



Support Vector Machine (SVM)

Prof. Seungchul Lee
Industrial AI Lab.

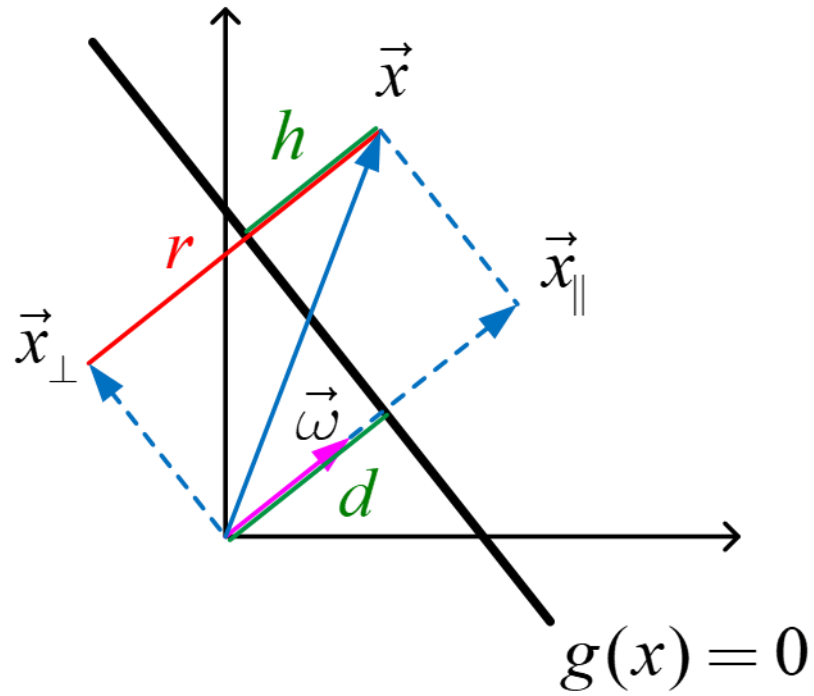
Classification (Linear)

- Autonomously figure out 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)

Distance

Distance from a Line

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

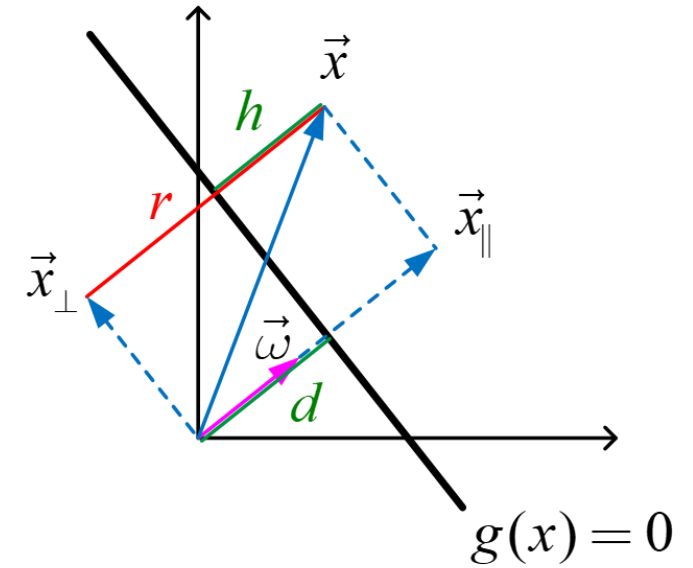


ω

- If \vec{p} and \vec{q} are on the decision line

$$\begin{aligned} g(\vec{p}) = g(\vec{q}) = 0 &\Rightarrow \omega_0 + \omega^T \vec{p} = \omega_0 + \omega^T \vec{q} = 0 \\ &\Rightarrow \omega^T (\vec{p} - \vec{q}) = 0 \end{aligned}$$

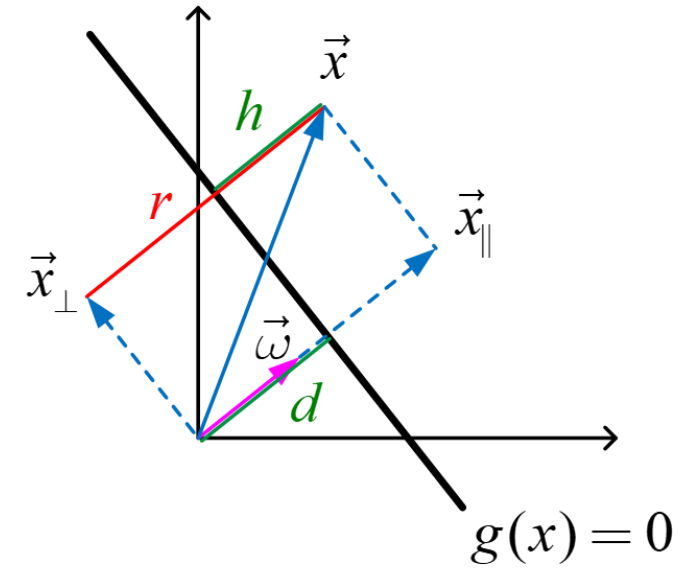
$\therefore \omega$: normal to the line (orthogonal)
 \Rightarrow tells the direction of the line



Signed Distance d

- If x is on the line and $x = d \frac{\omega}{\|\omega\|}$ (where d is a normal distance from the origin to the line)

$$\begin{aligned} g(x) &= \omega_0 + \omega^T x = 0 \\ \Rightarrow \omega_0 + \omega^T d \frac{\omega}{\|\omega\|} &= \omega_0 + d \frac{\omega^T \omega}{\|\omega\|} = \omega_0 + d \|\omega\| = 0 \\ \therefore d &= -\frac{\omega_0}{\|\omega\|} \end{aligned}$$



Distance from a Line: h

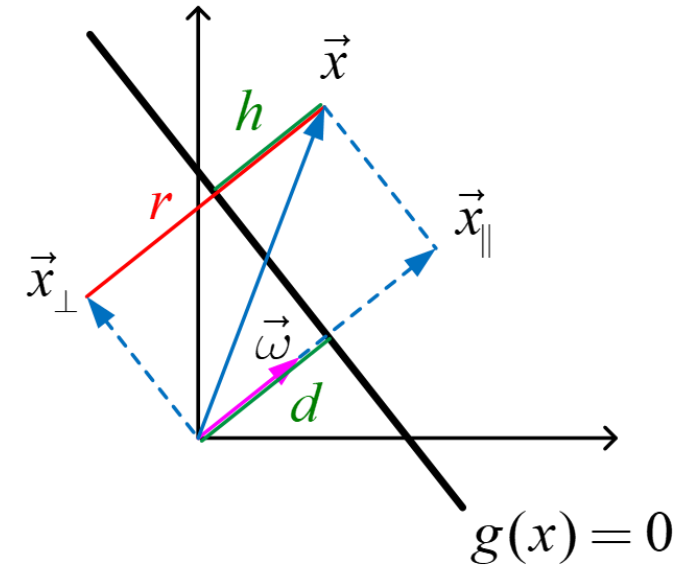
- for any vector of x

$$x = x_{\perp} + r \frac{\omega}{\|\omega\|}$$

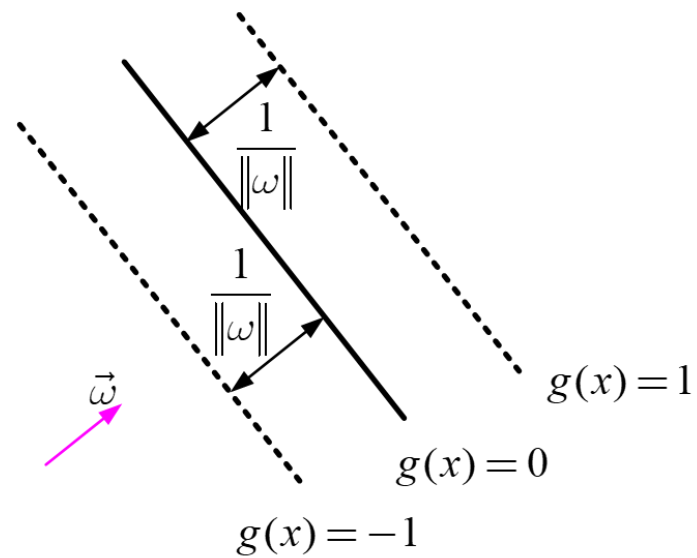
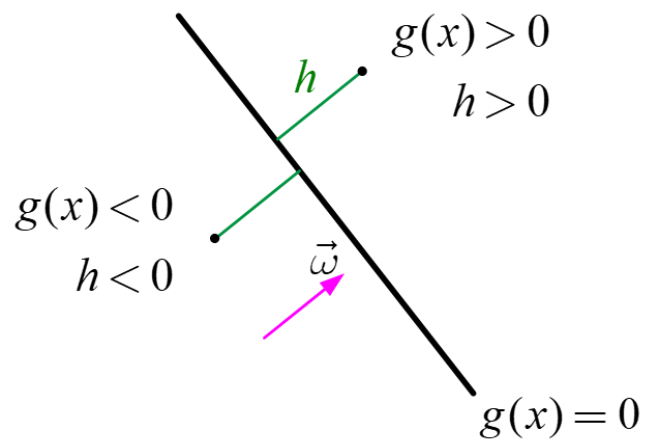
$$\omega^T x = \omega^T \left(x_{\perp} + r \frac{\omega}{\|\omega\|} \right) = r \frac{\omega^T \omega}{\|\omega\|} = r \|\omega\|$$

$$\begin{aligned} g(x) &= \omega_0 + \omega^T x \\ &= \omega_0 + r \|\omega\| \quad (r = d + h) \\ &= \omega_0 + (d + h) \|\omega\| \\ &= \omega_0 + \left(-\frac{\omega_0}{\|\omega\|} + h \right) \|\omega\| \\ &= h \|\omega\| \end{aligned}$$

$$\therefore h = \frac{g(x)}{\|\omega\|} \implies \text{orthogonal signed distance from the line}$$



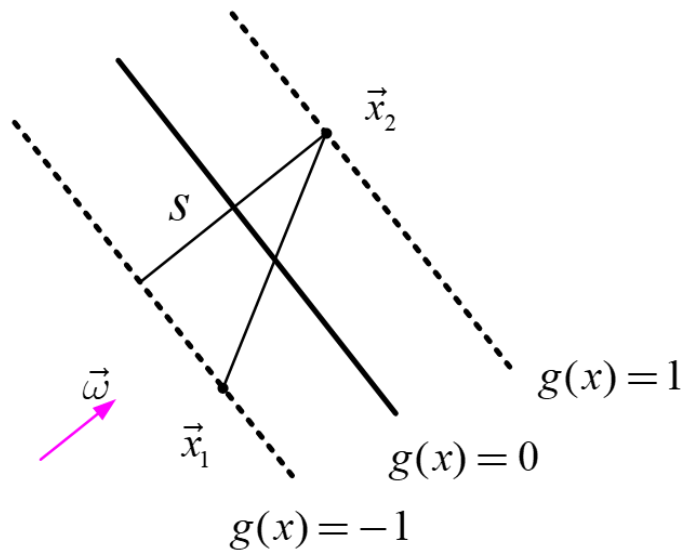
Distance from a Line: h



$$h = \frac{g(x)}{\|\omega\|}$$

Distance from a Line: h

- Another method to find a distance between $g(x) = 1$ and $g(x) = -1$
- Suppose $g(x_1) = -1$ and $g(x_2) = 1$



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

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

SVM Algorithm

Illustrative Example

- Binary classification
 - C_1 and C_0
- Features
 - The coordinate of the unknown animal i in the zoo

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

Hyperplane

- Is it possible to distinguish between C_1 and C_0 by its coordinates on a map of the zoo?
- We need to find a separating hyperplane (or a line in 2D)

$$\omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$$

$$\omega_0 + [\omega_1 \quad \omega_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$\omega_0 + \omega^T x = 0$$

Decision Making

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

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

$$\omega_0 + \omega^T x < 0 \implies x \text{ belongs to } C_0$$

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

Decision Boundary or Band

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

or

- Find ω and ω_0 such that
 - $x \in C_1$ given $\omega_0 + \omega^T x > 1$ and
 - $x \in C_0$ given $\omega_0 + \omega^T x < -1$

$$\omega_0 + \omega^T x > b, \quad (b > 0)$$

$$\iff \frac{\omega_0}{b} + \frac{\omega^T}{b} x > 1$$

$$\iff \omega'_0 + \omega'^T x > 1$$

Data Generation for Classification

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

```
g = 0.8*x1 + x2 - 3
g1 = g - 1
g0 = g + 1
```

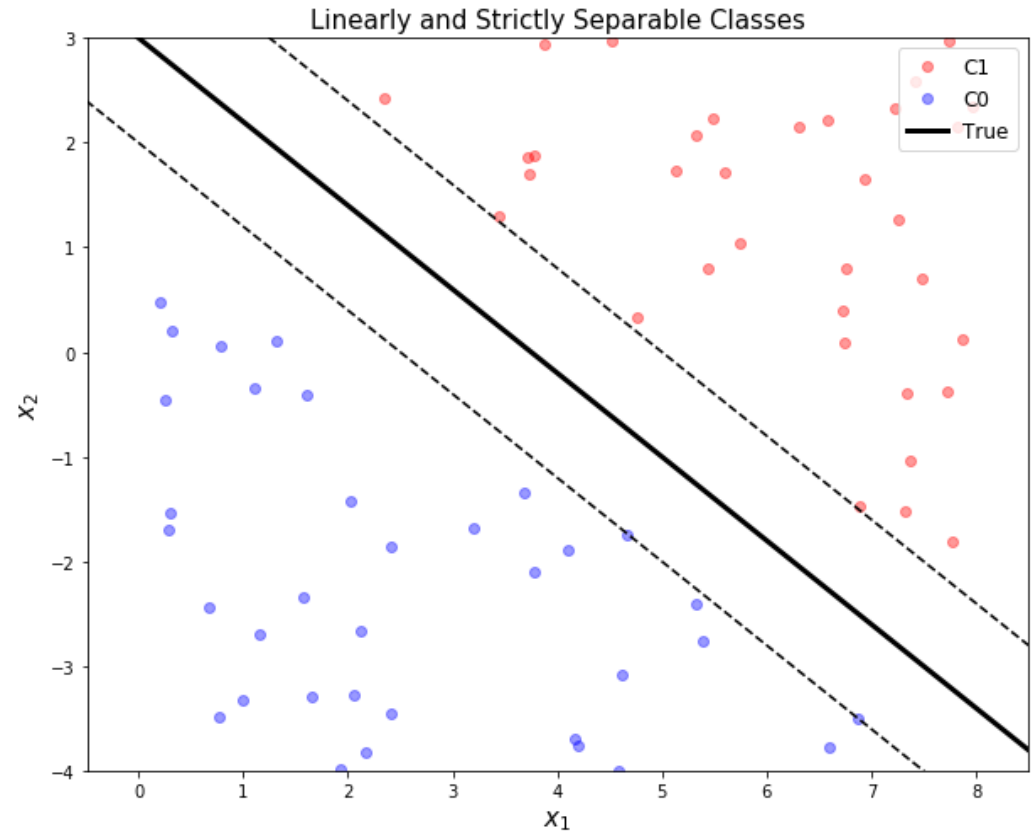
```
C1 = np.where(g1 >= 0)[0]
C0 = np.where(g0 < 0)[0]
```

```
xp = np.linspace(-1,9,100).reshape(-1,1)
ypt = -0.8*xp + 3
```

```
# see how data are generated
```

```
xp = np.linspace(-1,9,100).reshape(-1,1)
ypt = -0.8*xp + 3
```

```
plt.figure(figsize=(10, 8))
plt.plot(x1[C1], x2[C1], 'ro', alpha = 0.4, label = 'C1')
plt.plot(x1[C0], x2[C0], 'bo', alpha = 0.4, label = 'C0')
plt.plot(xp, ypt, 'k', linewidth = 3, label = 'True')
plt.plot(xp, ypt-1, '--k')
plt.plot(xp, ypt+1, '--k')
plt.title('Linearly and Strictly Separable Classes', fontsize = 15)
plt.xlabel(r'$x_1$', fontsize = 15)
plt.ylabel(r'$x_2$', fontsize = 15)
plt.legend(loc = 1, fontsize = 12)
plt.axis('equal')
plt.xlim([0, 8])
plt.ylim([-4, 3])
plt.show()
```



Optimization Formulation 1

- $n (= 2)$ features
- N belongs to C_1 in training set
- M belongs to C_0 in training set
- $m = N + M$ data points in training set

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

or

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

- ω and ω_0 are the unknown variables

Optimization Formulation 1

minimize something

$$\text{subject to } \begin{cases} \omega_0 + \omega^T x^{(1)} \geq 1 \\ \omega_0 + \omega^T x^{(2)} \geq 1 \\ \vdots \\ \omega_0 + \omega^T x^{(N)} \geq 1 \\ \omega_0 + \omega^T x^{(N+1)} \leq -1 \\ \omega_0 + \omega^T x^{(N+2)} \leq -1 \\ \vdots \\ \omega_0 + \omega^T x^{(N+M)} \leq -1 \end{cases}$$

minimize something

$$\text{subject to } \begin{cases} \omega^T x^{(1)} \geq 1 \\ \omega^T x^{(2)} \geq 1 \\ \vdots \\ \omega^T x^{(N)} \geq 1 \\ \omega^T x^{(N+1)} \leq -1 \\ \omega^T x^{(N+2)} \leq -1 \\ \vdots \\ \omega^T x^{(N+M)} \leq -1 \end{cases}$$

CVXPY 1

minimize something
subject to $X_1 \omega \geq 1$
 $X_0 \omega \leq -1$

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

$$X_0 = \begin{bmatrix} (x^{(N+1)})^T \\ (x^{(N+2)})^T \\ \vdots \\ (x^{(N+M)})^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(N+1)} & x_2^{(N+1)} \\ 1 & x_1^{(N+2)} & x_2^{(N+2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N+M)} & x_2^{(N+M)} \end{bmatrix}$$

```
import cvxpy as cvx

N = C1.shape[0]
M = C0.shape[0]

X1 = np.hstack([np.ones([N,1]), x1[C1], x2[C1]])
X0 = np.hstack([np.ones([M,1]), x1[C0], x2[C0]])

X1 = np.asmatrix(X1)
X0 = np.asmatrix(X0)
```

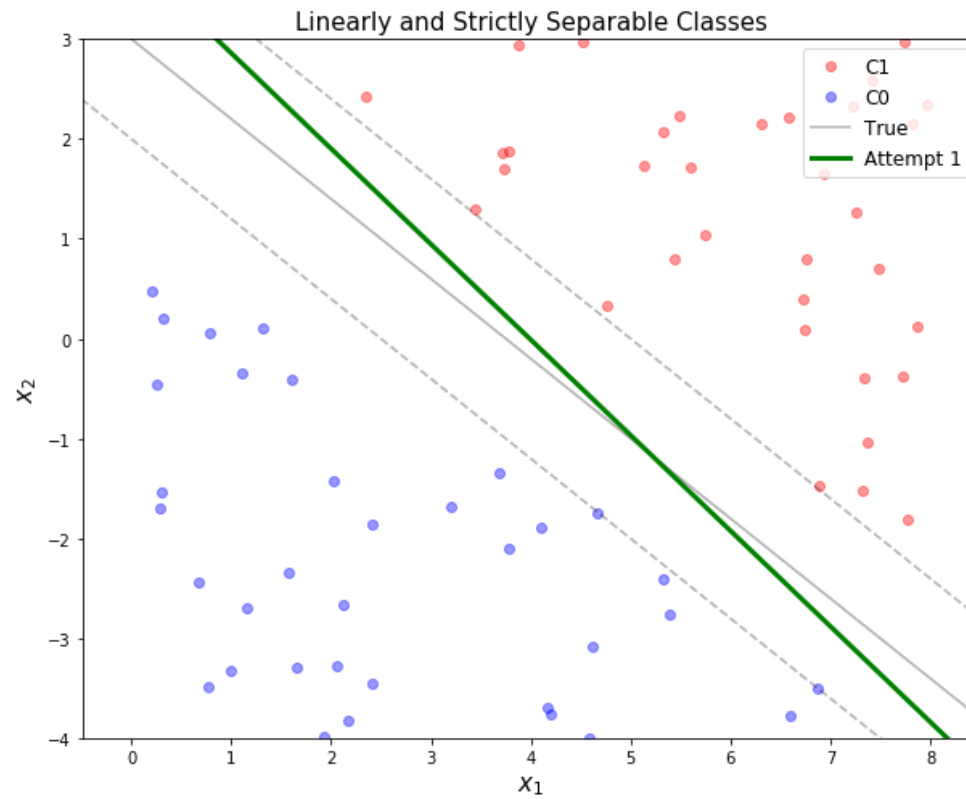
```
w = cvx.Variable([3,1])

obj = cvx.Minimize(1)
const = [X1*w >= 1, X0*w <= -1]
prob = cvx.Problem(obj, const).solve()

w = w.value
```

CVXPY 1

minimize something
subject to $X_1\omega \geq 1$
 $X_0\omega \leq -1$



Linear Classification: Outlier

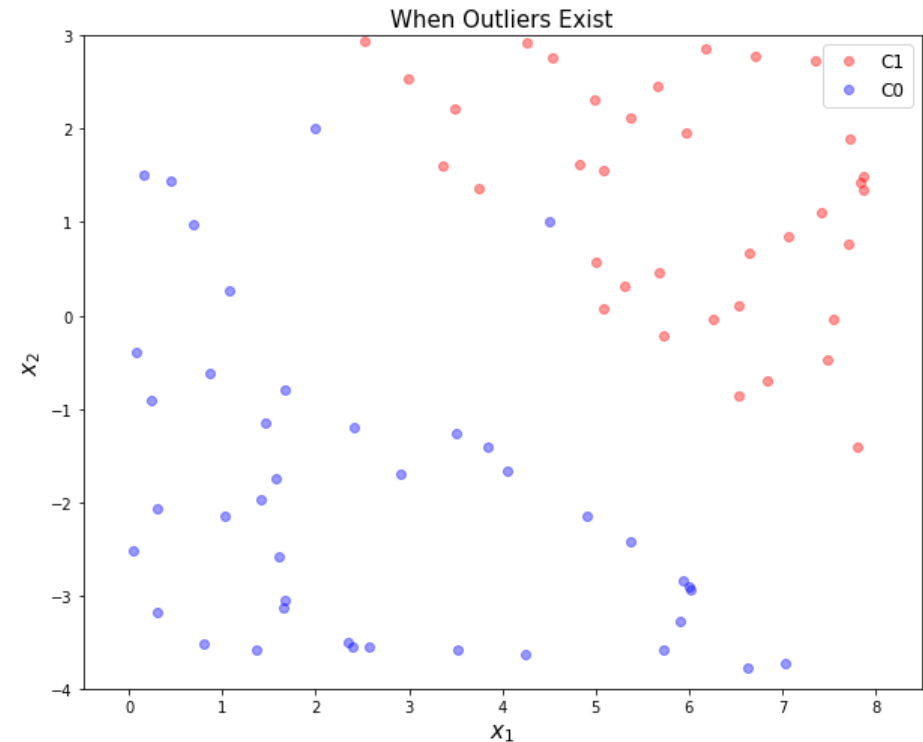
- Note that in the real world, you may have noise, errors, or outliers that do not accurately represent the actual phenomena
- Linearly non-separable case

Outliers

```
w = cvx.Variable([3,1])  
  
obj = cvx.Minimize(1)  
const = [X1*w >= 1, X0*w <= -1]  
prob = cvx.Problem(obj, const).solve()  
  
print(w.value)
```

None

- No solutions (hyperplane) exist
- We have to allow some training examples to be misclassified !
- but we want their number to be minimized



Optimization Formulation 2

- $n (= 2)$ features
- N belongs to C_1 in training set
- M belongs to C_0 in training set
- $m = N + M$ data points in training set

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

$$\begin{array}{ll} \text{minimize} & \text{something} \\ \text{subject to} & X_1 \omega \geq 1 \\ & X_0 \omega \leq -1 \end{array}$$

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

Optimization Formulation 2

- The optimization problem for the non-separable case

minimize something



$$\text{subject to } \begin{cases} \omega^T x^{(1)} \geq 1 \\ \omega^T x^{(2)} \geq 1 \\ \vdots \\ \omega^T x^{(N)} \geq 1 \\ \omega^T x^{(N+1)} \leq -1 \\ \omega^T x^{(N+2)} \leq -1 \\ \vdots \\ \omega^T x^{(N+M)} \leq -1 \end{cases}$$

minimize $\sum_{i=1}^N u_i + \sum_{i=1}^M v_i$

$$\text{subject to } \begin{cases} \omega^T x^{(1)} \geq 1 - u_1 \\ \omega^T x^{(2)} \geq 1 - u_2 \\ \vdots \\ \omega^T x^{(N)} \geq 1 - u_N \\ \omega^T x^{(N+1)} \leq -(1 - v_1) \\ \omega^T x^{(N+2)} \leq -(1 - v_2) \\ \vdots \\ \omega^T x^{(N+M)} \leq -(1 - v_M) \end{cases}$$
$$\begin{cases} u \geq 0 \\ v \geq 0 \end{cases}$$

Expressed in a Matrix Form

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

$$X_0 = \begin{bmatrix} (x^{(N+1)})^T \\ (x^{(N+2)})^T \\ \vdots \\ (x^{(N+M)})^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(N+1)} & x_2^{(N+1)} \\ 1 & x_1^{(N+2)} & x_2^{(N+2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(N+M)} & x_2^{(N+M)} \end{bmatrix}$$

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

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

$$\begin{array}{ll} \text{minimize} & 1^T u + 1^T v \\ \text{subject to} & X_1 \omega \geq 1 - u \\ & X_0 \omega \leq -(1 - v) \\ & u \geq 0 \\ & v \geq 0 \end{array}$$

$$\text{minimize} \quad \sum_{i=1}^N u_i + \sum_{i=1}^M v_i$$

$$\text{subject to} \quad \begin{cases} \omega^T x^{(1)} \geq 1 - u_1 \\ \omega^T x^{(2)} \geq 1 - u_2 \\ \vdots \\ \omega^T x^{(N)} \geq 1 - u_N \end{cases}$$

$$\begin{cases} \omega^T x^{(N+1)} \leq -(1 - v_1) \\ \omega^T x^{(N+2)} \leq -(1 - v_2) \\ \vdots \\ \omega^T x^{(N+M)} \leq -(1 - v_M) \end{cases}$$

$$\begin{cases} u \geq 0 \\ v \geq 0 \end{cases}$$

CVXPY 2

minimize something
subject to $X_1\omega \geq 1$
 $X_0\omega \leq -1$

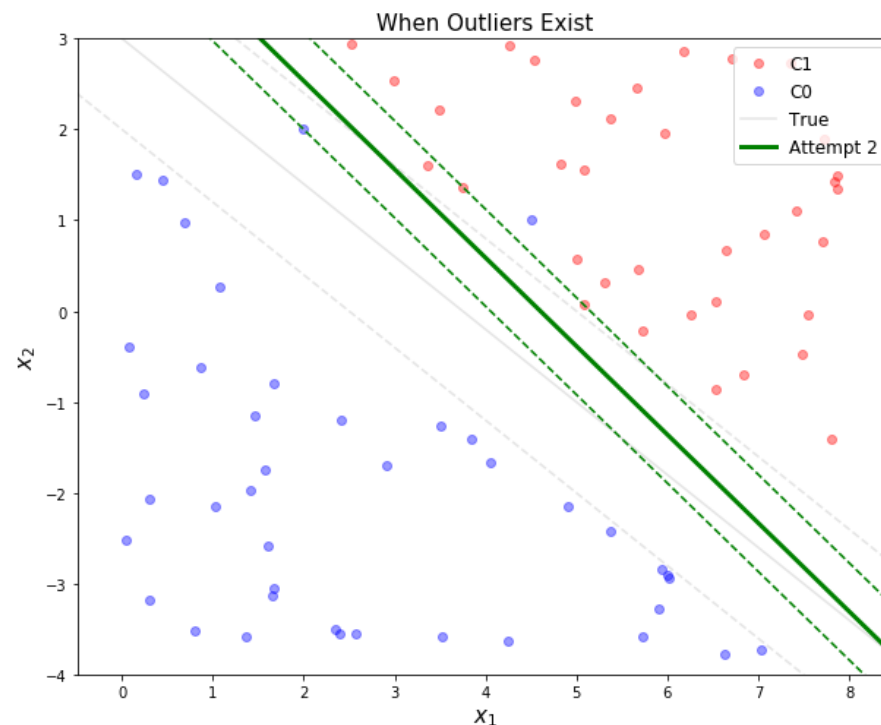


minimize $1^T u + 1^T v$
subject to $X_1\omega \geq 1 - u$
 $X_0\omega \leq -(1 - v)$
 $u \geq 0$
 $v \geq 0$

```
w = cvx.Variable([3,1])
u = cvx.Variable([N,1])
v = cvx.Variable([M,1])

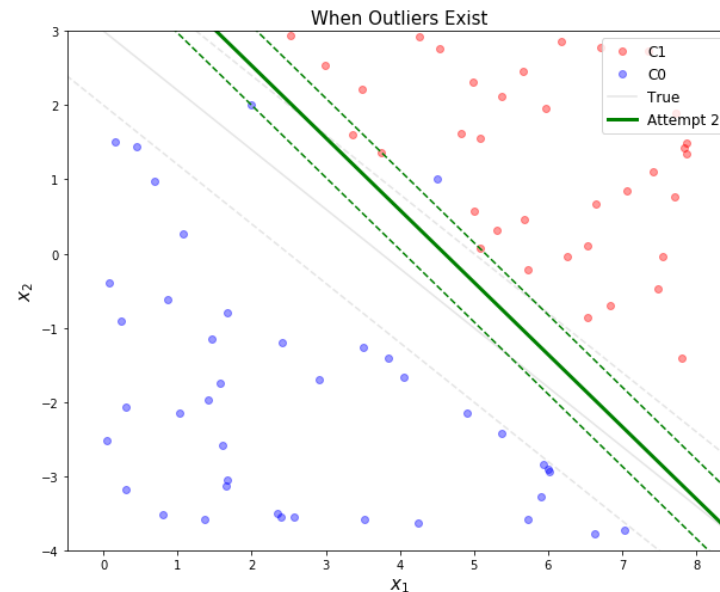
obj = cvx.Minimize(np.ones((1,N))*u + np.ones((1,M))*v)
const = [X1*w >= 1-u, X0*w <= -(1-v), u >= 0, v >= 0]
prob = cvx.Problem(obj, const).solve()

w = w.value
```



Further Improvement

- Notice that hyperplane is not as accurately represent the division due to the outlier

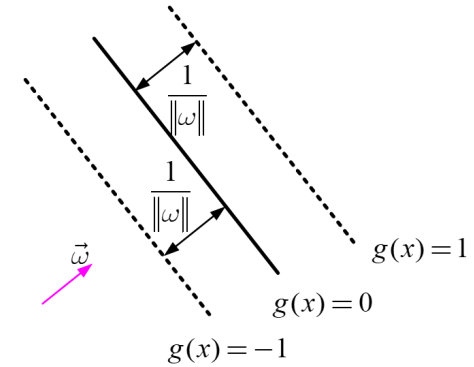


- Can we do better when there are noise data or outliers?
- Yes, but we need to look beyond linear programming
- Idea: large margin leads to good generalization on the test data

Maximize Margin

- Finally, it is Support Vector Machine (SVM)
- Distance (= margin)

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



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

maximize {minimum distance}

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

Support Vector Machine

$$\begin{aligned} & \text{minimize} && 1^T u + 1^T v \\ & \text{subject to} && X_1 \omega \geq 1 - u \\ & && X_0 \omega \leq -(1 - v) \\ & && u \geq 0 \\ & && v \geq 0 \end{aligned}$$



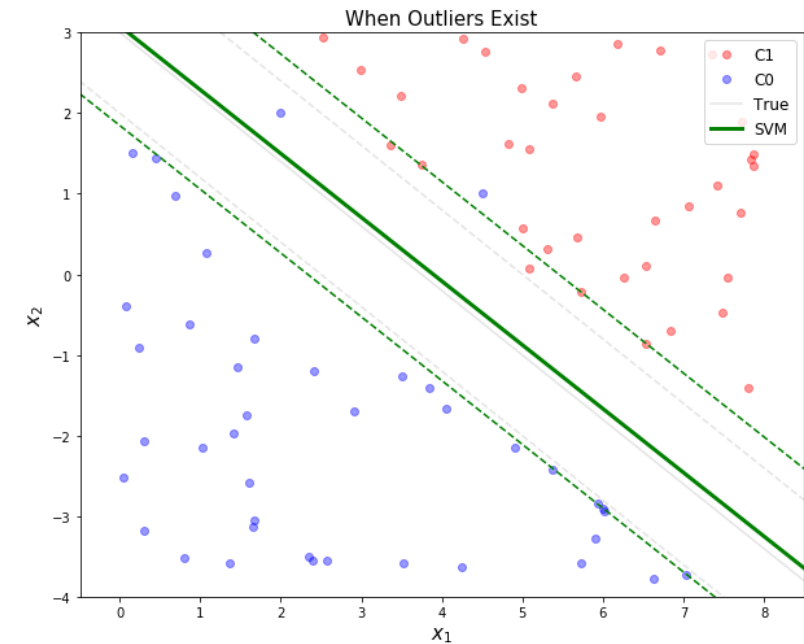
$$\begin{aligned} & \text{minimize} && \|\omega\|_2 + \gamma(1^T u + 1^T v) \\ & \text{subject to} && X_1 \omega \geq 1 - u \\ & && X_0 \omega \leq -(1 - v) \\ & && u \geq 0 \\ & && v \geq 0 \end{aligned}$$

```
g = 3

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, X0*w <= -(1-v), u >= 0, v >= 0]
prob = cvx.Problem(obj, const).solve()

w = w.value
```



Support Vector Machine

- In a more compact form

$$\begin{aligned} & \text{minimize} && \|\omega\|_2 + \gamma(1^T u + 1^T v) \\ & \text{subject to} && X_1 \omega \geq 1 - u \\ & && X_0 \omega \leq -(1 - v) \\ & && u \geq 0 \\ & && v \geq 0 \end{aligned}$$



$$\begin{aligned} & \text{minimize} && \|\omega\|_2 + \gamma(1^T \xi) \\ & \text{subject to} && y_n \cdot (\omega^T x_n) \geq 1 - \xi_n \\ & && \xi \geq 0 \end{aligned}$$

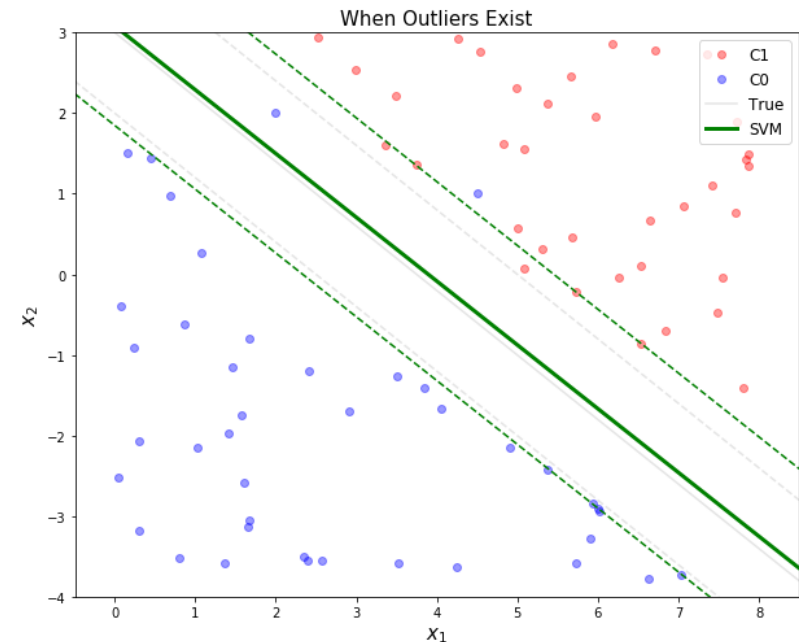
```
X = np.vstack([X1, X0])
y = np.vstack([np.ones([N,1]), -np.ones([M,1])])

m = N + M

w = cvx.Variable([3,1])
d = cvx.Variable([m,1])

obj = cvx.Minimize(cvx.norm(w,2) + g*(np.ones([1,m])*d))
const = [cvx.multiply(y, X*w) >= 1-d, d >= 0]
prob = cvx.Problem(obj, const).solve()

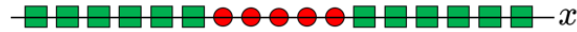
w = w.value
```



Nonlinear Classification

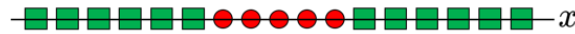
Classifying Non-linear Separable Data

- Consider the binary classification problem
 - each example represented by a single feature x
 - No linear separator exists for this data

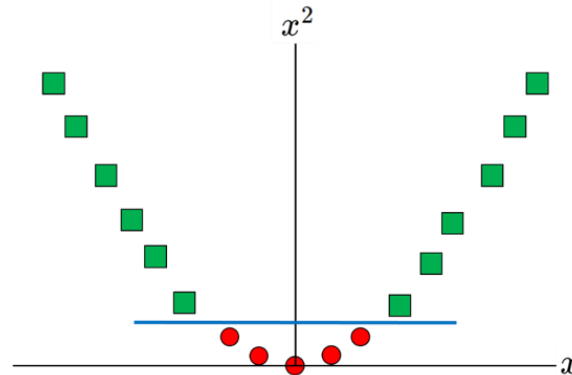


Classifying Non-linear Separable Data

- Consider the binary classification problem
 - each example represented by a single feature x
 - No linear separator exists for this data



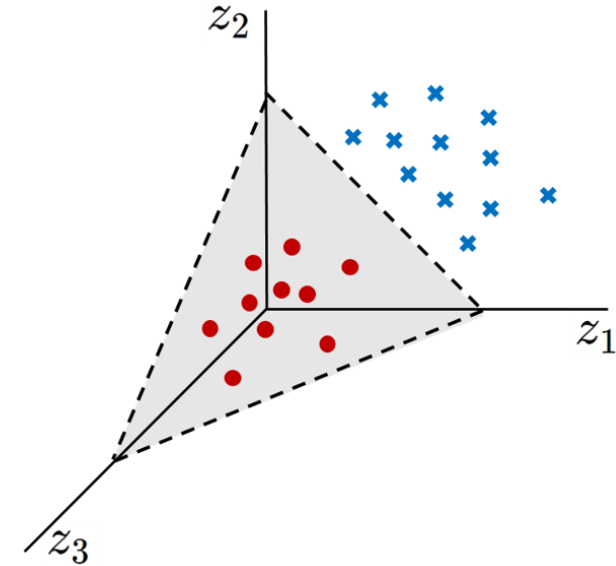
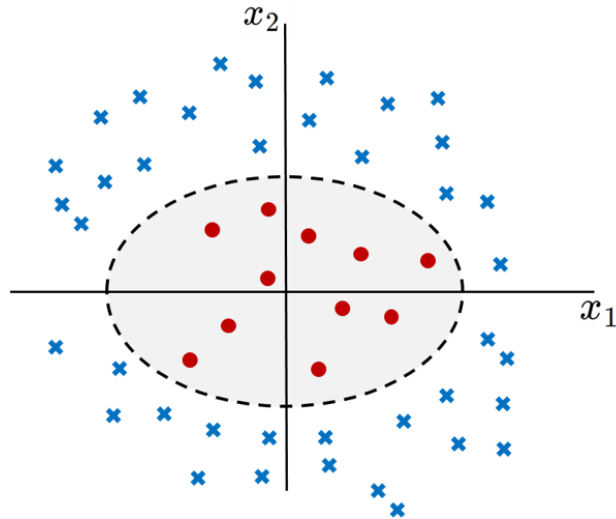
- Now map each example as $x \rightarrow \{x, x^2\}$
- Data now becomes linearly separable in the new representation



- Linear in the new representation = nonlinear in the old representation

Classifying Non-linear Separable Data

- Let's look at another example
 - Each example defined by a two features
 - No linear separator exists for this data $x = \{x_1, x_2\}$



- Now map each example as $x = \{x_1, x_2\} \rightarrow z = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$
 - Each example now has three features (derived from the old representation)
- Data now becomes linear separable in the new representation

Kernel

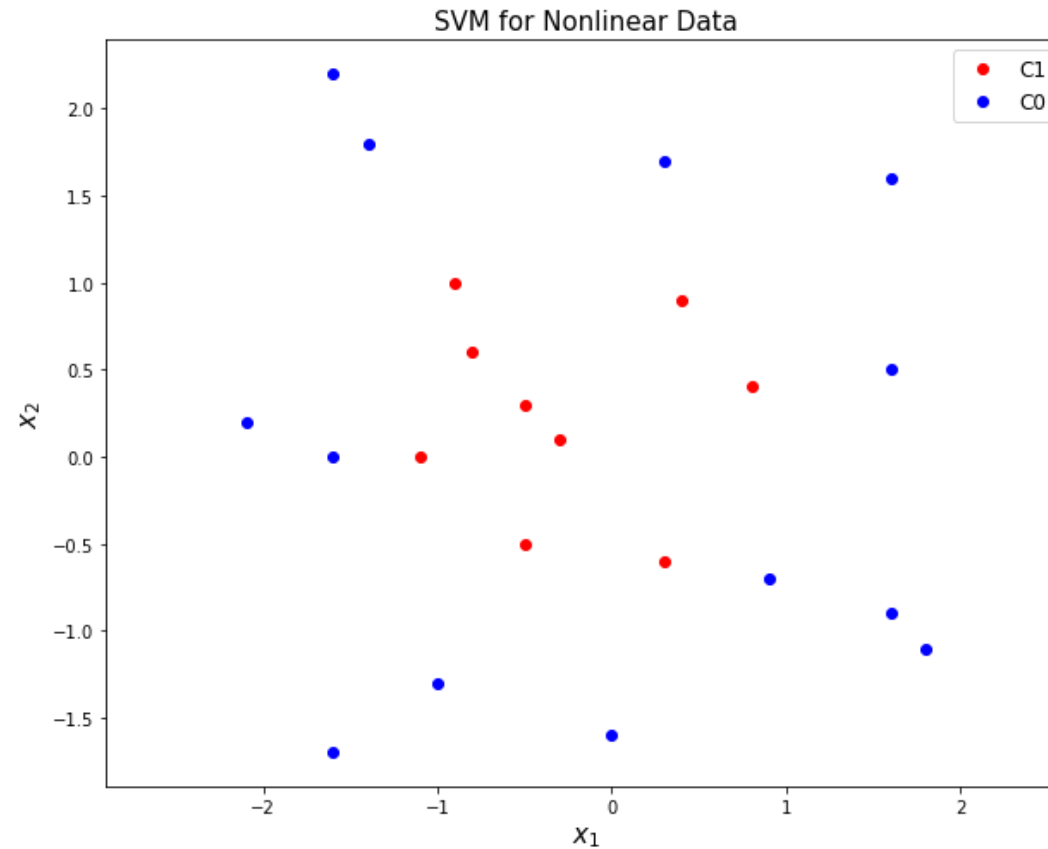
- Often we want to capture nonlinear patterns in the data
 - nonlinear regression: input and output relationship may not be linear
 - nonlinear classification: classes may not be separable by a linear boundary
- Linear models (e.g. linear regression, linear SVM) are not just rich enough
 - by mapping data to higher dimensions where it exhibits linear patterns
 - apply the linear model in the new input feature space
 - mapping = changing the feature representation
- Kernels: make linear model work in nonlinear settings

Nonlinear Classification

SVM with a polynomial
Kernel visualization

Created by:
Udi Aharoni

Classifying Non-linear Separable Data



Classifying Non-linear Separable Data

```
N = X1.shape[0]
M = X0.shape[0]

X = np.vstack([X1, X0])
y = np.vstack([np.ones([N,1]), -np.ones([M,1])])

X = np.asmatrix(X)
y = np.asmatrix(y)

m = N + M
Z = np.hstack([np.ones([m,1]), np.square(X[:,0]), np.sqrt(2)*np.multiply(X[:,0],X[:,1]), np.square(X[:,1])])

g = 10

w = cvx.Variable([4, 1])
d = cvx.Variable([m, 1])

obj = cvx.Minimize(cvx.norm(w, 2) + g*np.ones([1,m])*d)
const = [cvx.multiply(y, Z*w) >= 1-d, d >= 0]
prob = cvx.Problem(obj, const).solve()

w = w.value
print(w)
```

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \implies z = \phi(x) = \begin{bmatrix} 1 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$\begin{array}{ll} \text{minimize} & \|\omega\|_2 + \gamma(1^T \xi) \\ \text{subject to} & y_n \cdot (\omega^T x_n) \geq 1 - \xi_n \\ & \xi \geq 0 \end{array}$$

Classifying Non-linear Separable Data

