Supervised Learning

with Scikit Learn

by Prof. Seungchul Lee
Industrial AI Lab
http://isystems.unist.ac.kr/
POSTECH

Table of Contents

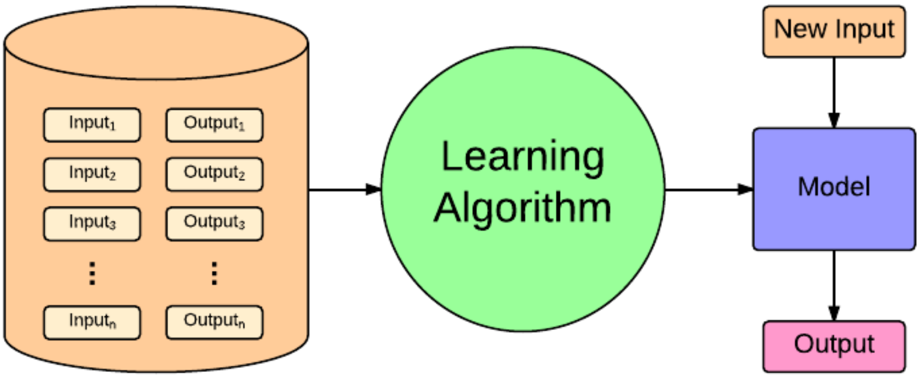
1. Supervised learning

- 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_ω with learning parameter, ω
  - g_ω desired to be as close as possible to y for future (x, y)
  - $i.e., g_\omega(x) \sim y$

- Define a loss function ℓ
- Solve the following optimization problem:

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



2. Regression

2.1. 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

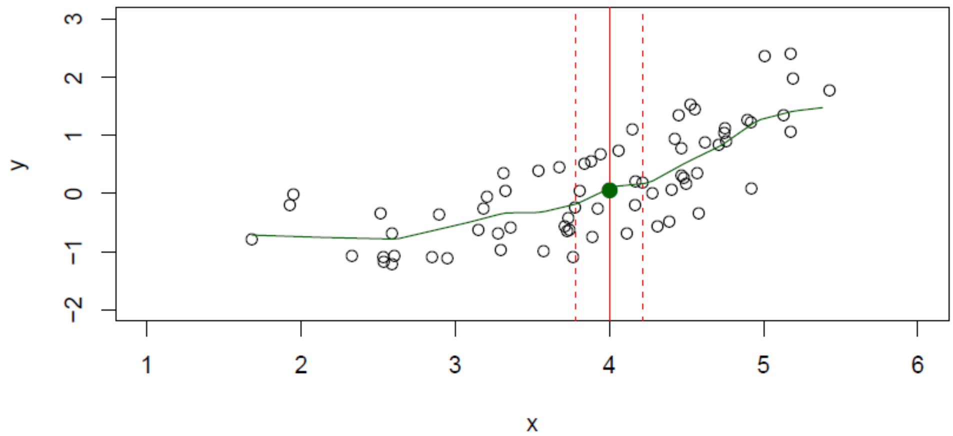
$$Y = f(X) + \epsilon$$

where ϵ 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:

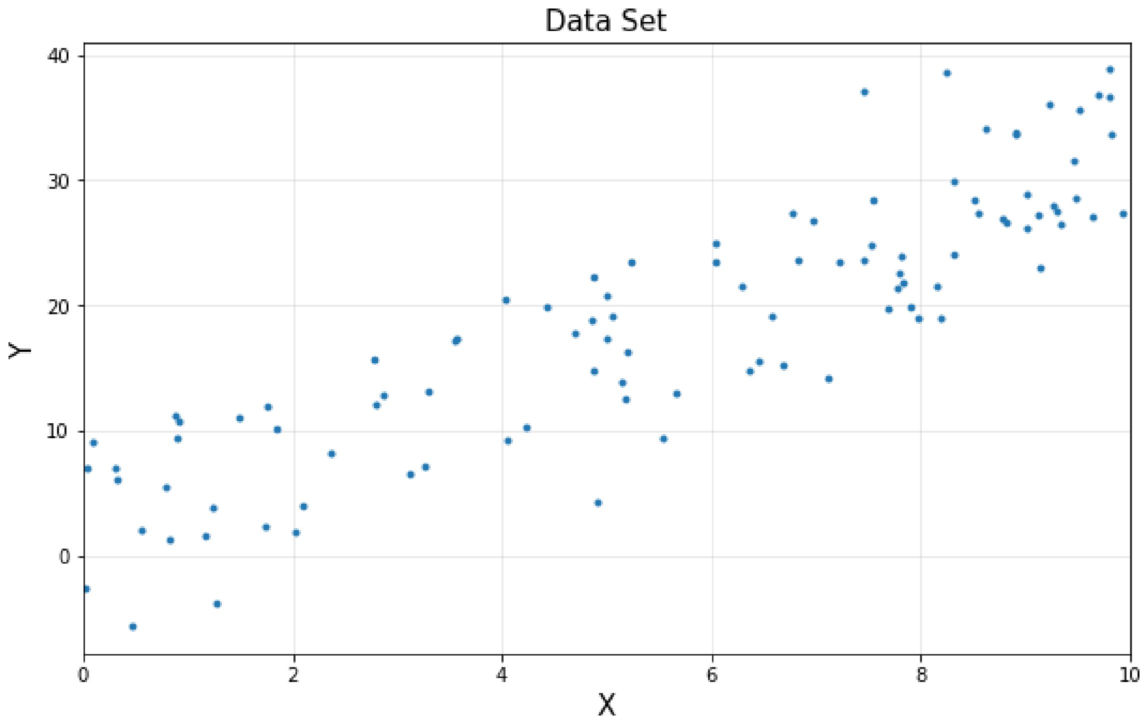
$$\hat{f} = \text{Ave}\,(Y \mid X \in \mathcal{N}(x))$$

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



In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

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

In [2]:
```
plt.figure(figsize=(10, 6))
plt.plot(x, y, '.')

plt.title('Data Set', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.xlim([0, 10])
plt.grid(alpha=0.3)
plt.show()
```



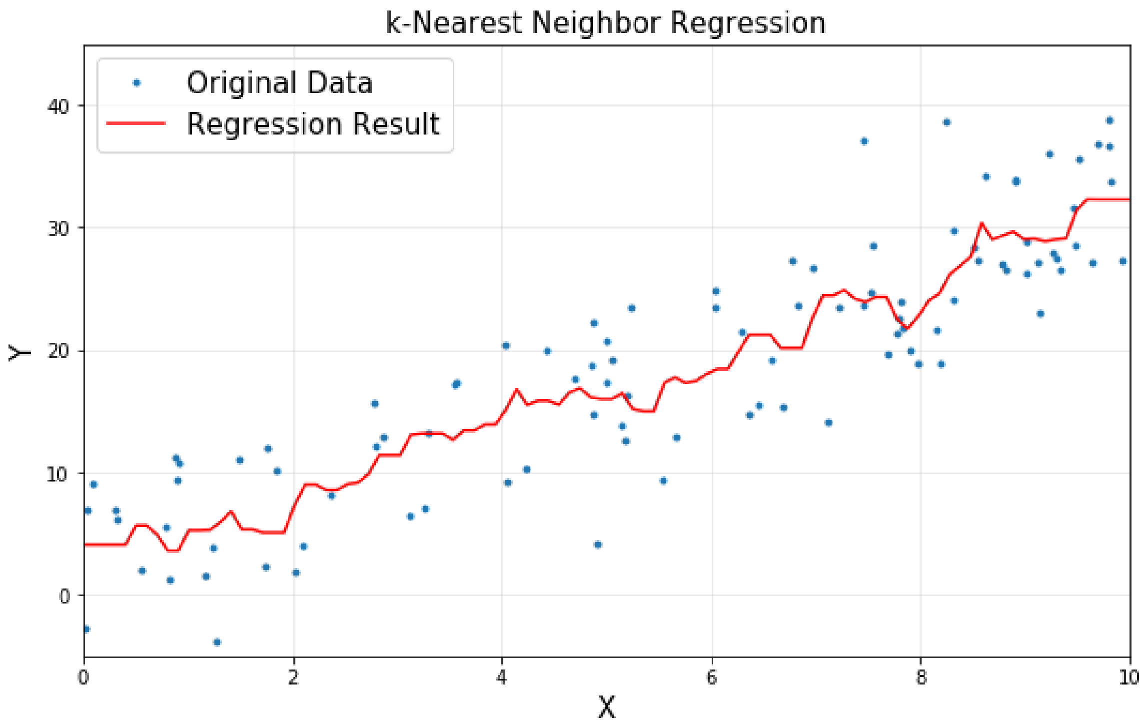
In [3]:
```
from sklearn.neighbors import KNeighborsRegressor

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

Out[3]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=10, p=2,
          weights='uniform')
```

In [4]:
```
pred = reg.predict(5)
print(pred)
```

```
[[15.98874782]]
```

In [5]:
```
xp = np.linspace(0, 10, 100).reshape(-1, 1)
yp = reg.predict(xp)

plt.figure(figsize=(10, 6))
plt.plot(x, y, '.', label='Original Data')
plt.plot(xp, yp, 'r', label='Regression Result')

plt.title('k-Nearest Neighbor Regression', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.legend(loc=2, fontsize=15)
plt.xlim([0, 10])
plt.ylim([-5, 45])
plt.grid(alpha=0.3)
plt.show()
```



1.2. Linear Regression

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

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

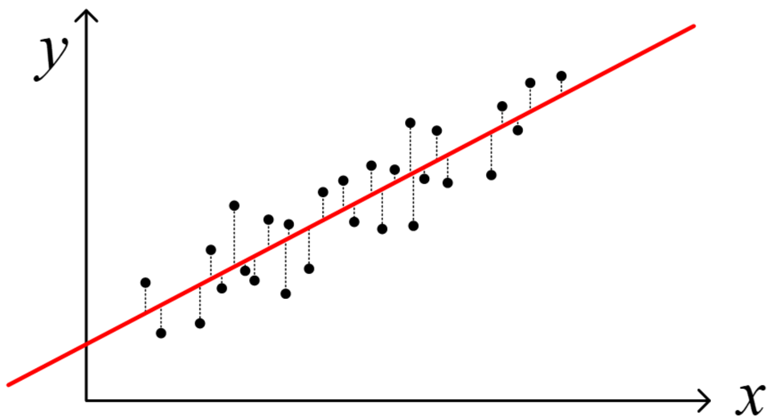
- $\hat{y}_i$: predicted output

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

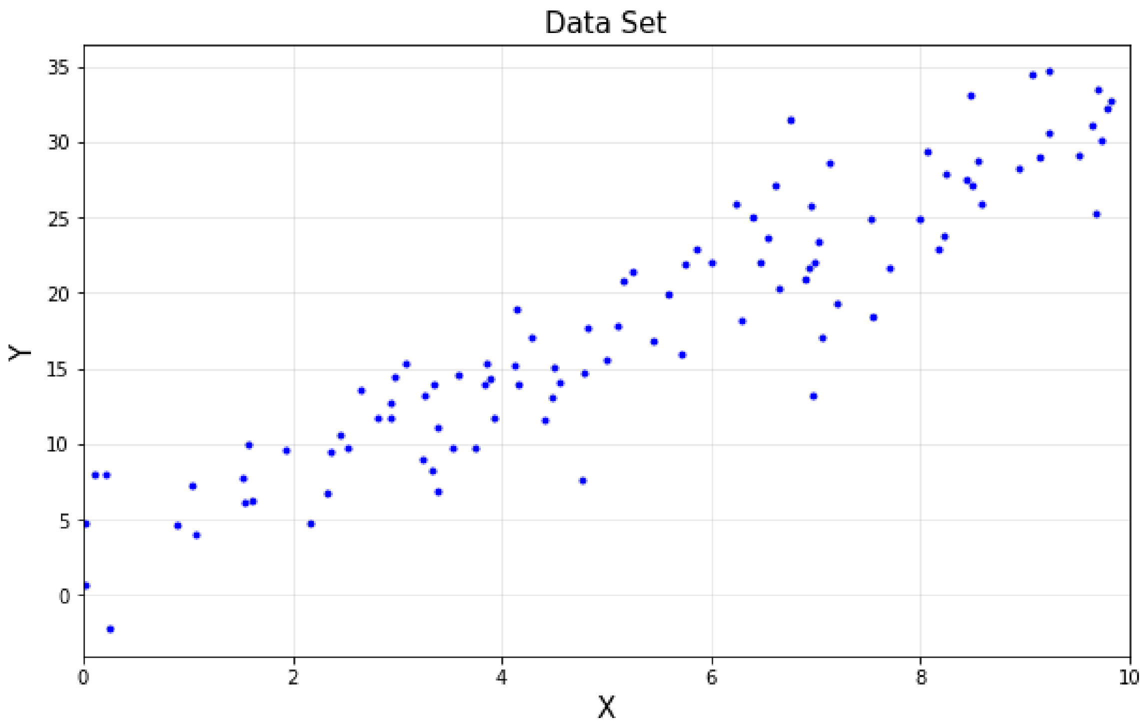
$$\hat{y}_i = f(x_i, \theta) \;\; \text{in general}$$

- 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 [6]:
```
N = 100
w1 = 3
w0 = 2
x = np.random.uniform(0, 10, (N,1))
y = w1*x + w0 + 3*np.random.normal(0, 1, (N,1))

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'b.')

plt.title('Data Set', fontsize=15)
plt.xlabel('X', fontsize=15)
plt.ylabel('Y', fontsize=15)
plt.xlim([0, 10])
plt.grid(alpha=0.3)
plt.show()
```



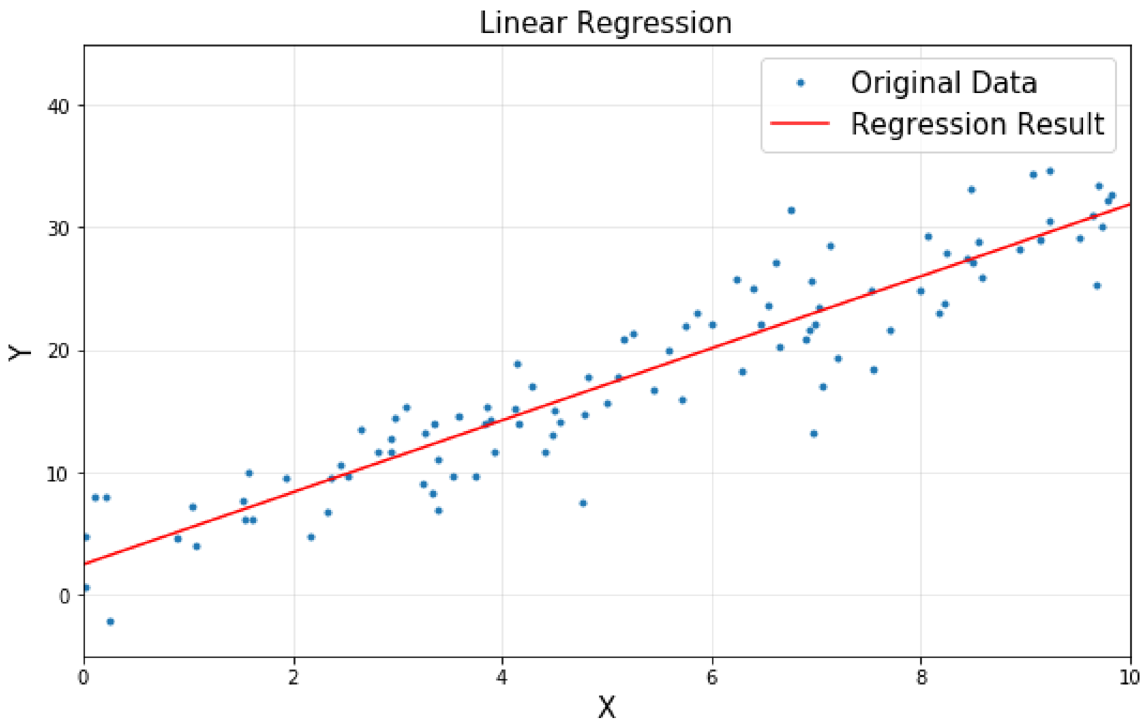
In [7]:
```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg.fit(x, y)
```

Out[7]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
e)
```

In [8]:
```
pred = reg.predict(5)
print(pred)
```

```
[[17.17945416]]
```

In [9]:
```
xp = np.linspace(0, 10).reshape(-1,1)
yp = reg.predict(xp)

plt.figure(figsize=(10, 6))
plt.plot(x, y, '.', label='Original Data')
plt.plot(xp, yp, 'r', label='Regression Result')

plt.title('Linear Regression', fontsize=15)
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()
```

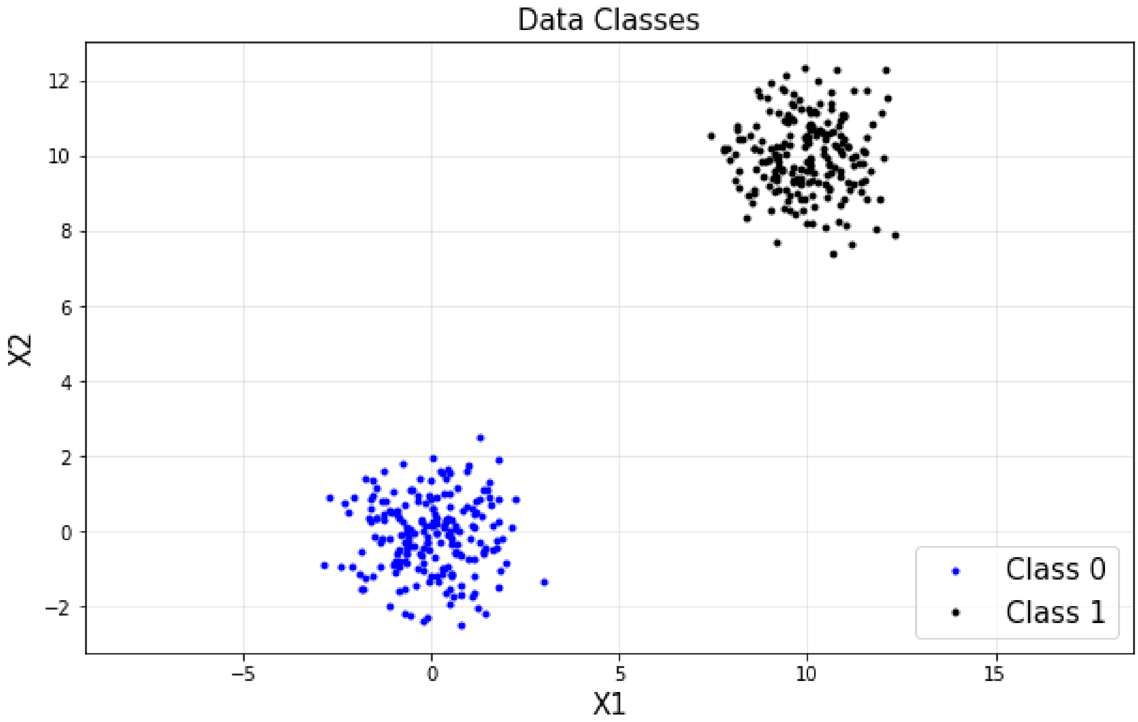


2. Classification

2.1. Data Generation for Classification

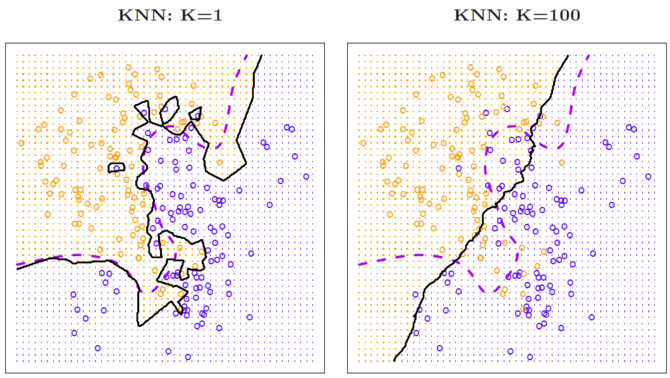
In [10]:
```
import matplotlib.pyplot as plt

m = 200

X0 = np.random.multivariate_normal([0, 0], np.eye(2), m)
X1 = np.random.multivariate_normal([10, 10], np.eye(2), m)

X = np.vstack([X0, X1])
y = np.vstack([np.zeros([m,1]), np.ones([m,1])])
```

In [11]:
```
plt.figure(figsize=(10, 6))
plt.plot(X0[:,0], X0[:,1], '.b', label='Class 0')
plt.plot(X1[:,0], X1[:,1], '.k', label='Class 1')

plt.title('Data Classes', fontsize=15)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.axis('equal')
plt.grid(alpha=0.3)
plt.show()
```

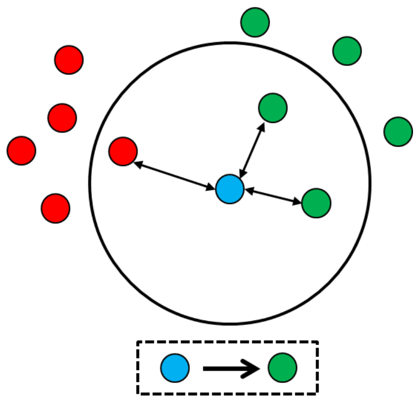


2.2. K-nearest neighbors

- 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,



In [12]:
```
from sklearn.neighbors import KNeighborsClassifier

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

Out[12]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
```

In [13]:
```
X_new = np.array([2,0]).reshape(1,-1)
pred = clf.predict(X_new)

print(pred)
```

```
[0.]
```

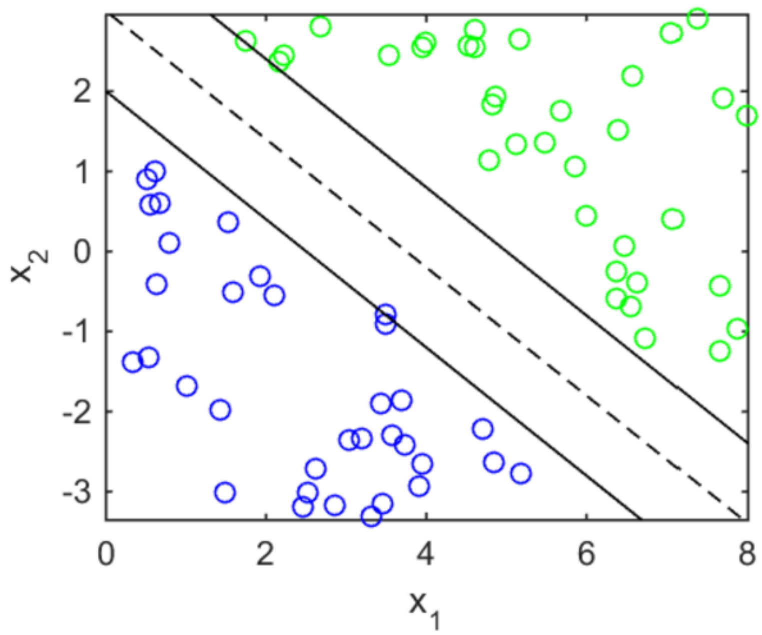
3. Support Vector Machine (SVM)

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

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

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

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



In [14]:
```
from sklearn.svm import SVC

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

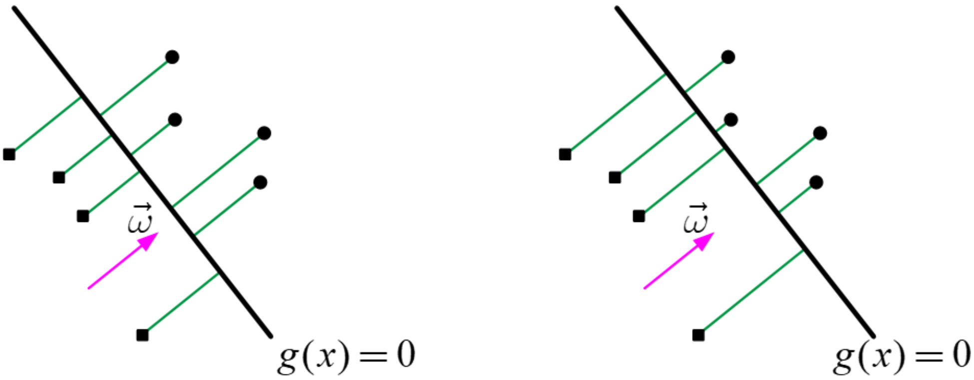
Out[14]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [15]:
```
X_new = np.array([7, 10]).reshape(1, -1)
pred = clf.predict(X_new)

print(pred)
```

```
[1.]
```

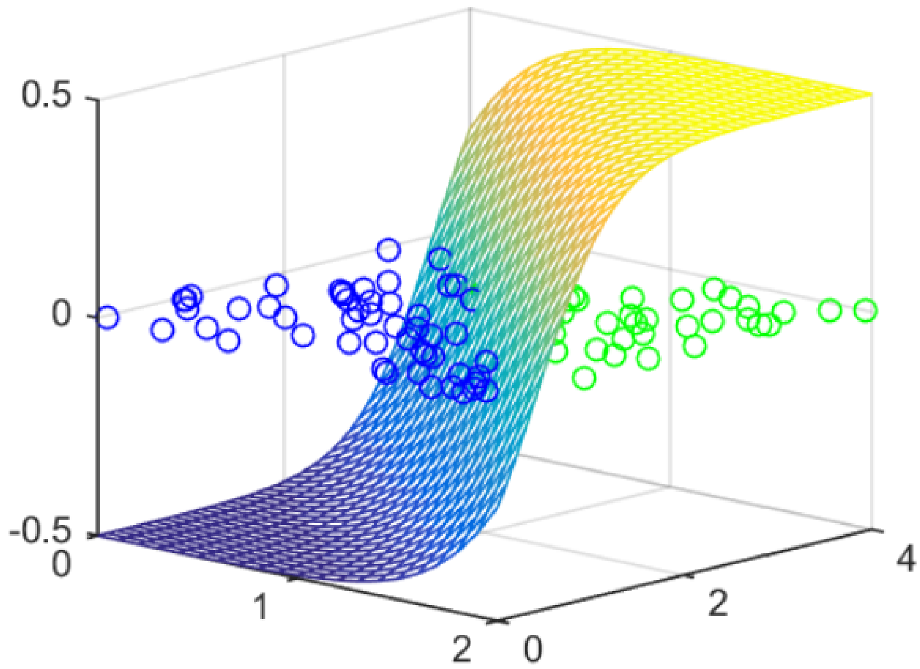
4. Logistic Regression

- basic idea: to find the decision boundary (hyperplane) of $g(x) = \omega^T x = 0$ such that maximizes $\prod_i |h_i| \rightarrow$ optimization



- Inequality of arithmetic and geometric means
$$\frac{x_1 + x_2 + \cdots + x_m}{m} \geq \sqrt[m]{x_1 \cdot x_2 \dots x_m}$$
and that equality holds if and only if $x_1 = x_2 = \cdots = x_m$

- Classified based on probability



In [16]:
```
m = 200

X0 = np.random.multivariate_normal([0, 0], np.eye(2), m)
X1 = np.random.multivariate_normal([10, 10], np.eye(2), m)

X = np.vstack([X0, X1])
y = np.vstack([np.zeros([m,1]), np.ones([m,1])])
```

In [17]:
```
from sklearn import linear_model

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

Out[17]:
```
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)
```

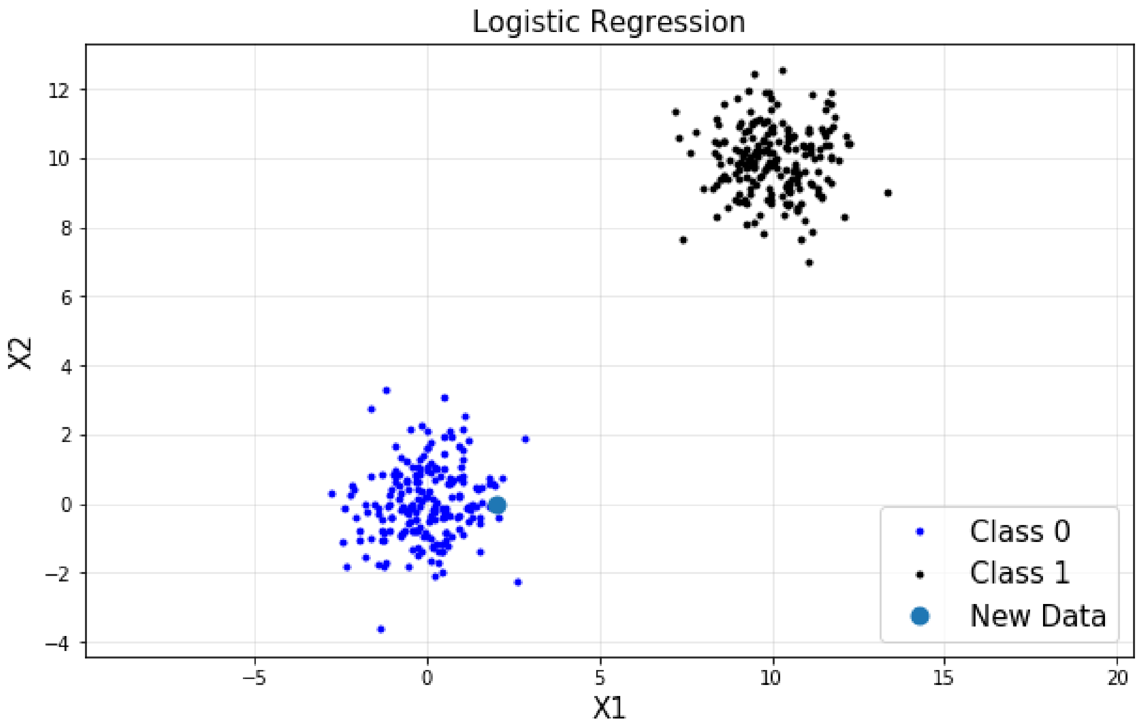
In [18]:
```
X_new = np.array([2, 0]).reshape(1, -1)
pred = clf.predict(X_new)

print(pred)
```

```
[0.]
```

In [19]:
```
pred = clf.predict_proba(X_new)

print(pred)
```

```
[[0.94680429 0.05319571]]
```

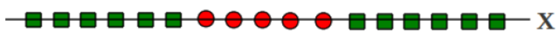
In [20]:
```
plt.figure(figsize=(10, 6))
plt.plot(X0[:,0], X0[:,1], '.b', label='Class 0')
plt.plot(X1[:,0], X1[:,1], '.k', label='Class 1')
plt.plot(X_new[0,0], X_new[0,1], 'o', label='New Data', ms=5, mew=5)

plt.title('Logistic Regression', fontsize=15)
plt.legend(loc='lower right', fontsize=15)
plt.xlabel('X1', fontsize=15)
plt.ylabel('X2', fontsize=15)
plt.grid(alpha=0.3)
plt.axis('equal')
plt.show()
```



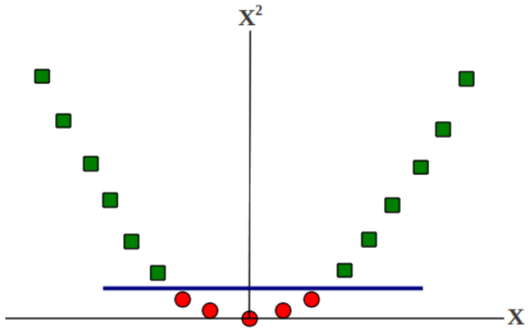
5. Nonlinear Classification

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

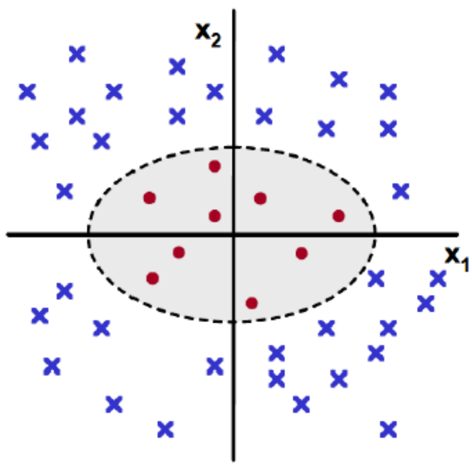


- Now map each example as $x \rightarrow \{x, x^2\}$
- Data now becomes linearly separable in the new representation



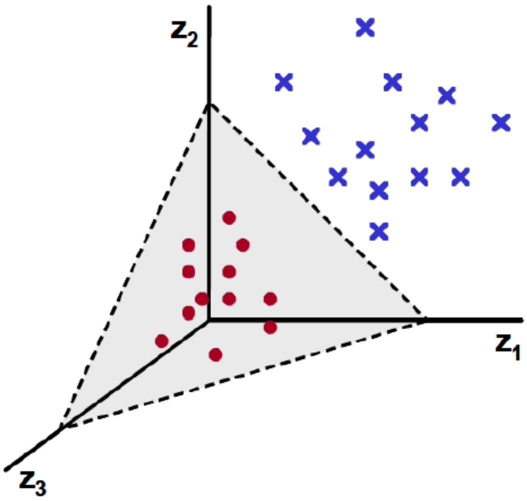
- Linear in the new representation $=$ nonlinear in the old representation

- 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

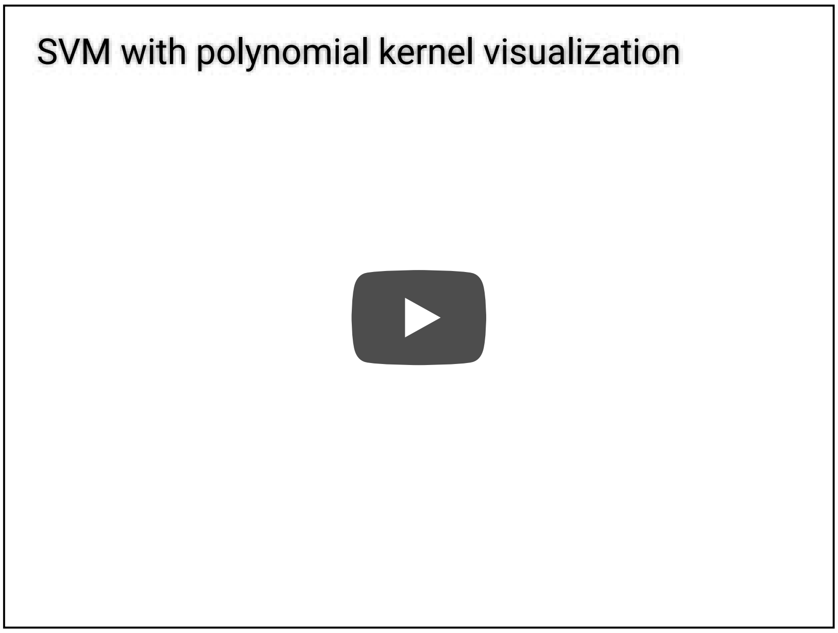


- Now map each example as $x = \{x_1, x_2\} \rightarrow z = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$
  - Each example now has three features (derived from the old represenation)

- Data now becomes linear separable in the new representation

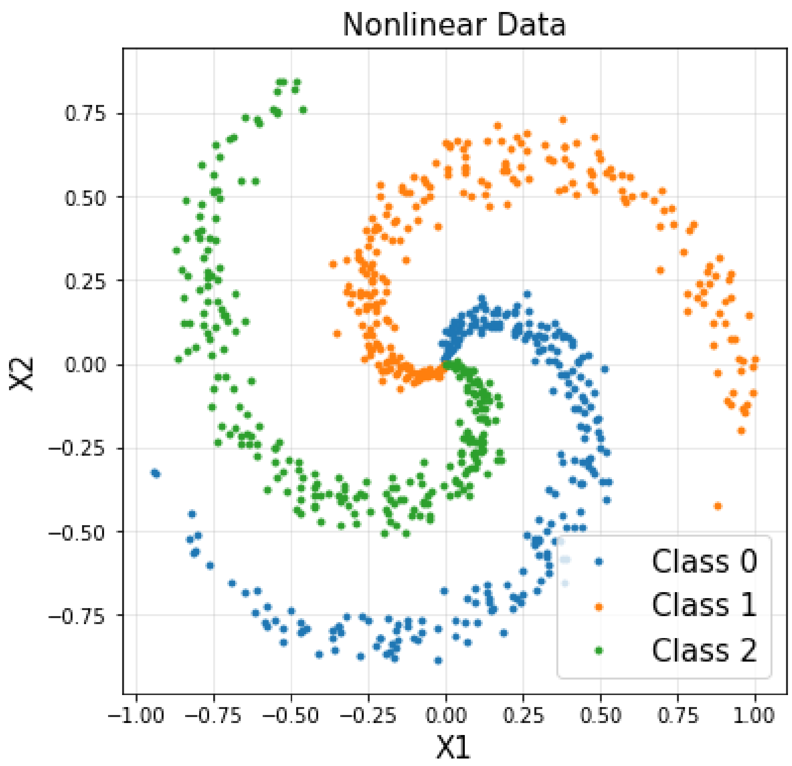


In [21]:
```
%%html
<center><iframe src="https://www.youtube.com/embed/3liCbRZPrZA?rel=0" 
width="420" height="315" frameborder="0" allowfullscreen></iframe></center>
```



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

In [22]:
```
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()
```



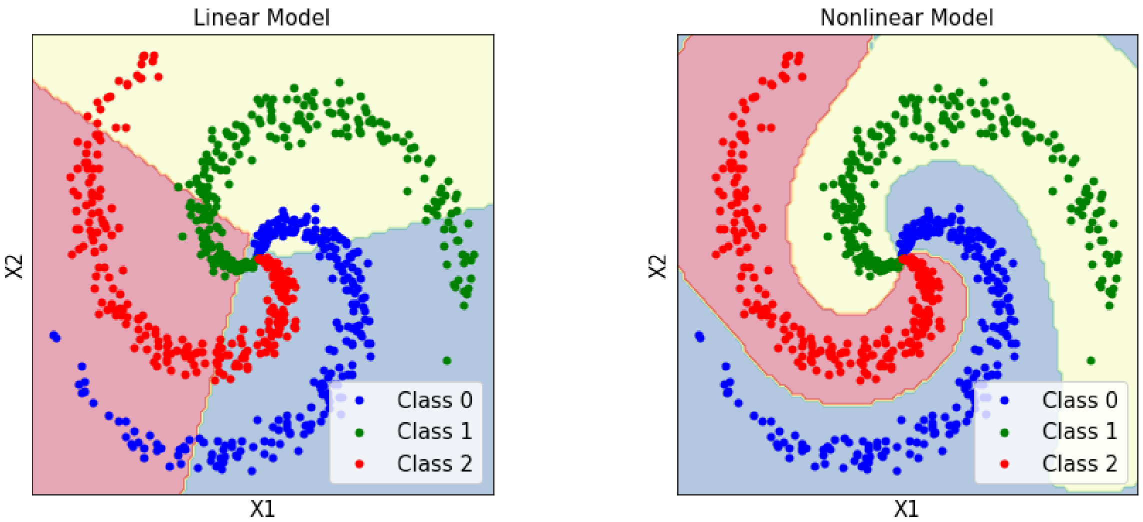
In [23]:
```
from sklearn.svm import SVC

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

In [24]:

```
# create a mesh to plot in

h = 0.02  # step size in the mesh
x_min, x_max = X[:,0].min() - 0.1, X[:,0].max() + 0.1
y_min, y_max = X[:,1].min() - 0.1, X[:,1].max() + 0.1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
```

In [25]:

```
# title for the plots
titles = ['Linear Model', 'Nonlinear Model']

fig = plt.figure(figsize=(14, 6))
for i, clf in enumerate((svc, rbf_svc)):
    plt.subplot(1, 2, i+1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral_r, alpha=0.4)

    # Plot also the training points
    plt.plot(X[y==0,0], X[y==0,1], 'b.', label='Class 0', mew=3)
    plt.plot(X[y==1,0], X[y==1,1], 'g.', label='Class 1', mew=3)
    plt.plot(X[y==2,0], X[y==2,1], 'r.', label='Class 2', mew=3)
    plt.legend(loc='lower right', fontsize=15)
    plt.xlabel('X1', fontsize=15)
    plt.ylabel('X2', fontsize=15)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks([])
    plt.yticks([])
    plt.title(titles[i], fontsize=15)

plt.show()
```



In [26]:

```
%%javascript
$.getScript('https://kmahelona.github.io/ipython_notebook_goodies/ipython_no
tebook_toc.js')
```