การเรียนรู้ของเครื่อง

มีคนเคยให้คำนิยามเรื่องการเรียนรู้ของเครื่องไว้ 2 นิยาม

Arthur Samuel: "*The field of study that gives computers the ability to learn without being explicitly programmed*."

ศาสตร์ที่ให้คอมพิวเตอร์สามารถเรียนรู้โดยที่ไม่ต้องโปรแกรมโดยตรง

Tom Mitchell: "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*"

คอมพิวเตอร์โปรแกรมที่เรียนรู้จากประสบการณ์ (E) ซึ่งขึ้นอยู่กับงาน (T) และมีการวัดประสิทธิภาพ (P) ถ้าประสิทธิภาพในการประมวลผลสามารถเพิ่มความถูกต้องให้กับงานได้

ตัวอย่างเช่น การเล่นหมากรุก

E = ประสบการณ์ที่เล่นหมากรุกมาแล้วหลายครั้ง

T = งานคือการเล่นหมากรุก

P = ความน่าจะเป็นที่โปรแกรมจะชนะในการเล่นครั้งถัดไป

การเรียนรู้ของเครื่อง

สรุปแล้ว ปัญหาการเรียนรู้ของเครื่องสามารถแบ่งได้เป็น 2 กลุ่ม

1. Supervised learning การเรียนรู้โดยมีผู้สอน
2. Unsupervised learning การเรียนรู้โดยไม่มีผู้สอน

การเรียนรู้แบบมีผู้สอน

หรือ supervised learning คือการให้ชุดข้อมูล ที่เรารู้ว่าคำตอบที่ถูกต้องคืออะไร การให้ข้อมูลเพื่อหาความสัมพันธ์ระหว่างอินพุตและเอาท์พุต

การเรียนรู้แบบมีผู้สอน นำมาแก้ไขปัญหา ซึ่งแบ่งลักษณะปัญหาได้เป็น 2 แบบ คือ

- Regression ซึ่งหมายถึง การทำนายผลลัพธ์ที่มีลักษณะต่อเนื่อง คือการพยายามที่จะจับคู่ระหว่างข้อมูลอินพุตกับฟังก์ชันต่อเนื่องบางอย่าง
- Classification ตรงกันข้ามกับ regression คือการทำนายผลลัพธ์ที่มีลักษณะไม่ต่อเนื่อง เป็นการจัดข้อมูลอินพุตเข้ากลุ่มใดกลุ่มหนึ่ง

การเรียนรู้แบบไม่มีผู้สอน

ตัวอย่างที่ 1: clustering

มีชุดข้อมูลที่มี 1 ล้านยีนส์ ต้องการหาทางที่จะจัดกลุ่มยีนส์เหล่านี้โดยอัตโนมัติตามลักษณะใกล้เคียงกันอยู่กลุ่มเดียวกัน ลักษณะเหล่านี้ เช่น เชื้อชาติ, โรคทางพันธุกรรม

ตัวอย่างที่ 2: Non-clustering

ปัญหา "Cocktail Party Algorithm", ต้องการหาโครงสร้างในสภาพแวดล้อมที่สับสนวุ่นวาย เช่น การระบุเจ้าของเสียง การแยกเสียงดนตรี

อัลกอริธึมประเภทต่างๆ

Supervised	Unsupervised
1. Naive Bayes 2. Random Forest, Decision Trees 3. Linear Regression 4. Logistic Regression 5. Support Vector Machines 6. Neural Networks	1. K-means Clustering 2. Principal Component Analysis 3. Single Value Decomposition

Decision Trees

- หรือต้นไม้ตัดสินใจ เป็นอัลกอริทึมแบบ supervised learning
- มีลักษณะโครงสร้างแบบต้นไม้ ประกอบด้วย จุดตัดสินใจ และผลลัพธ์ ของชุดข้อมูล
- พิจารณาจุดตัดสินใจจากข้อมูล training set โดยพิจารณา features และค่าที่เป็นไปได้ของ features
- ใช้การวัดประสิทธิภาพเพื่อเปรียบเทียบว่า features ใดดีที่สุด

วิธีการหา decision tree ด้วย Greedy algorithm

1. เริ่มจาก tree ว่าง
2. จาก feature ที่ยังไม่ได้พิจารณาเลือก feature มา 1 ชนิด ถ้าพิจารณาครบทุก feature แล้วให้หยุด
3. แบ่งตามค่าของ feature ที่เป็นไปได้ แต่ละค่านับจำนวนตามเอาท์พุต
4. ถ้า feature ใดให้ค่า classification error ต่ำสุดให้กำหนดเป็นจุดตัดสินใจ
    a. ถ้าข้อมูลทุกชุดในค่าของ feature นั้นได้เฉลยหมือนกัน หยุดสร้าง subtree ในคำตอบนั้น
    b. ถ้ามีหลายคำตอบที่เป็นไปได้ สร้าง tree วนไปข้อ 2 เพื่อพิจารณา feature อื่นเพื่อหาจุดตัดสินใจถัดไป

ดังนั้น จะใช้เวลาหา decision tree นานเท่าไร ขึ้นอยู่กับลำดับการเลือก feature ขึ้นมาพิจารณา

แต่ละอัลกอริทึม มีเกณฑ์วัดที่ต่างกัน เช่น ID3 ใช้ Information Gain, C4.5 ใช้ Gain Ratio

$$error = \frac{\#\text{wrong prediction}}{\#\text{total sample}}$$

Entropy และ Information Gain

Entropy ใช้วัดความแปรปรวนของข้อมูล ถ้าค่าต่ำแสดงว่าข้อมูลเกาะกลุ่มกัน ใกล้เคียงกัน

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

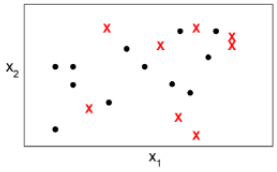
Information gain ใช้วัดความสัมพันธ์ระหว่างข้อมูล ค่าที่ได้ใช้บ่งบอกว่าสามารถแบ่งกลุ่มได้ดีแค่ไหน

$$Gain(X, A) = H(X) - \sum_{i=1}^{n} [P[X|A_i] * H(X|A_i)]$$

Entropy $H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$

ตัวอย่างเช่น

- มีข้อมูล 2 กลุ่ม คือ X และ O



คำนวณค่าความน่าจะเป็นของแต่ละกลุ่มได้

- P(X)=8/20=0.4,
- P(O)=12/20=0.6

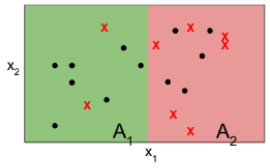
ได้ค่า entropy

$$H(X) = -[0.4 \log_2 0.4 + 0.6 \log_2 0.6] = -(-0.528 - 0.442) = 0.97$$

ค่า entropy ยิ่งต่ำแสดงว่า ความแปรปรวนของข้อมูลมีน้อย แต่ละ sample คล้ายกัน

Information Gain $Gain(X, A) = H(X) - \sum_{i=1}^{n} [P[X|A_i] * H(X|A_i)]$

กำหนดการแบ่งเขตดังรูป



เพื่อต้องการทำนายว่า

- จุดใดอยู่ในสีเขียวคือ O
- จุดใดอยู่ในสีเขียวคือ X

จากการแบ่งดังภาพ เราคำนวณค่า Information Gain ได้

$$Gain(X, A) = H(X) - [P[X|A_1] * H(X|A_1) + P[X|A_2] * H(X|A_2)]$$

$$Gain(X, A) = 0.97 - [(9/20 * (\frac{2}{9} \log_2 \frac{2}{9} + \frac{7}{9} \log_2 \frac{7}{9})) + (11/20 * (\frac{6}{11} \log_2 \frac{6}{11} + \frac{5}{11} \log_2 \frac{5}{11}))] = 1.858$$

อัลกอริทึม C4.5

- อัลกอริทึม C4.5 ปรับปรุงจาก ID3
- สามารถใช้กับ features ที่มีค่าต่อเนื่อง
- จัดการกับกรณีที่มี features มีค่าไม่ครบได้
- C4.5 เปลี่ยนจาก gain (ใช้ใน ID3) มาใช้ค่า gain ratio ในการเลือกจุดตัดสินใจ
- Gain ration คำนวณได้จากสมการ

$$GainRatio(A) = Gain(A)/H(A)$$

Random Forest

อัลกอริทึม Random forest มีขั้นตอนดังนี้

1) สุ่มข้อมูล และ feature ที่มีออกมากลุ่มหนึ่ง และ
2) หา decision tree ที่ดีที่สุดสำหรับข้อมูลกลุ่มนี้
3) ทำซ้ำข้อ 1)

ผลลัพธ์สุดท้ายได้จากการหาค่าเฉลี่ยของทุก trees

Random forest เปรียบได้กับ กลุ่มของ decision trees จำนวนหนึ่งที่สร้างขึ้นจากการสุ่มนั่นเอง

ปัญหา Overfitting

Overfitting คือปัญหาที่เกิดจาก การสร้างแบบจำลองที่ยึดติดกับความถูกต้องของ training data มากเกินไป ทำให้เมื่อใช้งานจริงเจอ unseen test data แล้วความถูกต้องของระบบลดลง

วิธีแก้ปัญหา Overfitting

- เมื่อสร้าง decision tree ไปเรื่อยๆ จนกระทั่ง ได้ความถูกต้องสูง อาจเกิดปัญหา overfitting ได้
- ปัญหา overfitting เกิดจาก decision tree ยึดติดกับผลทำนายของ training data มากเกินไป
- ซึ่งสามารถแก้ไขได้โดยเราจะเลือก tree ที่มีความซับซ้อนน้อยกว่าด้วยวิธี early stopping หยุดก่อนที่ tree จะซับซ้อน

Classification Problem with Linear Regression

นอกเหนือจากปัญหาที่กล่าวมา ยังพบว่า

$h_\theta(x)$ ที่ได้จากการคำนวณ linear regression มีขอบเขตไม่สอดคล้องกับผลลัพธ์ของปัญหาที่เราต้องการ

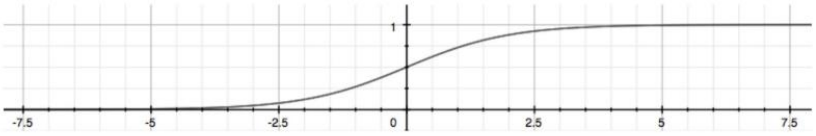
$h_\theta(x) > 1$ หรือ $h_\theta(x) < 0$ ได้

ในขณะที่ ผลลัพธ์ที่เราต้องการ มีเพียงค่า 0 หรือ 1

Logistic Regression : $0 \le h_\theta(x) \le 1$

Hypothesis function

วิธีการทำให้ผลลัพธ์ของ linear regression มีแค่ 0 และ 1 คือ ใส่ฟังก์ชัน g(h_θ(x)) เพื่อปรับค่า ฟังก์ชันนี้เรียกว่า logistic function หรือ sigmoid function



ทำให้ hypothesis function เปลี่ยนเป็น $h_\theta(x) = g(\theta^T x)$

$$z = \theta^T x$$
$$g(z) = \frac{1}{1 + e^{-z}}$$

Hypothesis function

เราแปลความหมายของค่าผลลัพธ์ของ hypothesis function ได้เป็น

- ความน่าจะเป็นที่เอาท์พุตจะมีค่าเป็น 1 ต่อเมื่อ x

$$h_\theta(x) = P[y = 1|x; \theta]$$

- ความน่าจะเป็นที่เอาท์พุตจะมีค่าเป็น 0 ต่อเมื่อ x

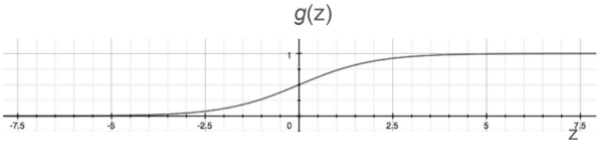
$$h_\theta(x) = P[y = 0|x; \theta]$$

ตามคุณสมบัติของความน่าจะเป็น จะได้

$$h_\theta(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$
$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$

Decision Boundary

จากสไลด์ก่อนหน้า เราแปลงให้ h_θ(x) มีคำตอบแค่ 0 และ 1 ด้วยการใส่ sigmoid function ดังภาพ



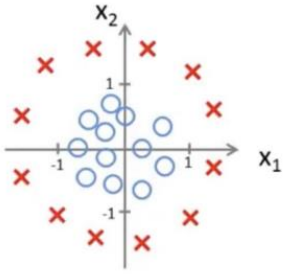
จาก $h_\theta(x) = g(\theta^T \mathbf{X}); \quad g(z) = \frac{1}{1+e^{-z}}$

เราทำนายว่า y = 1 เมื่อ $h_\theta(x) \geq 0.5$

y = 0 เมื่อ $h_\theta(x) < 0.5$

เส้นกราฟนี้ เรียกว่า decision boundary มีหน้าที่แบ่งเขตของการทำนายว่า y=0 หรือ y=1

Decision Boundary



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_2^2$$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

ทำนายว่า y = 1 เมื่อ $-1 + x_1^2 + x_2^2 \geq 0$

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Cost function

จาก linear regression เรามี cost function ดังสมการ

$$J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

แต่เนื่องจาก logistic regression ใช้ฟังก์ชัน $g(z) = \frac{1}{1+e^{-z}}$ ทำให้การคำนวณซับซ้อน

ดังนั้นเราต้องเปลี่ยน cost function เป็น $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x), y)$

โดยที่ $$cost(h_\theta(x), y) = \begin{cases} -\log h_\theta(x) & \text{if } y = 1, \\ -\log(1 - h_\theta(x)) & \text{if } y = 0. \end{cases}$$

Cost function $$cost(h_\theta(x), y) = \begin{cases} -\log h_\theta(x) & \text{if } y = 1, \\ -\log(1 - h_\theta(x)) & \text{if } y = 0. \end{cases}$$

เนื่องจาก cost function แยกเป็น 2 สมการ จะยากต่อการทำ gradient descent ดังนั้นเราจะยุบ cost function ให้เหลือเป็น**สมการเดียว** และยังคงให้ผลลัพธ์เหมือนเดิม ได้ดังนี้

$$cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

เมื่อแทนที่ ใน $J(\theta)$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

Gradient descent

จาก cost function เราสามารถทำ parameter learning ได้ดังนี้

ทำซ้ำจนกระทั่งลู่เข้า {

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

สมการนี้ได้จากการอนุพันธ์ $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

หรือเขียนอยู่ในรูปเวกเตอร์ได้

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

Optimization of $J(\theta)$

นอกเหนือจาก gradient descent ยังมีอัลกอริทึมอื่นอีก เช่น BFGS, L-BFGS ซึ่งอัลกอริทึมพวกนี้มีให้ในไลบรารี เราไม่จำเป็นต้องเขียนโค้ดเอง อีกทั้งยังทำให้ ลู่เข้าเร็วมากกว่า gradient descent

วิธีการใช้งาน เราต้องเขียนโค้ดเพื่อคำนวณค่า

1. $J(\theta)$
2. $\frac{\partial}{\partial \theta_j} J(\theta)$

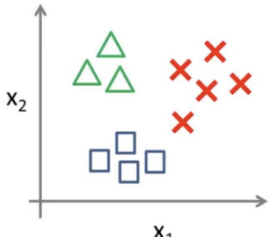
```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

ใน Octave มีฟังก์ชัน fminunc() ใช้สำหรับ optimize

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
     options);
```

Multiclass Classification (One vs. All)

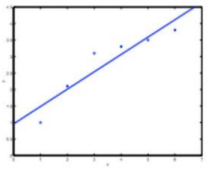
จากเดิมที่มีการทำนายผลลัพธ์เพียงแค่ 2 แบบ (binary classification) คือ y={0,1} เราขยายขอบเขตเป็นทำนายผลลัพธ์หลายแบบ y={0,1,...,n} โดยประยุกต์ใช้ binary classification ดังนี้



$$y \in \{0, 1 \ldots n\}$$
$$h_\theta^{(0)}(x) = P(y = 0|x; \theta)$$
$$h_\theta^{(1)}(x) = P(y = 1|x; \theta)$$
$$\cdots$$
$$h_\theta^{(n)}(x) = P(y = n|x; \theta)$$
$$\text{prediction} = \max_i(h_\theta^{(i)}(x))$$

Underfitting Problem

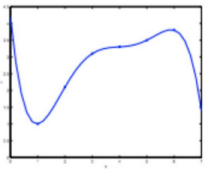
จาก linear regression เรามีสมการเส้นตรง ทำให้มี ความคาดเคลื่อนในการทำนายสูง



$$y = \theta_0 + \theta_1 x$$

Overfitting Problem

จากการพยายามปรับให้ระบบทำนายได้ถูกต้องแม่นยำที่สุดจากชุดข้อมูลที่มี พบว่า จะต้องเพิ่ม feature จนกระทั่งถึง polynomial ลำดับที่ 5



$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_5 x^5$$

Overfitting vs. Underfitting

- เมื่อเราพยายามปรับให้ระบบทำนายชุดข้อมูลที่มีได้ถูกต้องมากเกินไป เมื่อถึงตอนใช้งานจริงอาจมีอินพุตใหม่ๆ เข้ามา จะทำให้ระบบทำนายได้ไม่ถูกต้อง เราเรียกปัญหานี้ว่า overfitting
- ในทางตรงกันข้าม ถ้าระบบที่เราสร้างขึ้นไม่ได้ครอบคลุมหรือใกล้เคียงกับชุดข้อมูลเลย เช่นในภาพแรก ก็จะเกิดปัญหาระบบทำนายไม่ถูกต้อง เราเรียกปัญหานี้ว่า underfitting
- ทางแก้ปัญหาเหล่านี้ คือ
  a. ลดจำนวน feature ลง โดยใช้ algorithm หรือ เลือกด้วยตัวเอง
  b. Regularization ใช้วิธีลดน้ำหนักของ feature (ปรับค่า θ) ซึ่งเหมาะกับกรณีที่เรามี feature จำนวนมาก แต่ๆละ feature มีประโยชน์น้อย

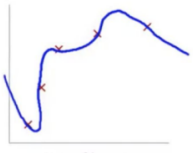
Regularization

หลักการคือ เราปรับค่าน้ำหนักของ feature บางตัว ด้วยการเพิ่มค่า cost ของ feature นั้นลงในสมการ cost function

ตัวอย่างเช่น เรามีสมการ quadratic $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

เราจะปรับน้ำหนักของ θ_3, θ_4 ได้โดยเพิ่มน้ำหนักที่ cost function ทำให้ parameter learning ลดน้ำหนักของ θ_3, θ_4 ลงเนื่องจากมันส่งผลเสียต่อการทำนาย

$$min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2$$



Regularization

สมการสำหรับ regularization เป็นดังนี้

$$min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

โดยที่ λ เรียกว่า regularization parameter เป็นค่าคงที่ เพื่อกำหนดว่าเราจะปรับเส้นแบ่งให้ราบลงเพียงใด

- สังเกตว่า เราจะไม่ปรับพารามิเตอร์ θ_0 ปรับแค่ $\theta_1, \theta_2, \ldots, \theta_n$

Regularization

จากสมการ regularization สิ่งที่เราต้องกำหนดคือ regularization parameter หรือ λ

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

- ถ้าเรากำหนดค่า λ สูงมาก ทำให้ค่าพารามิเตอร์ θ มีค่าต่ำลง จนเกิด underfitting

- ถ้าเรากำหนดค่า λ ต่ำเกินไป จะไม่เกิดอะไรขึ้น ผลคือ overfit ดังเดิม

Regularized Linear Regression

gradient descent เมื่อเราทำ regularization กับ linear regression

ทำซ้ำจนกระทั่งลู่เข้า {

$$\theta_0 := \theta_0 - \alpha\ \frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\left[\left(\frac{1}{m}\ \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right] \qquad j \in \{1, 2...n\}$$

}

Regularized Linear Regression

ถ้าเราใช้วิธี Normal Equation แทน gradient descent สามารถทำได้โดย

$$\theta = (X^T X + \lambda L)^{-1} X^T y$$

โดยที่ L คือ identity matrix ที่มีขนาด n+1 x n+1 และกำหนดให้ L(1,1)=0

$$L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

Regularized Logistic Regression

สำหรับ Logistic Regression สามารถทำ regularization ด้วยวิธีปรับ cost function เช่นเดียวกัน ดังนี้

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1 - y^{(i)})\ \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

สังเกตว่า เราจะไม่ปรับค่า θ_0 ปรับเริ่มจาก $\theta_1, \theta_2, \ldots, \theta_n$

Regularized Logistic Regression

ดังนั้นเมื่อเราทำ gradient descent จะเป็นดังนี้

ทำซ้ำจนกระทั่งลู่เข้า {

$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})x_0^i$$

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})x_j^i + \frac{\lambda}{m}\theta_j; \qquad j = 1, 2, \ldots, n$$

update ค่า θ พร้อมกัน

}

Support Vector Machine คืออะไร

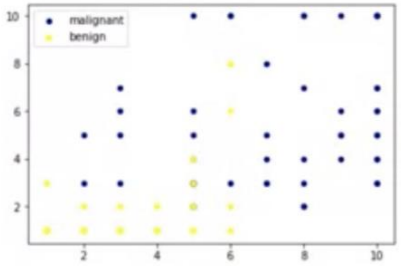
Support Vector Machine - SVM คือ supervised algorithm ที่ทำนายผลด้วยการหาเส้นแบ่ง (separator) ประกอบด้วย 2 ขั้นตอน

1. Map ข้อมูลไปยัง feature space ที่มิติที่สูงขึ้น (high dimensional)
2. หาระนาบที่แบ่งแยกข้อมูล (hyperplane)

เมื่อข้อมูลถูกย้ายไปยังมิติที่ซับซ้อนขึ้นในขั้นตอนที่ 1 จะทำให้ข้อมูลแต่ละประเภทแยกออกจากกัน จนสามารถหา hyperplane ในขั้นตอนที่ 2 ได้

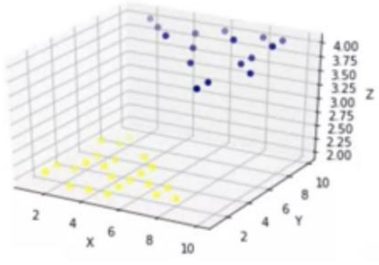
Linearly Separable

ลองพยายามลากเส้นแบ่งระหว่าง 2 ประเภท



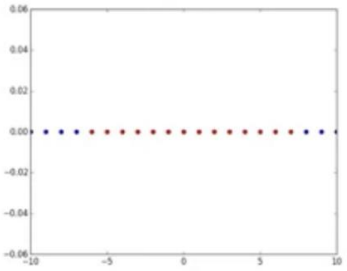
not linearly separable -แบ่งด้วยเส้นตรงไม่ได้

เปลี่ยน data จาก 2D ไปยัง 3D



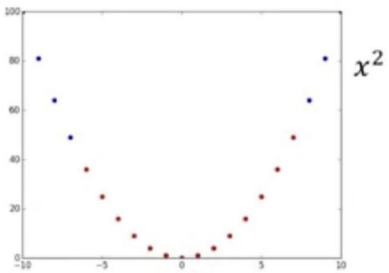
linearly separable -แบ่งด้วยเส้นตรงได้

Linearly Separable



not linearly separable -แบ่งด้วยเส้นตรงไม่ได้

เปลี่ยน x เป็น x^2

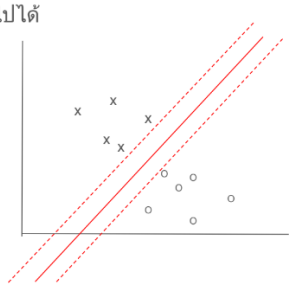


linearly separable -แบ่งด้วยเส้นตรงได้

Large Margin Classifier

- margin คือช่วงขยายจากเส้นแบ่งไปจนถึงข้อมูลจุดแรก
- hyperplane ที่ดีคือ มี margin ใหญ่สุดเท่าที่จะเป็นไปได้
- แค่จุดที่เป็น support vector เท่านั้นที่สำคัญ
- เราต้องใช้ training data และ optimizer เพื่อหา

hyperplane ที่ดีที่สุด

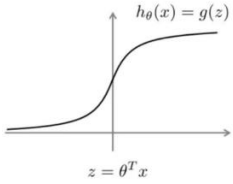


Hypothesis function

Hypothesis function ของ SVM เป็นดังสมการ

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Cost function

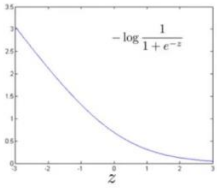


จาก logistic regression ซึ่ง

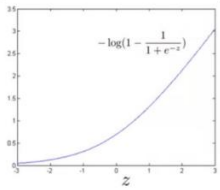
- y = 1 เราต้องการให้ $h_\theta(x) \approx 1$ หมายความว่า $\theta^T X >> 0$
- y = 0 เราต้องการให้ $h_\theta(x) \approx 0$ หมายความว่า $\theta^T X << 0$

cost ของ logistic regression คือ $J(\theta) = -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y) \log(1 - \frac{1}{1+e^{-\theta^T x}})$

y=1



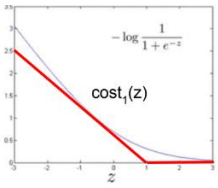
y=0

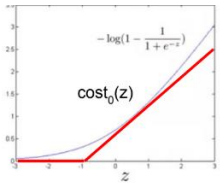


Cost function

เมื่อเราปรับ logistic cost ใหม่

จะได้ SVM cost function คือ





$$J(\theta) = y cost_1(\theta^T x) + (1-y) cost_0(\theta^T x)$$

จะได้ optimization objective function คือ

$$min_\theta \frac{1}{m} \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Optimization Objective Function

Optimization SVM ต่างจาก logistic regression

$$min_\theta \frac{1}{m} \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)}) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

กรณี Logistic regression สามารถเขียนสมการอยู่ในรูป A+λB

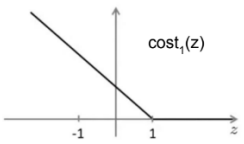
สำหรับ SVM มองสมการใหม่เป็น CA+B โดย C = 1/λ

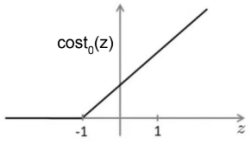
$$min_\theta \, C \sum_{i=1}^{m} [y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

Optimization

เราต้องการให้ A (cost) มีค่าต่ำมาก ≈ 0 ส่งผลให้ C มีค่าสูงมาก

$$min_\theta\ C\sum_{i=1}^{m}[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$





ถ้า y=1 เราต้องการให้ $\theta^T X \geq 1$

$cost_1$ จะได้เท่ากับ 0

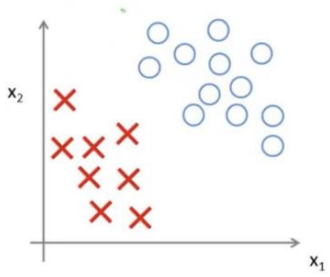
ถ้า y=0 เราต้องการให้ $\theta^T X \leq -1$

$cost_0$ จะได้เท่ากับ 0

Large Margin Classifier

จาก hypothesis function

$$h_\theta(x) = \begin{cases} 1 & \theta^T(x) \geq 1, \\ 0 & \theta^T(x) < -1. \end{cases}$$



Dot product และ Vector projection

ถ้าเรามีเวกเตอร์ U และ V เราสามารถหา

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad V = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

- dot product ได้
$$U \cdot V = u_1 v_1 + u_2 v_2$$
- U project onto V ได้
$$U \cdot V = |U||V|cos\theta$$
- ถ้าคำนวณเมตริกซ์ $U^T V$
$$U^T V = u_1 v_1 + u_2 v_2$$

เมื่อเทียบสมการทั้งหมด จะได้

$$U^T V = |U||V|cos\theta$$

ถ้ากำหนดให้ p คือ ขนาดของ V ที่ project ลงบน U ทำให้

$$U^T V = |U| * p$$

Gaussian Kernel

สมมติให้มีตัวแทน (L={l_1,l_2,...}) เพื่อหาค่าความเหมือนกับ X={x_1,x_2,...} เราจะแปลง x ให้เป็น f ดังนี้

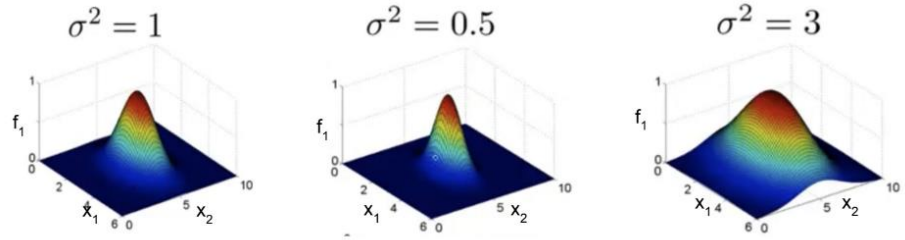
$$f_1 = similarity(l_1, x) = \exp(-\frac{||x-l_1||^2}{2\sigma^2})$$

if $x \approx l_1$; $\quad f_1 = \exp(-\frac{0}{2\sigma^2}) \approx 1$

if x ไกลจาก l_1; $\quad f_1 = \exp(-\frac{\text{ค่ามาก}}{2\sigma^2}) \approx 0$

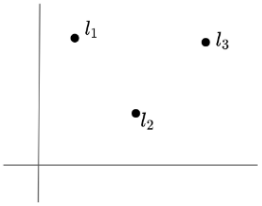
Gaussian Kernel

การกำหนดค่า σ $f_1 = similarity(l_1, x) = \exp(-\frac{||x-l_1||^2}{2\sigma^2})$



มีการเปลี่ยนแปลงเร็ว มีการเปลี่ยนแปลงช้า

Gaussian Kernel



ทำนาย y = 1 เมื่อ

$h_\theta(x) = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

โดยที่ $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$

ถ้า x อยู่ใกล้ l_1, f_1 = 1, $f_2 \approx 0$, $f_3 \approx 0$

ได้ $h_\theta(x) = 0.5$ ทำนายว่า y=1

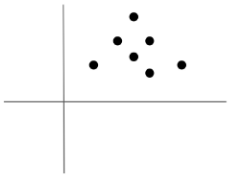
ถ้า x อยู่ไกลจากทุก l , $f_1 \approx 0$, $f_2 \approx 0$, $f_3 \approx 0$

ได้ $h_\theta(x) = -0.5$ ทำนายว่า y=0

วิธีหาค่า L

$L = \{l_1, l_2, l_3, \ldots\}$ ตัวแทนสำหรับหาค่าความเหมือน จะหาได้จากไหน

วิธีที่ง่ายที่สุดคือ กำหนดทุก $x^{(i)}$ เป็น l_i ถ้าเรามีชุดข้อมูล m ชุด เราจะได้



$$f^{(i)} = \begin{bmatrix} f_1^{(i)} \\ f_2^{(i)} \\ \cdot \\ \cdot \\ \cdot \\ f_m^{(i)} \end{bmatrix}$$

การกำหนดค่า SVM พารามิเตอร์

C หรือ regularization parameter มีค่า $c = \frac{1}{\lambda}$

- ถ้า high bias, low variance ให้กำหนดค่า C น้อย (λ มาก)
- ถ้า low bias, high variance ให้กำหนดค่า C มาก (λ น้อย)

σ^2 จาก Gaussian Kernel $f_1 = similarity(l_1, x) = \exp(-\frac{||x-l_1||^2}{2\sigma^2})$

- ถ้า high bias, low variance ให้กำหนดค่า σ^2 มาก
- ถ้า low bias, high variance ให้กำหนดค่า σ^2 น้อย

SVM ในการใช้งาน

- Polynomial Kernel อยู่ในรูป

$$k(x, l) = (x^T l + c)^d$$

*** ถ้าใช้ Polynomial kernel ต้องระวังไม่ให้ชุดข้อมูลมีค่าติดลบ ***

- String Kernel นิยมใช้เมื่ออินพุตเป็น text
- Chi-square Kernel
- Histogram Intersection Kernel

ข้อดี ข้อเสีย

ข้อดี

- ให้ความถูกต้องดีเมื่อเป็น high-dimensional data
- memory efficient

ข้อเสีย

- เกิด overfitting
- ไม่ให้ผลลัพธ์เป็นความน่าจะเป็น
- เหมาะกับข้อมูลขนาดเล็ก

Logistic VS. SVM

n คือ จำนวน feature, m คือ จำนวนชุดข้อมูล (training set)

- ถ้า n ≫ m เช่น n=10,000 , m=10-100

  ให้เลือก logistic หรือ SVM แบบ linear kernel

- ถ้า n น้อย m ก็ปานกลาง เช่น n=1000, m=10,000

  ให้เลือก SVM แบบ Gaussian kernel

- ถ้า n ≪ m มาก เช่น n=1000, m=50K+

  ให้เพิ่ม feature และใช้ logistic หรือ SVM แบบ linear kernel

แอพพลิเคชันที่นิยมใช้ SVM

- Image recognition
- Spam detection
- Sentiment analysis
- Gene classification