# Supervised Learning

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

Table of Contents


- I. 0. Supervised learning
- II. 1. Regression
  - I. 1.1. k-Nearest Neighbor Regression
  - II. 1.2. Linear Regression
- III. 2. Classification
  - I. 2.1. Data Generation for Classification
  - II. 2.2. K-nearest neighbors
- IV. 3. Support Vector Machine (SVM)
  - I. 3.0. Distance from a line
  - II. 3.1. Binary Classification
  - III. 3.2. Multi Classification
- V. 4. Logistic Regression
  - I. 4.1. Binary Classification
  - II. 4.2. Multi Classification
- VI. 5. Nonlinear Classification
- VII. 6. Save Model

# 0. Supervised learning

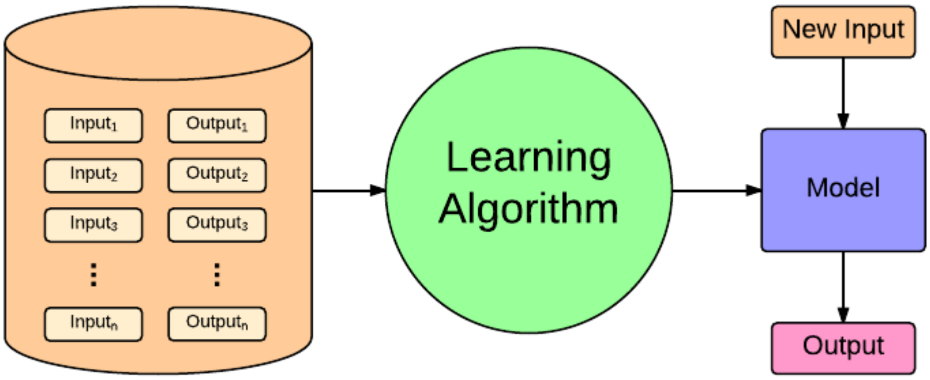
- Given training set $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \cdots, \left(x^{(m)}, y^{(m)}\right)\right\}$

- Want to find a function $g_\omega$ with learning parameter, $\omega$
  - $g_\omega$ desired to be as close as possible to $y$ for future $(x, y)$
  - $i.e., g_\omega(x) \sim y$

- Define a loss function $\ell$

- Solve the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\omega) = \frac{1}{m}\sum_{i=1}^{m} \ell\left(g_\omega\left(x^{(i)}\right), y^{(i)}\right) \\ \text{subject to} \quad & \omega \in \boldsymbol{\omega} \end{aligned}$$



# 1. Regression

## 1.1. k-Nearest Neighbor Regression

The goal is to make quantitative (real valued) predictions on the basis of a (vector of) features or attributes.

We write our model as

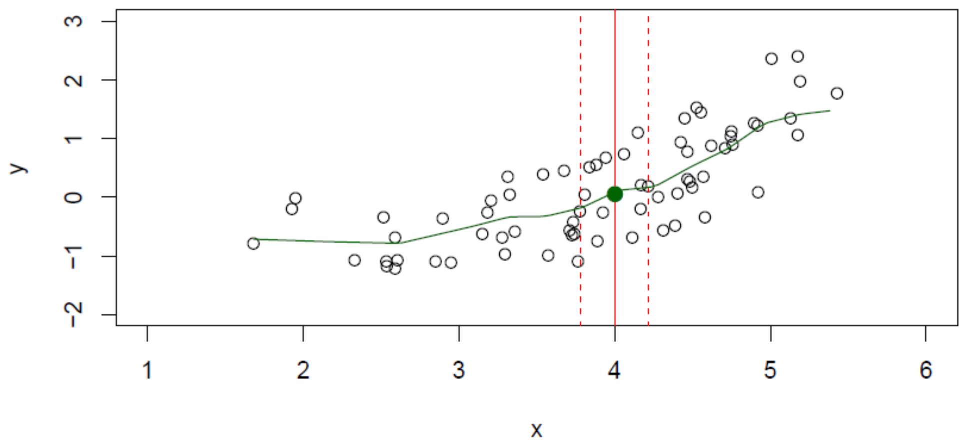
$$Y = f(X) + \epsilon$$

where $\epsilon$ captures measurement errors and other discrepancies.

Then, with a good $f$ we can make predictions of $Y$ at new points $X = x$. One possible way so called "nearest neighbor method" is:

$$\hat{f} = \text{Ave}\ (Y \mid X \in \mathcal{N}(x))$$

where $\mathcal{N}(x)$ is some neighborhood of $x$




- Regression 에 사용할 데이터 생성

In [1]:

```
import numpy as np
```

In [2]:

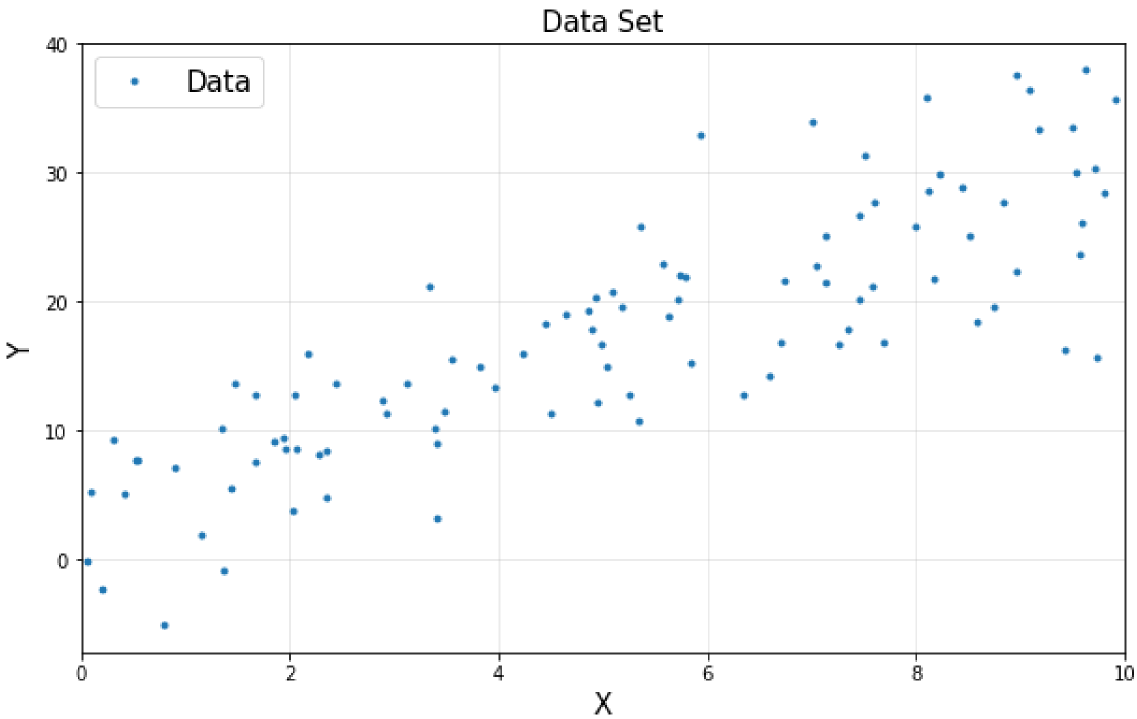
```
N = 100
w1 = 3
w0 = 2
x = np.random.uniform(0, 10, N)
y = w1*x + w0 + 5*np.random.normal(0, 1, N)
```

In [3]:

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

In [4]:

```
plt.figure(figsize=(10, 6))
plt.title('Data Set', fontsize=15)
plt.plot(x, y, '.', label='Data')
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.legend(fontsize=15)
plt.xlim([0, 10])
plt.grid(alpha=0.3)
plt.show()
```




- sklearn.neighbors에 있는 KNeighborsRegressor import

In [5]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [6]:

```
reg = KNeighborsRegressor(n_neighbors=10)
reg.fit(x.reshape(-1, 1), y)
```

Out[6]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=10, p=2,
          weights='uniform')
```

In [7]:

```
x_new = np.array([[5]])
```

In [8]:

```
pred = reg.predict(5)
```

In [9]:

```
print(pred)
```

```
[ 16.5196895]
```

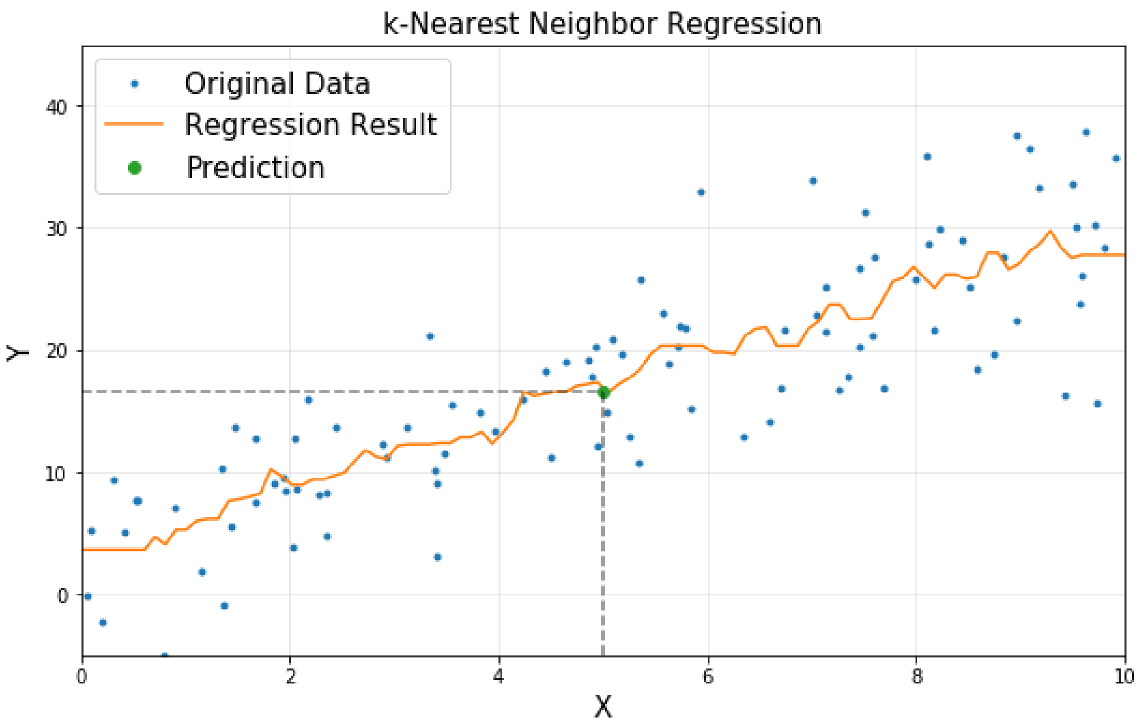
- plot

In [10]:

```
xp = np.linspace(0, 10, 100).reshape(-1, 1)
yp = reg.predict(xp)
```

In [11]:

```
plt.figure(figsize=(10, 6))
plt.title('k-Nearest Neighbor Regression', fontsize=15)
plt.plot(x, y, '.', label='Original Data')
plt.plot(xp, yp, label='Regression Result')
plt.plot(x_new, pred, 'o', label='Prediction')
plt.plot([x_new[0,0], x_new[0,0]], [-5, pred[0]], 'k--', alpha=0.5)
plt.plot([0, x_new[0,0]], [pred[0], pred[0]], 'k--', alpha=0.5)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.legend(fontsize=15)
plt.xlim([0, 10])
plt.ylim([-5, 45])
plt.grid(alpha=0.3)
plt.show()
```



# 1.2. Linear Regression

선형 회귀 분석 (fitting)

$$\text{Given } \begin{cases} x_i \text{ : inputs} \\ y_i \text{ : outputs} \end{cases}$$ , Find $\omega_1$ and $\omega_0$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \approx \hat{y}_i = \omega_1 x_i + \omega_0$$

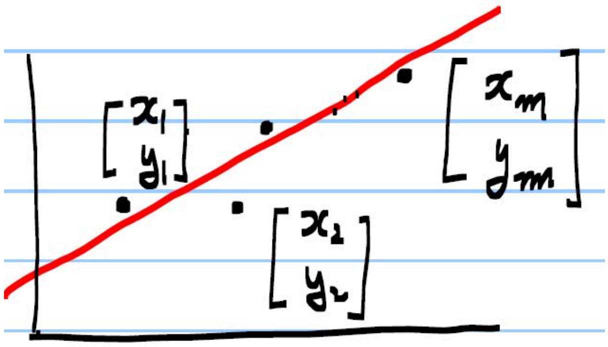
- $\hat{y}_i$ : predicted output
- $\omega = \begin{bmatrix} \omega_1 \\ \omega_0 \end{bmatrix}$ : Model parameters

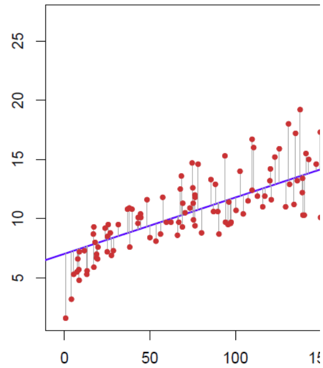
$$\hat{y}_i = f(x_i, \omega) \;\; \text{in general}$$

- in many cases, a linear model to predict $y_i$ used

$$\hat{y}_i = \omega_1 x_i + \omega_0$$

$$\text{such that } \min_{\omega_1, \omega_0} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$




To see how it works, click [here (http://i-systems.github.io/HSE545/machine%20learning%20all/03%20Regression/iSystems_01_Regression.html)](http://i-systems.github.io/HSE545/machine%20learning%20all/03%20Regression/iSystems_01_Regression.html)

- Regression 에 사용할 데이터 생성

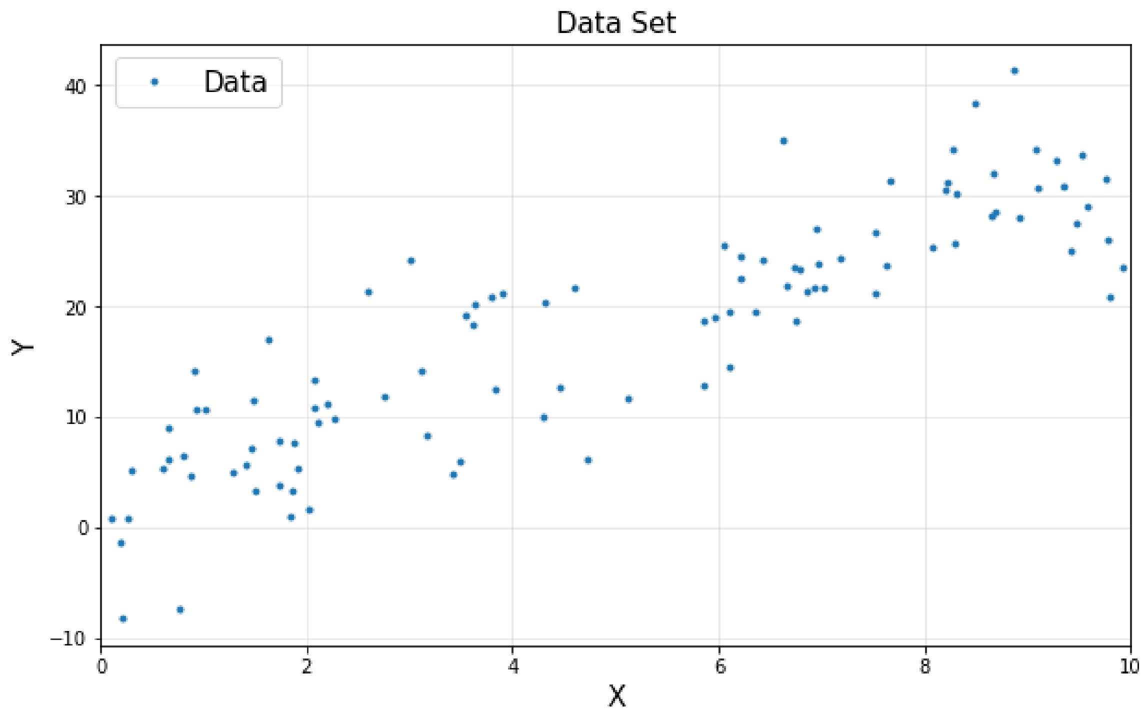
In [12]:

```
import numpy as np

N = 100
w1 = 3
w0 = 2
x = np.random.uniform(0, 10, N)
y = w1*x + w0 + 5*np.random.normal(0, 1, N)

import matplotlib.pyplot as plt
% matplotlib inline

plt.figure(figsize=(10, 6))
plt.title('Data Set', fontsize=15)
plt.plot(x, y, '.', label='Data')
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.legend(fontsize=15)
plt.xlim([0, 10])
plt.grid(alpha=0.3)
plt.show()
```




- sklearn.linear_model 에 있는 LinearRegression import

In [13]:

```
from sklearn.linear_model import LinearRegression
```

In [14]:

```
reg = LinearRegression()
reg.fit(x.reshape(-1, 1), y)
```

Out[14]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

- 새로운 데이터에 대하여 predict

In [15]:

```
x_new = np.array([[6]])
```

In [16]:

```
pred = reg.predict(x_new)
```

In [17]:

```
print(pred)
```

```
[ 20.78413979]
```

- parameters 확인 및 plot

In [18]:

```
w1_pred = reg.coef_
w0_pred = reg.intercept_
print('w1 pred : ', w1_pred[0])
print('w1 original : ', w1)
print('w0 pred : ', w0_pred)
print('w0 : ', w0)
```

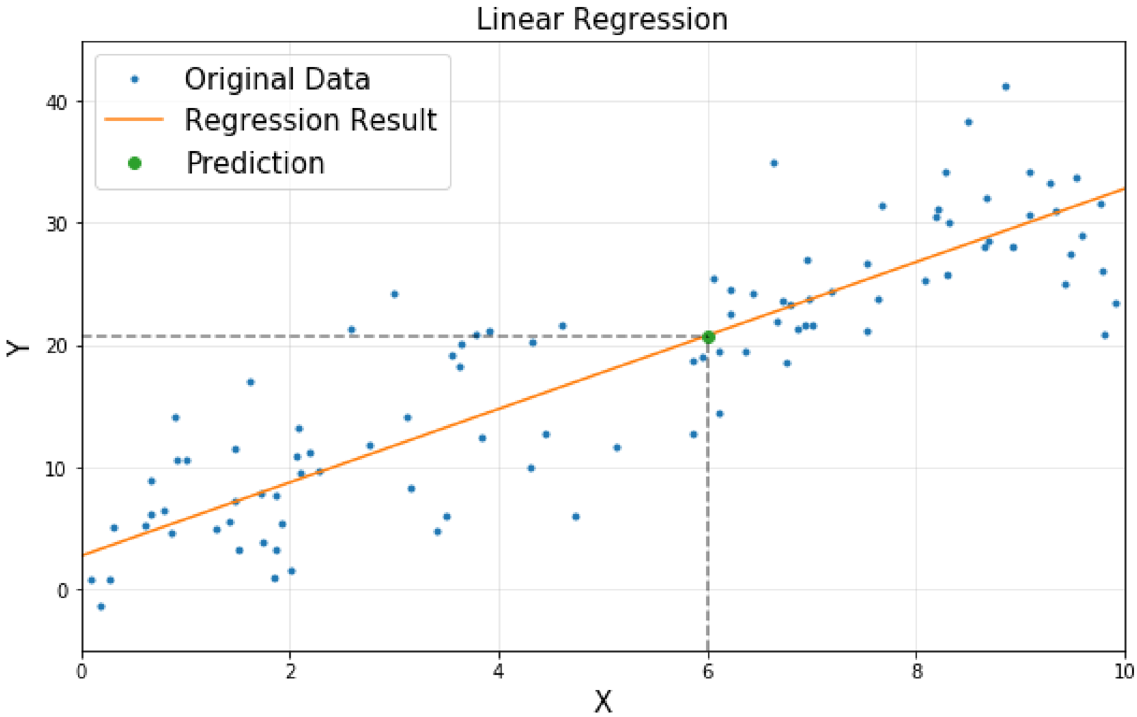
```
w1 pred :  3.00789939237
w1 original :  3
w0 pred :  2.73674343365
w0 :  2
```

In [19]:

```
xp = np.linspace(0, 10)
yp = w1_pred*xp + w0_pred
```

In [20]:

```
plt.figure(figsize=(10, 6))
plt.title('Linear Regression', fontsize=15)
plt.plot(x, y, '.', label='Original Data')
plt.plot(xp, yp, label='Regression Result')
plt.plot(x_new, pred, 'o', label='Prediction')
plt.plot([x_new[0,0], x_new[0,0]], [-5, pred[0]], 'k--', alpha=0.5)
plt.plot([0, x_new[0,0]], [pred[0], pred[0]], 'k--', alpha=0.5)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.legend(fontsize=15)
plt.xlim([0, 10])
plt.ylim([-5, 45])
plt.grid(alpha=0.3)
plt.show()
```




# 2. Classification

## 2.1. Data Generation for Classification

- Classification에 사용할 데이터 생성

In [21]:

```
import matplotlib.pyplot as plt

C0 = np.random.multivariate_normal([0, 0], np.eye(2), 200)
C1 = np.random.multivariate_normal([10, 10], np.eye(2), 200)
C2 = np.random.multivariate_normal([-5, 5], np.eye(2), 200)

y0 = np.array(C1.shape[0]*[0])
y1 = np.array(C1.shape[0]*[1])
y2 = np.array(C1.shape[0]*[2])
```

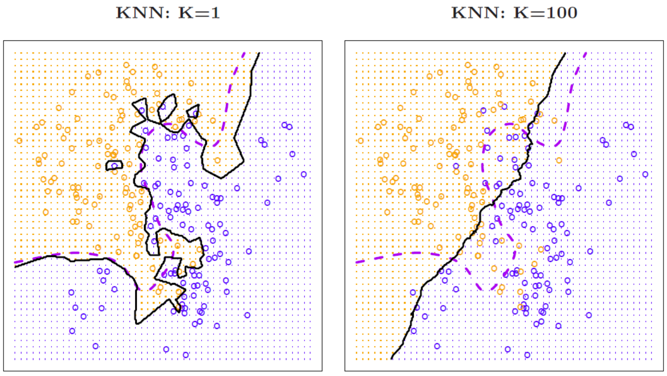
- Plot을 통하여 데이터 파악

In [22]:

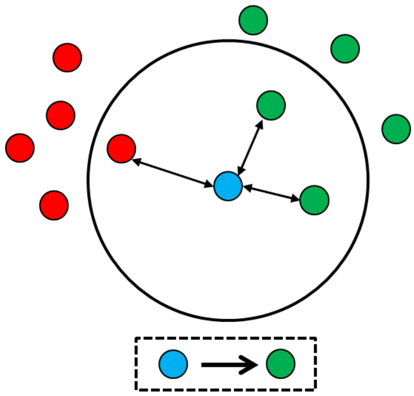
```
plt.figure(figsize=(10, 6))
plt.title('Data Classes', fontsize=15)
plt.plot(C0[:,0], C0[:,1], '.', label='Class 0')
plt.plot(C1[:,0], C1[:,1], '.', label='Class 1')
plt.plot(C2[:,0], C2[:,1], '.', label='Class 2')
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```

# 2.2. K-nearest neighbors

- In k-NN classification, an object is assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small).
- If k = 1, then the object is simply assigned to the class of that single nearest neighbor.




- Zoom in,



### Binary Classification

- C0와 C1 데이터를 분류
- 데이터를 X, y로 병합

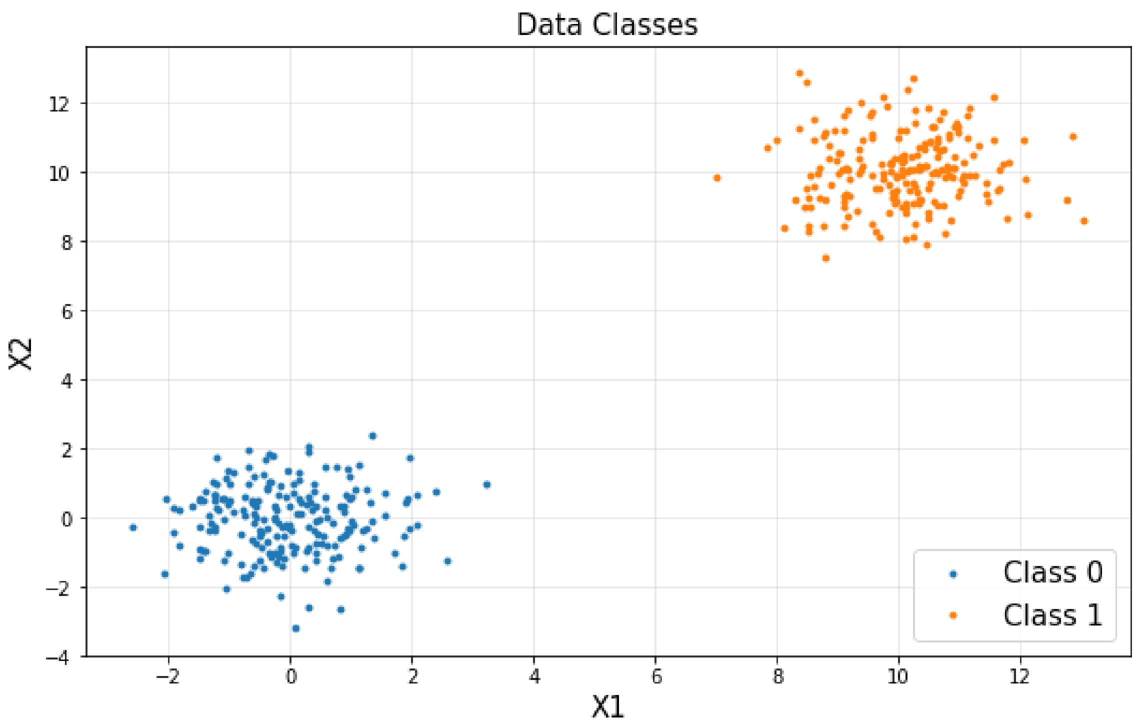
In [23]:

```
X = np.vstack([C0, C1])
y = np.hstack([y0, y1])
```

- Plot을 통하여 결과 확인

In [24]:

```
plt.figure(figsize=(10, 6))
plt.title('Data Classes', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- Sklearn neighbors을 import
- KNeighborsClassifier 개체를 선언 후 피팅

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [26]:

```
clf = KNeighborsClassifier(n_neighbors=2)
clf.fit(X, y)
```

Out[26]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

- 새로운 데이터에 대한 결과 확인
- Input shape을 맞추는 것에 주의

In [27]:

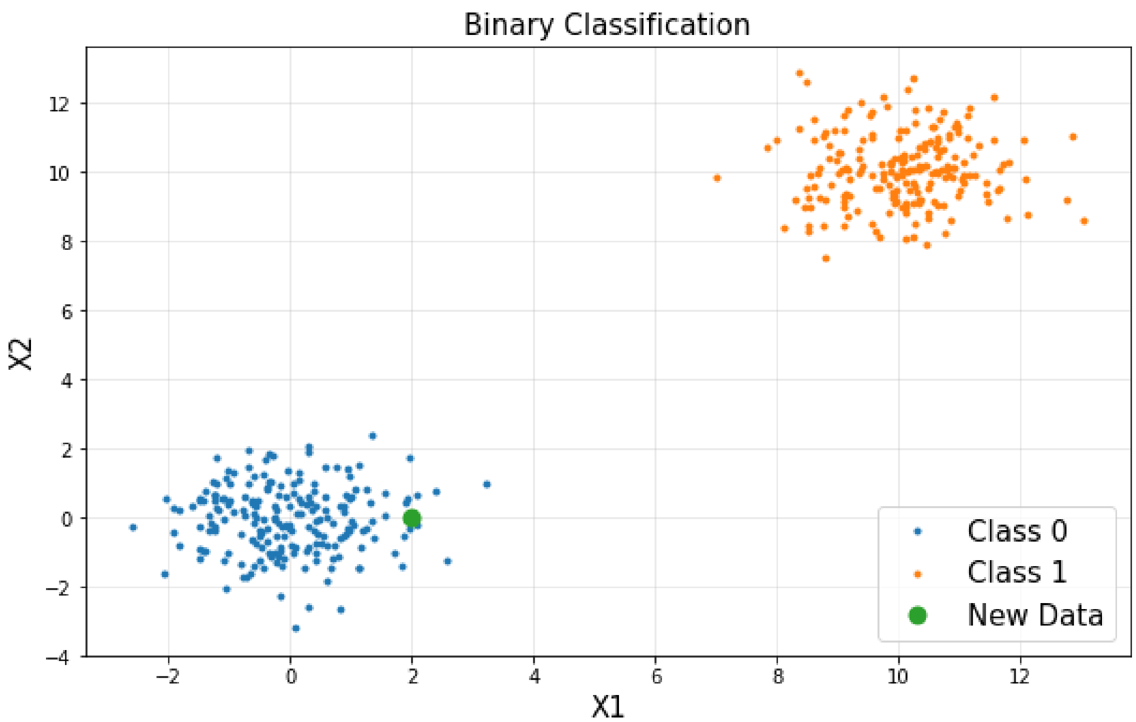
```
X_new = np.array([2, 0])
X_new = X_new.reshape(1, -1)
X_new.shape
```

Out[27]:

```
(1, 2)
```

In [28]:

```
plt.figure(figsize=(10, 6))
plt.title('Binary Classification', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.plot(X_new[0,0], X_new[0,1], 'o', label='New Data', ms=5, mew=5)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- Class 0에 속함

In [29]:

```
pred = clf.predict(X_new)
print(pred)
```

```
[0]
```

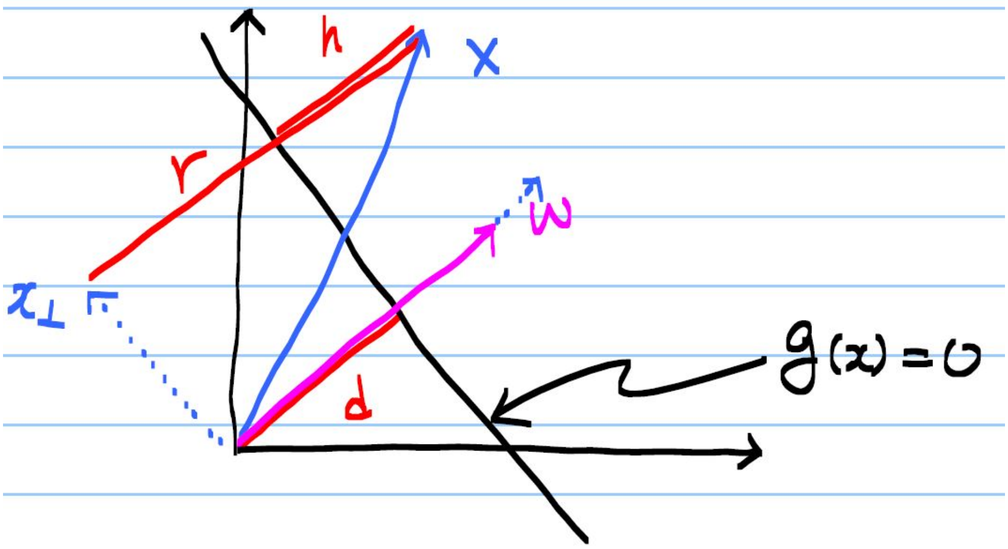
# 3. Support Vector Machine (SVM)

To see how it works, click here (http://i-systems.github.io/HSE545/machine%20learning%20all/04%20Classification/iSystems_02_SVM.html)

- 가장 많이 쓰이는 모델
- 경계선과 데이터 사이의 거리 (margin) 을 최대화 하는 모델

# 3.0. Distance from a line

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




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

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

$$\therefore \omega : \text{normal to the line (orthogonal)} \implies \text{tells the direction of the line}$$

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

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

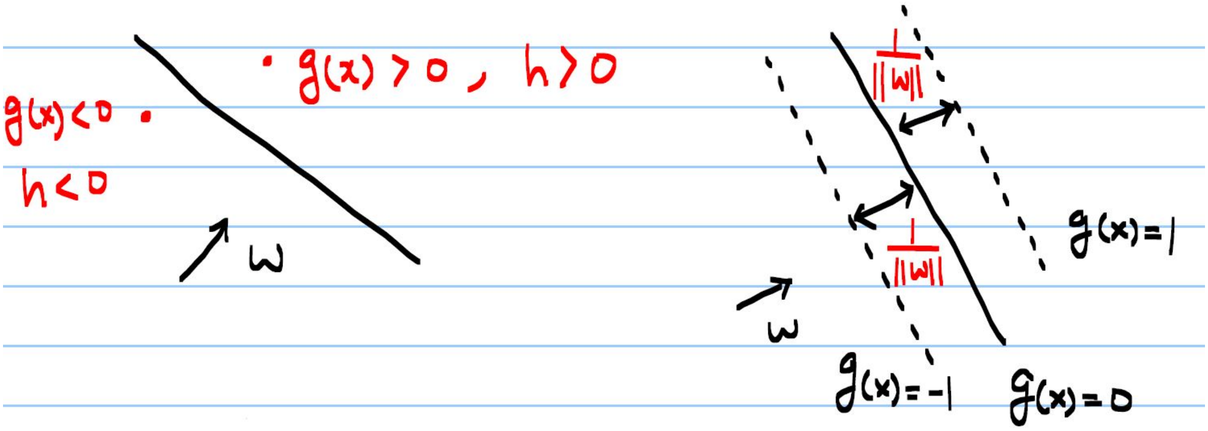
- for any vector of $x$

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

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

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

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

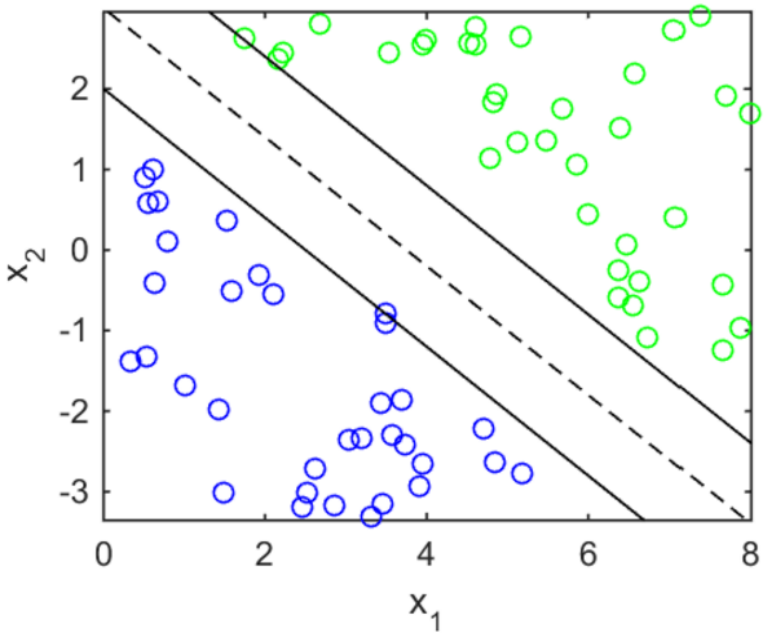



- Distance (= margin)

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

- Minimize $\|\omega\|_2$ to maximize the margin

$$
\begin{aligned}
\text{minimize} \quad & \|\omega\|_2 \\
\text{subject to} \quad & C_1\omega + \omega_0 \geq 1 \\
& C_2\omega + \omega_0 \leq -1
\end{aligned}
$$



# 3.1. Binary Classification

- C0와 C1 데이터를 분류
- 데이터를 X, y로 병합

In [30]:

```
X = np.vstack([C0, C1])
y = np.concatenate([y0, y1])
```

- sklearn.svm 모듈에서 SVC import
- svc 개체를 선언 후 피팅

In [31]:

```
from sklearn.svm import SVC
```

In [32]:

```
clf = SVC()
clf.fit(X, y)
```

Out[32]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

- 새로운 데이터에 대한 결과 확인
- Input shape을 맞추는 것에 주의

In [33]:

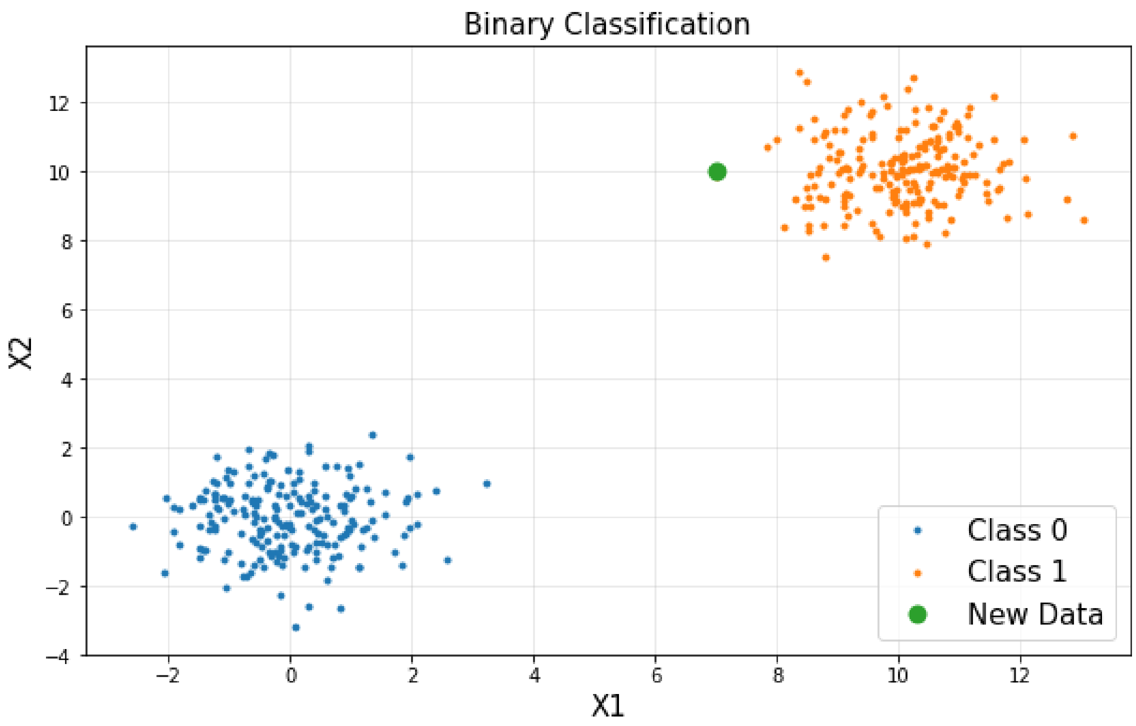
```
X_new = np.array([7, 10])
X_new = X_new.reshape(1, -1)
X_new.shape
```

Out[33]:

```
(1, 2)
```

In [34]:

```
plt.figure(figsize=(10, 6))
plt.title('Binary Classification', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.plot(X_new[0,0], X_new[0,1], 'o', label='New Data', ms=5, mew=5)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- 새로운 데이터는 Class 1에 속함

In [35]:

```
clf.predict(X_new)
```

Out[35]:

```
array([1])
```

## 3.2. Multi Classification

- C0, C1, C2 데이터를 분류
- Binary classification 에 이용된 코드와 동일
- X, y로 병합

In [36]:

```
X = np.vstack([C0, C1, C2])
y = np.concatenate([y0, y1, y2])
```

- sklearn.svm 모듈에서 SVC import
- svc 개체를 선언 후 피팅

In [37]:

```
from sklearn.svm import SVC
```

In [38]:

```
clf = SVC()
clf.fit(X, y)
```

Out[38]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

- 새로운 데이터에 대한 결과 확인
- Input shape을 맞추는 것에 주의

In [39]:

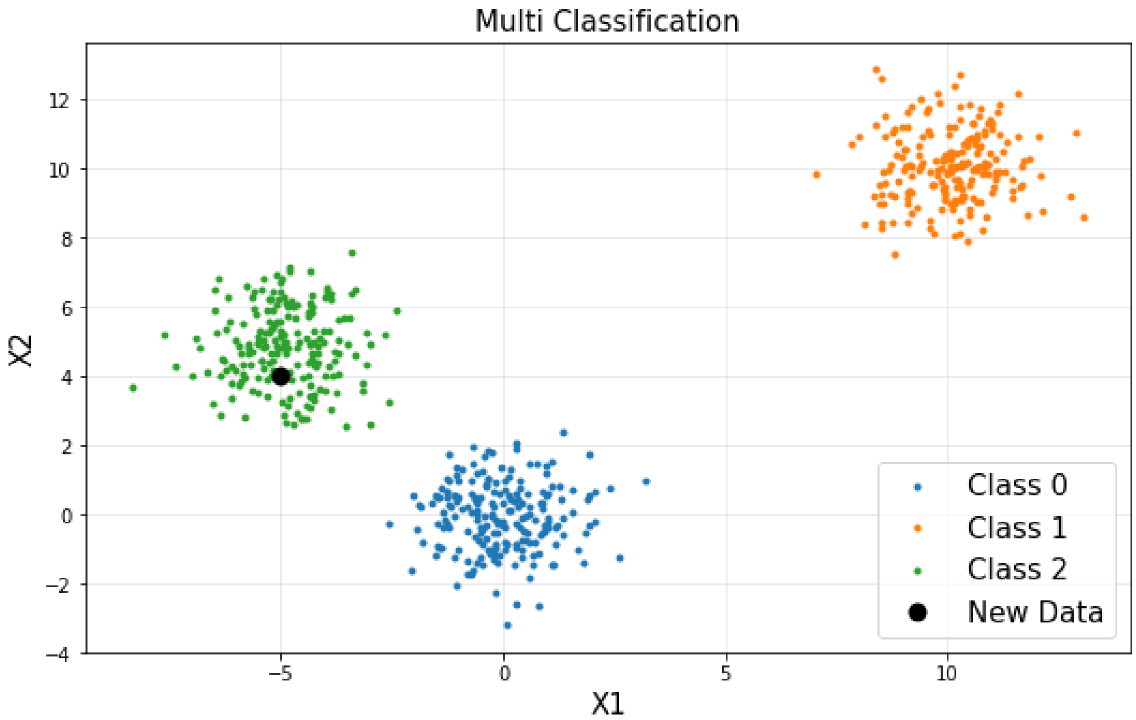
```
X_new = np.array([-5, 4])
X_new = X_new.reshape(1, -1)
X_new.shape
```

Out[39]:

```
(1, 2)
```

In [40]:

```
plt.figure(figsize=(10, 6))
plt.title('Multi Classification', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.plot(X[y==2,0], X[y==2,1], '.', label='Class 2')
plt.plot(X_new[0,0], X_new[0,1], 'ko', label='New Data', ms=5, mew=5)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- 새로운 데이터는 Class1에 속함

In [41]:

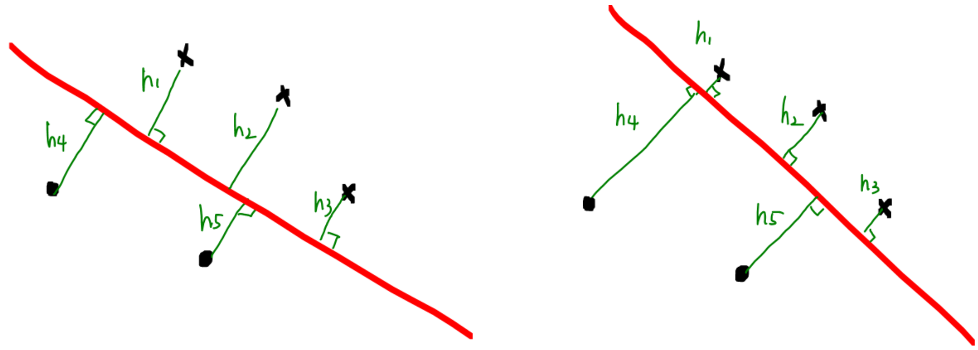
```
clf.predict(X_new)
```

Out[41]:

```
array([2])
```

# 4. Logistic Regression

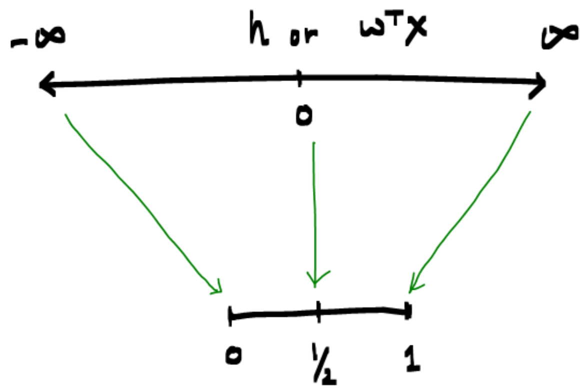
- Logistic regression is a classification algorithm - don't be confused
- We want to use *distance information of all data points* $\rightarrow$ logistic regression




- basic idea: find the decision boundary (hyperplane) of $g(x) = \omega^T x = 0$ such that maximizes $\prod_i |h_i|$
  - Inequality of arithmetic and geometric means
$$\frac{h_1 + h_2}{2} \geq \sqrt{h_1 h_2}$$
and that equality holds if and only if $h_1 = h_2$

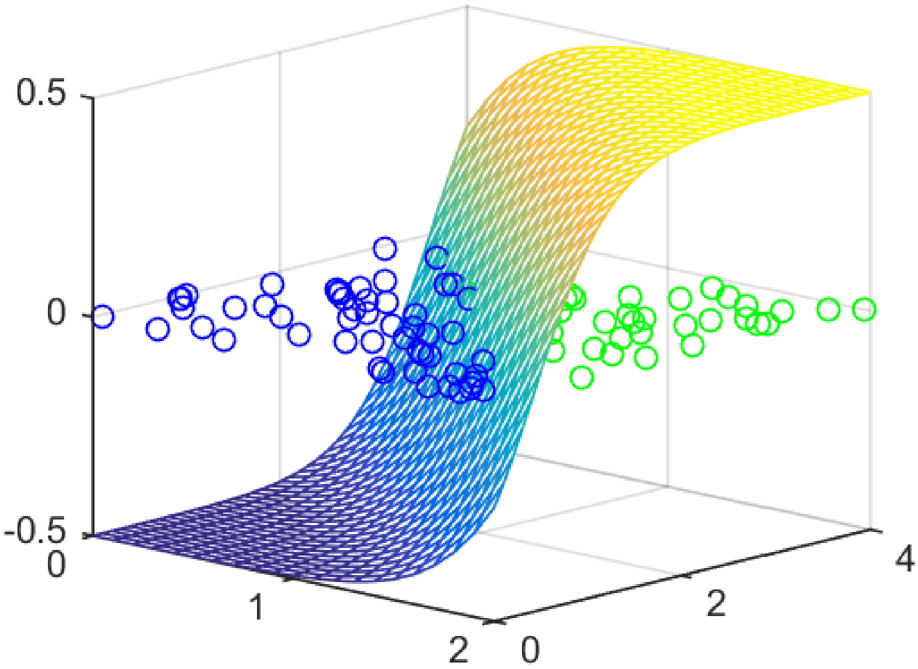
- Roughly speaking, this optimization of $\max \prod_i |h_i|$ tends to position a hyperplane in the middle of two classes
$$h = \frac{g(x)}{\lVert\omega\rVert} = \frac{\omega^T x}{\lVert\omega\rVert} \approx \omega^T x$$

- We link or squeeze $(-\infty, +\infty)$ to $(0, 1)$ for several reasons:




- If $\sigma(z)$ is the sigmoid function, or the logistic function
$$\sigma(z) = \frac{1}{1 + e^{-z}} \implies \sigma(\omega^T x) = \frac{1}{1 + e^{-\omega^T x}}$$
  - logistic function generates a value where is always either 0 or 1
  - Crosses 0.5 at the origin, then flattens out

- Classified based on probability




To see how it works, click [here (http://i-systems.github.io/HSE545/machine%20learning%20all/04%20Classification/iSystems_03_logistic_regression.](http://i-systems.github.io/HSE545/machine%20learning%20all/04%20Classification/iSystems_03_logistic_regression.)

## 4.1. Binary Classification

- C0와 C1 데이터를 분류
- 데이터를 X, y로 병합

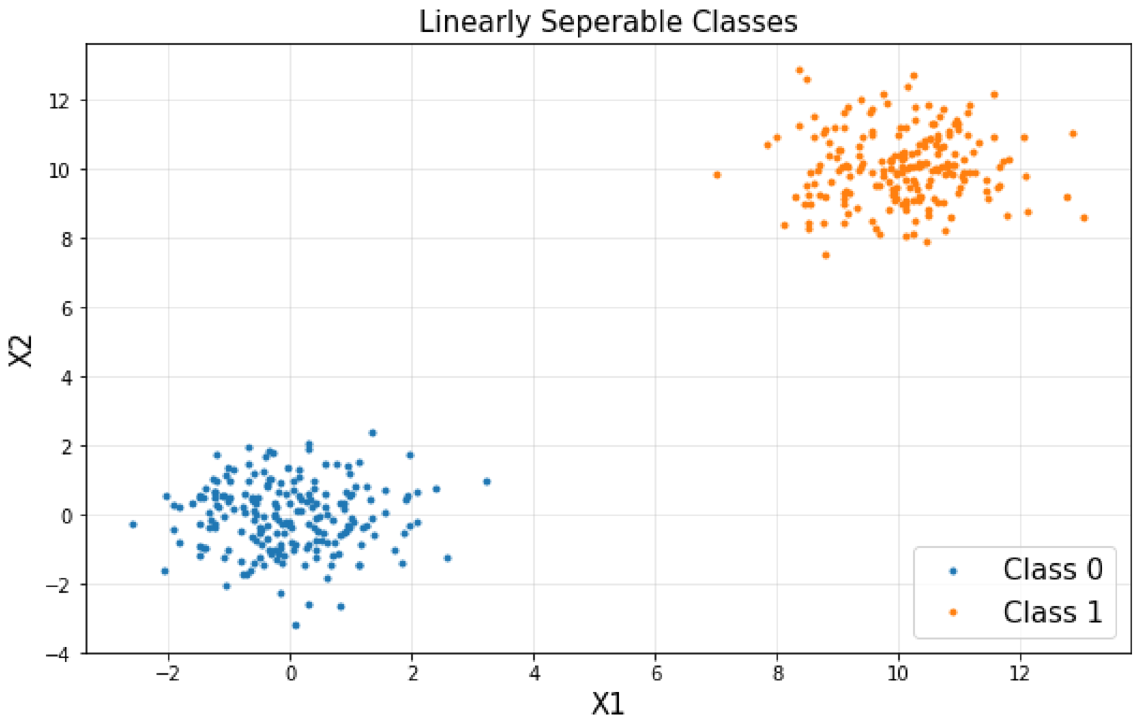
In [42]:

```
X = np.vstack([C0, C1])
y = np.hstack([y0, y1])
```

- Plot을 통하여 결과 확인

In [43]:

```
plt.figure(figsize=(10, 6))
plt.title('Linearly Seperable Classes', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- Sklearn linear_model을 import
- LogisticRegression 개체를 선언 후 피팅

In [44]:

```
from sklearn import linear_model
```

In [45]:

```
clf = linear_model.LogisticRegression()
clf.fit(X, y)
```

Out[45]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

- 새로운 데이터에 대한 결과 확인
- Input shape을 맞추는 것에 주의

In [46]:

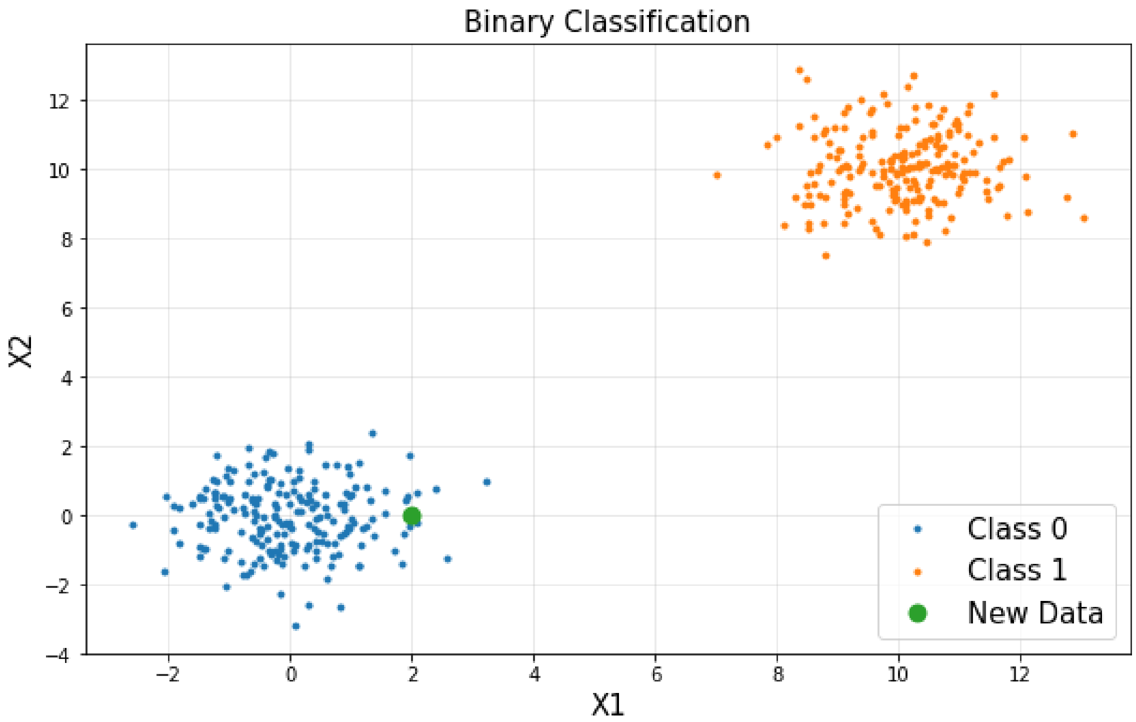
```
X_new = np.array([2, 0])
X_new = X_new.reshape(1, -1)
X_new.shape
```

Out[46]:

```
(1, 2)
```

In [47]:

```
plt.figure(figsize=(10, 6))
plt.title('Binary Classification', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.plot(X_new[0,0], X_new[0,1], 'o', label='New Data', ms=5, mew=5)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- Class 0에 속함

In [48]:

```
pred = clf.predict(X_new)
print(pred)
```

```
[0]
```

In [49]:

```
pred = clf.predict_proba(X_new)
print(pred)
```

```
[[ 0.9538944  0.0461056]]
```

## 4.2. Multi Classification

- C0, C1, C2 데이터를 분류
- Binary classification 에 이용된 코드와 동일
- X, y로 병합

In [50]:

```
X = np.vstack([C0, C1, C2])
y = np.hstack([y0, y1, y2])
```

- Plot을 통하여 결과 확인

In [51]:

```
plt.figure(figsize=(10, 6))
plt.title('Linearly Seperable Classes', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class1')
plt.plot(X[y==2,0], X[y==2,1], '.', label='Class2')
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```


- Sklearn linear_model을 import
- LogisticRegression 개체를 선언 후 피팅

In [52]:

```
from sklearn import linear_model
```

In [53]:

```
clf = linear_model.LogisticRegression()
clf.fit(X, y)
```

Out[53]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

- 새로운 데이터에 대한 결과 확인
- Input shape을 맞추는 것에 주의

In [54]:

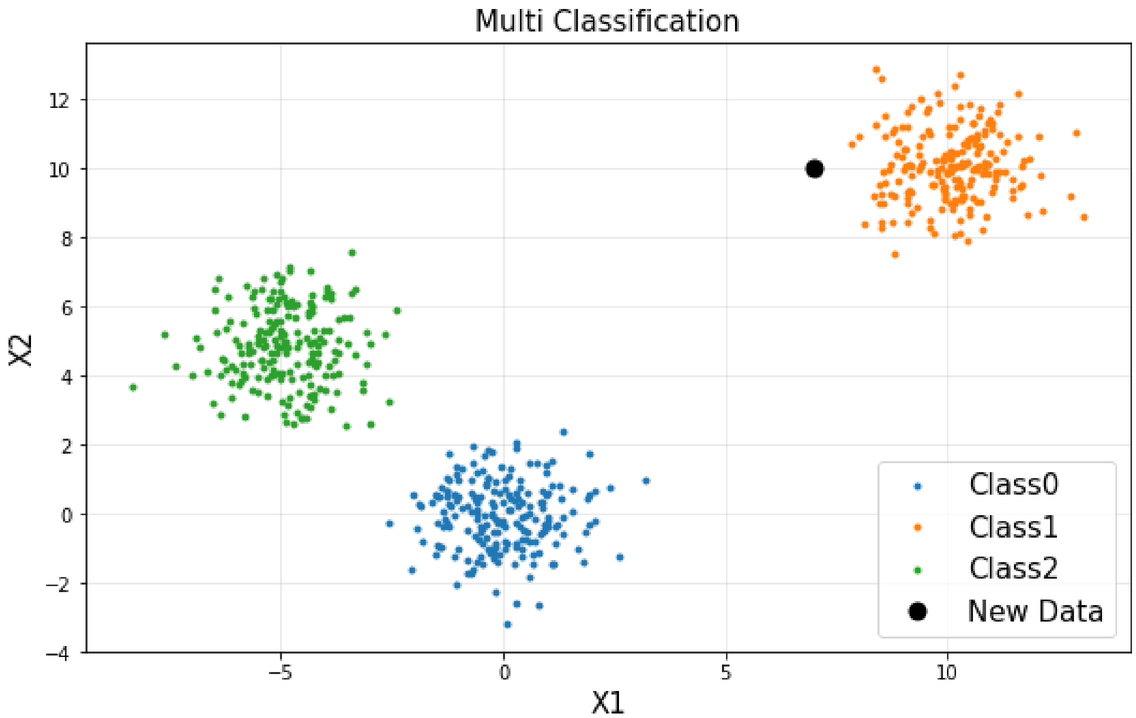
```
X_new = np.array([7, 10])
X_new = X_new.reshape(1, -1)
X_new.shape
```

Out[54]:

```
(1, 2)
```

In [55]:

```
plt.figure(figsize=(10, 6))
plt.title('Multi Classification', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class1')
plt.plot(X[y==2,0], X[y==2,1], '.', label='Class2')
plt.plot(X_new[0,0], X_new[0,1], 'ko', label='New Data', ms=5, mew=5)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




- Predict로 예측

In [56]:

```
prob = clf.predict(X_new)
print(prob)
```

```
[1]
```

In [57]:

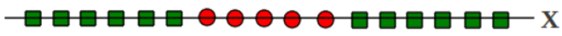
```
prob = clf.predict_proba(X_new)
print(prob)
```

```
[[  1.15846006e-04   9.90478147e-01   9.40600702e-03]]
```

# 5. Nonlinear Classification

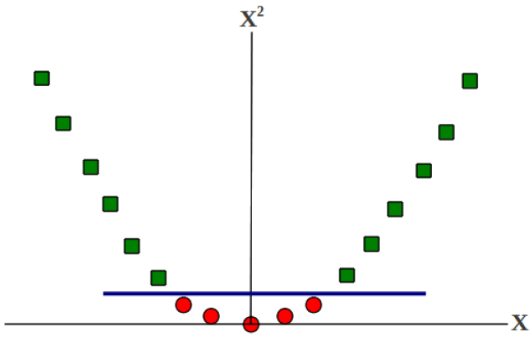
**Classifying non-linear separable data**

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

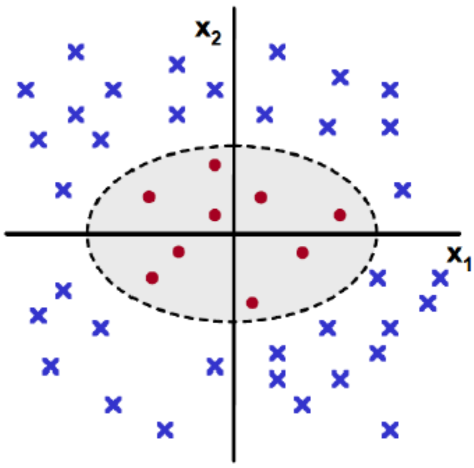



- Now map each example as $x \rightarrow \{x, x^2\}$

- Data now becomes linearly separable in the new representation

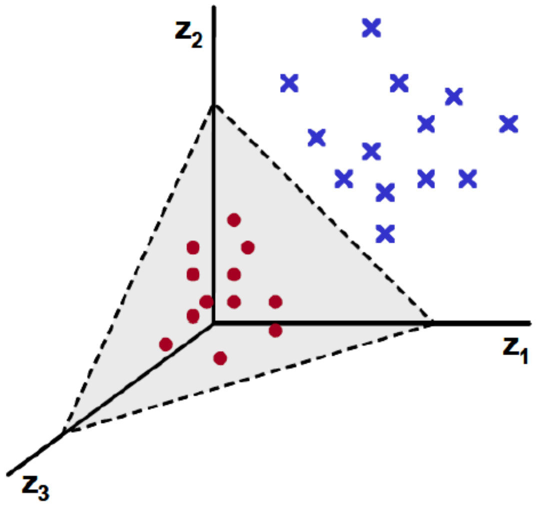



- Linear in the new representation $=$ nonlinear in the old representation
- Let's look at another example
  - Each example defined by a two features $x = \{x_1, x_2\}$
  - No linear separator exists for this data




- 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 represenation)

- Data now becomes linear separable in the new representation

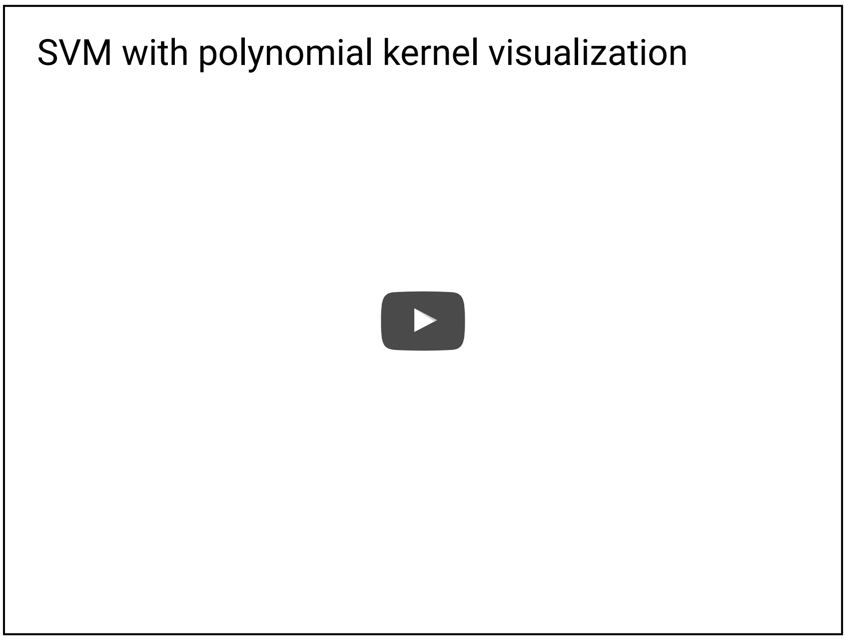



To see how it works, click here (http://i-systems.github.io/HSE545/machine%20learning%20all/04%20Classification/iSystems_02_SVM.html#4.-Nonlinear-Support-Vector-Machine)

- 이 부분 코드는 이해할 필요가 없으며, 개념적인 것만 이해하시면 됩니다
- Nonlinear Example

In [58]:

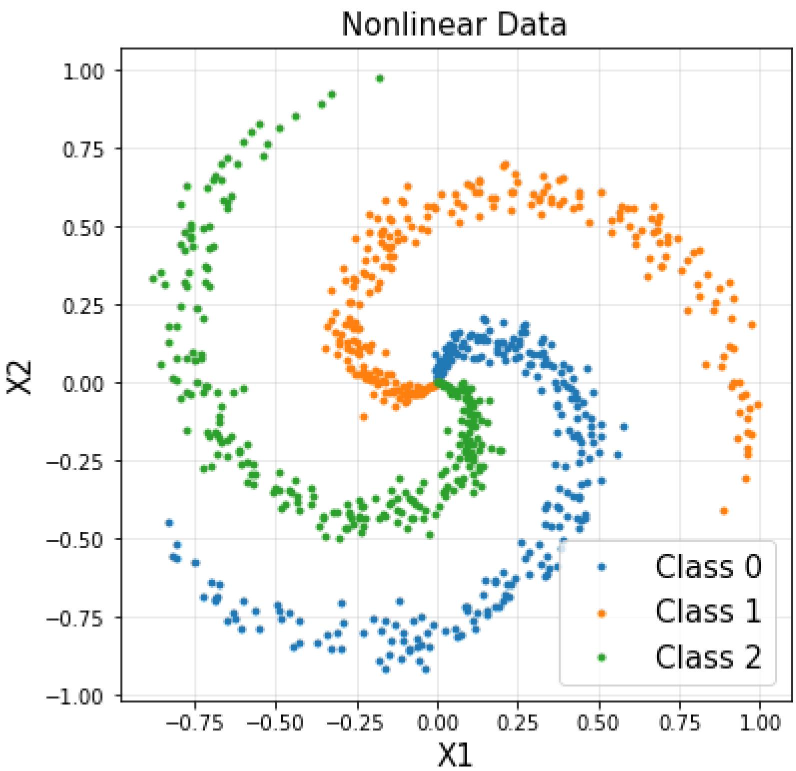
```
%%html
<center><iframe src="https://www.youtube.com/embed/3liCbRZPrZA"
width="420" height="315" frameborder="0" allowfullscreen></iframe></center>
```



In [59]:

```
N = 250 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros([N*K, D]) # data matrix (each row = single example)
y = np.zeros(N*K) # class labels
for j in range(K):
    ix = range(N*j,N*(j+1))
    r = np.linspace(0.0, 1, N) # radius
    t = np.linspace(j*4, (j+1)*4, N) + np.random.randn(N)*0.2 # theta
    X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
    y[ix] = j

plt.figure(figsize=(6, 6))
plt.title('Nonlinear Data', fontsize=15)
plt.plot(X[y==0,0], X[y==0,1], '.', label='Class 0')
plt.plot(X[y==1,0], X[y==1,1], '.', label='Class 1')
plt.plot(X[y==2,0], X[y==2,1], '.', label='Class 2')
plt.xlim(min(X[:,0]) - 0.1, max(X[:,0]) + 0.1)
plt.ylim(min(X[:,1]) - 0.1, max(X[:,1]) + 0.1)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.show()
```




In [60]:

```
from sklearn.svm import SVC
```

In [61]:

```
svc = SVC(kernel='linear', C=1).fit(X, y)
rbf_svc = SVC(kernel='rbf', C=1, gamma=5).fit(X, y)
```