
- $\hat{y}_i$: predicted output

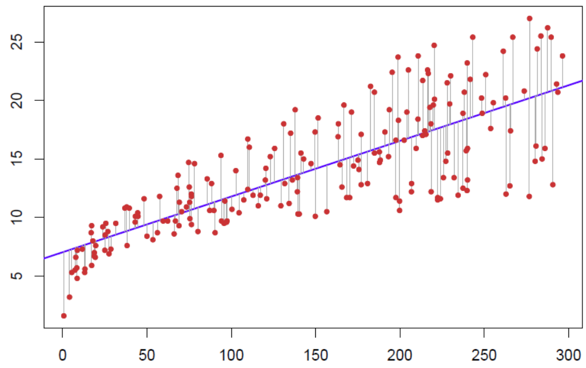
- $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$: Model parameters

$$\hat{y}_i = f(x_i, \theta) \ \text{ in general}$$

- In many cases, a linear model to predict y_i can be used

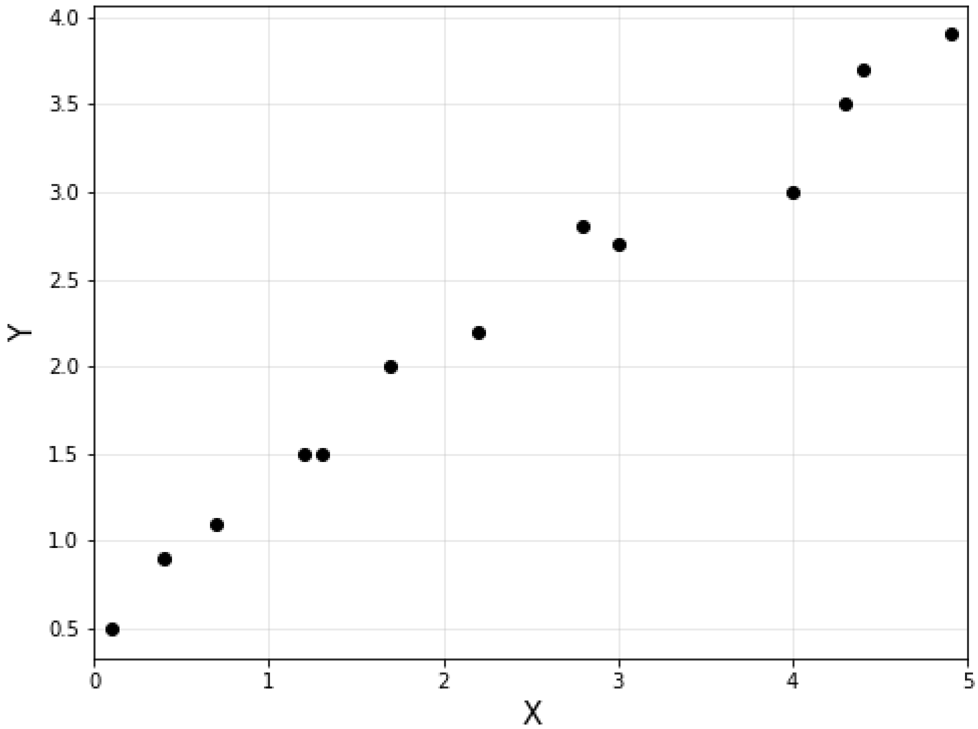
$$\hat{y}_i = \theta_1 x_i + \theta_2 \qquad \text{such that} \qquad \min_{\theta_1,\theta_2} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$





In [2]:

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

# data points in column vector [input, output]
x = np.array([0.1, 0.4, 0.7, 1.2, 1.3, 1.7, 2.2, 2.8, 3.0, 4.0, 4.3, 4.4,
4.9]).reshape(-1, 1)
y = np.array([0.5, 0.9, 1.1, 1.5, 1.5, 2.0, 2.2, 2.8, 2.7, 3.0, 3.5, 3.7,
3.9]).reshape(-1, 1)

# to plot
plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label="data")
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.axis('scaled')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```



Use CVXPY optimization (least squared)

For convenience, we define a function that maps inputs to feature vectors, ϕ

$$
\begin{aligned}
\hat{y}_i &= \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \\
&= \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad , \qquad \text{feature vector } \phi(x_i) = \begin{bmatrix} x_i \\ 1 \end{bmatrix} \\
&= \phi^T(x_i)\theta
\end{aligned}
$$

$$
\Phi = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_m & 1 \end{bmatrix} = \begin{bmatrix} \phi^T(x_1) \\ \phi^T(x_2) \\ \vdots \\ \phi^T(x_m) \end{bmatrix} \quad \Longrightarrow \quad \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \Phi\theta
$$

Model parameter estimation

$$
\min_{\theta} \; \lVert \hat{y} - y \rVert_2 = \min_{\theta} \; \lVert \Phi\theta - y \rVert_2
$$

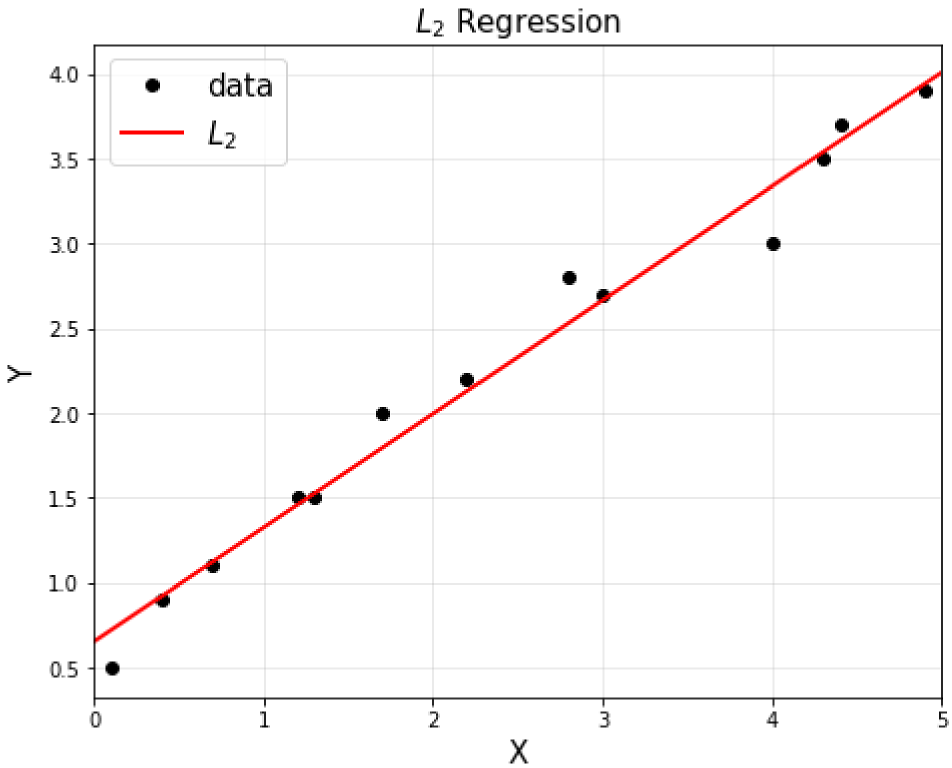
In [3]:

```
import cvxpy as cvx

m = y.shape[0]
#A = np.hstack([x, np.ones([m, 1])])
A = np.hstack([x, x**0])
A = np.asmatrix(A)

theta2 = cvx.Variable(2, 1)
obj = cvx.Minimize(cvx.norm(A*theta2-y, 2))
cvx.Problem(obj,[]).solve()

print('theta:\n', theta2.value)
```

```
theta:
 [[ 0.67129519]
 [ 0.65306531]]
```

In [4]:

```
# to plot
plt.figure(figsize=(10, 6))
plt.title('$L_2$ Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.plot(x, y, 'ko', label="data")

# to plot a straight line (fitted line)
xp = np.arange(0, 5, 0.01).reshape(-1, 1)
theta2 = theta2.value
yp = theta2[0,0]*xp + theta2[1,0]

plt.plot(xp, yp, 'r', linewidth=2, label="$L_2$")
plt.legend(fontsize=15)
plt.axis('scaled')
plt.grid(alpha=0.3)
plt.xlim([0, 5])
plt.show()
```



3. Classification (Linear)

- Figure out, autonomously, which category (or class) an unknown item should be categorized into

- Number of categories / classes
  - Binary: 2 different classes
  - Multiclass: more than 2 classes

- Feature
  - The measurable parts that make up the unknown item (or the information you have available to categorize)

3.1. Distance between Two Parallel Lines

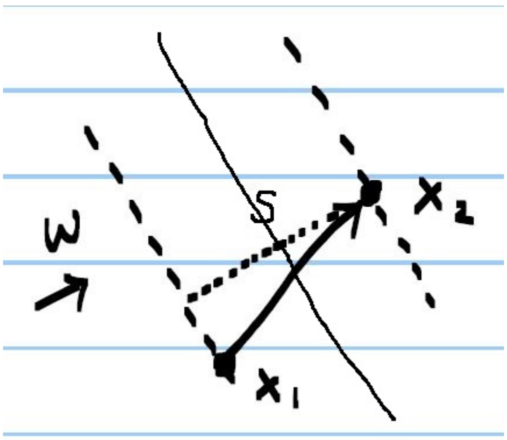
$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix},\ x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \Longrightarrow \quad g(x) = \omega^T x + \omega_0 \ = \omega_1 x_1 + \omega_2 x_2 + \omega_0$$

Find a distance between $g(x) = -1$ **and** $g(x) = 1$

suppose $g(x_1) = -1,\ g(x_2) = 1$

$$\begin{aligned} \omega^T x_1 + \omega_0 &= -1 \\ \omega^T x_2 + \omega_0 &= 1 \end{aligned} \quad \Longrightarrow \quad \omega^T (x_2 - x_1) = 2$$

$$s = \left\langle \frac{\omega}{\|\omega\|}, x_2 - x_1 \right\rangle = \frac{1}{\|\omega\|} \omega^T (x_2 - x_1) = \frac{2}{\|\omega\|}$$



3.2. Illustrative Example

- Binary classification: C_1 and C_2

- Features: the coordinate of ith data

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Is it possible to distinguish between C_1 and C_2 by its coordinates?

- We need to find a separating hyperplane (or a line in 2D)

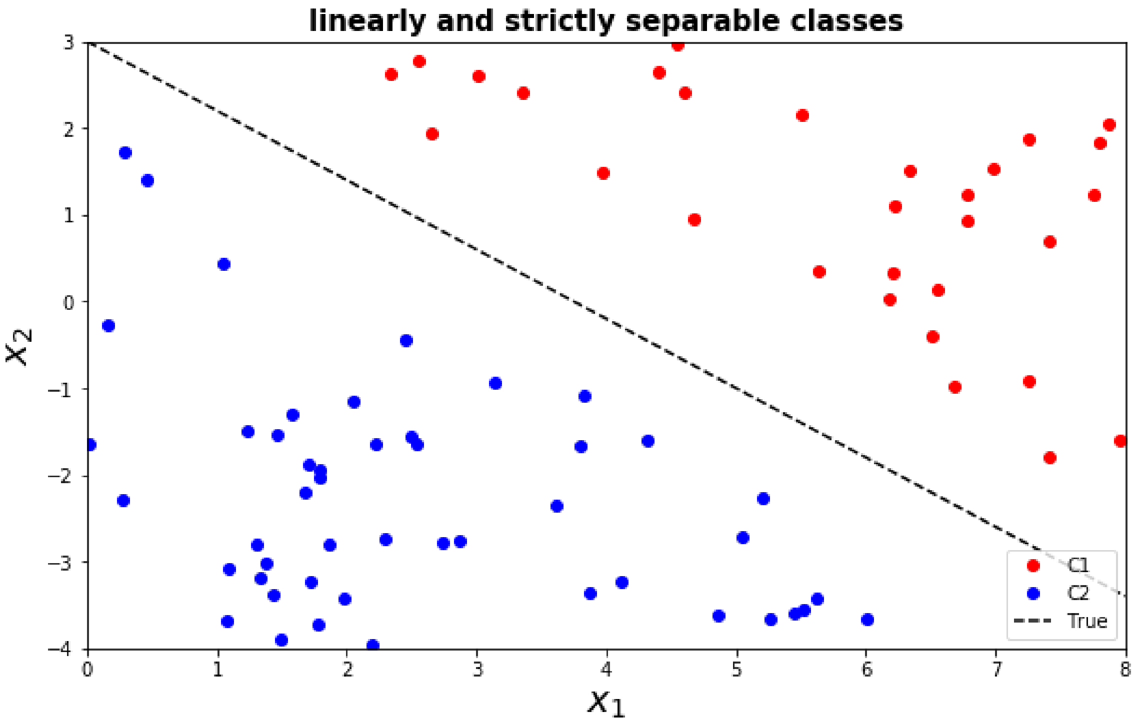
$$\begin{aligned} \omega_1 x_1 + \omega_2 x_2 + \omega_0 &= 0 \\ \begin{bmatrix} \omega_1 & \omega_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \omega_0 &= 0 \\ \omega^T x + \omega_0 &= 0 \end{aligned}$$

In [5]:

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

#training data gerneration
x1 = 8*np.random.rand(100, 1)
x2 = 7*np.random.rand(100, 1) - 4

g0 = 0.8*x1 + x2 - 3
g1 = g0 - 1
g2 = g0 + 1

C1 = np.where(g1 >= 0)[0]
C2 = np.where(g2 < 0)[0]
```

In [6]:

```
xp = np.linspace(0,8,100).reshape(-1,1)
ypt = -0.8*xp + 3

plt.figure(figsize=(10, 6))
plt.plot(x1[C1], x2[C1], 'ro', label='C1')
plt.plot(x1[C2], x2[C2], 'bo', label='C2')
plt.plot(xp, ypt, '--k', label='True')
plt.title('linearly and strictly separable classes', fontweight = 'bold', fontsize = 15)
plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.legend(loc = 4)
plt.xlim([0, 8])
plt.ylim([-4, 3])
plt.show()
```



- Given:
  - Hyperplane defined by ω and ω_0
  - Animals coordinates (or features) x

- Decision making:

$$\omega^T x + \omega_0 > 0 \implies x \text{ belongs to } C_1$$
$$\omega^T x + \omega_0 < 0 \implies x \text{ belongs to } C_2$$

- Find ω and ω_0 such that x given $\omega^T x + \omega_0 = 0$

  or

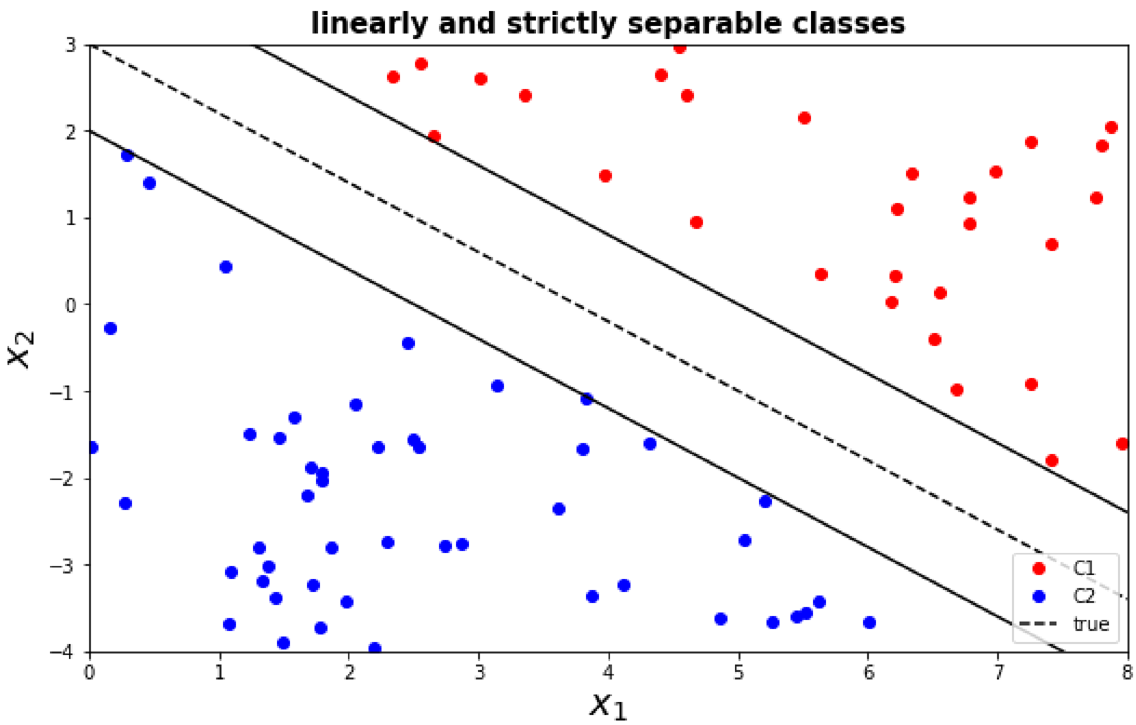
- Find ω and ω_0 such that $x \in C_1$ given $\omega^T x + \omega_0 > 1$ and $x \in C_2$ given $\omega^T x + \omega_0 < -1$

$$\begin{aligned} & \omega^T x + \omega_0 > b \\ \iff & \frac{\omega^T}{b} x + \frac{\omega_0}{b} > 1 \\ \iff & \omega'^T x + \omega'_0 > 1 \end{aligned}$$

- Same problem if strictly separable

In [7]:

```
#  see how data are generated
xp = np.linspace(0,8,100).reshape(-1,1)
ypt = -0.8*xp + 3

plt.figure(figsize=(10, 6))
plt.plot(x1[C1], x2[C1], 'ro', label='C1')
plt.plot(x1[C2], x2[C2], 'bo', label='C2')
plt.plot(xp, ypt, '--k', label='true')
plt.plot(xp, ypt-1, '-k')
plt.plot(xp, ypt+1, '-k')
plt.title('linearly and strictly separable classes', fontweight = 'bold', fontsize =
15)
plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.legend(loc = 4)
plt.xlim([0, 8])
plt.ylim([-4, 3])
plt.show()
```



3.2.1. LP Formulation 1

- $n\ (=2)$ features
- $m = N + M$ data points in training set

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \text{ with } \omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix} \qquad \text{or} \qquad x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \text{ with } \omega = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

- N belongs to C_1 in training set
- M belongs to C_2 in training set
- ω and ω_0 are the unknown variables

$$
\begin{array}{ll}
\text{minimize} & \text{something} \\
\text{subject to} & \left\{\begin{array}{l}
\omega^T x^{(1)} + \omega_0 \geq 1 \\
\omega^T x^{(2)} + \omega_0 \geq 1 \\
\quad\vdots \\
\omega^T x^{(N)} + \omega_0 \geq 1
\end{array}\right. \\
& \left\{\begin{array}{l}
\omega^T x^{(N+1)} + \omega_0 \leq -1 \\
\omega^T x^{(N+2)} + \omega_0 \leq -1 \\
\quad\vdots \\
\omega^T x^{(N+M)} + \omega_0 \leq -1
\end{array}\right.
\end{array}
\quad \text{or} \quad
\begin{array}{ll}
\text{minimize} & \text{something} \\
\text{subject to} & \left\{\begin{array}{l}
\omega^T x^{(1)} \geq 1 \\
\omega^T x^{(2)} \geq 1 \\
\quad\vdots \\
\omega^T x^{(N)} \geq 1
\end{array}\right. \\
& \left\{\begin{array}{l}
\omega^T x^{(N+1)} \leq -1 \\
\omega^T x^{(N+2)} \leq -1 \\
\quad\vdots \\
\omega^T x^{(N+M)} \leq -1
\end{array}\right.
\end{array}
$$

Code (CVXPY)

$$
X_1 = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(N)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} \end{bmatrix}
$$

$$
X_2 = \begin{bmatrix} \left(x^{(N+1)}\right)^T \\ \left(x^{(N+2)}\right)^T \\ \vdots \\ \left(x^{(N+M)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(N+1)} & x_2^{(N+1)} \\ 1 & x_1^{(N+2)} & x_2^{(N+2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N+M)} & x_2^{(N+M)} \end{bmatrix}
$$

$$
\begin{array}{rl}
\text{minimize} & \text{something} \\
\text{subject to} & X_1\omega \geq 1 \\
& X_2\omega \leq -1
\end{array}
$$

$$
\begin{array}{rl}
\text{minimize} & \text{something} \\
\text{subject to} & X_1\omega \geq 1 \\
& X_2\omega \leq -1
\end{array}
$$

In [8]:

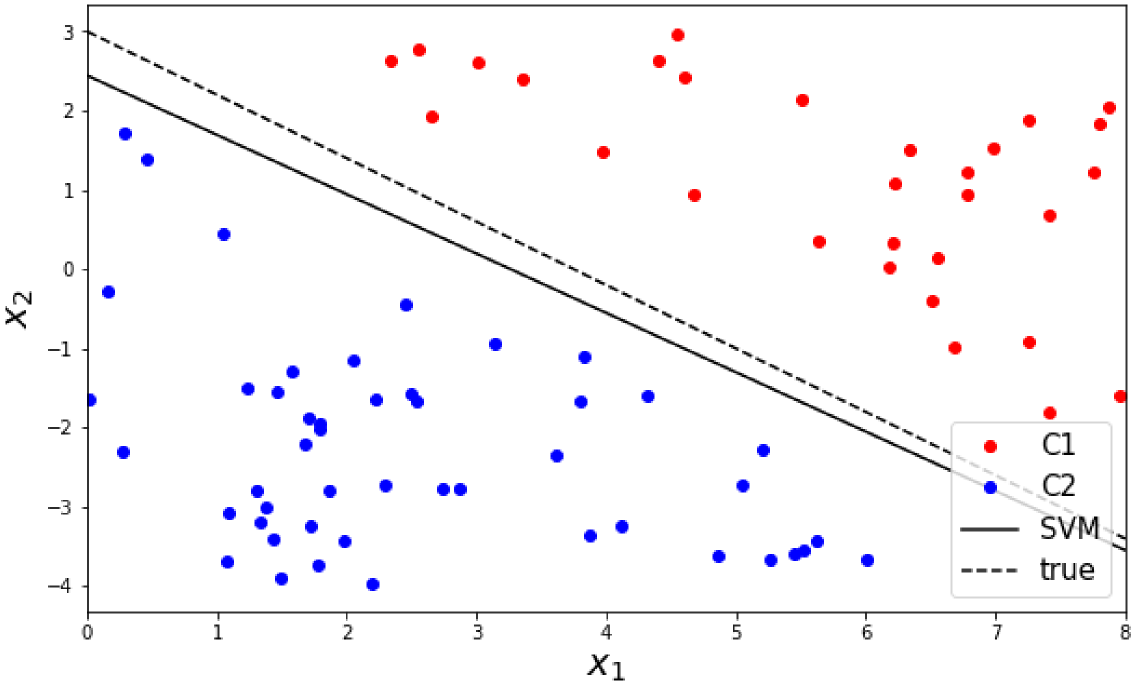
```
# CVXPY using simple classification
import cvxpy as cvx

N = C1.shape[0]
M = C2.shape[0]

X1 = np.hstack([np.ones([N,1]), x1[C1], x2[C1]])
X2 = np.hstack([np.ones([M,1]), x1[C2], x2[C2]])

X1 = np.asmatrix(X1)
X2 = np.asmatrix(X2)
```

In [9]:

```
w = cvx.Variable(3,1)
obj = cvx.Minimize(1)
const = [X1*w >= 1, X2*w <= -1]
prob = cvx.Problem(obj, const).solve()

w = w.value
```

In [10]:

```
xp = np.linspace(0,8,100).reshape(-1,1)
yp = - w[1,0]/w[2,0]*xp - w[0,0]/w[2,0]

plt.figure(figsize=(10, 6))
plt.plot(X1[:,1], X1[:,2], 'ro', label='C1')
plt.plot(X2[:,1], X2[:,2], 'bo', label='C2')
plt.plot(xp, yp, 'k', label='SVM')
plt.plot(xp, ypt, '--k', label='true')
plt.xlim([0,8])
plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.legend(loc = 4, fontsize = 15)
plt.show()
```

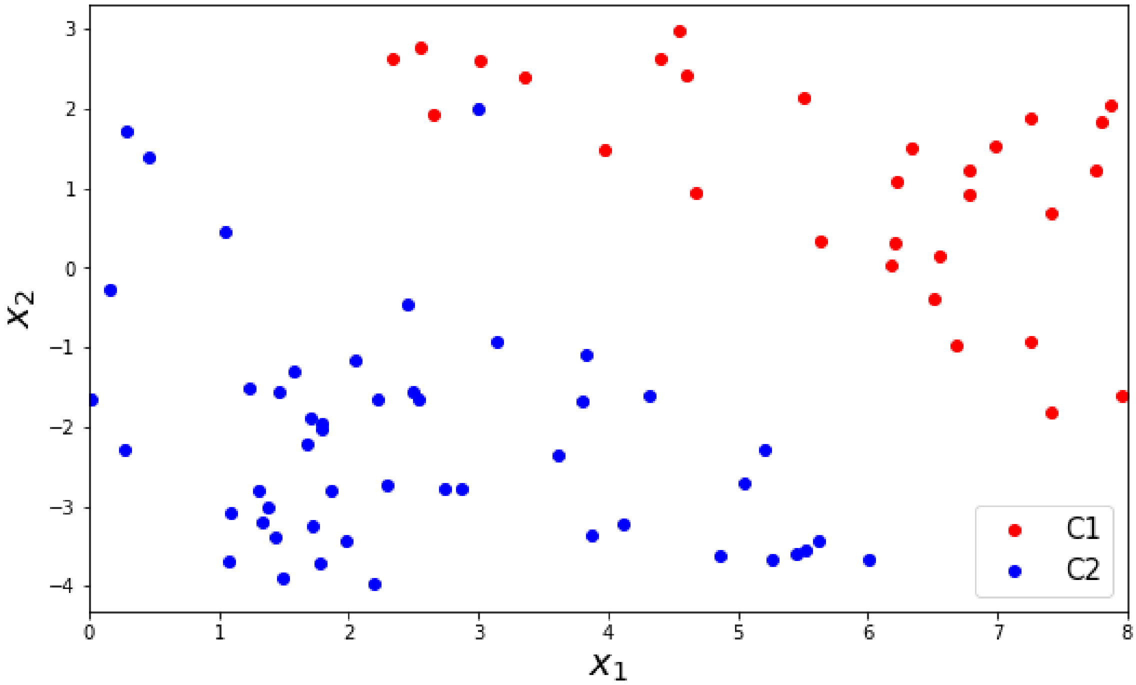


3.2.2. Outlier

- Note that in the real world, you may have noise, errors, or outliers that do not accurately represent the actual phenomena
- Non-separable case
- No solutions (hyperplane) exist
  - We will allow some training examples to be misclassified !
  - but we want their number to be minimized

In [11]:

```
X1 = np.hstack([np.ones([N,1]), x1[C1], x2[C1]])
X2 = np.hstack([np.ones([M,1]), x1[C2], x2[C2]])

outlier = np.array([1, 3, 2]).reshape(-1,1)
X2 = np.vstack([X2, outlier.T])

X1 = np.asmatrix(X1)
X2 = np.asmatrix(X2)

plt.figure(figsize=(10, 6))
plt.plot(X1[:,1], X1[:,2], 'ro', label='C1')
plt.plot(X2[:,1], X2[:,2], 'bo', label='C2')
plt.xlim([0,8])
plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.legend(loc = 4, fontsize = 15)
plt.show()
```



In [12]:

```
w = cvx.Variable(3,1)
obj = cvx.Minimize(1)
const = [X1*w >= 1, X2*w <= -1]
prob = cvx.Problem(obj, const).solve()

print(w.value)
```

None

3.2.3. LP Formulation 2

- $n \ (=2)$ features
- $m = N + M$ data points in a training set

$$x^i = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$$

- N belongs to C_1 in training set
- M belongs to C_2 in training set
- ω and ω_0 are the variables (unknown)

- For the non-separable case, we relex the above constraints
- Need slack variables u and v where all are positive

The optimization problem for the non-separable case

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{N} u_i + \sum_{i=1}^{M} v_i \\
\text{subject to} \quad & \begin{cases} \omega^T x^{(1)} \geq 1 - u_1 \\ \omega^T x^{(2)} \geq 1 - u_2 \\ \quad\quad \vdots \\ \omega^T x^{(N)} \geq 1 - u_N \end{cases} \\
& \begin{cases} \omega^T x^{(N+1)} \leq -(1 - v_1) \\ \omega^T x^{(N+2)} \leq -(1 - v_2) \\ \quad\quad \vdots \\ \omega^T x^{(N+M)} \leq -(1 - v_M) \end{cases} \\
& \begin{cases} u \geq 0 \\ v \geq 0 \end{cases}
\end{aligned}$$

- Expressed in a matrix form

$$X_1 = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \left(x^{(2)}\right)^T \\ \vdots \\ \left(x^{(N)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N)} & x_2^{(N)} \end{bmatrix}$$

$$X_2 = \begin{bmatrix} \left(x^{(N+1)}\right)^T \\ \left(x^{(N+2)}\right)^T \\ \vdots \\ \left(x^{(N+M)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(N+1)} & x_2^{(N+1)} \\ 1 & x_1^{(N+2)} & x_2^{(N+2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N+M)} & x_2^{(N+M)} \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix}$$

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix}$$

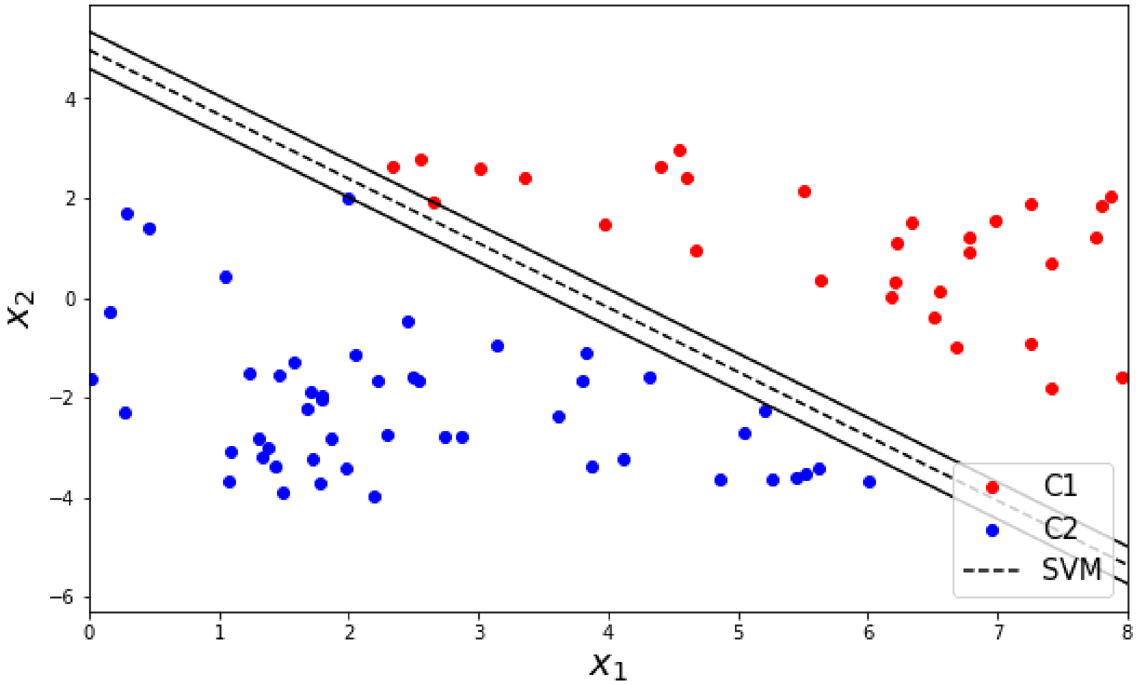
$$\begin{aligned} \text{minimize} \quad & 1^T u + 1^T v \\ \text{subject to} \quad & X_1 \omega \geq 1 - u \\ & X_2 \omega \leq -(1 - v) \\ & u \geq 0 \\ & v \geq 0 \end{aligned}$$

In [13]:

```
X1 = np.hstack([np.ones([C1.shape[0],1]), x1[C1], x2[C1]])
X2 = np.hstack([np.ones([C2.shape[0],1]), x1[C2], x2[C2]])

outlier = np.array([1, 2, 2]).reshape(-1,1)
X2 = np.vstack([X2, outlier.T])

X1 = np.asmatrix(X1)
X2 = np.asmatrix(X2)

N = X1.shape[0]
M = X2.shape[0]

w = cvx.Variable(3,1)
u = cvx.Variable(N,1)
v = cvx.Variable(M,1)
obj = cvx.Minimize(np.ones((1,N))*u + np.ones((1,M))*v)
const = [X1*w >= 1-u, X2*w <= -(1-v), u >= 0, v >= 0 ]
prob = cvx.Problem(obj, const).solve()

w = w.value
```

In [14]:

```
xp = np.linspace(0,8,100).reshape(-1,1)
yp = - w[1,0]/w[2,0]*xp - w[0,0]/w[2,0]

plt.figure(figsize=(10, 6))
plt.plot(X1[:,1], X1[:,2], 'ro', label='C1')
plt.plot(X2[:,1], X2[:,2], 'bo', label='C2')
plt.plot(xp, yp, '--k', label='SVM')
plt.plot(xp, yp-1/w[2,0], '-k')
plt.plot(xp, yp+1/w[2,0], '-k')
plt.xlim([0,8])
plt.xlabel('$x_1$', fontsize = 20)
plt.ylabel('$x_2$', fontsize = 20)
plt.legend(loc = 4, fontsize = 15)
plt.show()
```



Further improvement

- Notice that hyperplane is not as accurately represent the division due to the outlier
- Can we do better when there are noise data or outliers?
- Yes, but we need to look beyond LP
- Idea: large margin leads to good generalization on the test data

4. Maximize Margin (Finally, it is Support Vector Machine)

- Distance (= margin)

$$\text{margin} = \frac{2}{\|\omega\|_2}$$

- Minimize $\|\omega\|_2$ to maximize the margin (closest samples from the decision line)

$$\text{maximize } \{\text{minimum distance}\}$$

- Use gamma (γ) as a weighting betwwen the followings:
  - Bigger margin given robustness to outliers
  - Hyperplane that has few (or no) errors

$$\begin{array}{rl} \text{minimize} & \|\omega\|_2 + \gamma(1^T u + 1^T v) \\ \text{subject to} & X_1\omega + \omega_0 \geq 1 - u \\ & X_2\omega + \omega_0 \leq -(1 - v) \\ & u \geq 0 \\ & v \geq 0 \end{array}$$

In [15]:

```
g = 1
w = cvx.Variable(3,1)
u = cvx.Variable(N,1)
v = cvx.Variable(M,1)
obj = cvx.Minimize(cvx.norm(w,2) + g*(np.ones((1,N))*u + np.ones((1,M))*v))
const = [X1*w >= 1-u, X2*w <= -(1-v), u >= 0, v >= 0 ]
prob = cvx.Problem(obj, const).solve()

w = w.value
```