

Linear Regression

Gradient Descent

อ. ปรัชญ์ ปิยะวงศ์วิศาล

Pratch Piyawongwisal

Today

- Recap – kNN, MNIST
- Linear Regression
- Gradient Descent
- Polynomial Regression
- Regularization
- Logistic Regression

Recap: Supervised Learning

• Classification

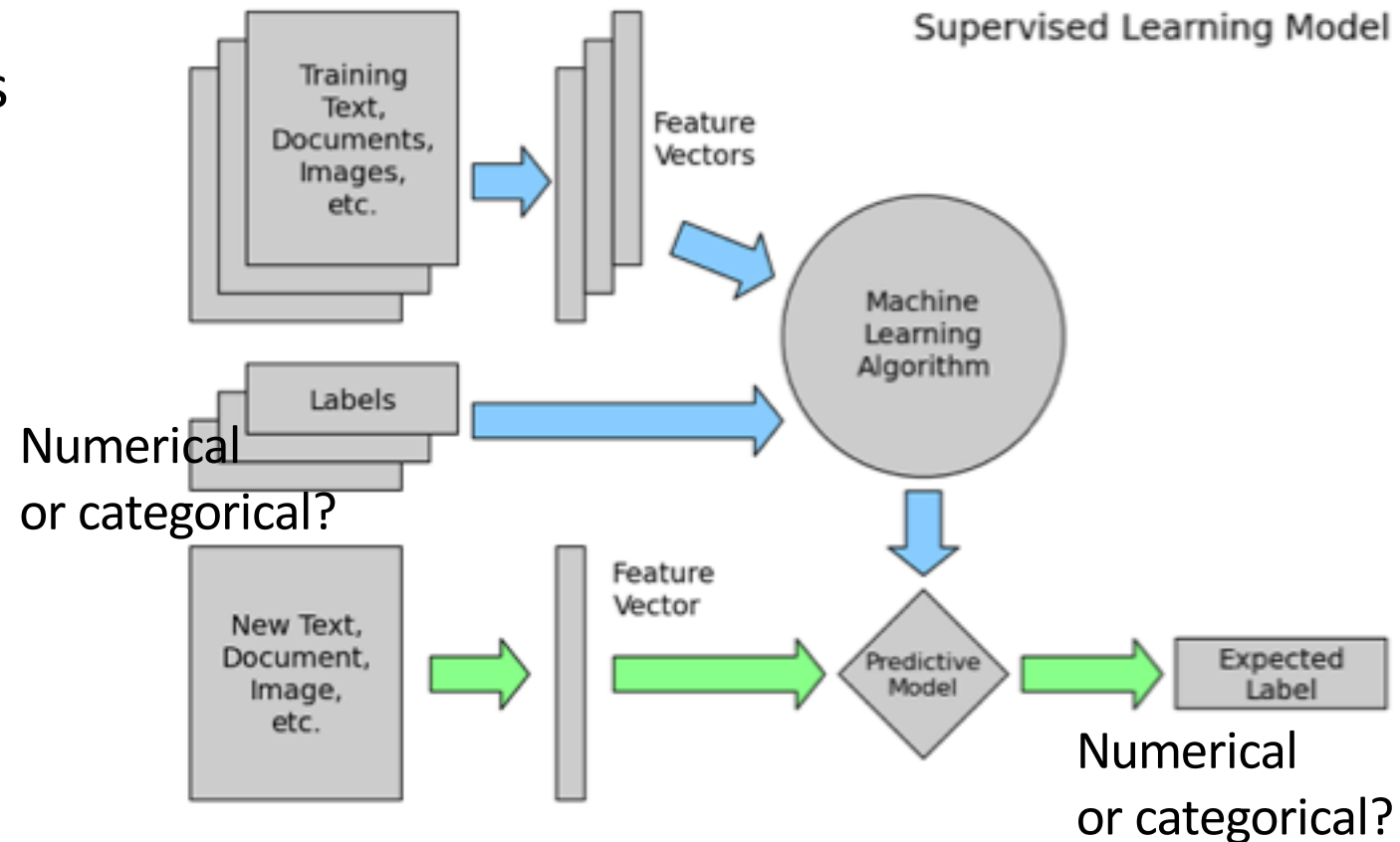
kNN

- Predicts class labels/categories
- ทำนายค่าที่เป็นหมวดหมู่ = จำแนกประเภท
- อาจมองเป็นการหา **boundary** ที่แบ่งข้อมูลในแต่ละหมวดหมู่ ออกจากกัน

• Regression

Linear
Regression

- Predicts continuous values
- ทำนายค่าที่เป็นจำนวนจริง
- อาจมองเป็นการหา **hyperplane** ที่ **fit** กับข้อมูลที่มีมากที่สุด



Recap: K-Nearest Neighbor Algorithm

```
def train(train_images, labels):  
    # ML  
    return model
```



จดจำทุก **training data**
และ **training label**

```
def predict(test_images):  
    # use model to predict labels  
    return test_labels
```



หา **train_image** ที่ใกล้เคียง
กับ **test_image** มากที่สุด
แล้วทำนายว่าเป็น **label** ของ
train_image นั้น

X คือ feature ของข้อมูล (เช่น น้ำหนัก อายุ)
Y คือ label (เช่น เป็นมะเร็ง/สุขภาพดี)

ขั้นการ test/predict ใน kNN

- ในการ **test** เราจะหาเพื่อนบ้าน **k** คน ที่ใกล้ที่สุด แล้วดูว่าเพื่อนบ้านเป็นมะเร็ง (Y1) หรือไม่ (Y0) มากกว่ากัน เป็นต้น
- ในการหาเพื่อนบ้านที่ใกล้ เราวัดระยะทางจากอะไร -> นิยมใช้ **Euclidean Distance**

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \dots + (x_n - x'_n)^2}$$

- **formally**, เราต้องการหาฟังก์ชัน $h: X \rightarrow Y$ ซึ่ง $h(x) = P(y=\text{มะเร็ง?} \mid X=x)$

$$P(y = j \mid X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

โดยที่ $I()$ เป็น indicator จะเท่ากับ 1 ถ้า $y_i = j$ หรือเท่ากับ 0 ถ้า $y_i \neq j$

การ implement kNN ด้วย scikit-learn

- โหลดข้อมูล **MNIST** ซึ่งประกอบด้วย
 - **images** ภาพตัวเลข
 - **target** label เฉลย
- แบ่งข้อมูลเป็นชุด **train** ชุด **test**
- สร้าง **KNeighborsClassifier**
- ทำการ **train** โดยเรียก **classifier.fit(ชุด train)**
- ทำการ **test** โดยเรียก **classifier.predict(ชุด test)**
- เปรียบเทียบผลการ **predict** กับเฉลยใน **target** แล้วประเมินประสิทธิภาพ
 - **accuracy** = จำนวนชุด **test** ที่ทายถูก / จำนวนชุด **test** ทั้งหมด
 - ใช้ **metrics.confusion_matrix(เฉลย, ทาย)** เพื่อหาว่าทำนายผิดอย่างไร

X คือ **feature** ของข้อมูล (เช่น น้ำหนัก อายุ)
Y คือ **label** (เช่น เป็นมะเร็ง/สุขภาพดี)

สรุปเกี่ยวกับ kNN

- เป็นอัลกอริทึมแบบ
 - supervised
 - non-parametric
 - ใช้สำหรับทำ **classification**
- ในการ **train** แค่จำข้อมูลทั้งหมดไว้ -> **ข้อเสีย**: เปลืองเนื้อที่
- ในการ **test** ทำการหาเพื่อนบ้าน **k** คน ที่ใกล้ที่สุด แล้วดูว่าเพื่อนบ้านเป็นมะเร็ง (**Y1**) หรือไม่ (**Y0**) มากกว่ากัน -> **ข้อเสีย**: ใช้เวลาคำนวณนาน
- ค่าของ **k** เป็น **hyperparameter** ที่เราเลือกปรับได้

มีผู้ช่วย (ใช้ข้อมูล **train** ที่มี **label** เฉลย ในการหา $h:X \rightarrow Y$)

ไม่มี **assumption** เกี่ยวกับหน้าตาของฟังก์ชัน $h: X \rightarrow Y$
ไม่มีโมเดลทางคณิตศาสตร์ที่ประกอบด้วยตัวแปร (**parameter**)

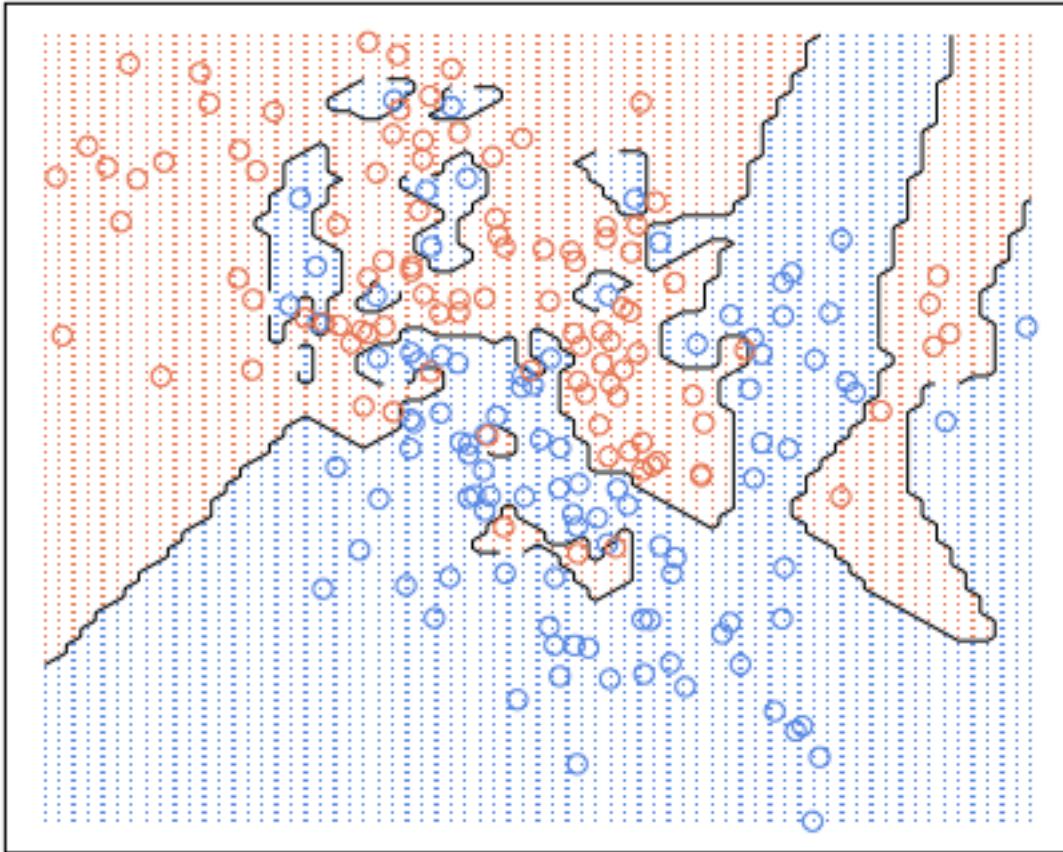
จำแนกหมวดหมู่ของข้อมูล

More on kNN...

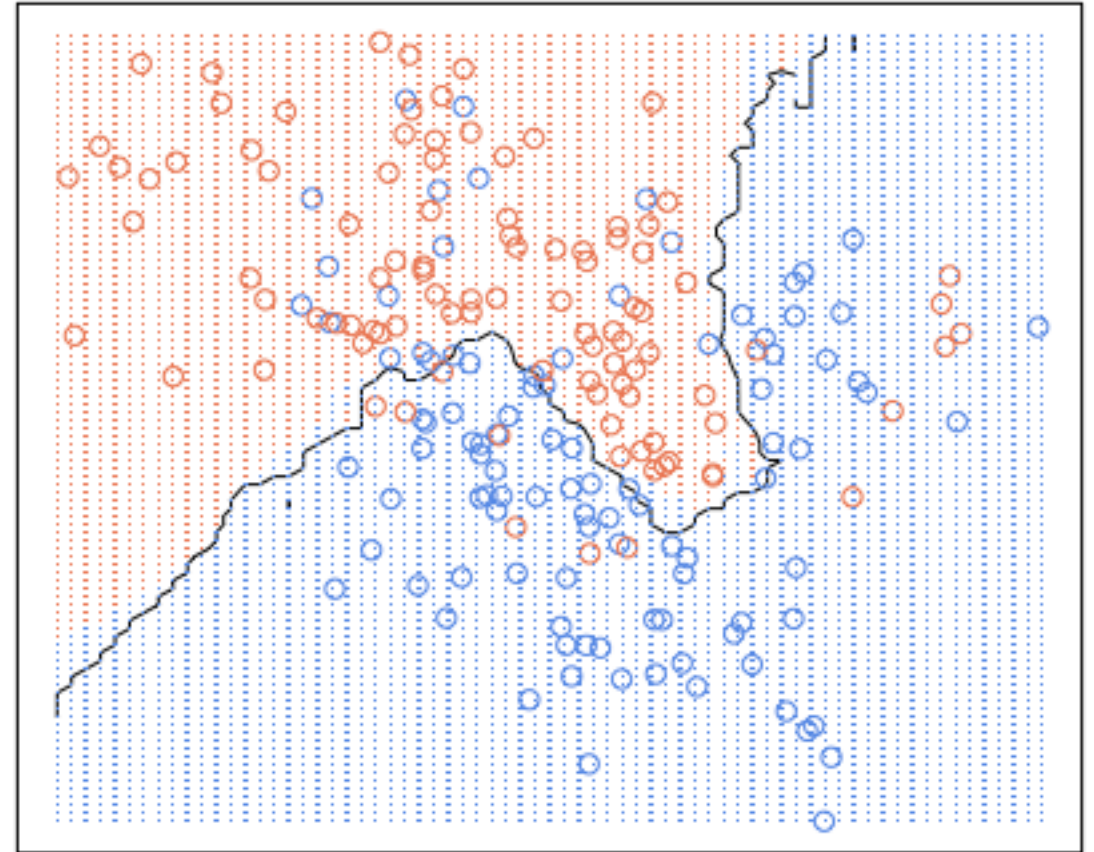
- Kevin Zakka's complete guide
 - <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>
- Stanford's CS231n slides
 - http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture02.pdf
- Visualize kNN classifier's boundary
 - http://wittawat.com/posts/knn_boundary.html
 - <http://vision.stanford.edu/teaching/cs231n-demos/knn/>

How to choose the best k ?

nearest neighbour ($k = 1$)

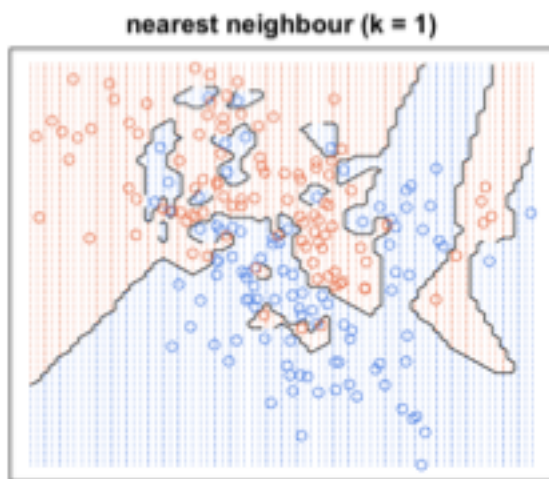


20-nearest neighbour

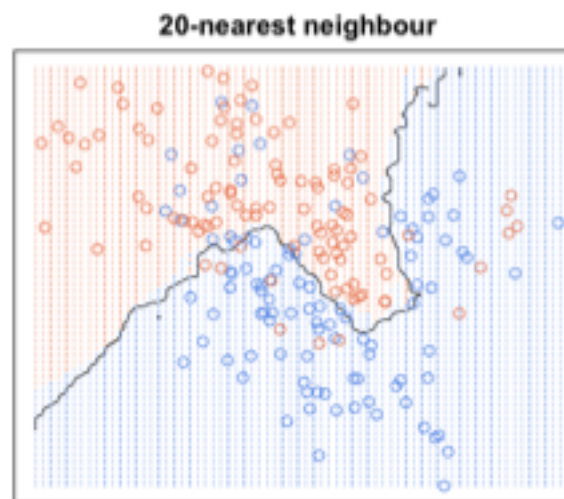


Choosing k: Bias-Variance Tradeoff

- ค่าของ **k** เป็น **hyperparameter** ที่เราเลือกปรับได้ แต่เราจะเลือกค่าที่ดีที่สุดได้อย่างไร?
- **error** ของการเรียนรู้ มีสองประเภท
 - **bias** = ทำนายผิดเพราะเราใช้ **model** ซึ่งเป็นการประมาณจากโลกจริง อาจเกิดจากการที่เราลืมพิจารณาบางปัจจัย บาง **feature** ไป
 - **variance** = ทำนายผิดเพราะความผันผวนของข้อมูล เจออะไรที่ไม่คาดฝัน อาจเกิดจาก **noise** ในข้อมูล



low bias 😊
high variance ☹️

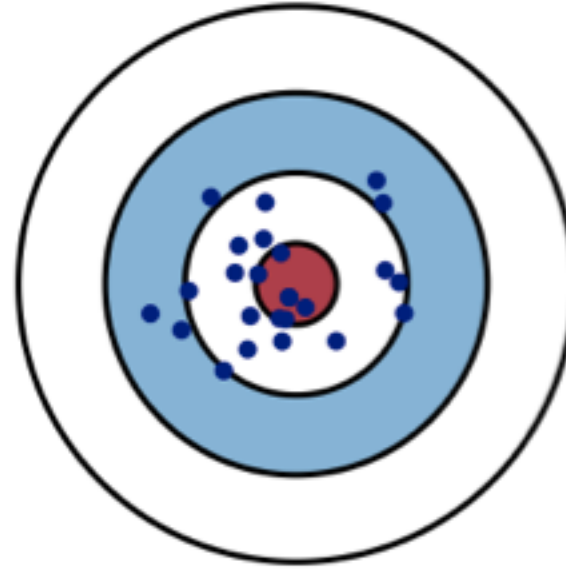
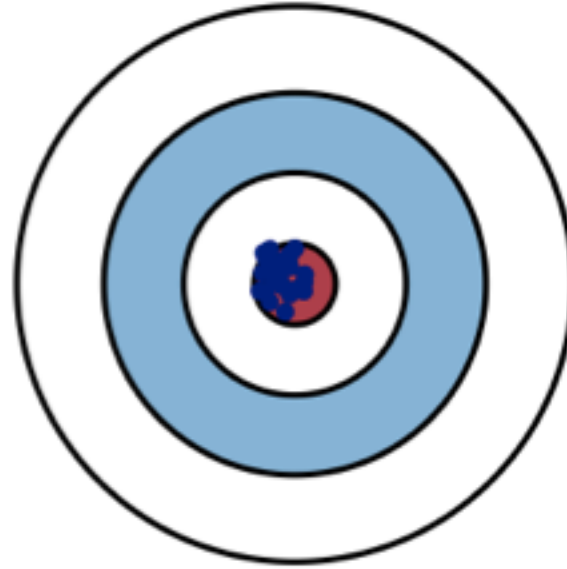


high bias ☹️
low variance 😊

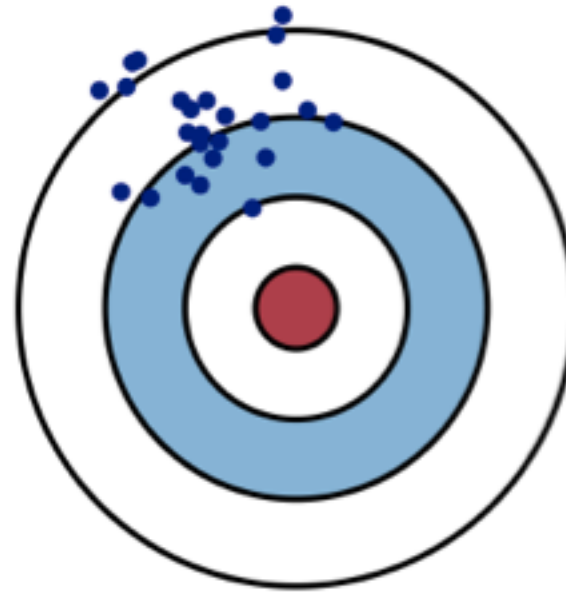
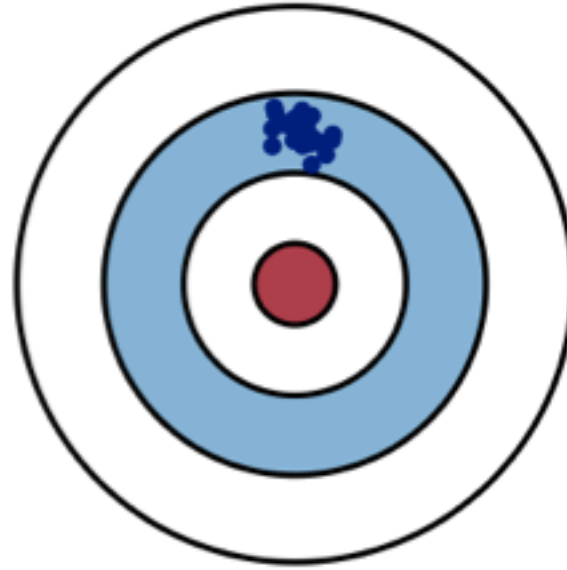
Low Variance

High Variance

Low Bias

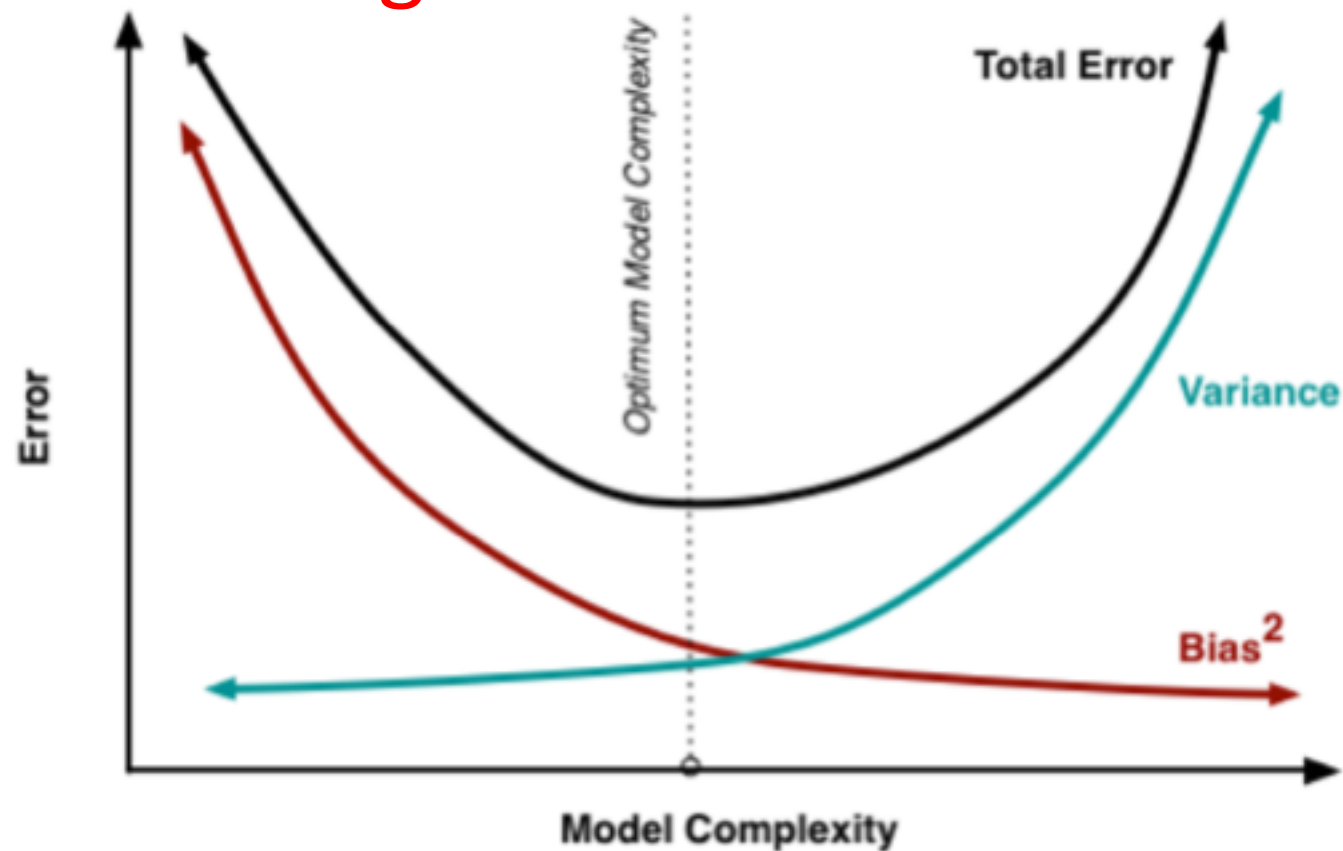


High Bias



Bias-Variance Tradeoff

underfitting overfitting



← **k** มาก

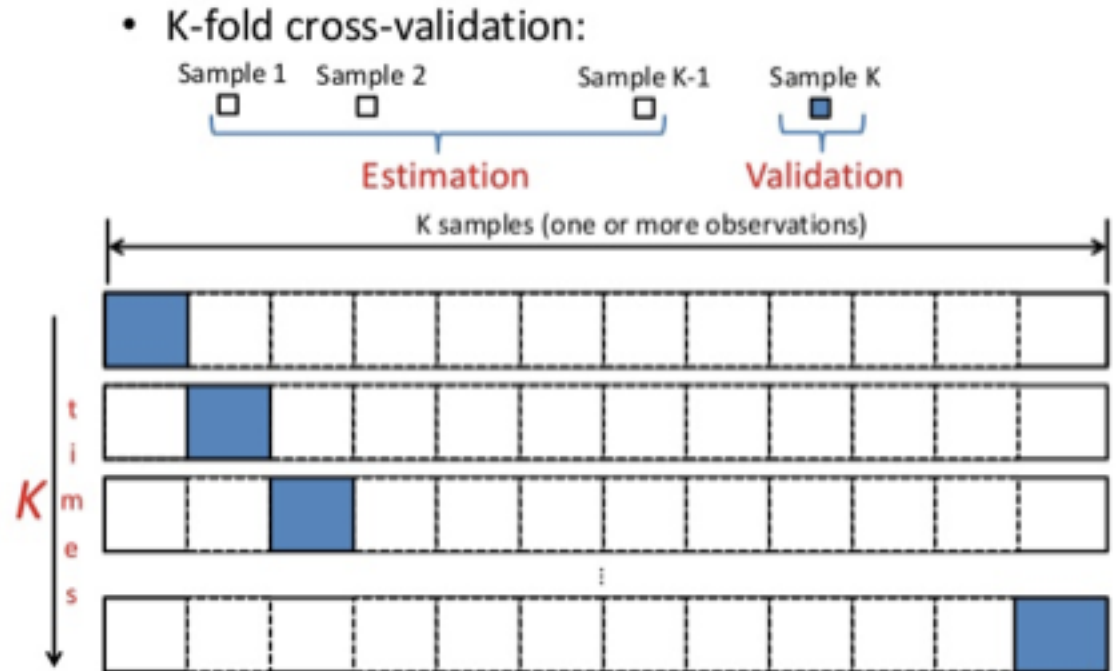
Choosing k

- เป้าหมาย: เราต้องการค่า **k** ที่ทำให้ **test error** ต่ำสุด
- **solution1**: ทำการ **train, test** ซ้ำๆ กับข้อมูลชุดเดิม โดยเปลี่ยนค่า **k** ไปเรื่อยๆ แล้วหา **k** ที่ทำให้ **test error** ต่ำสุด
 - problem?
- **Overfitting** คือการที่โมเดล **fit** กับข้อมูลชุดหนึ่งมากเกินไป ทำให้ **generalize** กับข้อมูลชุดอื่นๆ ไม่ได้

Cross Validation

- แบ่งข้อมูลออกเป็น **train, validate, test**
- ใช้ชุด **train, validate** ในการหาค่า **k** ที่ดีที่สุด
- ไม่ยุ่งกับข้อมูลชุด **test** จนกว่าจะหา **k** ที่ดีที่สุดได้
- ใช้ชุด **test** ในการประเมินสุดท้าย

Cross-validation: How it works?



Linear Regression

Recap: Supervised Learning

• Classification

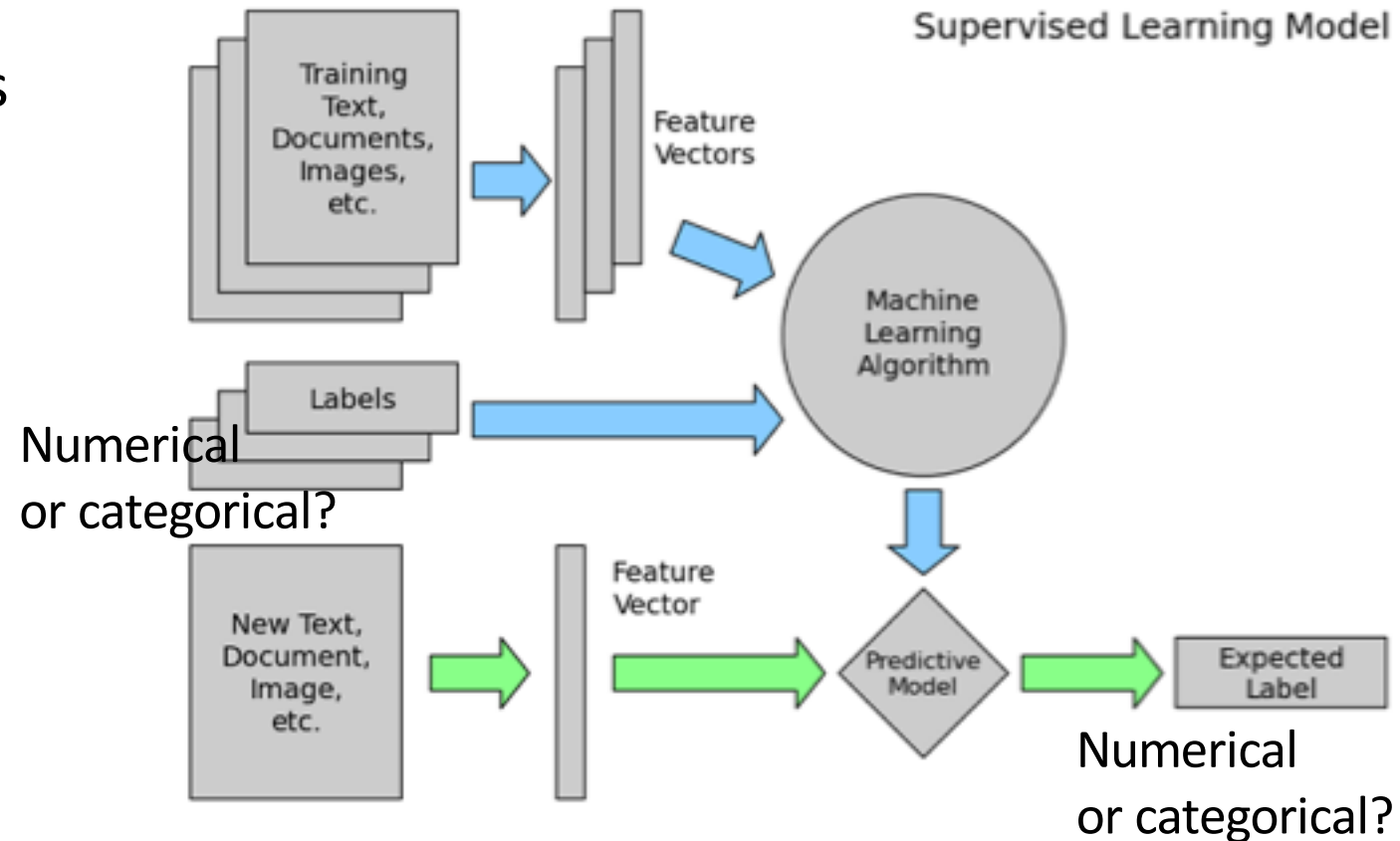
kNN

- Predicts class labels/categories
- ทำนายค่าที่เป็นหมวดหมู่ = จำแนกประเภท
- อาจมองเป็นการหา **boundary** ที่แบ่งข้อมูลในแต่ละหมวดหมู่ ออกจากกัน

• Regression

Linear
Regression

- Predicts continuous values
- ทำนายค่าที่เป็นจำนวนจริง
- อาจมองเป็นการหา **hyperplane** ที่ **fit** กับข้อมูลที่มีมากที่สุด



Linear Regression

- เป็นอัลกอริทึมแบบ

- supervised
- parametric

- ใช้สำหรับทำ regression

มีผู้ช่วย (ใช้ข้อมูล train ที่มี label เหนย ในการหา $h:X \rightarrow Y$)

มี assumption ว่าฟังก์ชัน $h: X \rightarrow Y$ เป็น Linear

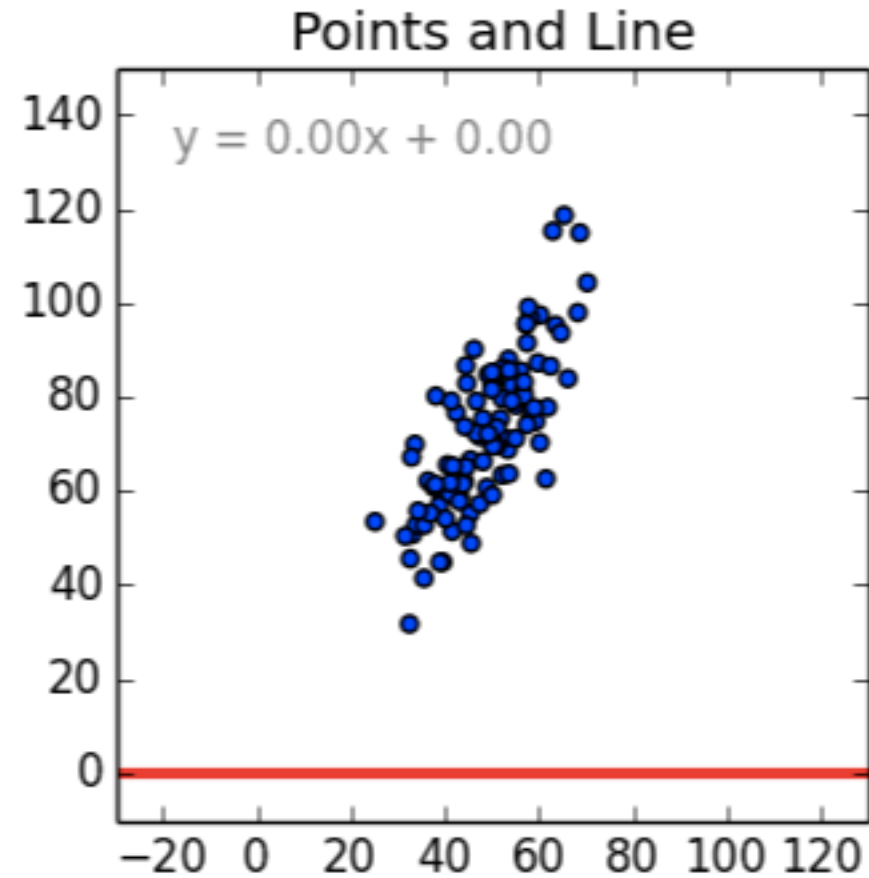
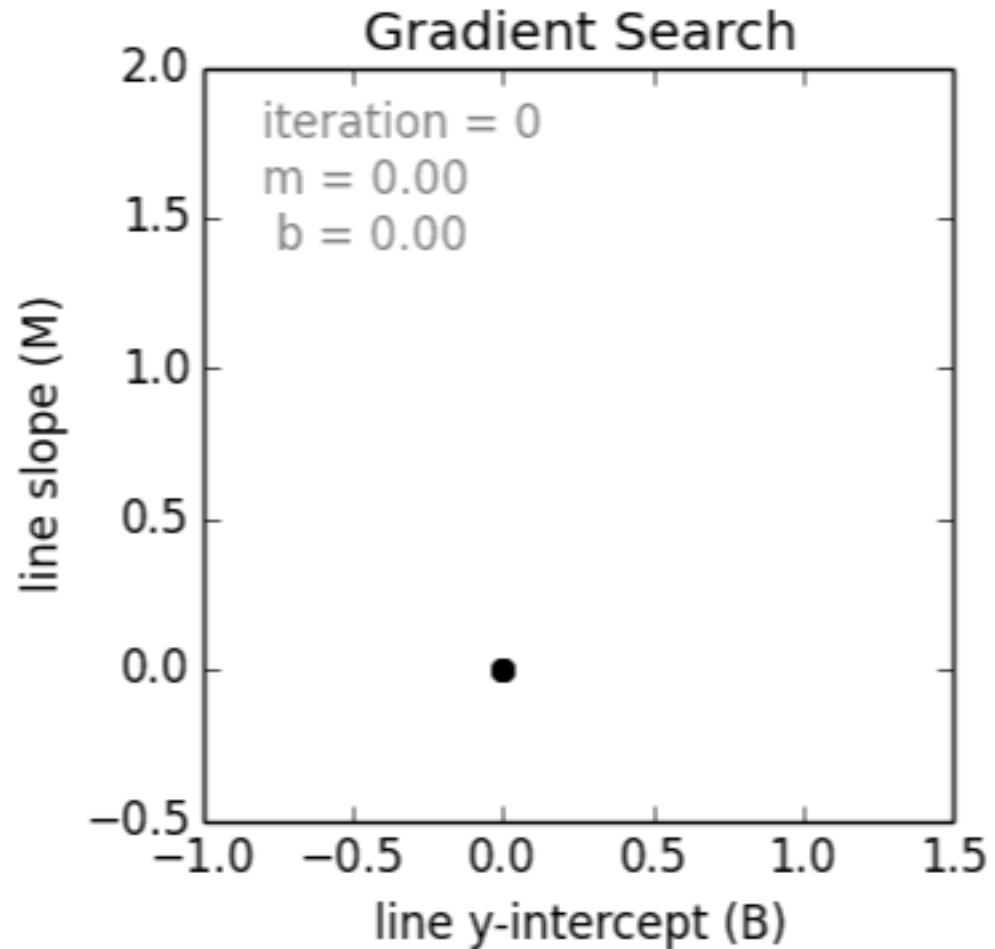
มีตัวแปร (parameter) ในโมเดล

ทำนายค่าจำนวนจริง


- Example

- โมเดล: $\text{life_satisfaction} = h(\theta) = \theta_0 + \theta_1 \times GDP$

Linear Regression



Linear Regression

- เราสามารถหาโมเดลที่ **fit** ข้อมูลได้ดีที่สุดด้วย คณิตศาสตร์ ได้อย่างไร?
- โมเดลการทำนาย: $\hat{y} = h_{\theta}(x) = \theta^T x$ (เช่น $h_{\theta}(x) = \theta_0 + \theta_1 \times GDP$)
- กำหนดเป้าหมาย $J(\theta)$  เรียกว่า **cost function**
- ต้องการหาพารามิเตอร์ θ ที่ทำให้ **cost** $J(\theta)$ น้อยที่สุด
- **cost function** ของ **linear regression** คือความคาดเคลื่อนกำลังสอง **Mean-Square Error**

$$J(\theta) = MSE(\theta) = \frac{1}{M} \sum_{i=1}^M (\theta^T x^{(i)} - y^{(i)})^2$$

- จะได้ว่า $\hat{\theta}_{MSE} = \operatorname{argmin}_{\theta} J(\theta)$
** ต้องการหาพารามิเตอร์ θ ที่ทำให้ **cost** ต่ำที่สุด **

Solution 1 – Closed-form solution

- $\hat{\theta}_{MSE} = \underset{\theta}{\operatorname{argmin}} J(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^M (\theta^T x^{(i)} - y^{(i)})^2$

**** ต้องการหา θ ที่ทำให้ cost ต่ำที่สุด ****

- หากใช้ **matrix calculus** แก้สมการตรง ๆ จะได้ว่า

$$\hat{\theta}_{MSE} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- สมการนี้มีชื่อว่า **Normal Equation**

ไม่ต้องจำ พิสูจน์ Normal Equation ด้วย Matrix Calculus

$$\begin{aligned}\frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta)\end{aligned}$$

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \frac{1}{2} (X\theta - \vec{y})^T (X\theta - \vec{y}) \\ &= \frac{1}{2} \nabla_{\theta} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y}) \\ &= \frac{1}{2} \nabla_{\theta} \text{tr} (\theta^T X^T X \theta - \theta^T X^T \vec{y} - \vec{y}^T X \theta + \vec{y}^T \vec{y}) \\ &= \frac{1}{2} \nabla_{\theta} (\text{tr} \theta^T X^T X \theta - 2 \text{tr} \vec{y}^T X \theta) \\ &= \frac{1}{2} (X^T X \theta + X^T X \theta - 2 X^T \vec{y}) \\ &= X^T X \theta - X^T \vec{y}\end{aligned}$$



ให้พจน์นี้เป็น 0 ก็จะได้ **Normal Equation**

Solution 1 – Closed-form solution

- ลองทำ linear regression ด้วยวิธี Normal Equation ใน Jupyter

```
data = genfromtxt('data.csv', delimiter=',')
X = data[:,0]
Y = data[:,1]
Xb = c_[ones((100,1)), X]
theta_mse = linalg.inv(Xb.T.dot(Xb)).dot(Xb.T).dot(Y)
x_test = arange(10,80).reshape((-1,1))
x_testb = c_[ones((len(x_test),1)), x_test]
y_hat = x_testb.dot(theta_mse)
```

train

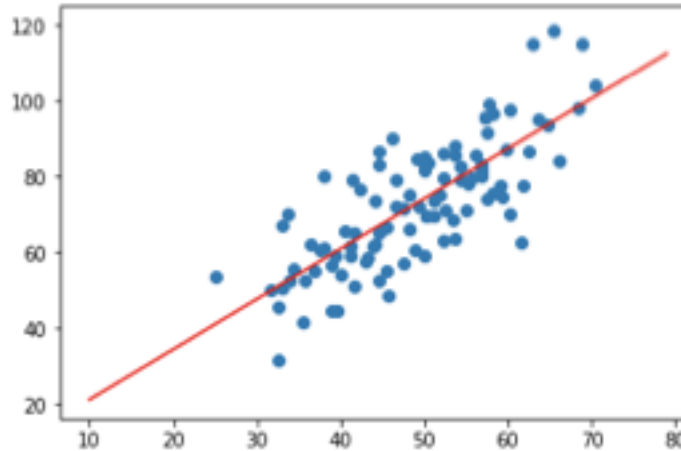
$$\hat{\theta}_{MSE} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

predict

$$\hat{y} = h_{\theta}(x) = \theta^T x$$

Solution 1 – Closed-form solution

ผลลัพธ์:



เส้นสีแดงเป็นตามสมการ

$$\hat{y} = h_{\theta}(x) = \theta^T x$$

- ข้อเสียของวิธี **Normal Equation/ Closed-form**
 - การคำนวณ matrix inverse $(X^T \cdot X)^{-1}$ มี complexity เป็น $O(n^{2.4}) \sim O(n^3)$
 - หากจำนวน feature เยอะ X จะใหญ่และทำให้คำนวณช้า
- ข้อดี
 - แม้มีจำนวน training data มาก ก็ไม่ทำให้ช้า $O(m)$

Solution 2 - Gradient Descent

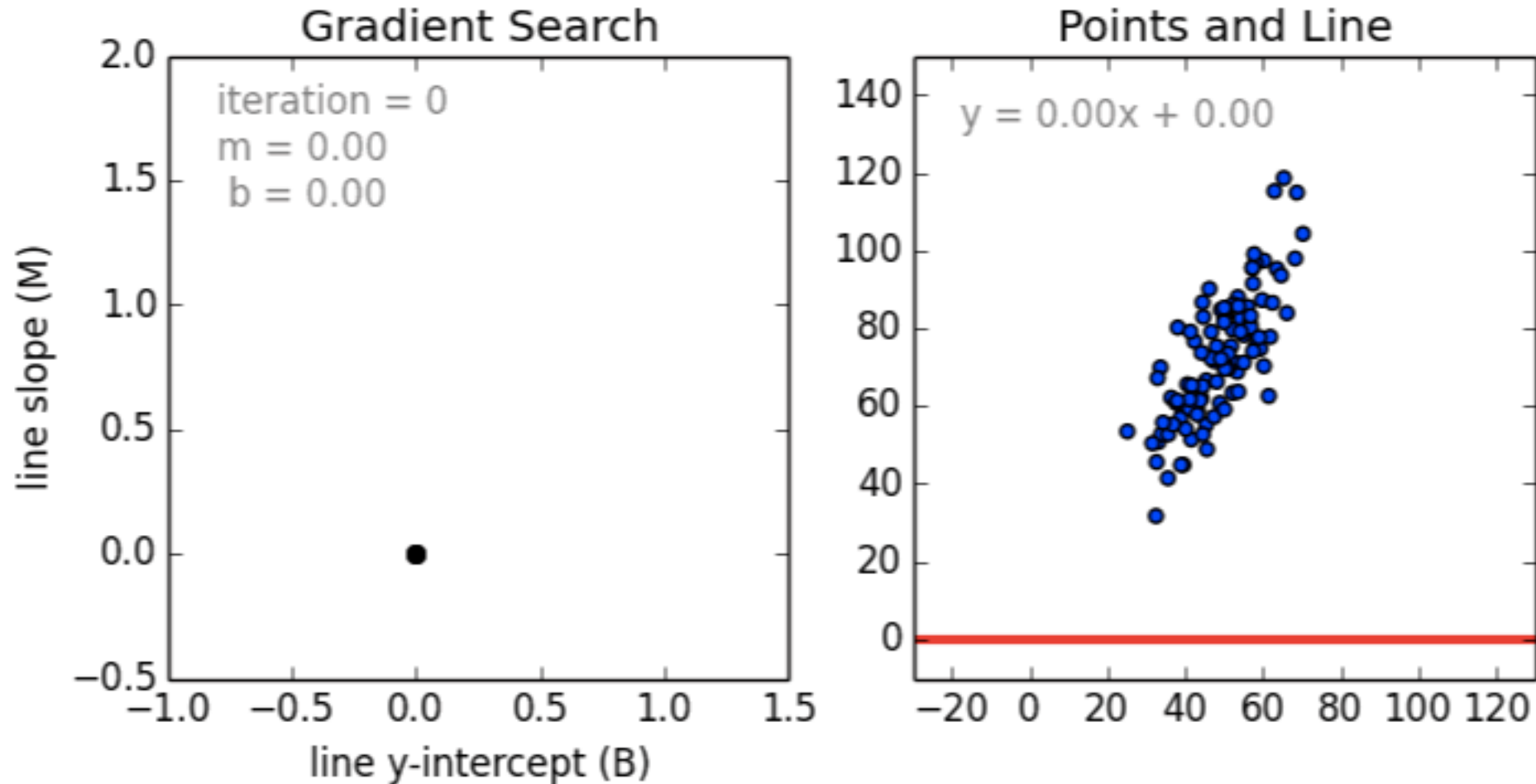
- ในกรณีที่เรามี **feature** เยอะ (น้ำหนัก, ส่วนสูง, ความดัน, ... เป็นพจน์ๆ **feature**) หรือ **training data** ใหญ่มาก เราอาจจะต้องหันมาใช้วิธี **Gradient Descent** แทนวิธี **normal equation**

- เป้าหมายเดิมคือ ต้องการหา θ ที่ทำให้ **MSE cost** ต่ำที่สุด

$$\hat{\theta}_{MSE} = \underset{\theta}{\operatorname{argmin}} J(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{M} \sum_{i=1}^M (\theta^T x^{(i)} - y^{(i)})^2$$

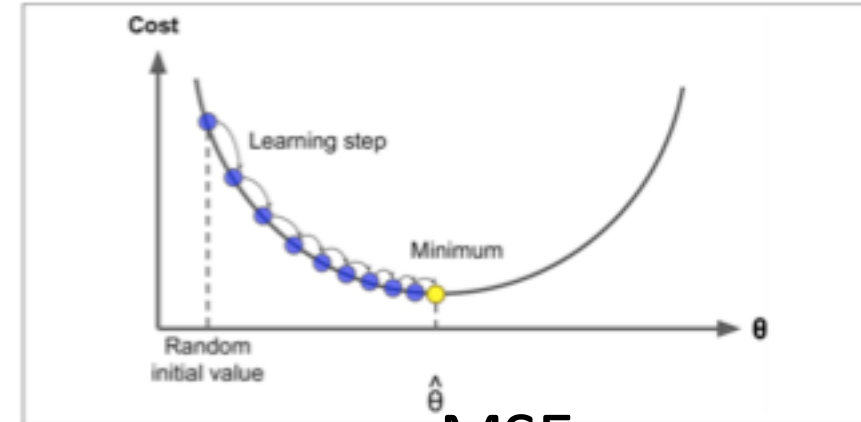
- แต่แทนที่จะแก้สมการตรงๆ เราจะเริ่มจากเดาค่า θ มาสักค่า มั่วๆ
- แล้วค่อยปรับ θ ทีละนิด ให้ **cost** ลดทีละนิด จนเราพอใจ

Solution 2 - Gradient Descent



Solution 2 - Gradient Descent

- อัลกอริทึม: วนทำสมการนี้ซ้ำไปเรื่อยๆ จนกว่า **MSE** จะเล็กมากพอ



ภูเขา **MSE**

$$\theta(\text{next step}) = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

θ เริ่มต้นเป็นค่า random

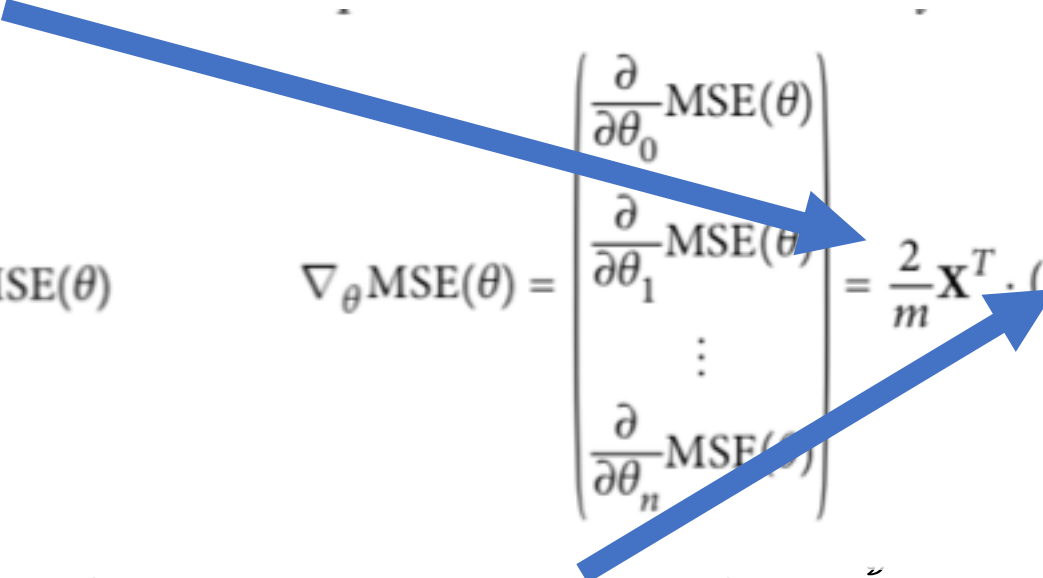
learning rate
จะก้าวใหญ่แค่ไหน

gradient ของ $\text{MSE}(\theta)$ w.r.t. θ
เป็นตัวบอกทิศทางลงภูเขา

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

Solution 2 - Gradient Descent

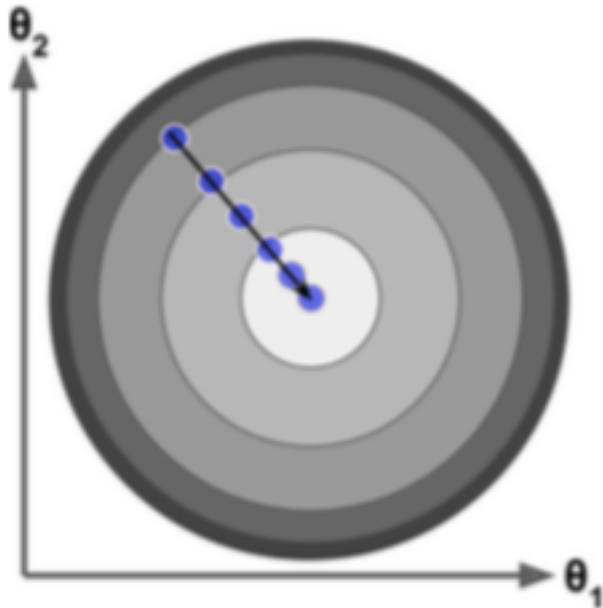
- ข้อดี: ไม่ต้องหา **matrix inverse** = เร็วกว่าวิธีแรกเมื่อฟีเจอร์เยอะ

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$
$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$


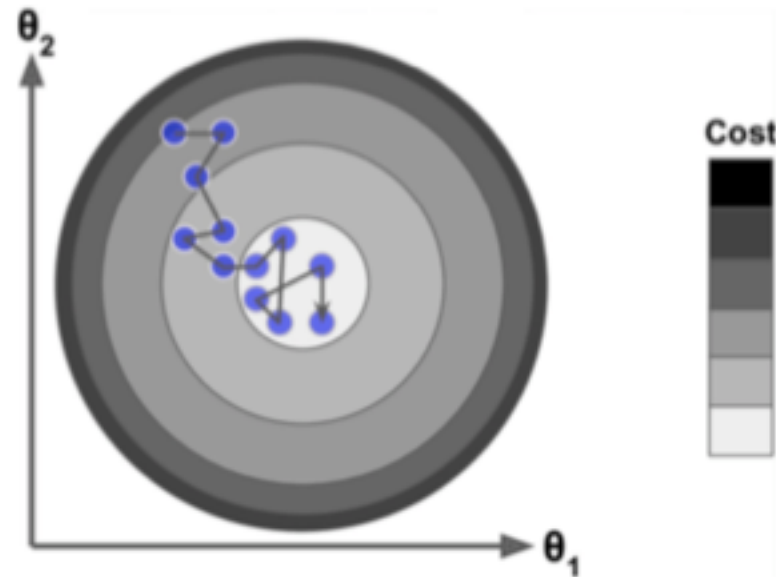
- ข้อเสีย: ทุกๆ **step** ต้องคำนวณ **gradient** โดยใช้ **X** (training data ทั้งหมด) = ช้า

Solution 3 – Stochastic Gradient Descent

- แทนที่จะใช้ \mathbf{X} ทั้งหมด ในแต่ละ **step**
- ให้เลือกข้อมูล \mathbf{x}_i มาบางตัวโดยสุ่ม เพื่อคำนวณ **gradient** = เร็วขึ้นมาก แต่ **converge** ช้าหน่อย



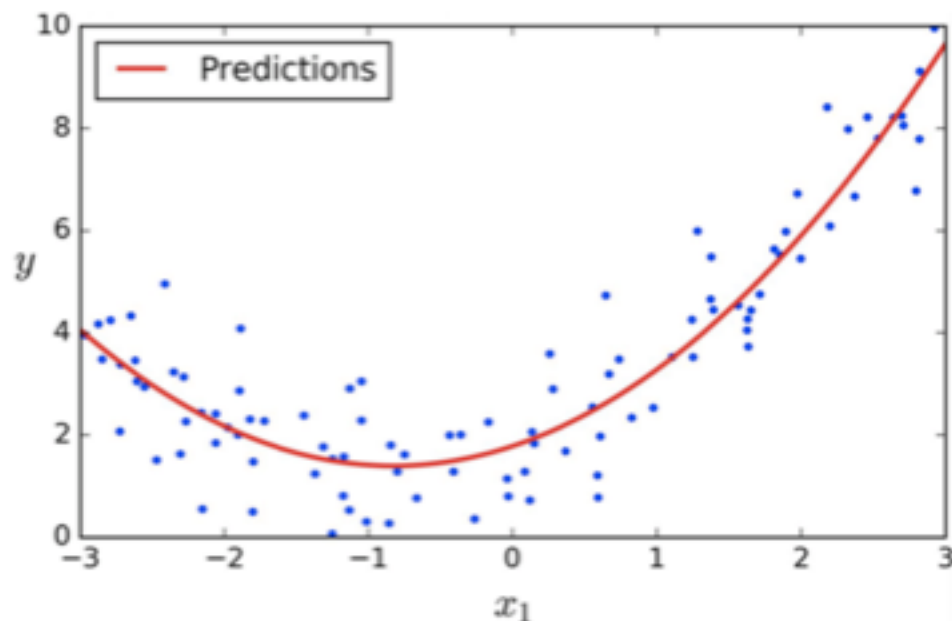
Solution 2: Batch GD



Solution 3: Stochastic GD

Polynomial Regression

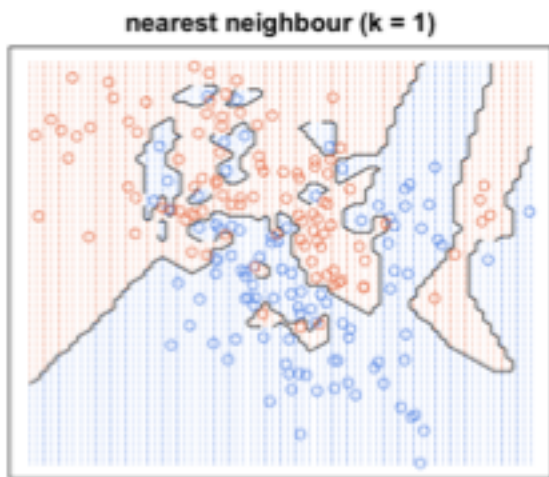
- โมเดล **linear regression** นี้สามารถใช้ทำนายความสัมพันธ์แบบ **non-linear** อย่าง x^2 ได้ด้วย
- เพียงแค่เปลี่ยนข้อมูล **training X** ให้มีค่าของ x^2 เพิ่มขึ้นมาด้วย เช่น
 - x_i เดิม = [น้ำหนัก, ส่วนสูง, อายุ] = [70, 150, 30]
 - x_i ใหม่ = [70, 150, 30, 4900, 22500, 900] → กลายเป็น 6 ฟีเจอร์ 6 พารามิเตอร์



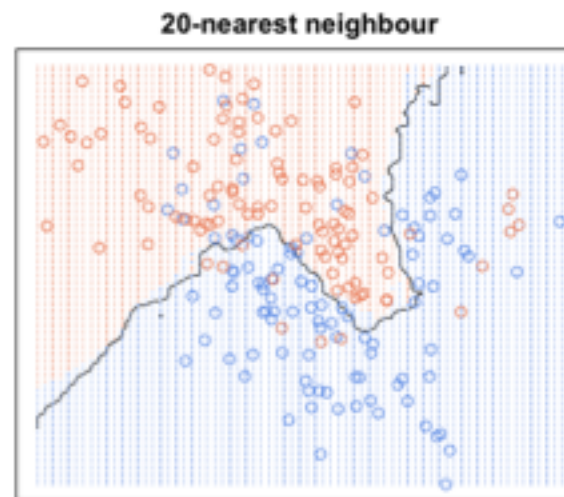
```
from sklearn.preprocessing import PolynomialFeatures  
poly_features = PolynomialFeatures(degree=2, include_bias=False)
```

Problem: Overfitting

- ปัญหาสุดท้าย **polynomial regression** อาจทำให้เกิดการ **overfit** ข้อมูล



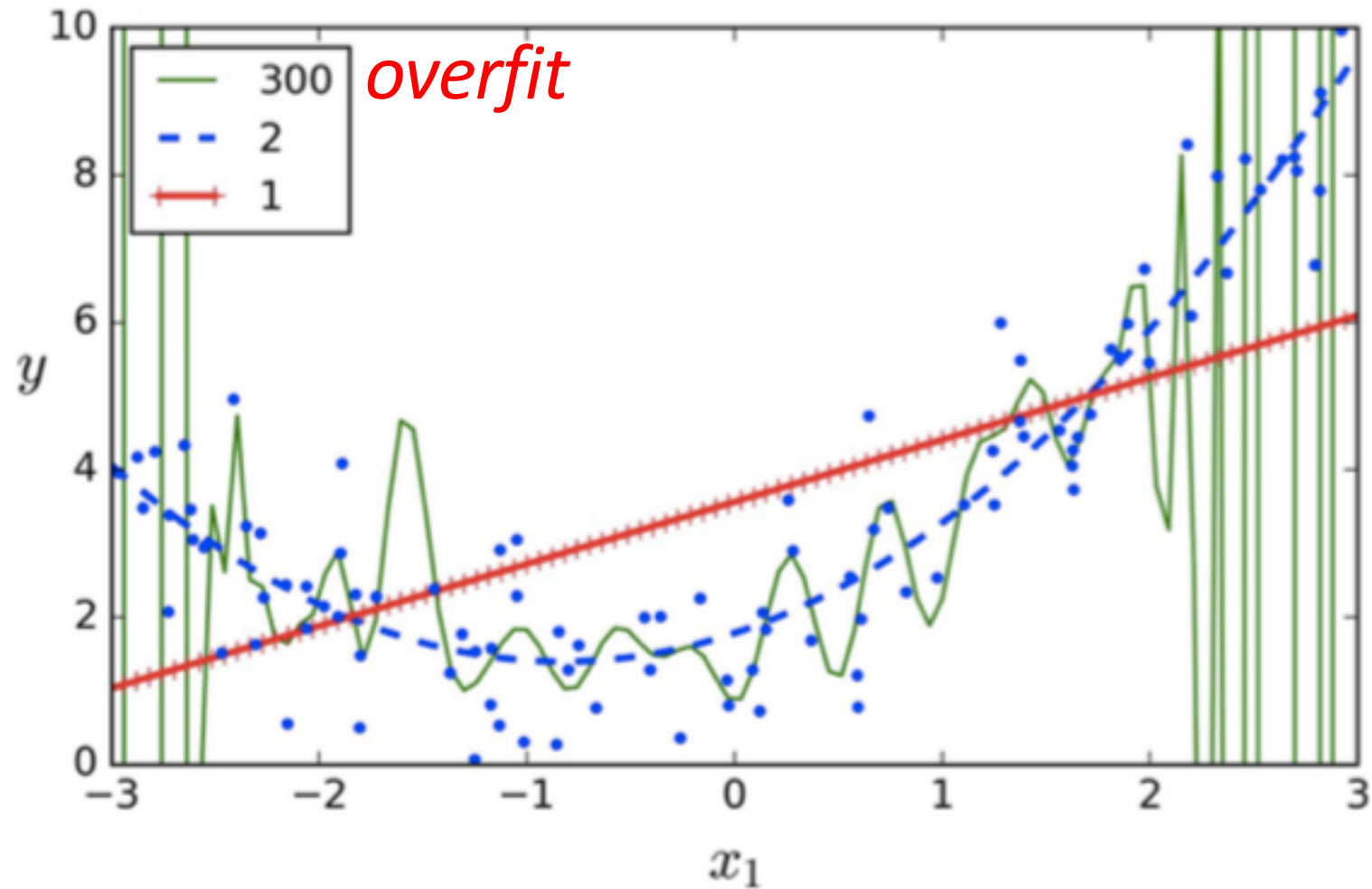
low bias 😊
high variance ☹️



high bias ☹️
low variance 😊

remember???

Problem: Overfitting



Solution to Overfitting: Regularization

- ทางออกคือเพิ่มพจน์ regularization ใน cost function

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

Ridge (L2) regularization

- พยายามให้ **MSE** น้อย แต่ก็พยายามให้ θ เล็กด้วย
- θ ที่เล็ก จะช่วยลดอิทธิพลของ polynomial degree สูงๆ อย่าง x^4, x^5 = ลด overfitting

Solution to Overfitting: Regularization

- ได้ด regularize สำหรับวิธี Normal Equation

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[ 1.55071465]])
```

- ได้ด regularize สำหรับวิธี SGD

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([[ 1.13500145]])
```

Recap

- Linear Regression
- Minimizing cost function
- Solutions
 - Normal Equation
 - (Stochastic) Gradient Descent
- Polynomial Regression
- Regularization -> solves overfitting

Recap: Supervised Learning

• Classification

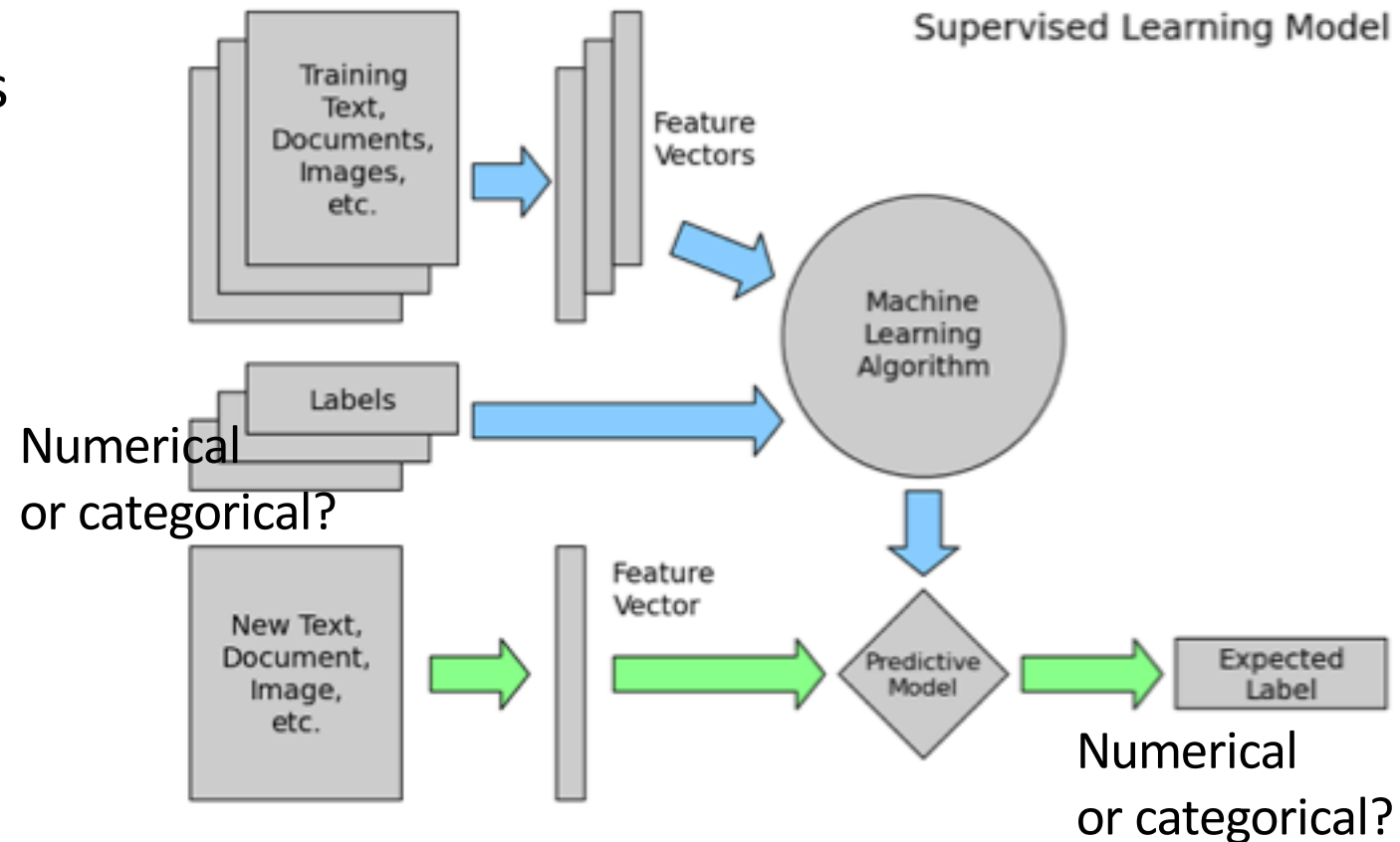
kNN

- Predicts class labels/categories
- ทำนายค่าที่เป็นหมวดหมู่ = จำแนกประเภท
- อาจมองเป็นการหา **boundary** ที่แบ่งข้อมูลในแต่ละหมวดหมู่ ออกจากกัน

• Regression

Linear
Regression

- Predicts continuous values
- ทำนายค่าที่เป็นจำนวนจริง
- อาจมองเป็นการหา **hyperplane** ที่ **fit** กับข้อมูลที่มีมากที่สุด



Logistic Regression

- เราสามารถใช้ **regression algo** ในการทำ **classification** ได้ด้วย !?
- โอเคดีกว่า... คือเราจะให้โมเดล $\hat{y} = h_{\theta}(x)$ ทำนายค่าในช่วง **0 – 1** เท่านั้น
- ค่าที่ได้ คือ ความน่าจะเป็นที่ข้อมูลนั้นอยู่ในคลาส **+** (เช่น เป็นมะเร็ง)
- **threshold** ที่ **50%** ถ้าเกินคือเป็นมะเร็ง น้อยกว่าคือไม่เป็น



* ใช้ตัว \hat{p} แทน y เพราะเป็นความน่าจะเป็น

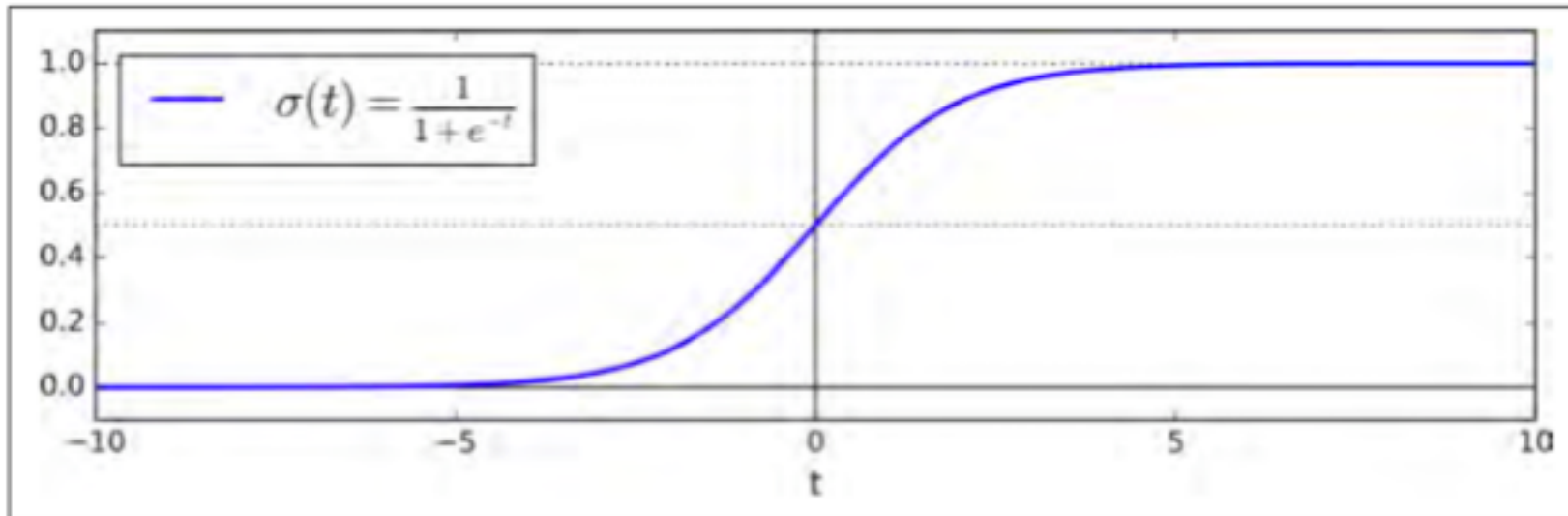
Logistic Regression

- วิธี: แคะเปลี่ยน model เป็น

$$\hat{p} = h_{\theta}(x) = \sigma(\theta^T x)$$

- โดยที่ $\sigma(t)$ คือ sigmoid (logistic) function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Logistic Regression

- **cost function** จะต้องเปลี่ยนไปด้วย
- **cost** สำหรับแต่ละ **instance** ข้อมูลคือ

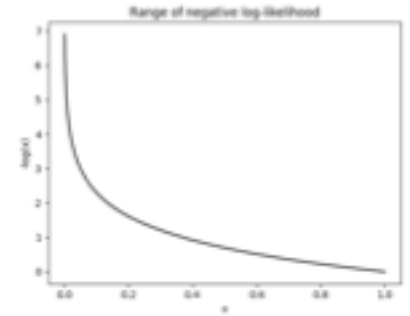
$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1, \\ -\log(1 - \hat{p}) & \text{if } y = 0. \end{cases}$$



ไอดีคือ ถ้าเฉลยเป็นมะเร็ง ($y=1$)

แต่ \hat{p} ทำนายว่าใกล้ 0 ค่า $\text{cost} = -\log(\sim 0)$ จะใหญ่มาก

ถ้า \hat{p} ทำนายว่าใกล้ 1 ค่า cost จะเป็น 0



- **cost** สำหรับทั้งชุดข้อมูลคือ

Go to page 158

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]$$

Logistic Regression

- จะ train โมเดลนี้อย่างไร? (minimize cost อย่างไร?)

Go to page 158

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]$$

- ไม่มี closed-form solution แบบ Normal Eq ☹️

Logistic Regression

- จะ **train** โมเดลนี้อย่างไร? (minimize cost อย่างไร?)

Go to page 158

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]$$

- ไม่มี closed-form solution แบบ Normal Eq ☹️
- แต่สามารถหา gradient/partial derivative ได้ 😊

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m \left(\sigma(\theta^T \cdot \mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

- ดังนั้น ใช้วิธี Stochastic Gradient Descent แบบเต็มได้

Lab: ใช้ Logistic Regression จำแนกพันธุ์ดอกไม้

```
from sklearn.linear_model import LogisticRegression
```



เป้าหมาย: จำแนก Iris-Virginica ออกจากชนิดอื่น โดยใช้ขนาดของกลีบ Sepal/Petal - width/length