

Introduction to Machine Learning

Machine Learning applications:

- Speech conversion from one language to other
- Suspicious activity detection from CCTV
- Medical diagnostics for detecting diseases
- Content (image, video, text) categorization
- Face Detection
- Credit Card Fraud Detection
- Digit Recognition
- Speech Understanding
- Product Recommendation
- Medical Diagnosis
- Stock Trading
- Customer Segmentation
- Shape Detection
- Autonomous driving
- Recommendations that you get when you go on websites like Amazon, Flipkart, Netflix etc
- Google Now / Siri
- Web page classification: various spam and junk pages.
- Entity extraction from web page and queries, like names, addresses.
- Speller correction, running on each queries into Bing.
- Search ranking.
- Facebook Ads ranking: various events prediction.
- Facebook news feed ranking.
- Facebook PYMK (People You Might Know), aka friend suggestions.
- Quora home page feed that you are reading now.
- Quora digest email
- Quora related questions
- AirBnB Search Ranking
- screening large molecule databases and identify which (drug-like) molecules are likely binding to a particular receptor protein
- predict the potency of a receptor agonist or antagonist
- ...

Introduction

What is Machine Learning

Machine Learning definitions:

- “Learning is any process by which a system improves performance from experience.” (Herbert Alexander Simon)
- “Machine Learning is concerned with computer programs that automatically improve their performance through experience.” (Herbert Alexander Simon)
- “Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.” (Arthur Samuel)
- “A computer program is said to learn from experience (E) with some class of tasks (T) and a performance measure (P) if its performance at tasks in T as measured by P improves with E” (Tom Mitchell)

Machine Learning:

Learning = Improving with experience at some task

- Improve over task T,
- With respect to performance measure, P
- Based on experience, E.

An example:

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.

What is the task T in this setting? The experience E? the performance P?

- Classifying emails as spam or not spam.
- Watching you label emails as spam or not spam.
- The number (or fraction) of emails correctly classified as spam/not spam.
- None of the above—this is not a machine learning problem

An example:

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.

What is the task T in this setting? The experience E? the performance P?

T

Classifying emails as spam or not spam.

Watching you label emails as spam or not spam.

The number (or fraction) of emails correctly classified as spam/not spam.

None of the above—this is not a machine learning problem

An example:

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.

What is the task T in this setting? The experience E? the performance P?

T

Classifying emails as spam or not spam.

E

Watching you label emails as spam or not spam.

The number (or fraction) of emails correctly classified as spam/not spam.

None of the above—this is not a machine learning problem

An example:

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.

What is the task T in this setting? The experience E? the performance P?

T

Classifying emails as spam or not spam.

E

Watching you label emails as spam or not spam.

P

The number (or fraction) of emails correctly classified as spam/not spam.

None of the above—this is not a machine learning problem

Machine Learning classification 1:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning
- Recommender Systems

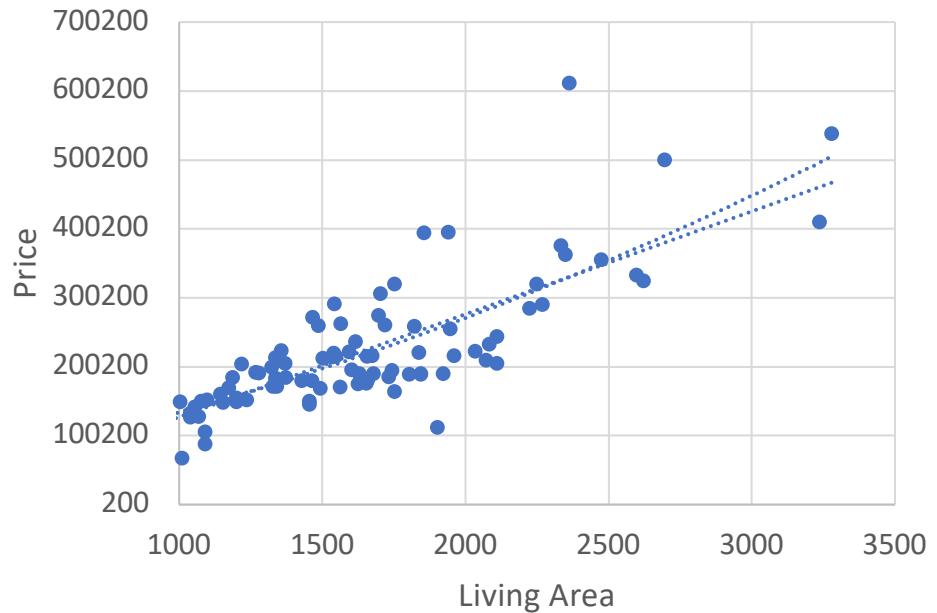
....

Supervised Learning

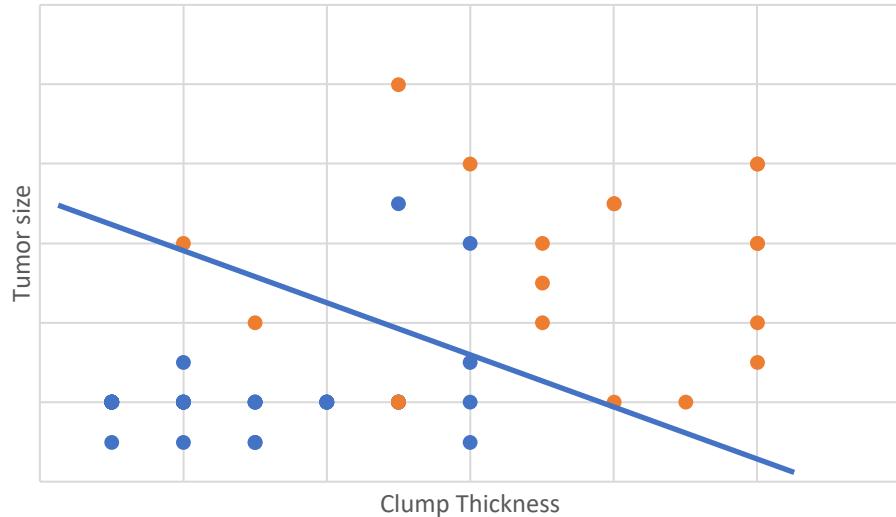
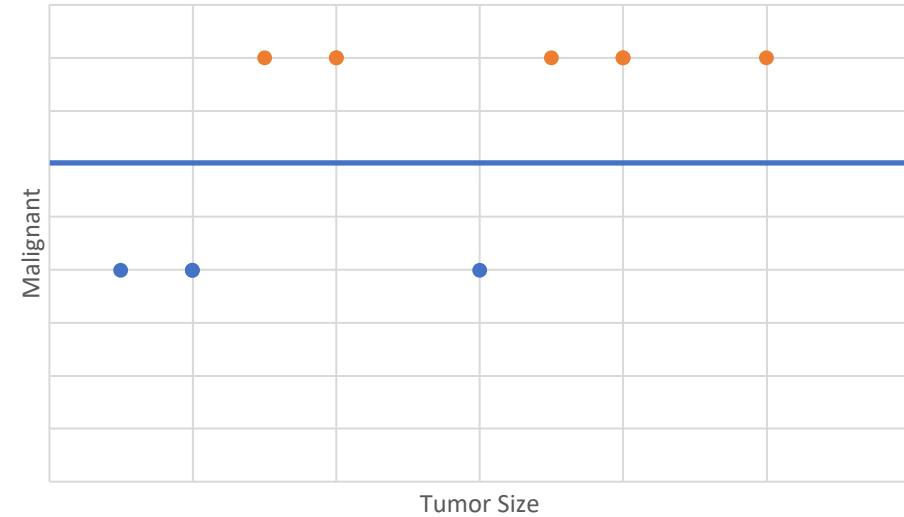
Housing price prediction in Ames (Iowa)

- “right answers” given
- Regression: Predict continuous valued output (price)

Living area (feet ²)	Price (1000\$)
1656	215
896	105
1329	172
2110	244
...	...



Supervised Learning



Regression or classification?

Problem 1: You want to analize all your emails and to predict whether an email is a *spam* and should be delivered to the Junk folder.

Problem 2: You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months.

Should you treat these problems as classification or as regression problems?

- Treat both as classification problems.
- Treat problem 1 as a classification problem, problem 2 as a regression problem.
- Treat problem 1 as a regression problem, problem 2 as a classification problem.
- Treat both as regression problems.

Regression or classification?

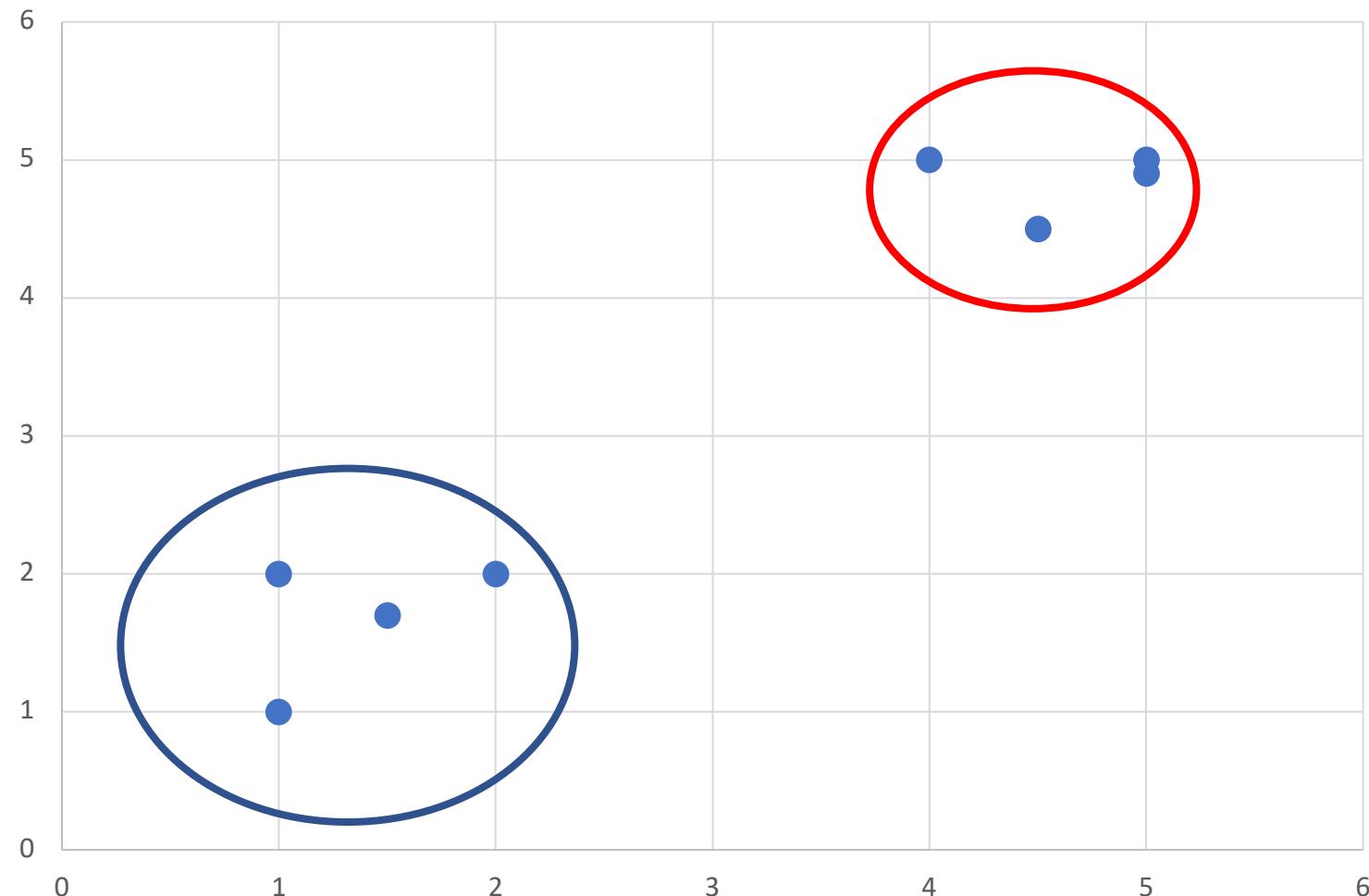
Problem 1: You want to analize all your emails and to predict whether an email is a *spam* and should be delivered to the Junk folder.

Problem 2: You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months.

Should you treat these problems as classification or as regression problems?

- Treat both as classification problems.
- Treat problem 1 as a classification problem, problem 2 as a regression problem.
- Treat problem 1 as a regression problem, problem 2 as a classification problem.
- Treat both as regression problems.

Unsupervised Learning

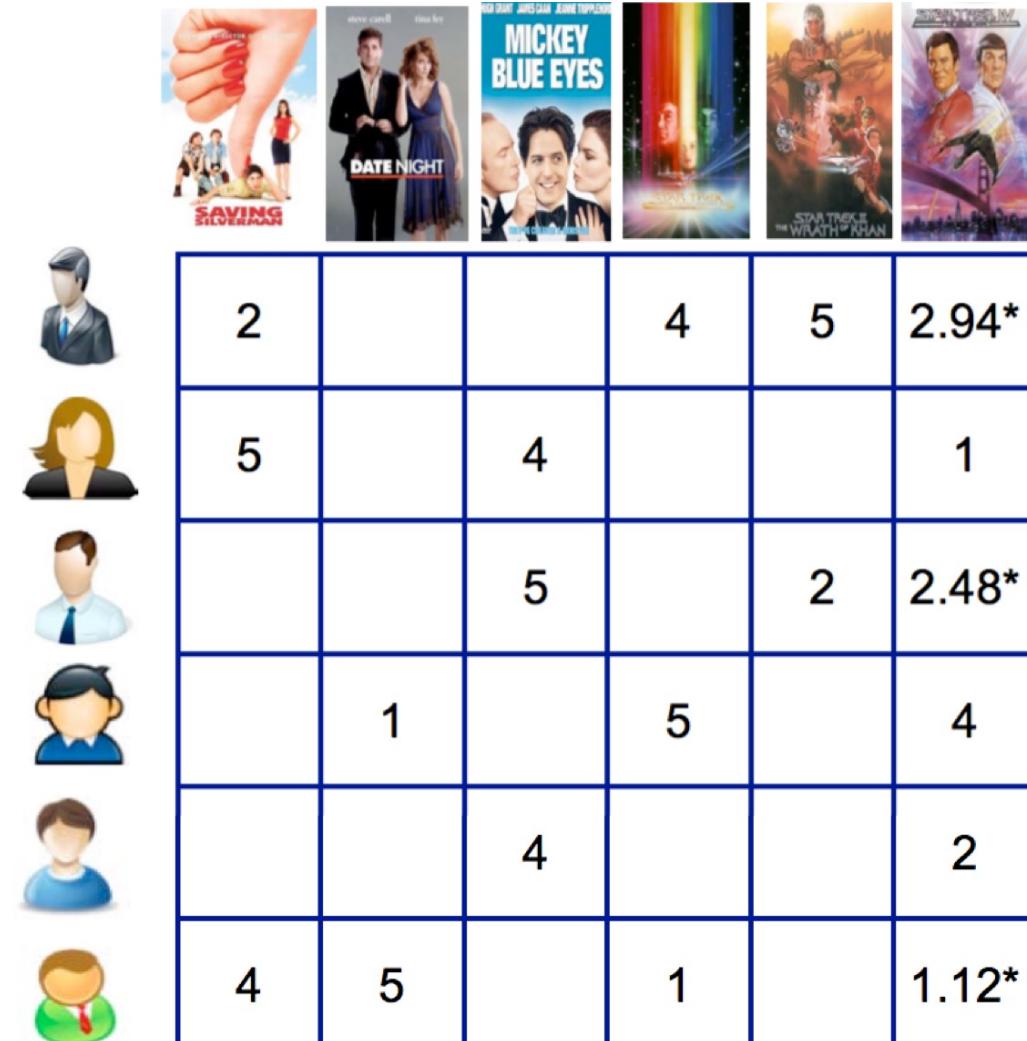


Reinforcement Learning

Programmers incentivised it
to go from point A to point B



Recommender Systems



Machine Learning classification 2:

- Discriminative models
- Generative models

Discriminative Models

These models are trained over a training set.

When these models take an input, they estimate the most probable output. The purpose is to estimate the conditional probability $p(y|x)$.

E.g.

Housing price prediction

Image classification

Text classification

Speech recognition

Machine Translation

Genes

Topic modeling

Generative models

The purpose is to estimate the joint probability $p(x, y)$.

These are probabilistic models that produce both input and output. After the model is trained, the conditional probability can be inferred.

E.g.

Deep fake

Text 2 Image

ChatGPT

Music Generation

Supervised Learning

Linear regression with one variable

Model Representation

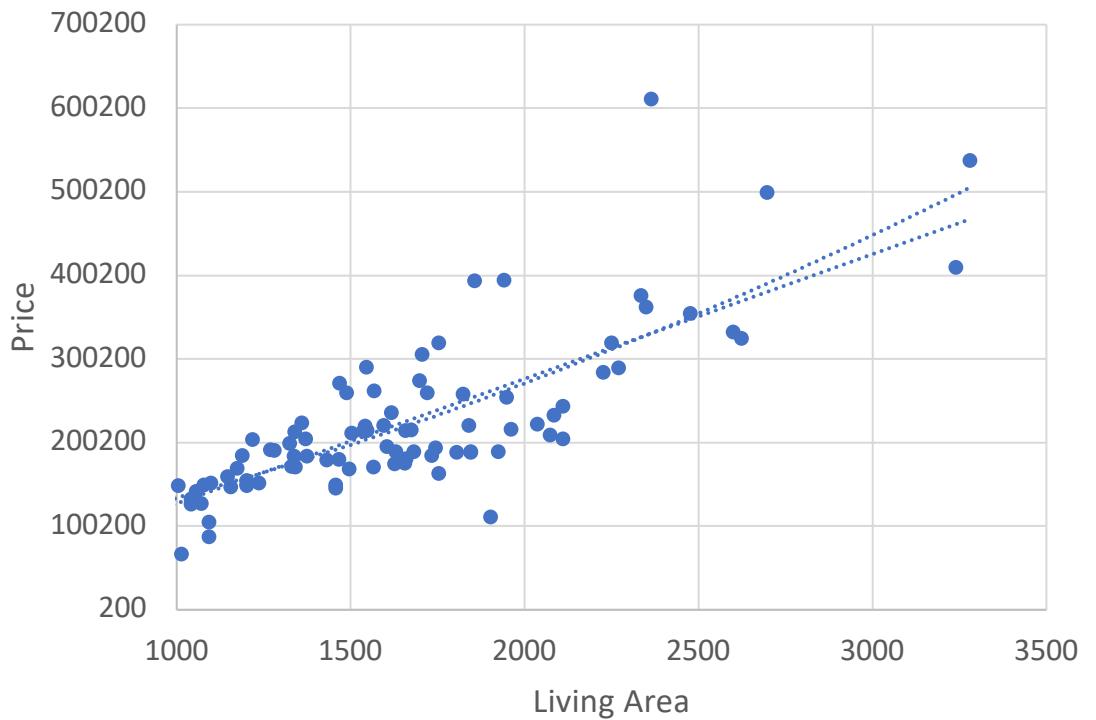
Housing prediction problem in Ames

Which kind of data we have?

- Actual answers from real data examples

What we want to predict?

- A real valued output



Housing prediction problem in Ames

Training set of housing prices

Notation:

m = Number of training examples

x = “input” variable / feature

y = “output” variable / “target” variable

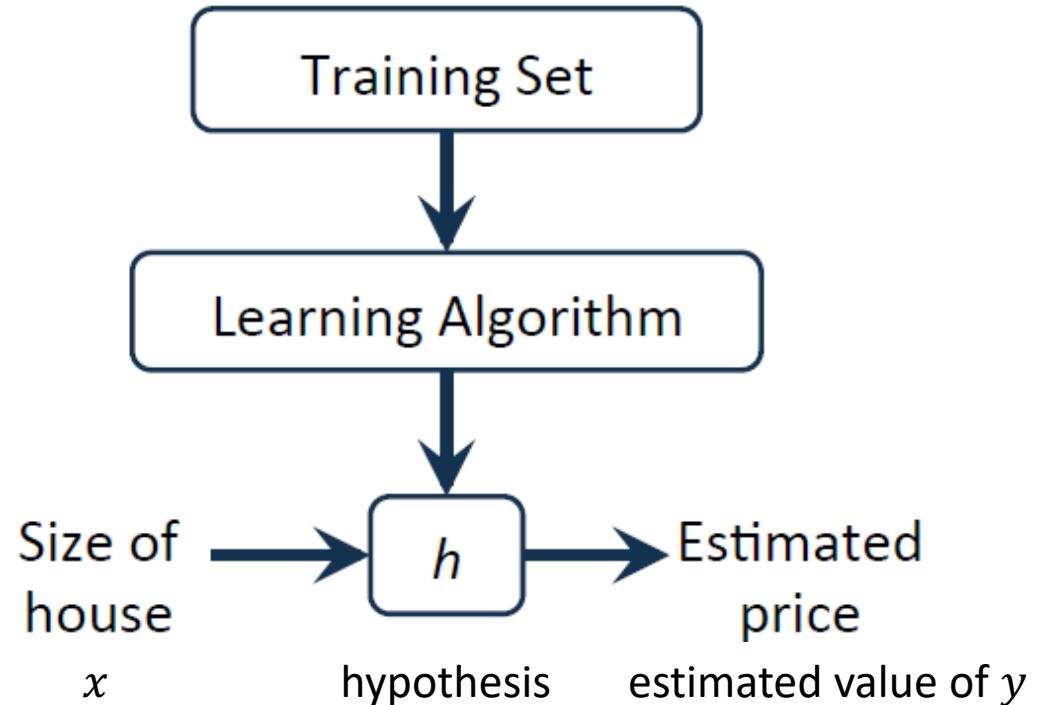
Living area (feet2)	Price (1000\$s)
1656	215
896	105
1329	172
2110	244
...	...

(x, y) = tuple representing one training example

$(x^{(i)}, y^{(i)})$ = i^{th} training example

$x^{(1)}$	1656
$x^{(2)}$	896
$y^{(1)}$	215

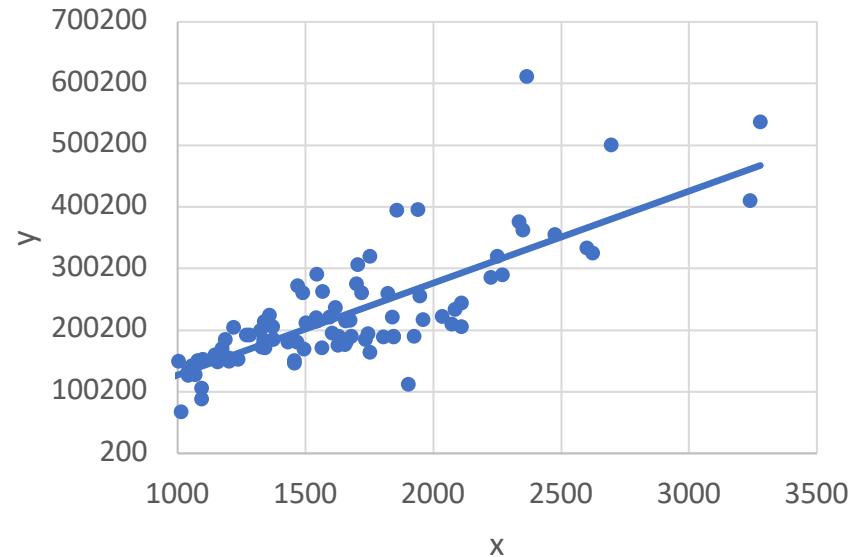
What we are going to realize



We want to realize a function that, given a value of x , it will return the estimated value of y .

How h could be?

$h_{\theta}(x) = \theta_0 + \theta_1 x$ Usually abbreviated as $h(x)$



Linear regression with one variable
(Univariate linear regression)

Linear regression with one variable

Cost function

Linear hypothesis

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

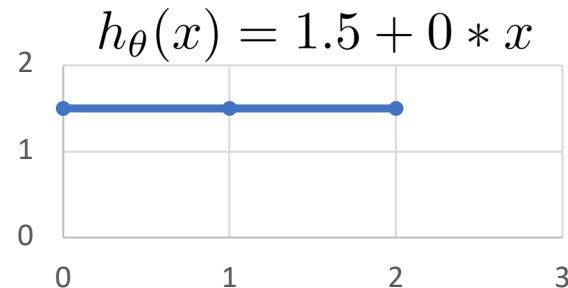
θ_i : parameters/weights

How θ_i can be chosen?

Living area (feet ²)	Price (1000\$)
1656	215
896	105
1329	172
2110	244
...	...

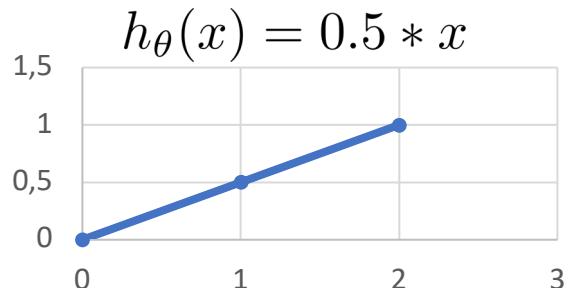
Univariate linear regression

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



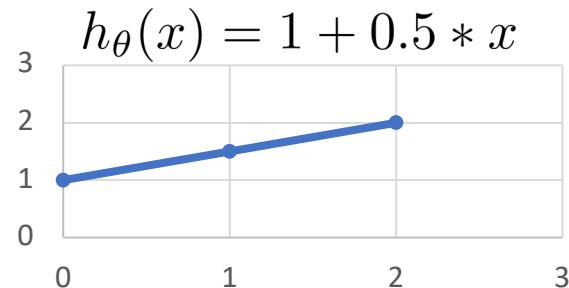
$$\theta_0 = 1.5$$

$$\theta_1 = 0$$



$$\theta_0 = 0$$

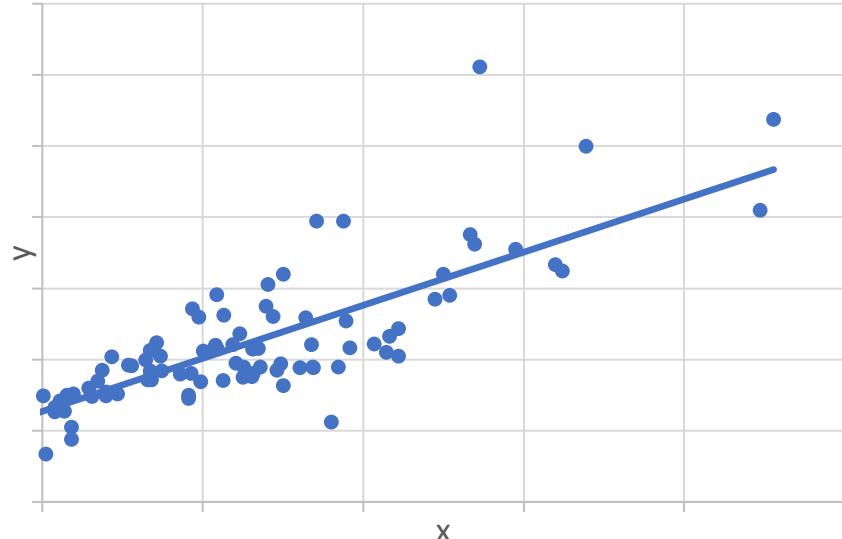
$$\theta_1 = 0.5$$



$$\theta_0 = 1$$

$$\theta_1 = 0.5$$

Cost Function: Intuition



We can choose the **parameters**
so the **hypothesis** is as **close** as possible
to y for our **training examples**

$$\underset{\theta_0, \theta_1}{\operatorname{argmin}} \frac{1}{2m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2$$

#training examples Squared error function
 ↓ ↓

$$h_\theta(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

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

(CostFunction)

$$\theta_{min} = \underset{\theta}{\operatorname{argmin}}(J(\theta_0, \theta_1)) = \underset{\theta}{\operatorname{argmin}}(J(\theta)).$$

Linear regression with one variable

Cost function Intuition I

Cost Function Example

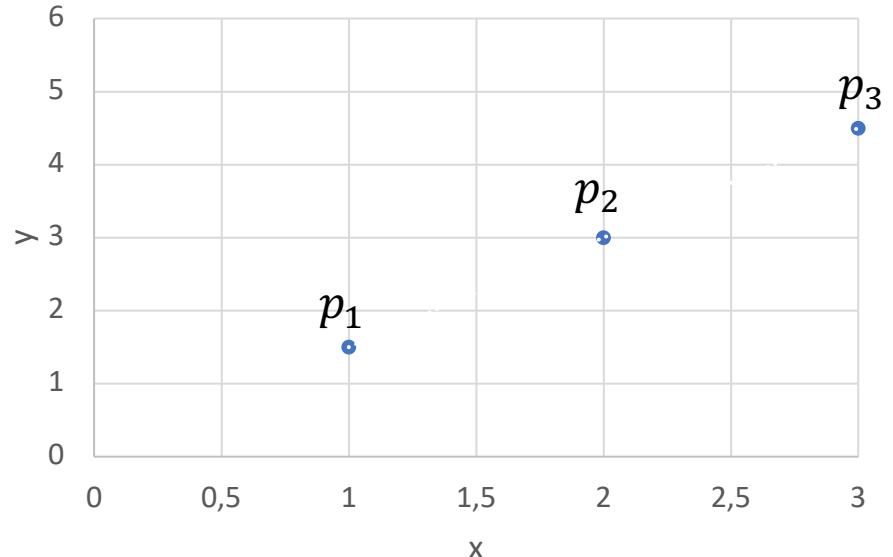
$$p_1 = (1, 1.5)$$

$$p_2 = (2, 3)$$

$$p_3 = (3, 4.5)$$

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

$$J(\theta) = \frac{1}{2 \cdot 3} ((\theta_0 + 1 \cdot \theta_1 - 1.5)^2 + (\theta_0 + 2 \cdot \theta_1 - 3)^2 + (\theta_0 + 3 \cdot \theta_1 - 4.5)^2)$$



Cost Function Example

$$p_1 = (1, 1.5)$$

$$p_2 = (2, 3)$$

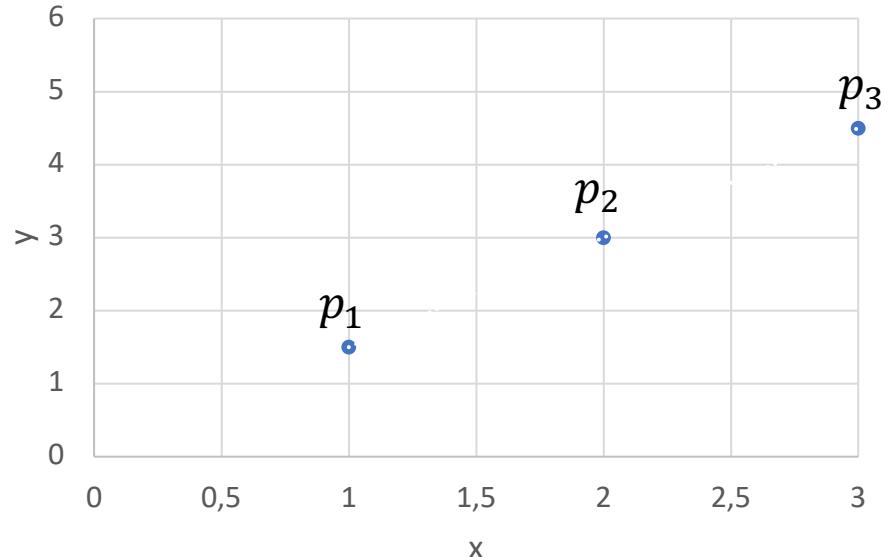
$$p_3 = (3, 4.5)$$

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

$$J(\theta) = \frac{1}{2 \cdot 3} ((\theta_0 + 1 \cdot \theta_1 - 1.5)^2 + (\theta_0 + 2 \cdot \theta_1 - 3)^2 + (\theta_0 + 3 \cdot \theta_1 - 4.5)^2)$$

$h_\theta(x^{(1)})$

m $x^{(1)}$ $y^{(1)}$



Univariate Linear regression

Hypothesis:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal:

$$\underset{\theta_0, \theta_1}{\text{minimize}}(J(\theta_0, \theta_1))$$

Simplified toy example

Hypothesis:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_1 x^{(i)} \quad (\theta_0 = 0)$$

Parameters:

$$\theta_1$$

Cost Function:

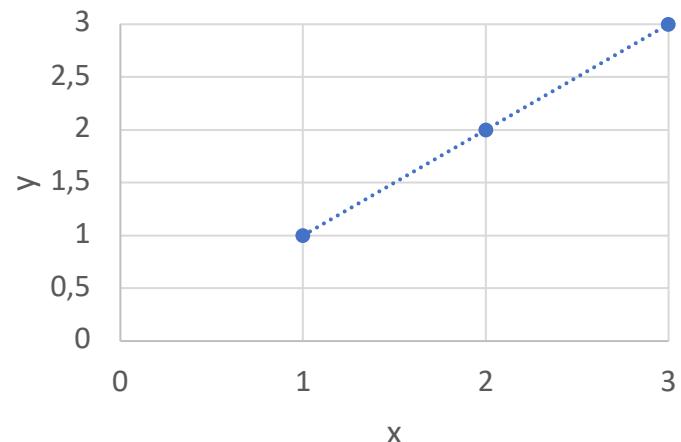
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{\theta_1}{\text{minimize}}(J(\theta_1))$$

$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)

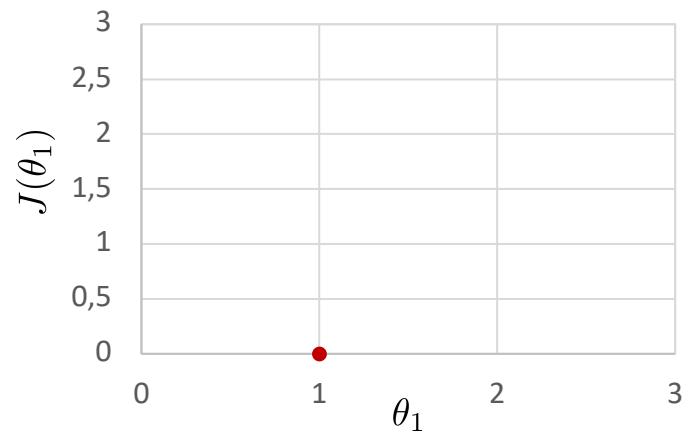


$$\theta_1 = 1$$

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

$$J(\theta_1)$$

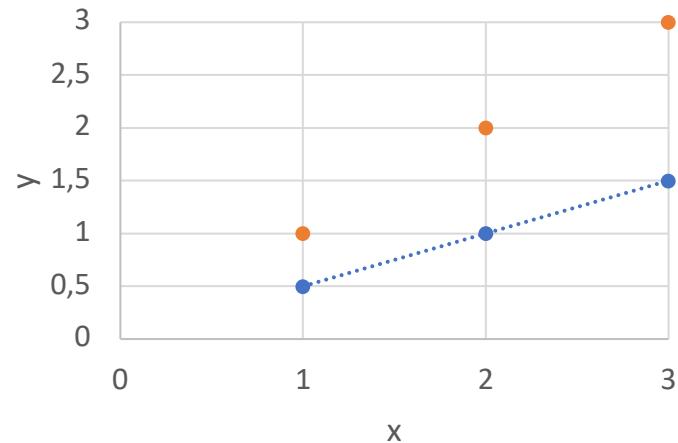
(function of θ_1 , an aggregate function over all x 's)



$$\begin{aligned}
 J(\theta_1) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (\theta_1 x^{(i)} - y^{(i)})^2 \\
 &= \frac{1}{2 * 3} [0^2 + 0^2 + 0^2] \\
 J(1) &= 0
 \end{aligned}$$

$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)

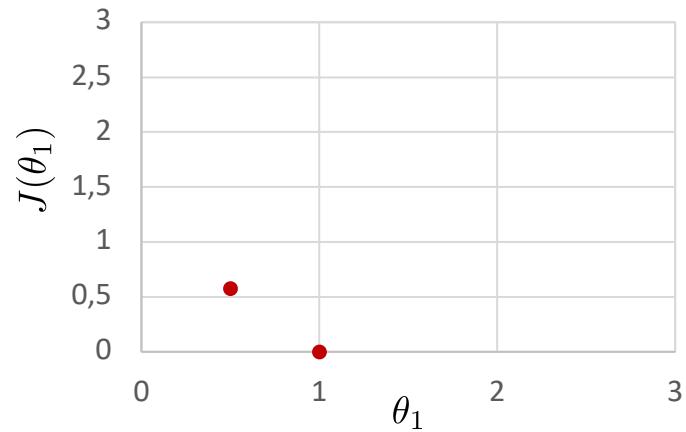


$$\theta_1 = 0.5$$

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

$$J(\theta_1)$$

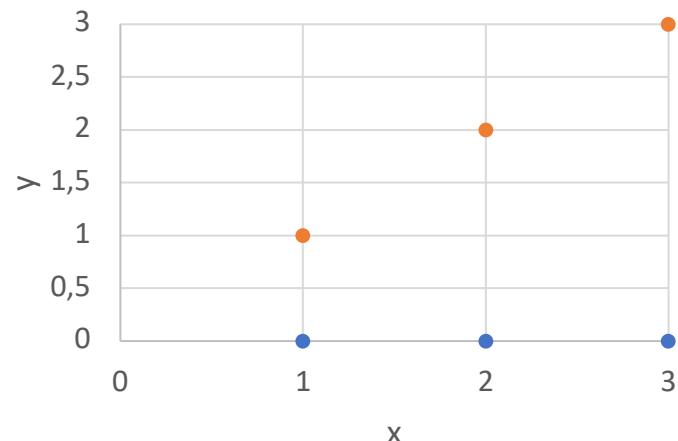
(function of θ_1 , an aggregate function over all x 's)



$$\begin{aligned}
 J(\theta_1) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (\theta_1 x^{(i)} - y^{(i)})^2 \\
 &= \frac{1}{2 * 3} [(0.5 - 1)^2 + (1 - 2)^2 + (1.5 - 3)^2] \\
 J(0.5) &= \frac{3.5}{6} = 0.58
 \end{aligned}$$

$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)

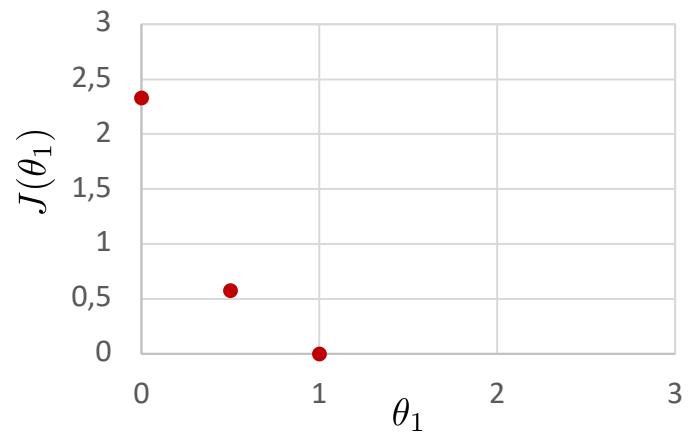


$$\theta_1 = 0$$

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

$$J(\theta_1)$$

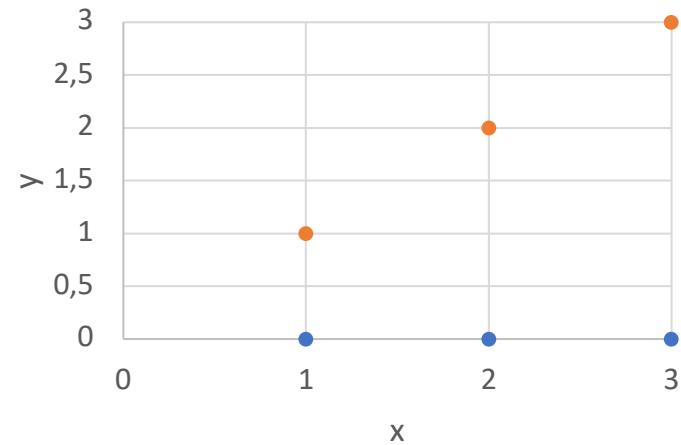
(function of θ_1 , an aggregate function over all x 's)



$$\begin{aligned}
 J(\theta_1) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (\theta_1 x^{(i)} - y^{(i)})^2 \\
 &= \frac{1}{2 * 3} [(0 - 1)^2 + (0 - 2)^2 + (0 - 3)^2] \\
 J(0) &= \frac{14}{6} = 2.33
 \end{aligned}$$

$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)

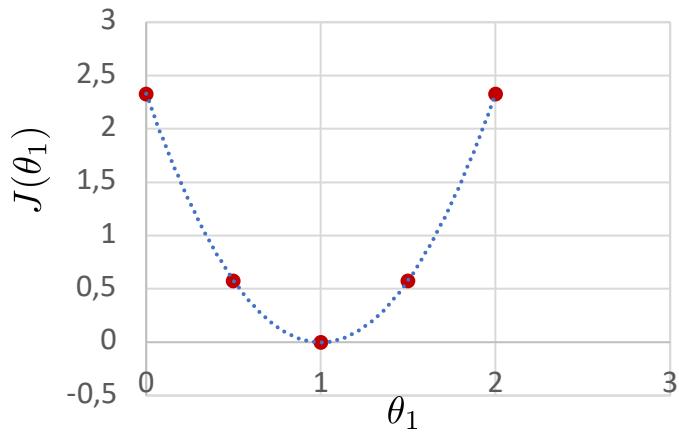


$$\theta_1 = 0$$

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

$$J(\theta_1)$$

(function of θ_1 , an aggregate function over all x 's)



$$\underset{\theta_1}{\text{minimize}}(J(\theta_1))$$

Linear regression with one variable

Cost function Intuition II

Recap

Univariate Linear regression

Hypothesis:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

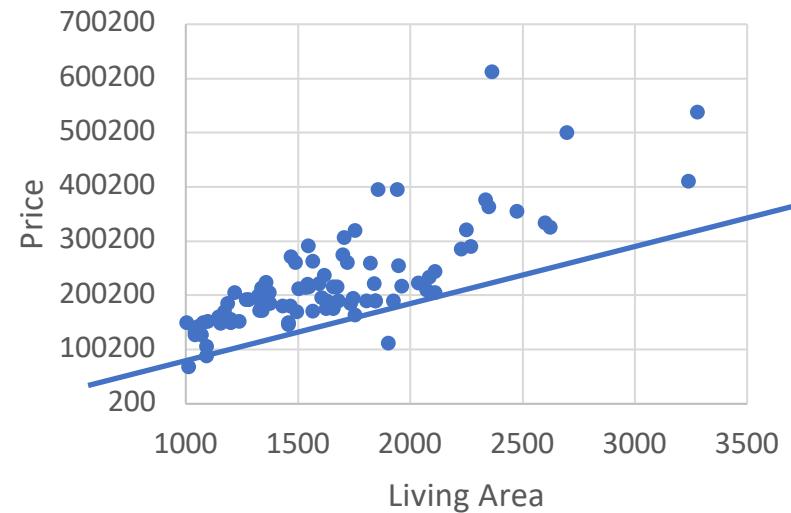
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{\theta_0, \theta_1}{\text{minimize}}(J(\theta_0, \theta_1))$$

$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)



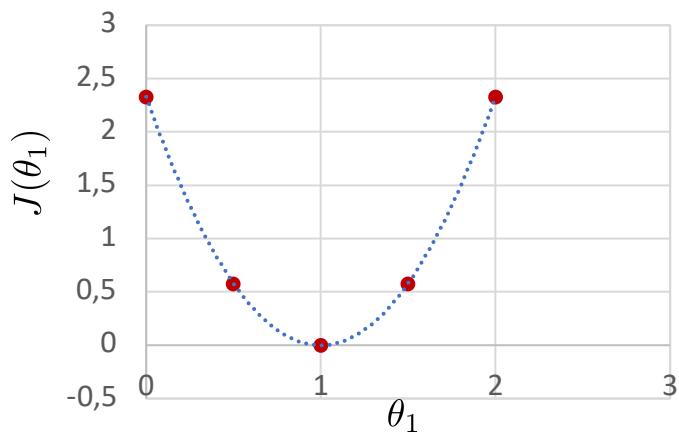
$$\theta_0 = 100000$$

$$\theta_1 = 66$$

$$h_{\theta}(x^{(i)}) = 100000 + 66x$$

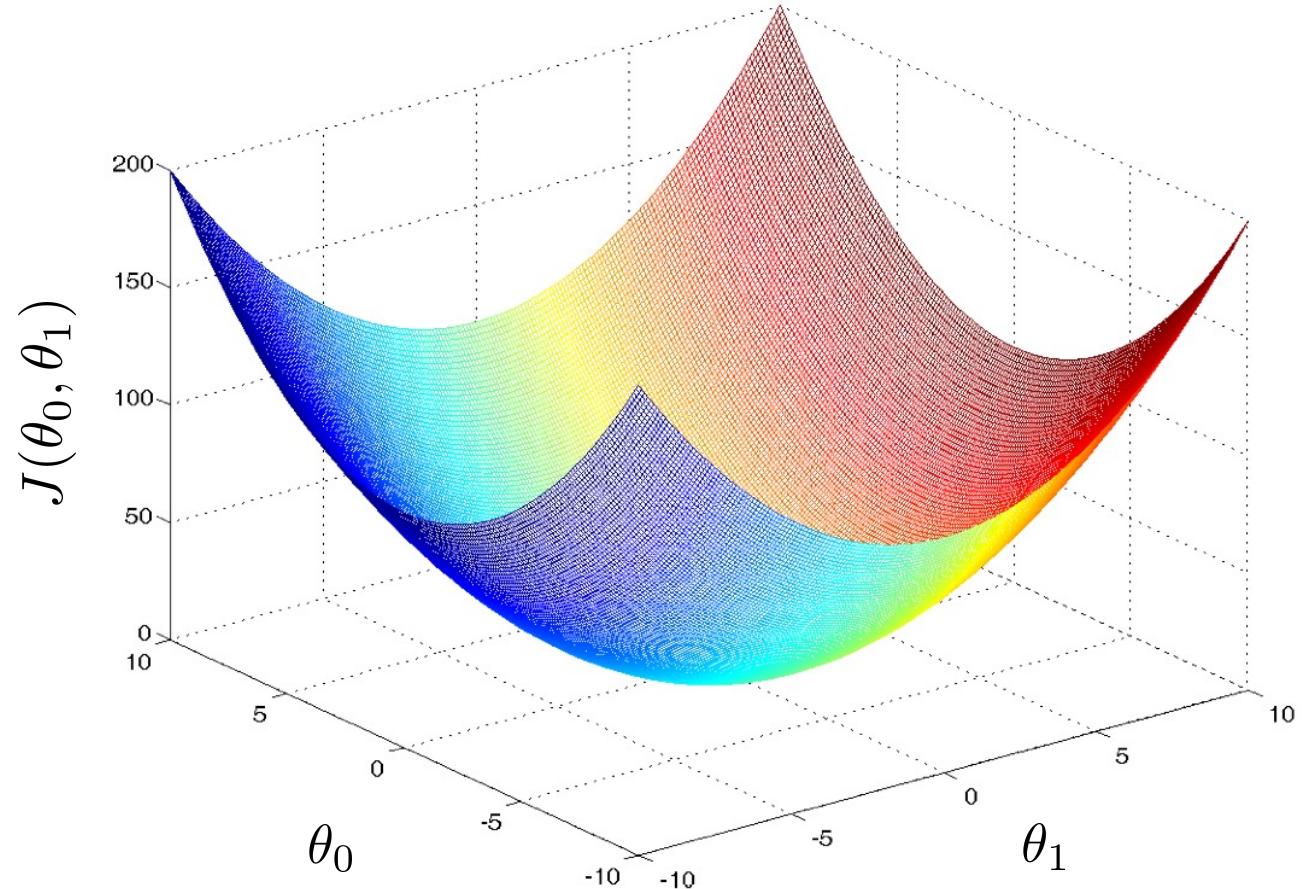
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



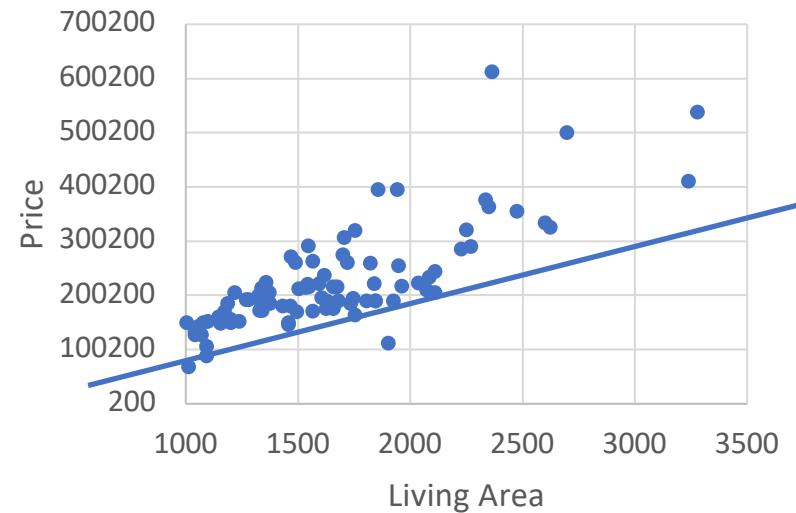
$$\underset{\theta_0, \theta_1}{\text{minimize}}(J(\theta_0, \theta_1))$$

$J(\theta_0, \theta_1)$: contour plot



$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)



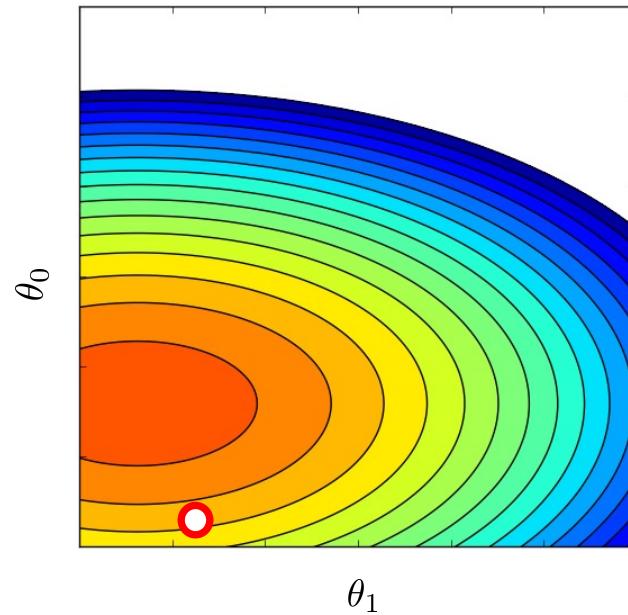
$$\theta_0 = 100000$$

$$\theta_1 = 66$$

$$h_{\theta}(x^{(i)}) = 100000 + 66x$$

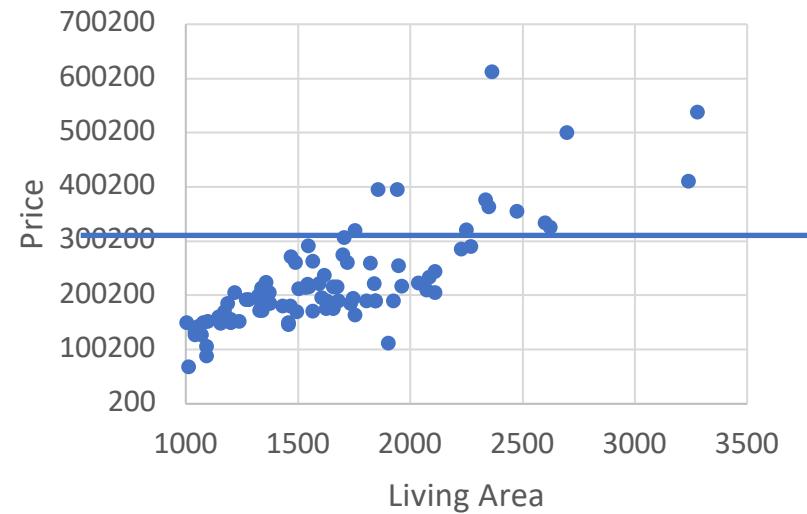
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)



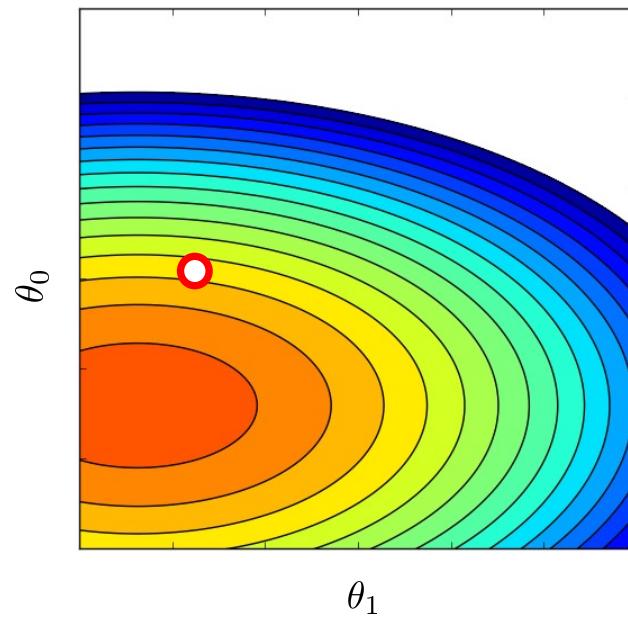
$$\theta_0 = 300200$$

$$\theta_1 = 0$$

$$h_{\theta}(x^{(i)}) = 300200 + 0x$$

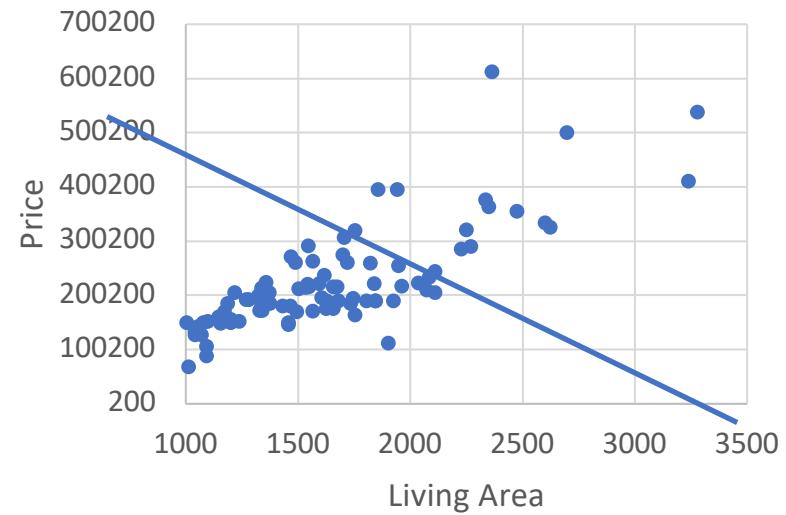
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)



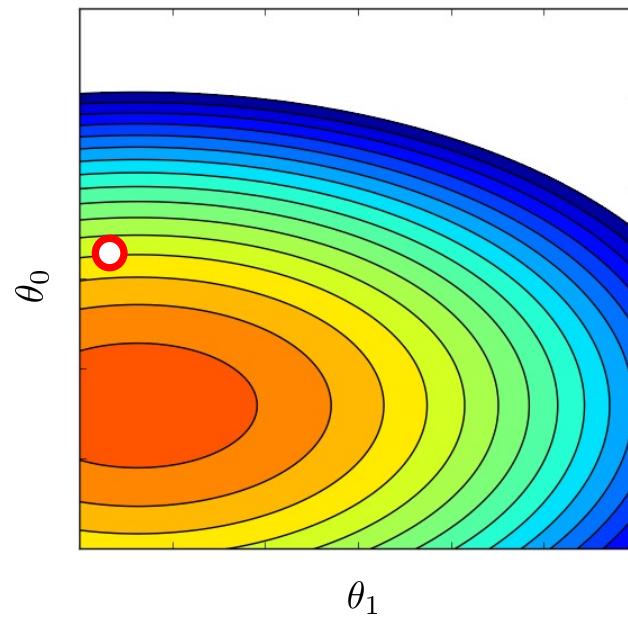
$$\theta_0 = 600000$$

$$\theta_1 = -90$$

$$h_{\theta}(x^{(i)}) = 600000 - 90x$$

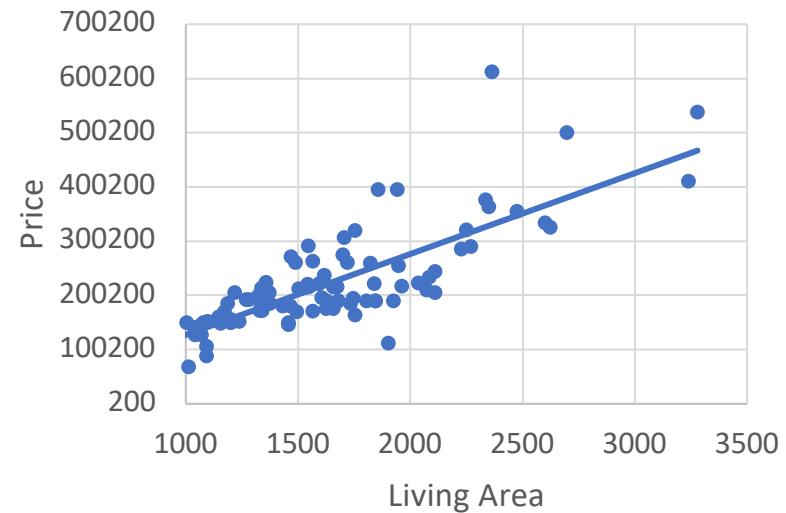
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



$$h_{\theta}(x)$$

(function of x , with θ_1 fixed)



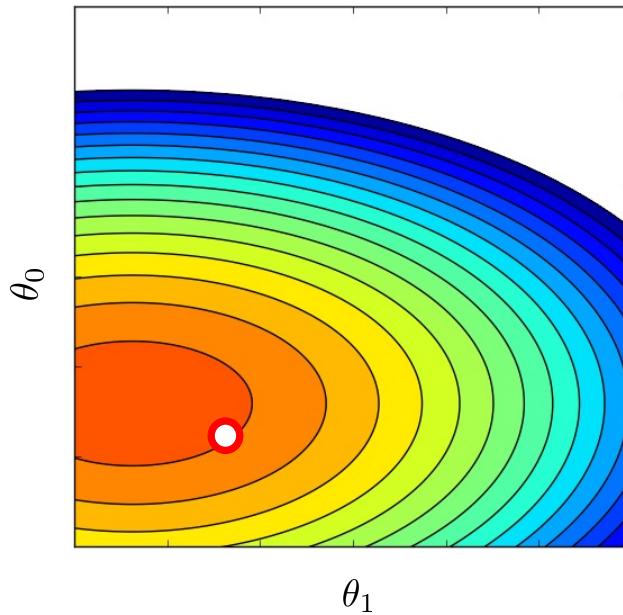
$$\theta_0 = 200000$$

$$\theta_1 = 70$$

$$h_{\theta}(x^{(i)}) = 200000 + 70x$$

$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



Linear regression with one variable

Gradient Descent

Cost function (we will deal with it as a generic function)

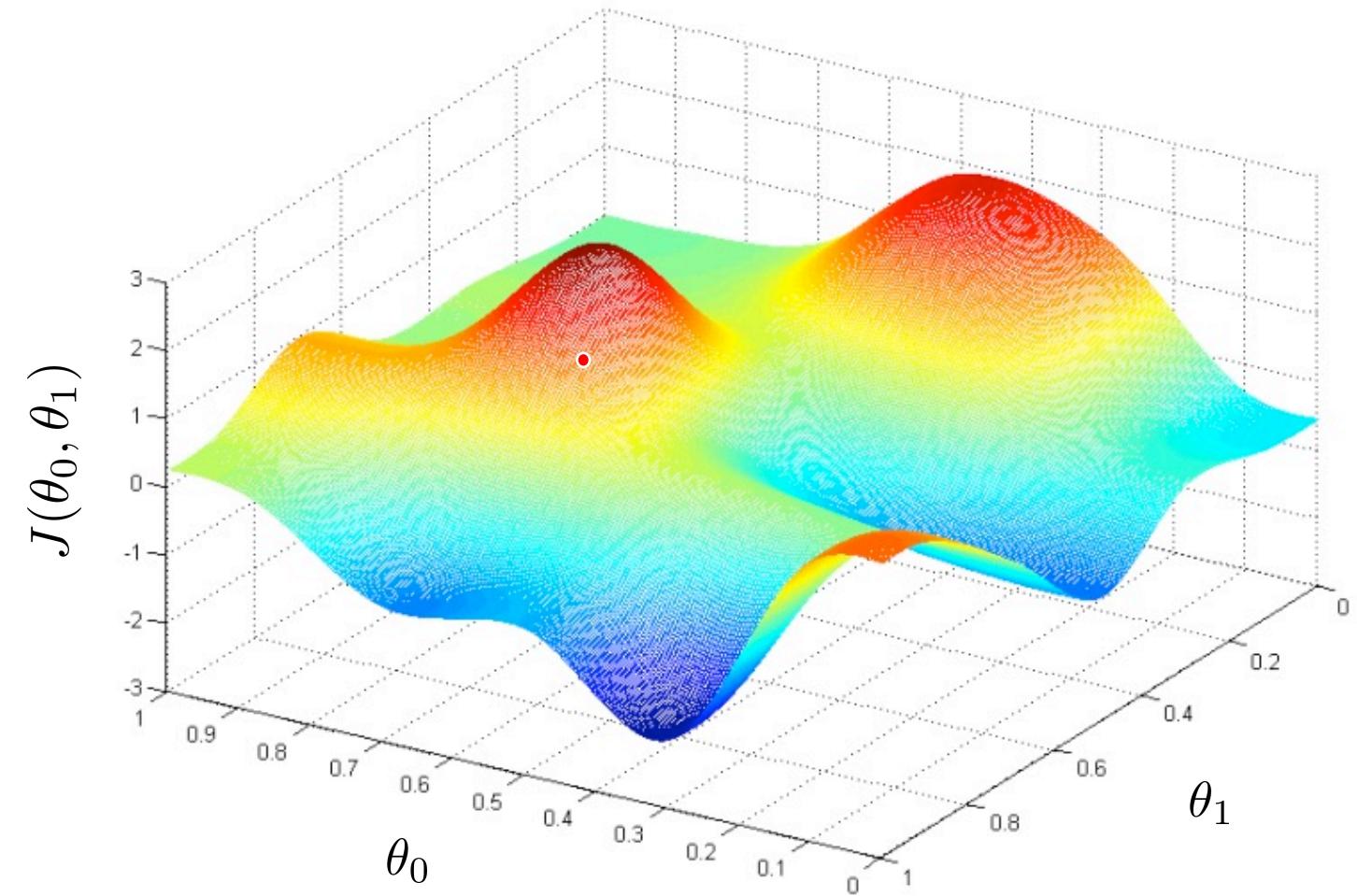
$$J(\theta_0, \theta_1)$$

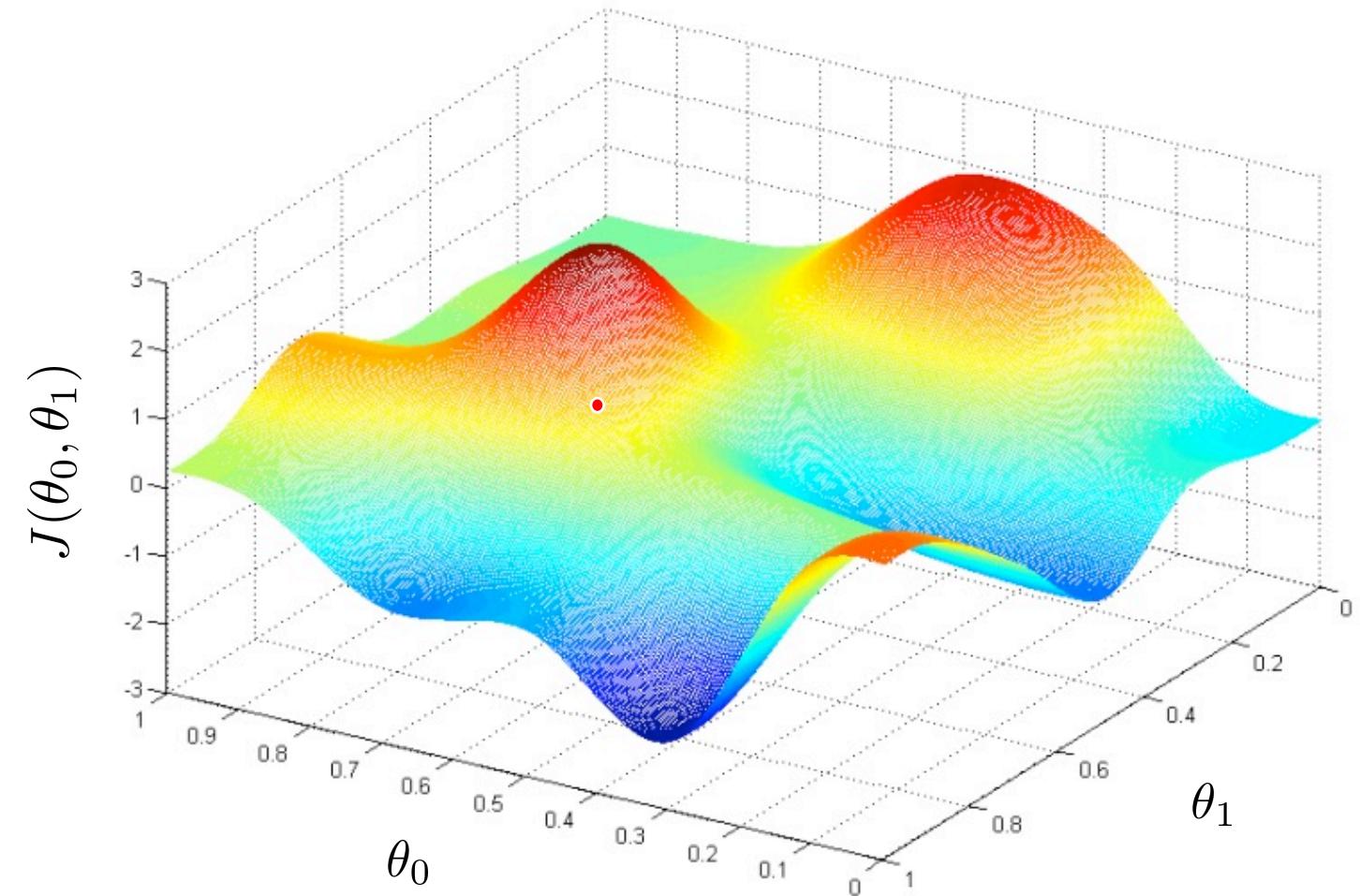
$$J(\theta_0, \theta_1, \dots, \theta_n)$$

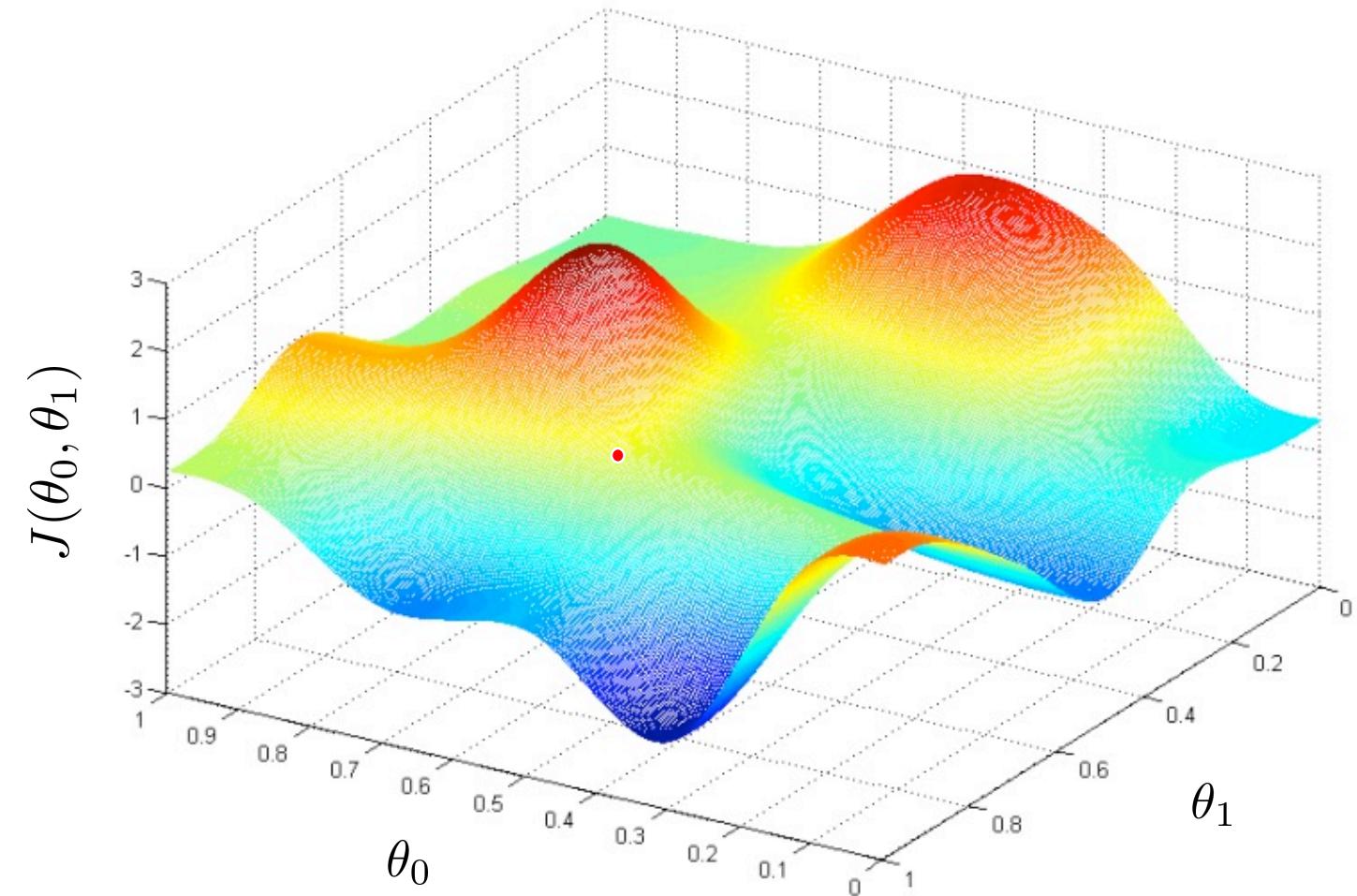
Goal $\min_{\theta_0, \theta_1} (J(\theta_0, \theta_1))$ $\min_{\theta_0, \theta_1, \dots, \theta_n} (J(\theta_0, \theta_1, \dots, \theta_n))$

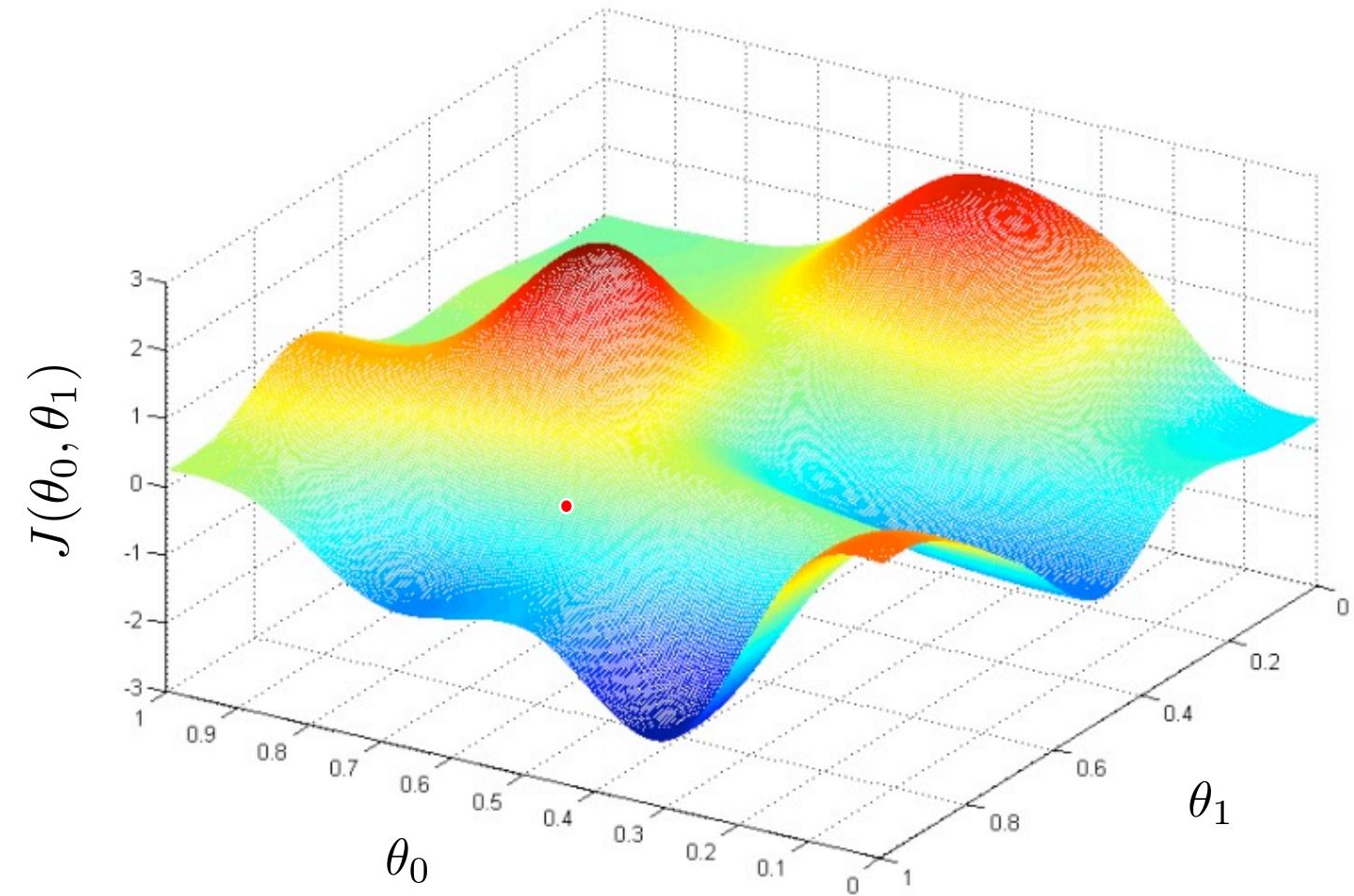
How to:

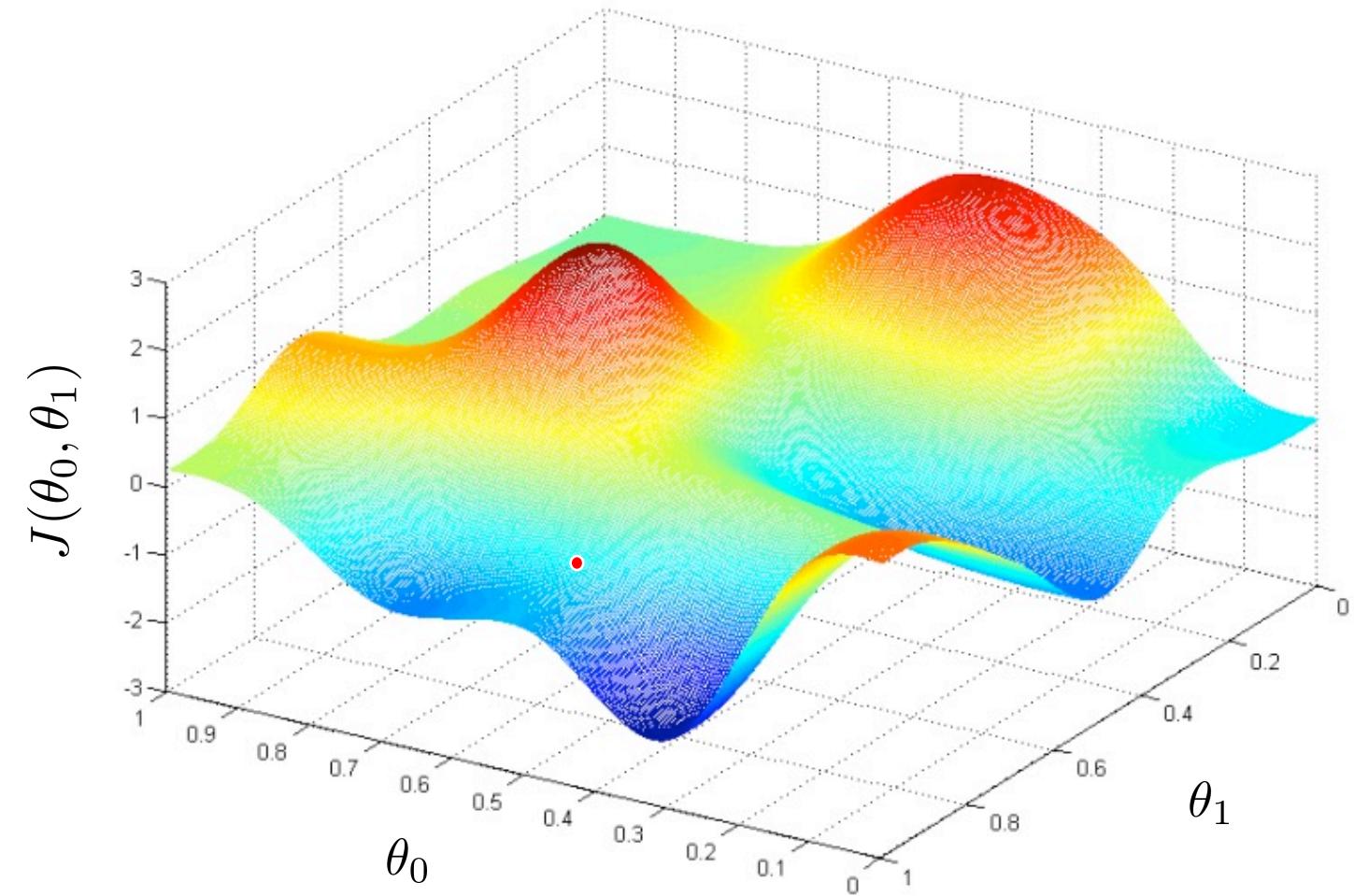
- Initialize θ_0, θ_1 to some values
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ to get closer and reach the minimum

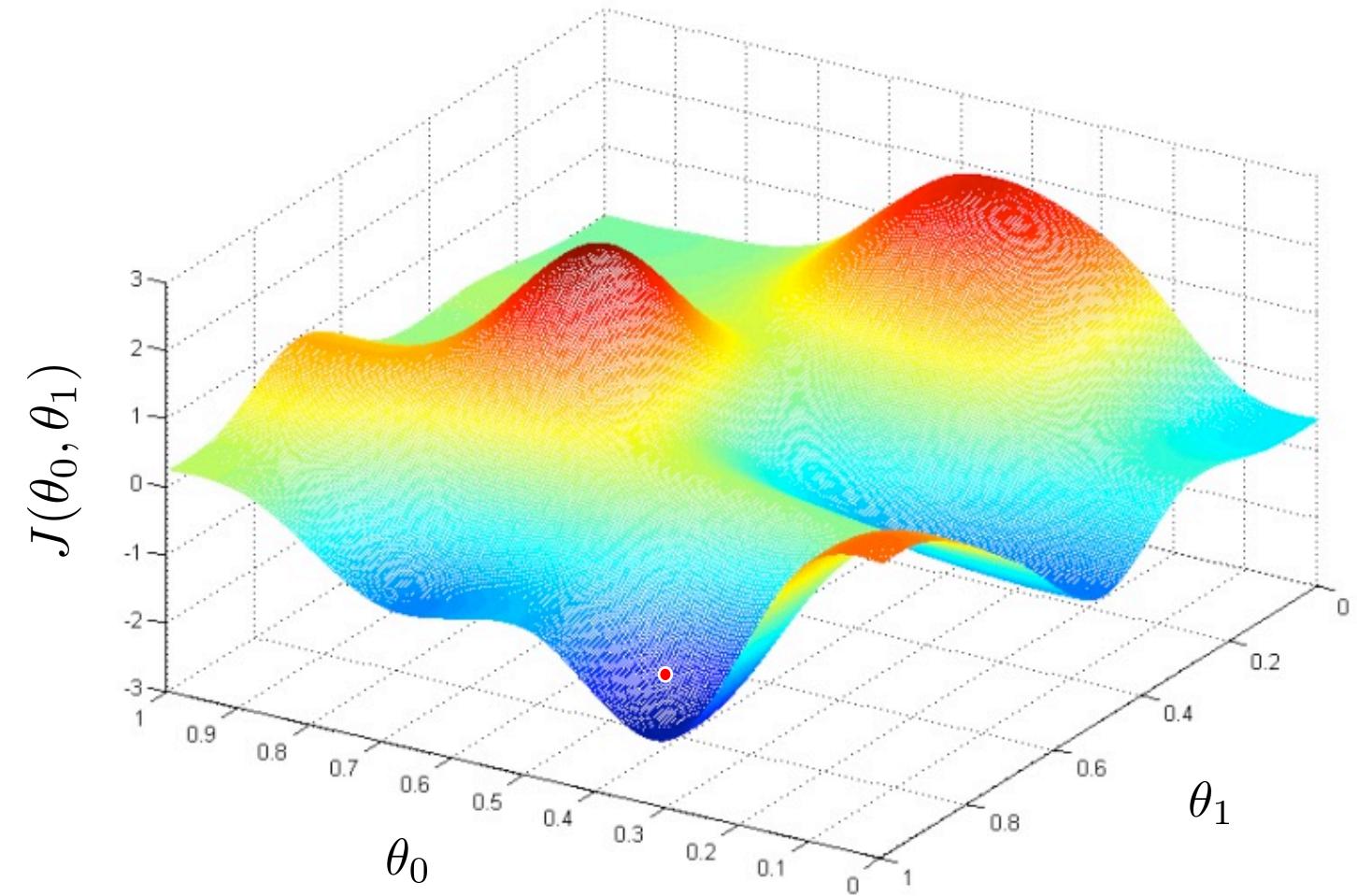


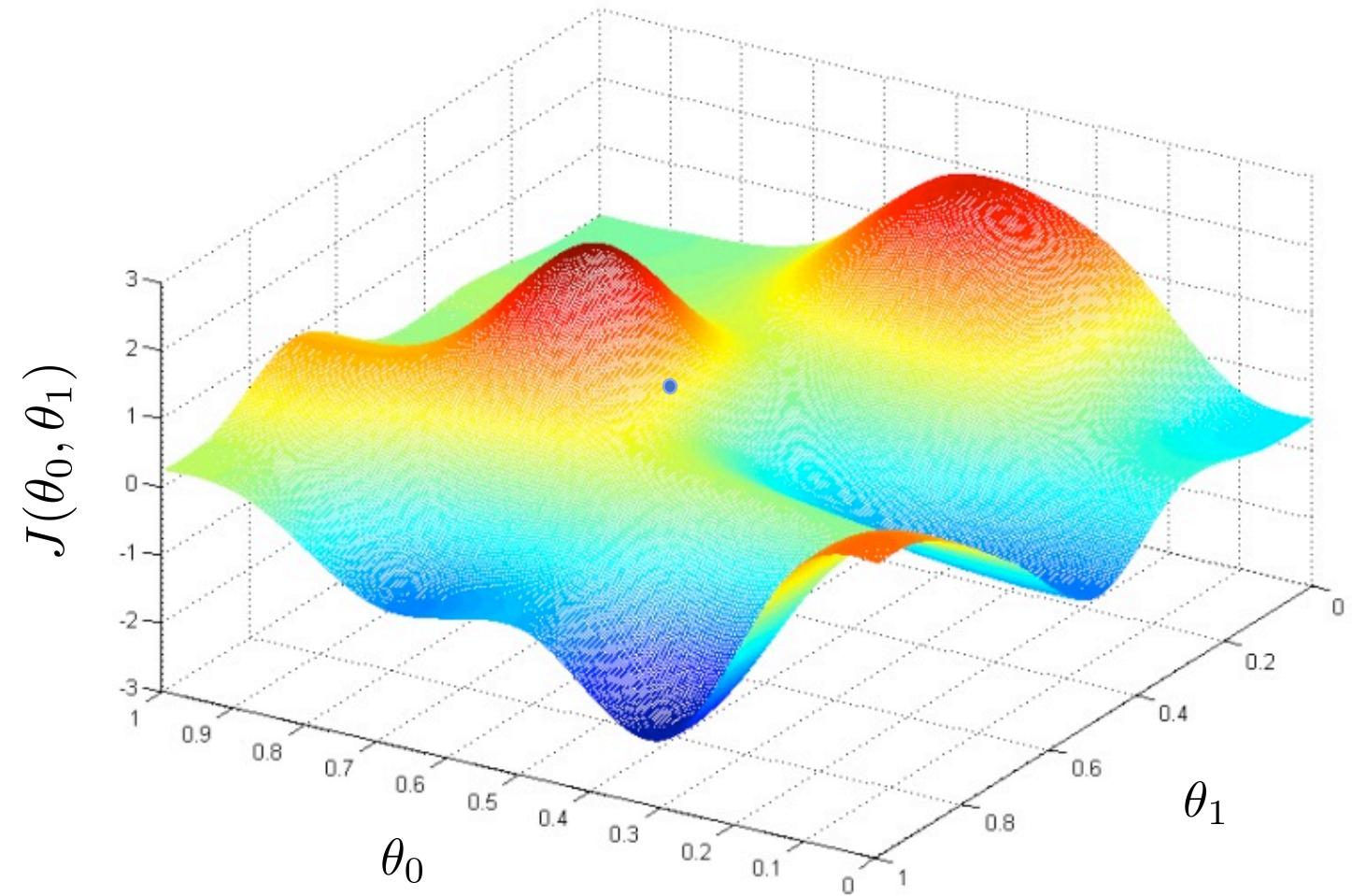


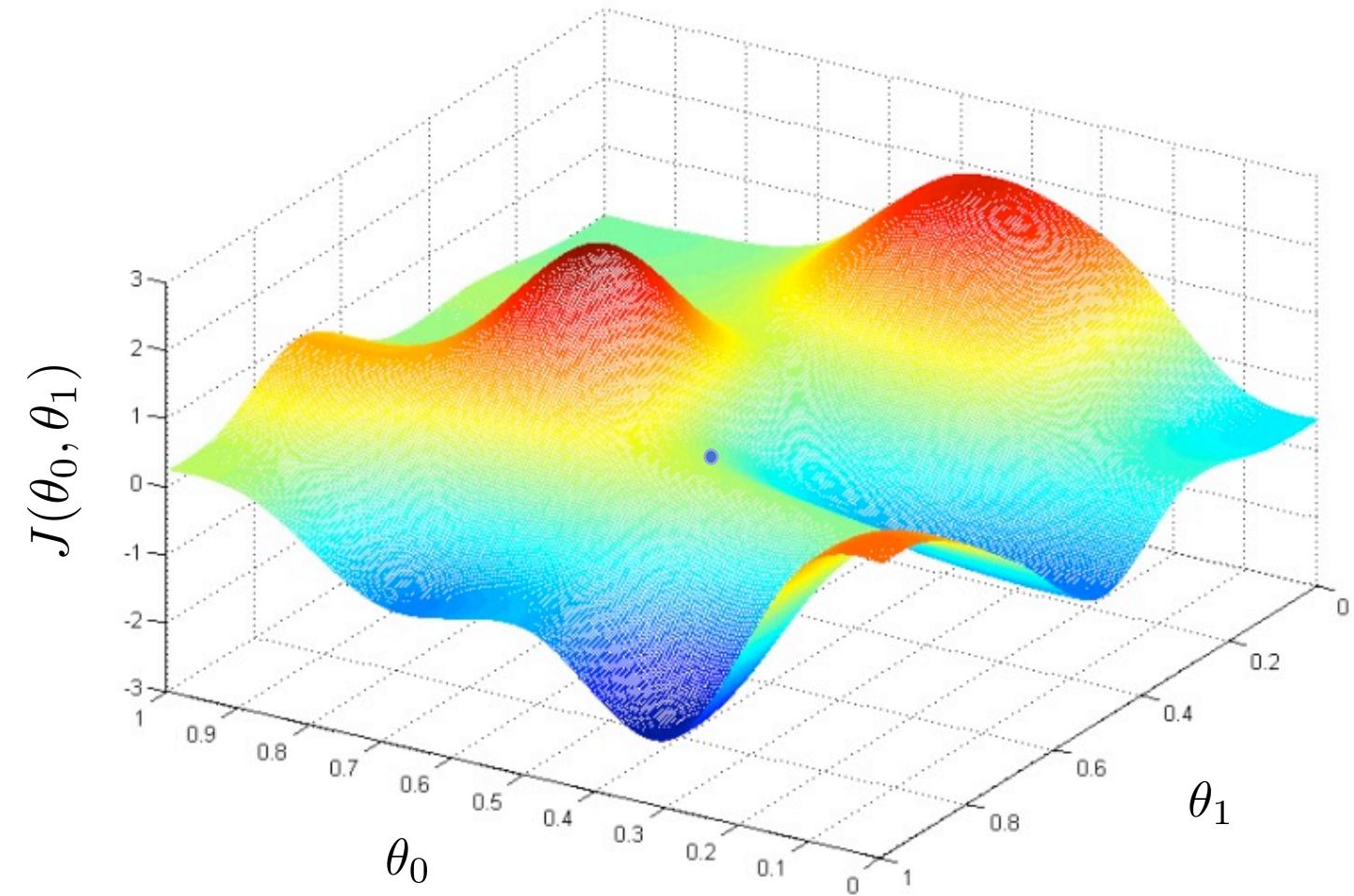


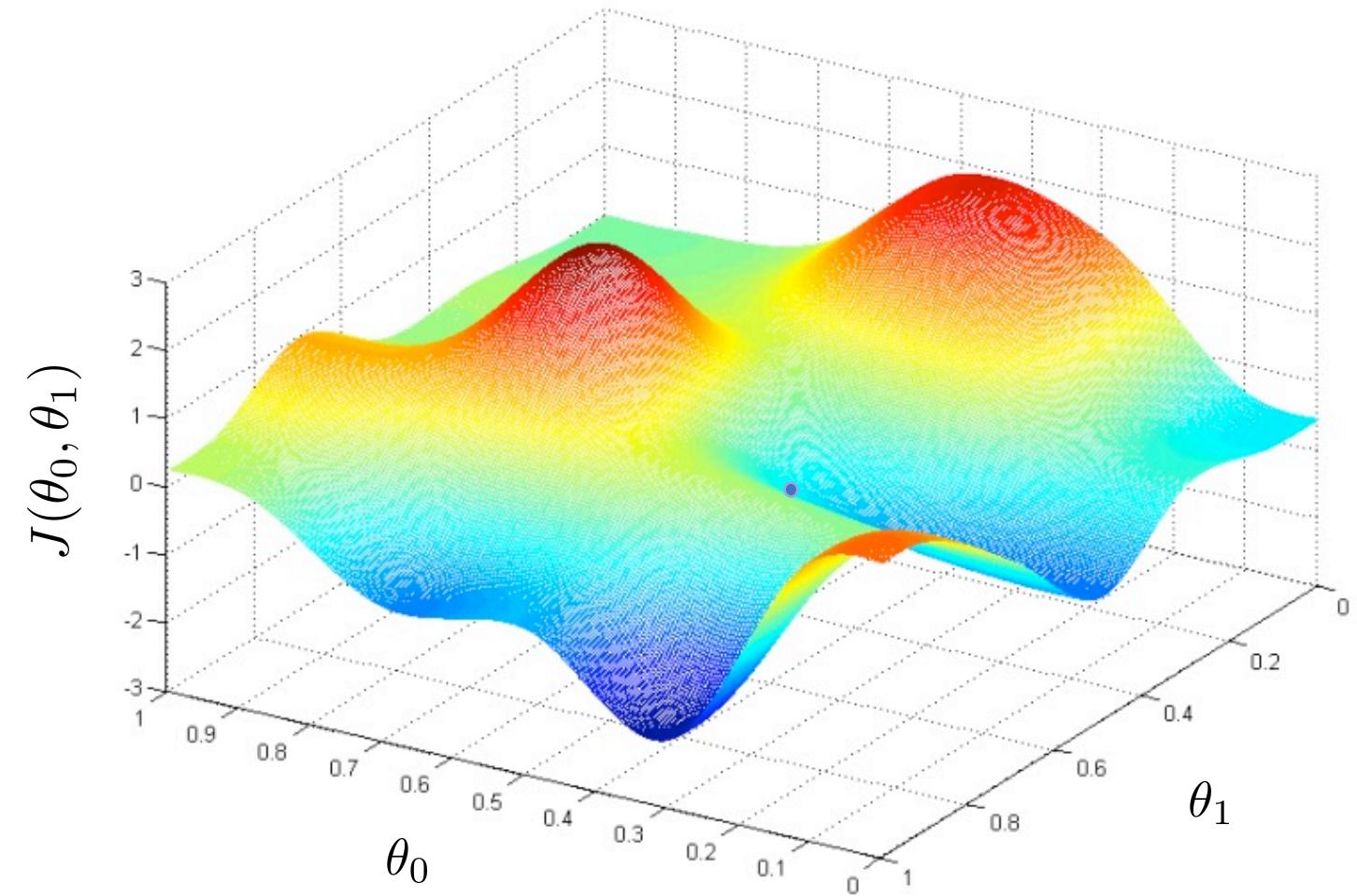


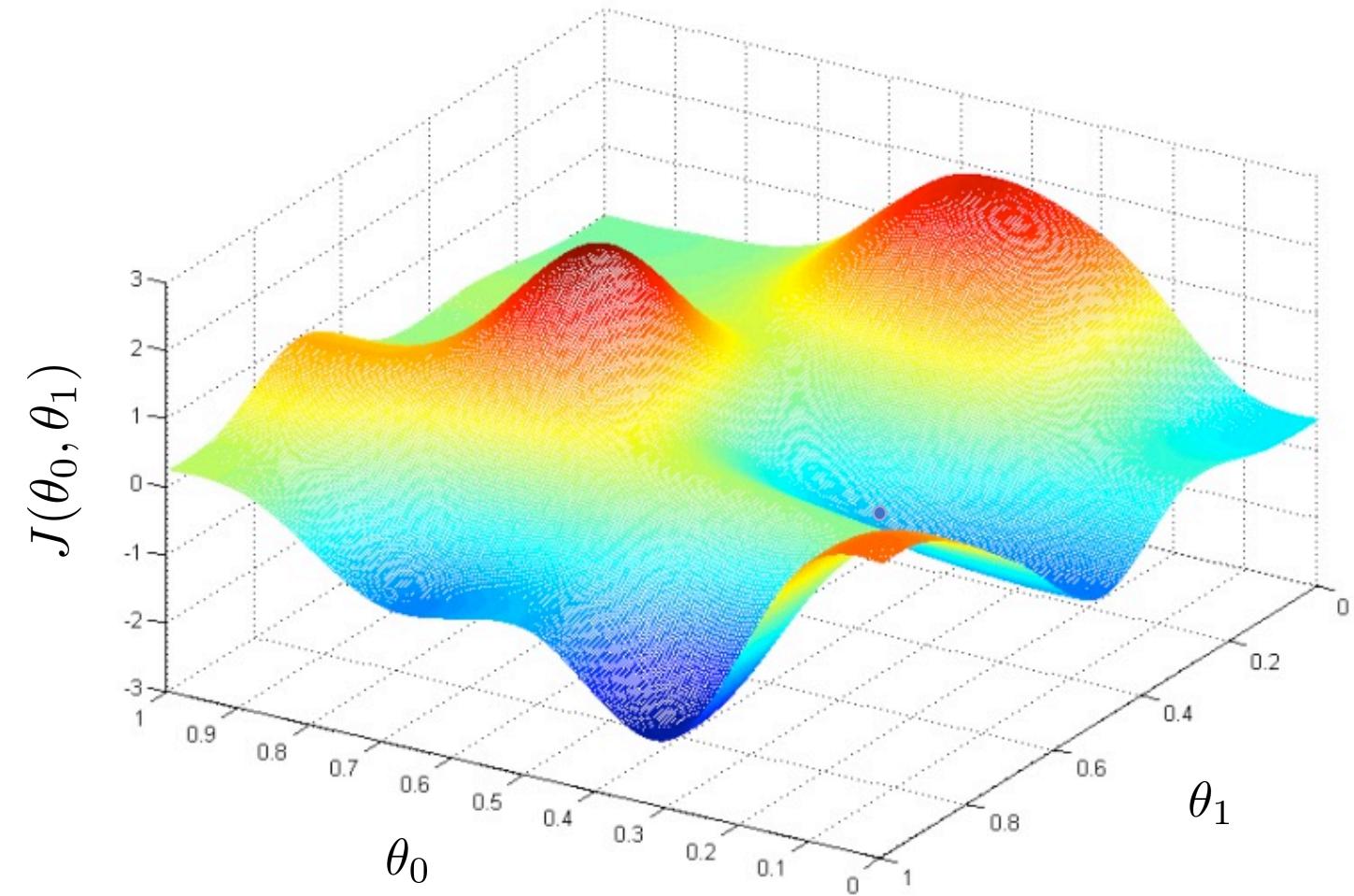


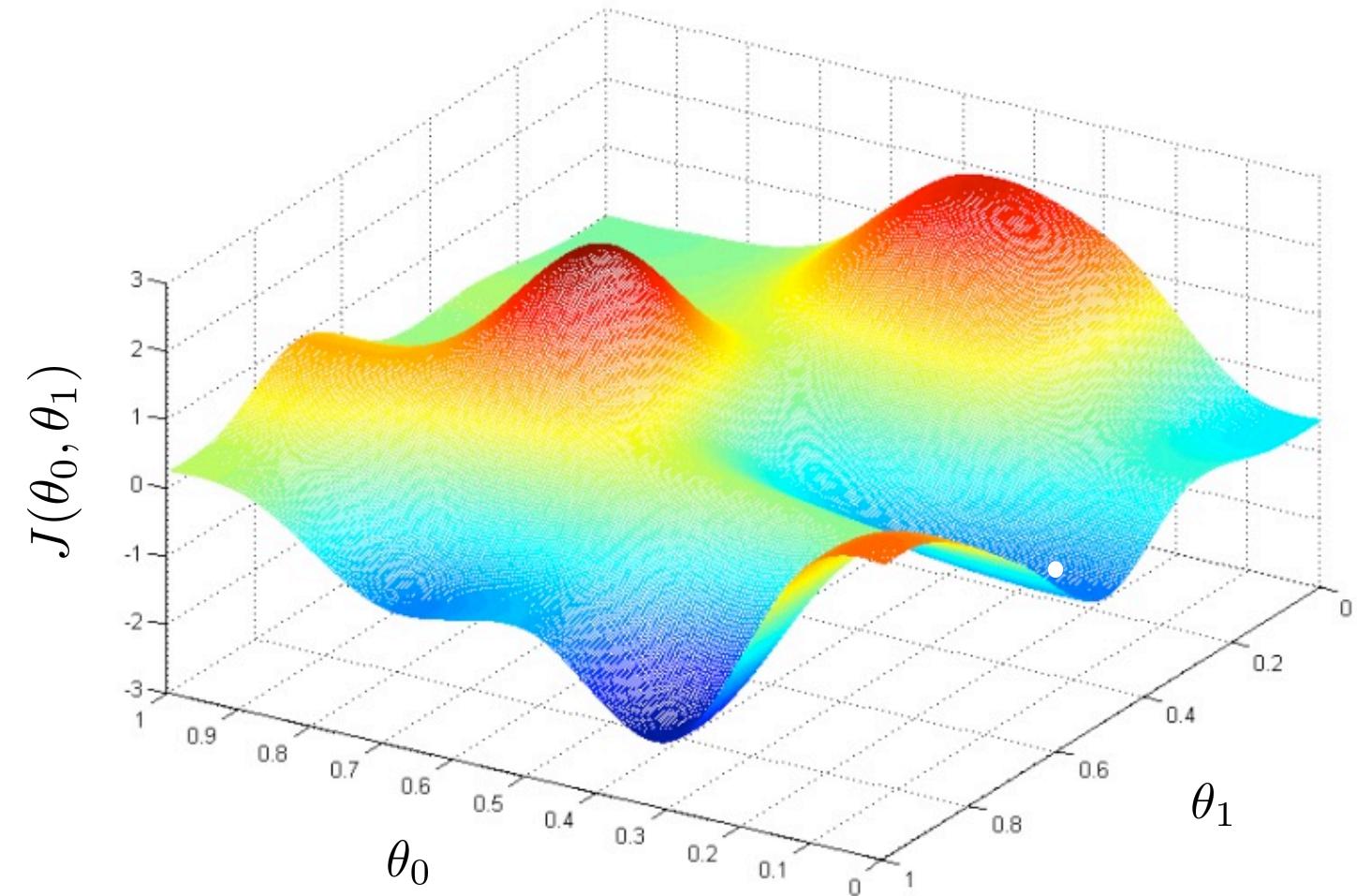












Gradient descent algorithm

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update

$$\text{Temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{Temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp0}$$

$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{Temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp0}$$

$$\text{Temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_1 := \text{temp1}$$

Linear regression with one variable

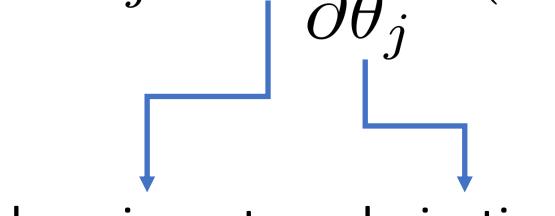
Gradient Descent Intuition

Gradient descent algorithm

Repeat until convergence {

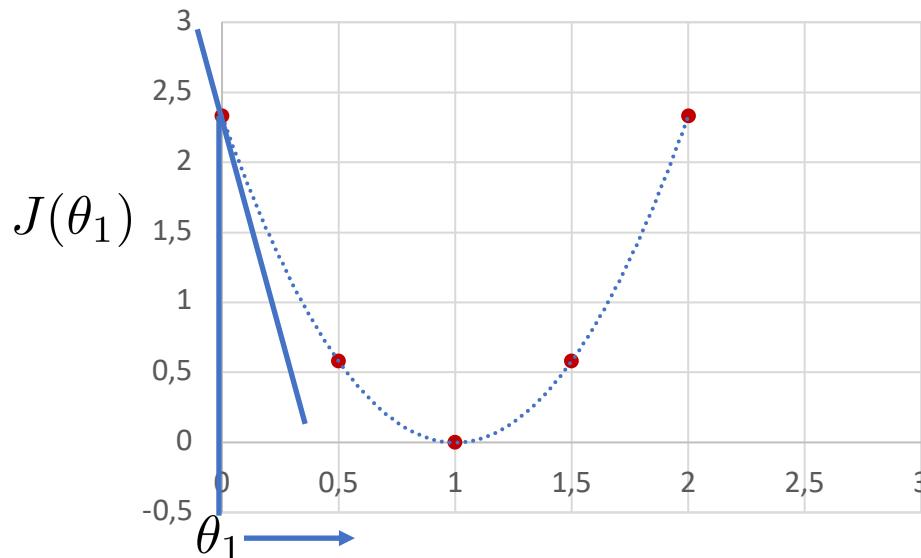
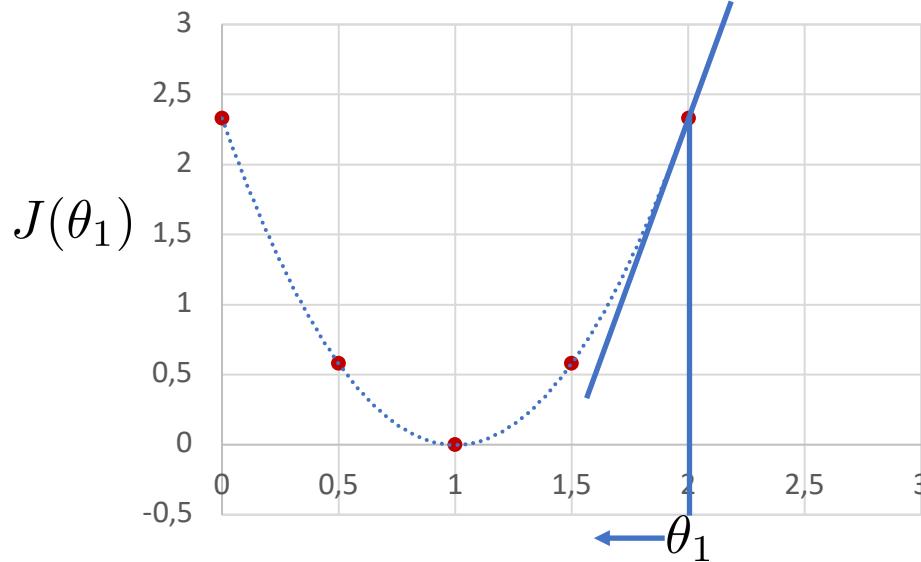
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (for \quad j = 0 \quad and \quad j = 1)$$

}



learning rate derivative

Gradient intuition



$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

≥ 0

$\theta_1 := \theta_1 - \alpha * (\text{positive number})$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

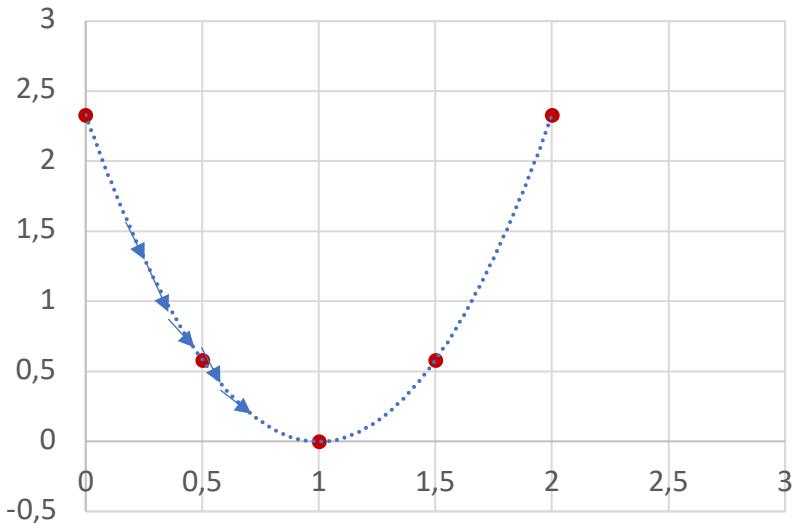
≤ 0

$\theta_1 := \theta_1 - \alpha * (\text{negative number})$

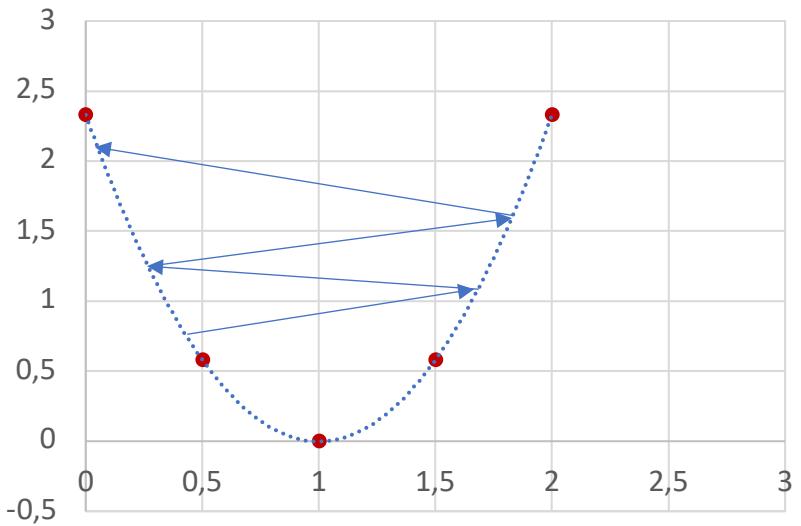
Learning rate

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

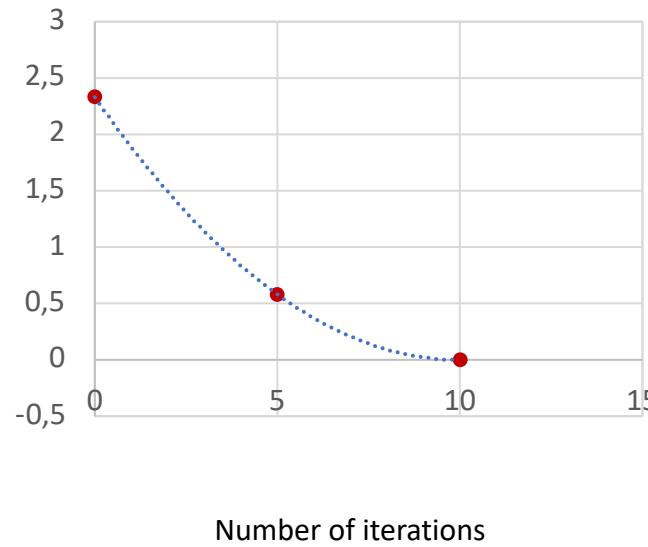
If alpha is too small,
gradient descent is slow



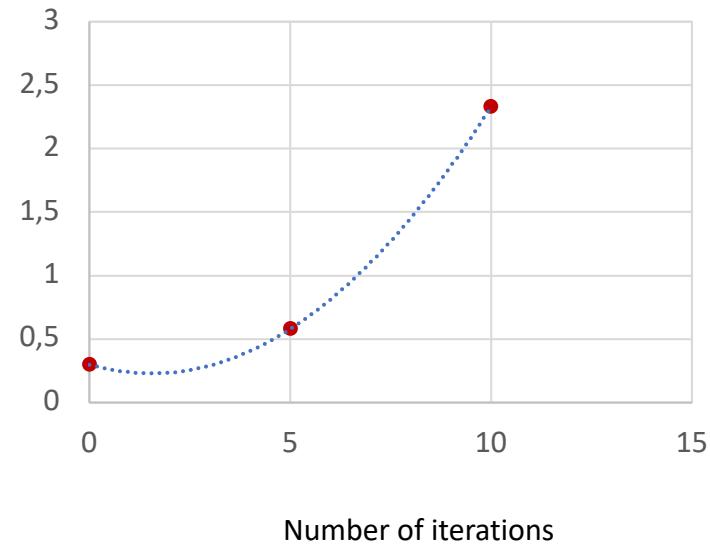
If alpha is too large,
gradient descent can diverge



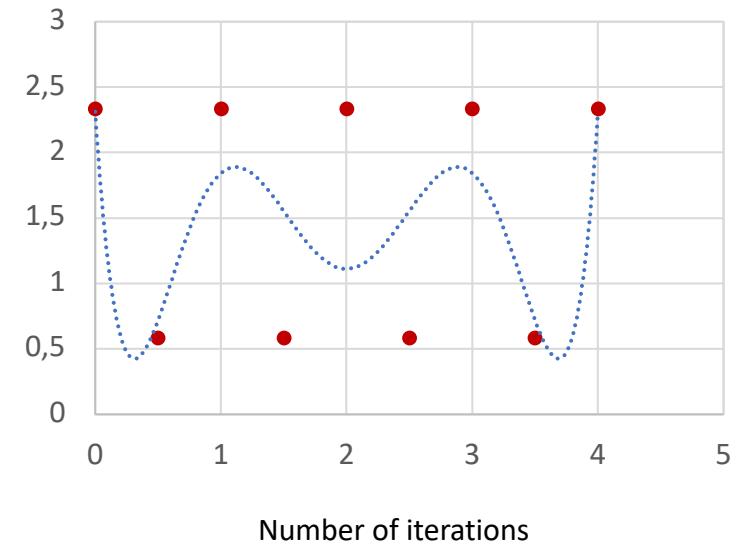
Learning rate (cont.d)



Gradient descent is working



Gradient descent is **NOT** working
Use smaller alpha



Gradient descent is **NOT** working
use smaller alpha

Linear regression with one variable

Gradient Descent for linear regression

Gradient descent for a linear regression model

Repeat *until convergence* {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 0$ and $j = 1$)

Linear Regression Model

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

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

Deriving the updates

$$\begin{aligned}
 \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) &= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2 \\
 &= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 \mathbf{x}^{(i)} - y^{(i)})^2
 \end{aligned}$$

$$\begin{aligned}
 j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) &= \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \\
 j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) &= \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \mathbf{x}^{(i)}
 \end{aligned}$$

Gradient descent algorithm

Repeat until convergence {

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

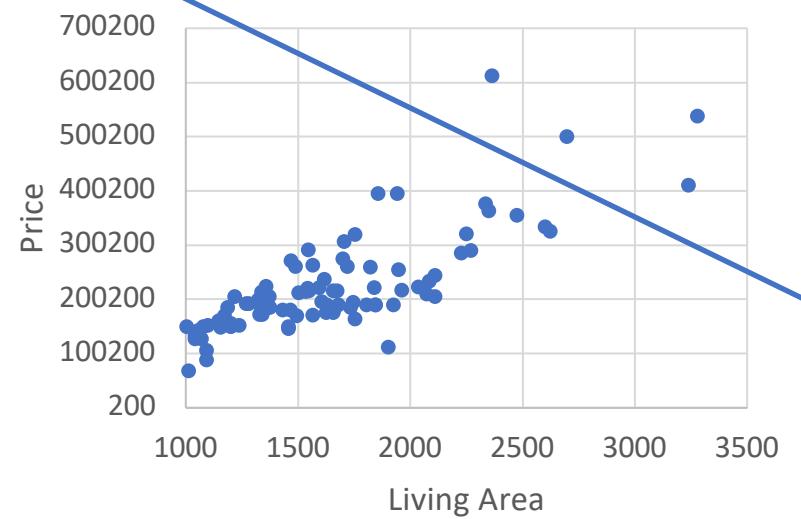
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \mathbf{x}^{(i)}$$

} simultaneous update



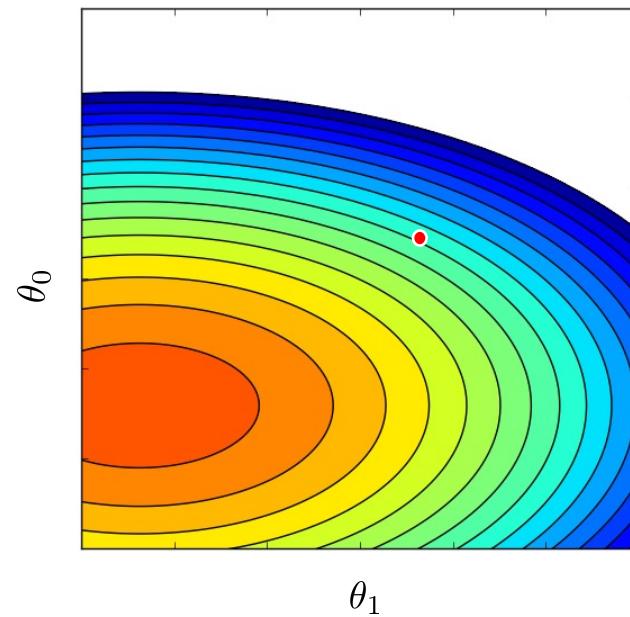
$$h_{\theta}(x)$$

(function of x , with θ_0, θ_1 fixed)



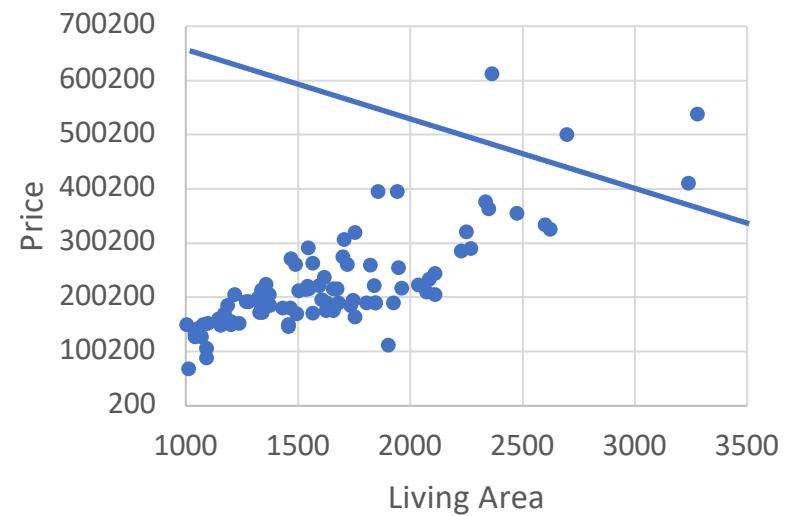
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



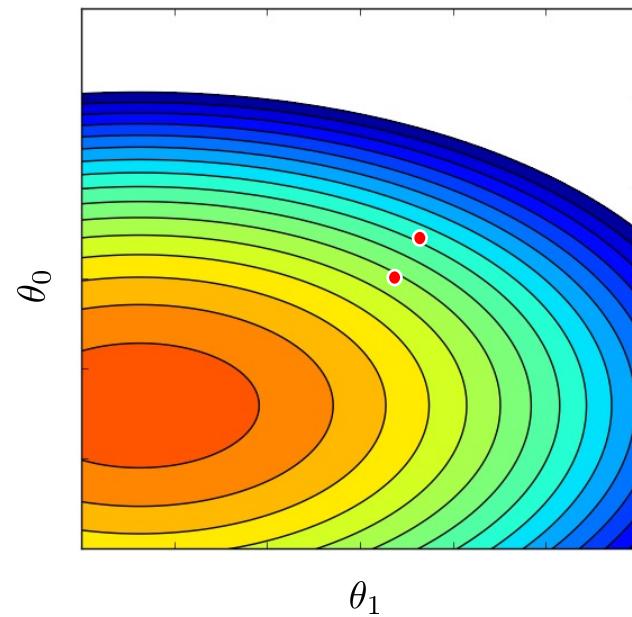
$$h_{\theta}(x)$$

(function of x , with θ_0, θ_1 fixed)



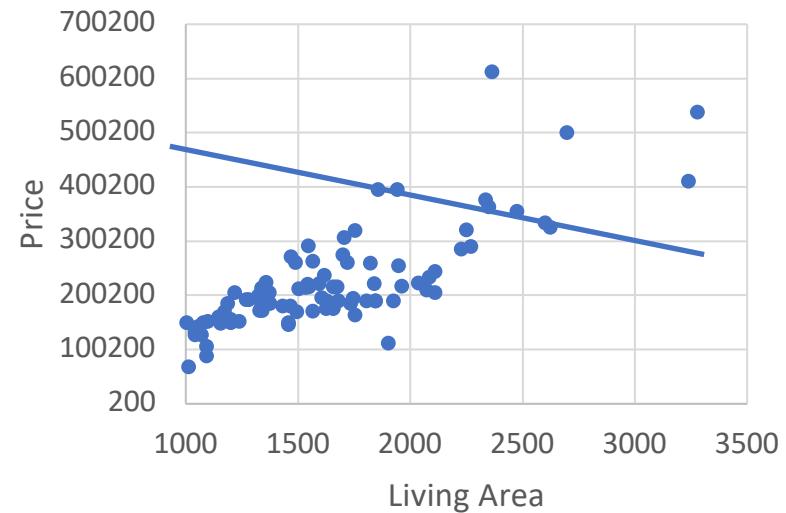
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



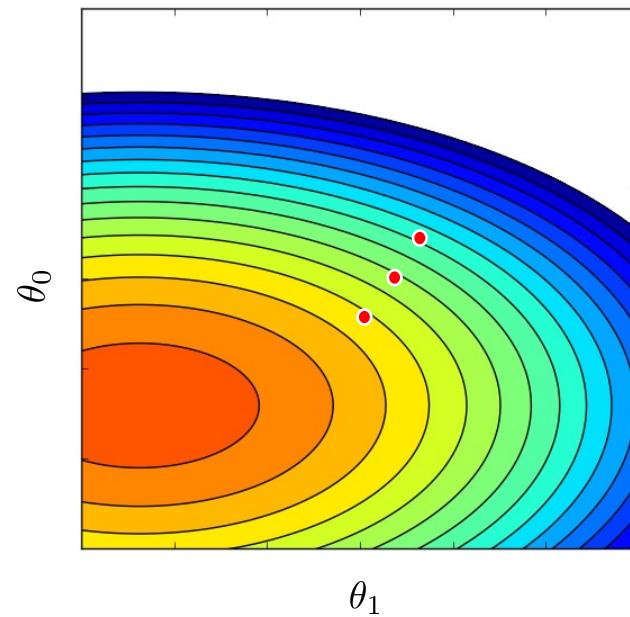
$$h_{\theta}(x)$$

(function of x , with θ_0, θ_1 fixed)



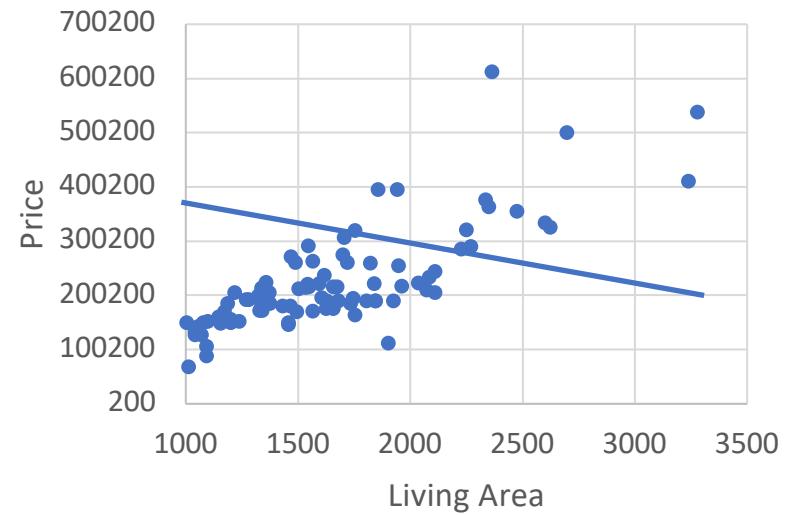
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



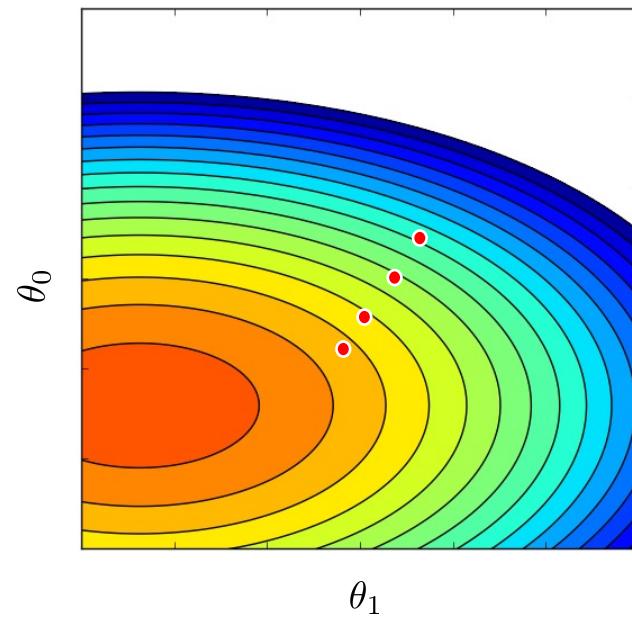
$$h_{\theta}(x)$$

(function of x , with θ_0, θ_1 fixed)



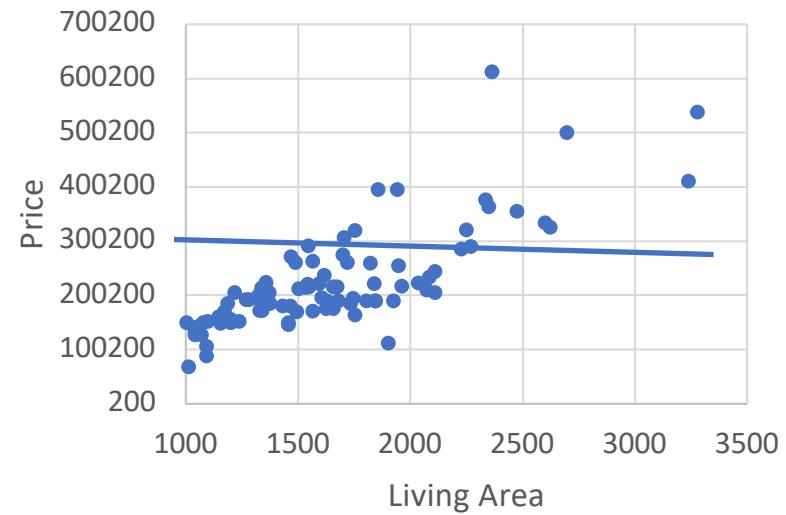
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



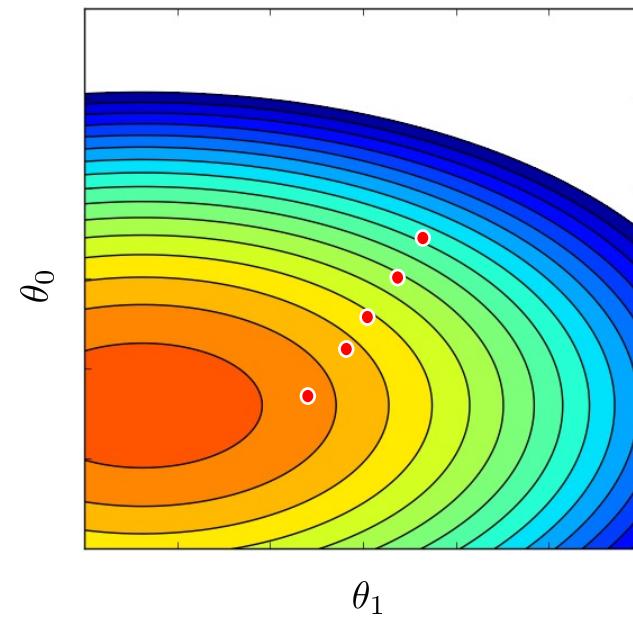
$$h_{\theta}(x)$$

(function of x , with θ_0, θ_1 fixed)



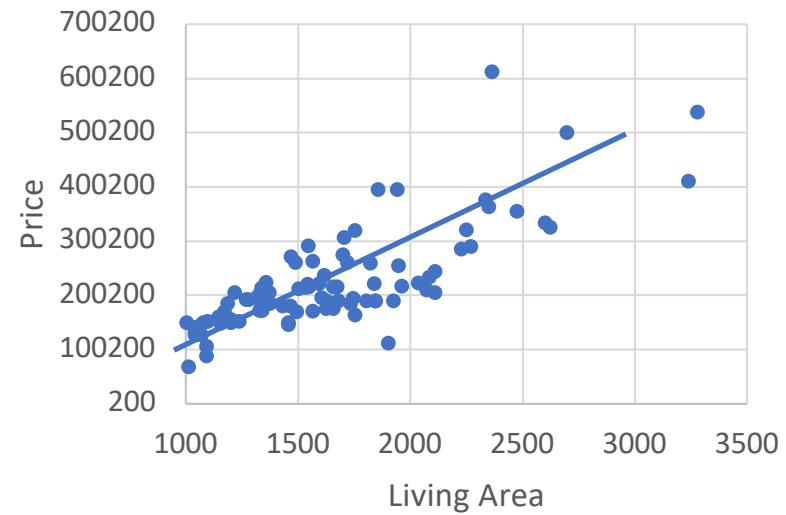
$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



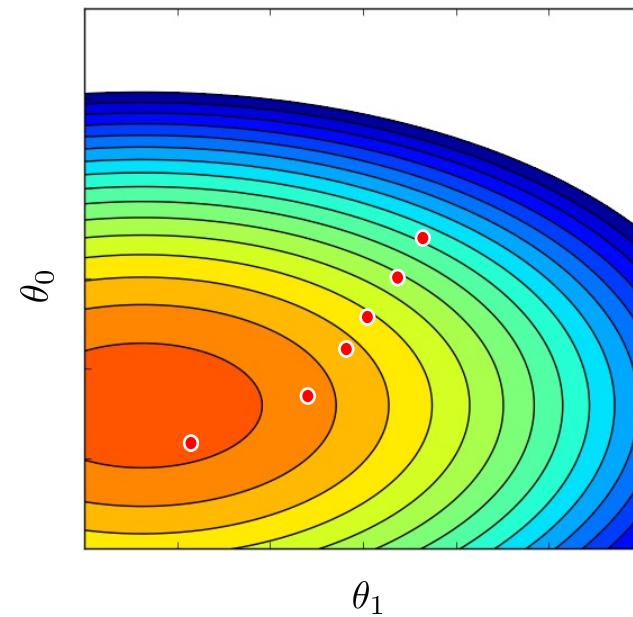
$$h_{\theta}(x)$$

(function of x , with θ_0, θ_1 fixed)



$$J(\theta_0, \theta_1)$$

(function of θ_0, θ_1 , an aggregate function over all x 's)



Linear regression

Multiple features

Housing prediction problem in Ames

Training set of housing prices

Notation:

m = Number of training examples

n = Number of features

$x^{(i)}$ = input features of the i^{th} training example

$y^{(i)}$ = “output” variable / “target” variable

$x_j^{(i)}$ = value of feature j in i^{th} training example

Living area (feet ²)	Number of bedrooms	Number of floors	Price (1000\$)
1656	5	1	215
896	3	2	105
1329	4	2	172
2110	2	1	244
...

$x_1^{(0)}$	1656
$x_1^{(1)}$	896
$x_2^{(3)}$	2

Hypothesis with multiple features

Previous hypothesis:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$



$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)}$$

Multivariate linear regression

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_n x_n^{(i)}$$

For convenience we set $x_0^{(i)} = 1$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Multivariate linear regression

$$\begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix}^T \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

$$\begin{aligned} h_{\theta}(x^{(i)}) &= \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_n x_n^{(i)} \\ &= \theta^T x. \end{aligned}$$

Batch Gradient descent

1. Randomly initialize parameters
2. Repeat *until convergence* {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \mathbf{x}_j^{(i)} \quad (\text{for } j = 0, 1, \dots, n)$$

}

Stochastic Gradient descent

1. Randomly initialize parameters

2. Repeat *until convergence* {

for each **training case** {

$$\theta_j := \theta_j - \alpha(h_\theta(\mathbf{x}^{(i)}) - y^{(i)})\mathbf{x}_j^{(i)} \quad (for \quad j = 0, 1, \dots, n)$$

}

}

- This version is called ***stochastic*** or ***incremental*** because the parameters are updated after each training sample, therefore the gradient is a "stochastic approximation" of the "true" cost gradient.
- The optimal solution is generally approximated faster, but the algorithm could swing around it without never getting convergence (zig-zag problem). A solution close to the optimal one is often acceptable.
- For large datasets, Stochastic gradient descent must be preferred over Batch version.
- The parameters are continuously updated. It means that the function h will use different values at each iteration

Batch GD vs Stochastic GD

```

theta = rand();
while (not convergence){
# iteration over theta parameters
for( j from 1 to n){
  update = 0;
  # iteration over training samples
  for( i from 1 to m)
    update = update + (h(x[i]) - y[i]) x[i];
  }
  theta_new[j] = theta[j] - alpha * update / m;
}
theta = theta_new;
}
  
```

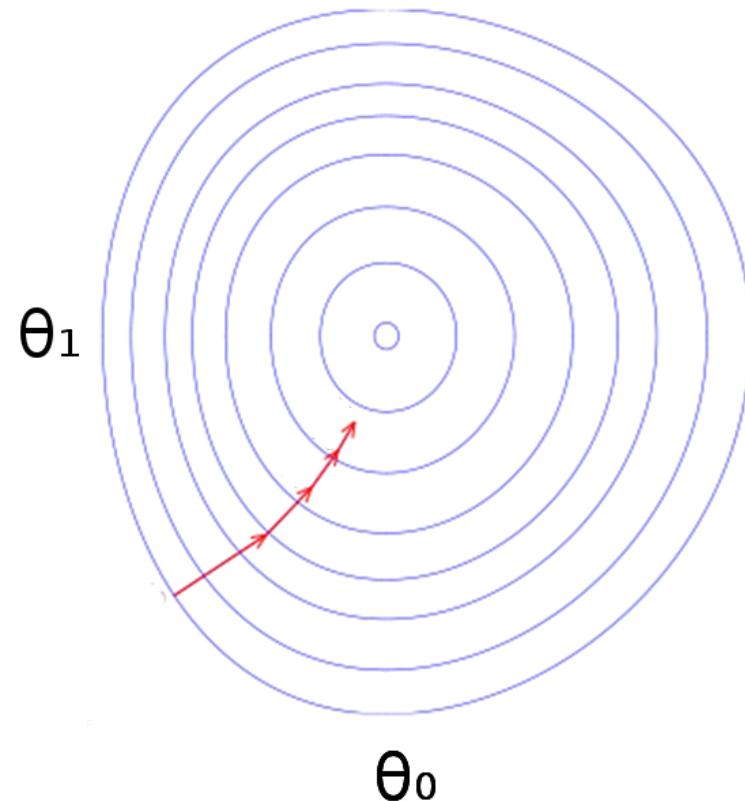
```

theta = rand();
while (not convergence){
# iteration over training samples
for(i from 1 to m){
# iteration over theta parameters
for(j from 1 to n){
  theta_new[j] = theta[j] - alpha * (h(x[i]) - y[i]) x[i];
}
theta = theta_new;
}
  
```



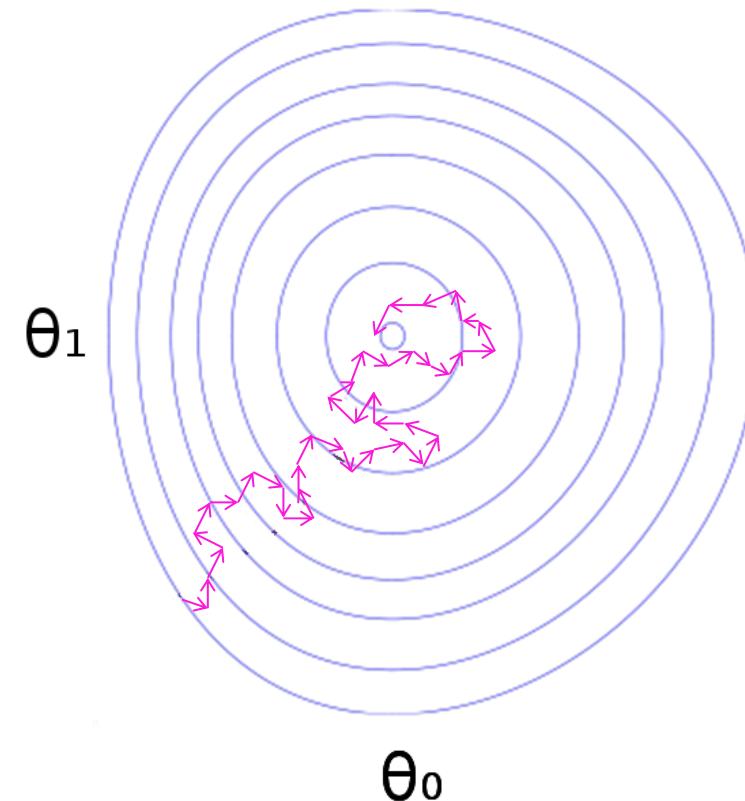
Pay attention at the position of the for loops

Batch GD vs Stochastic GD



Each “jump” is direct toward the minimum, but the relative update step goes through all the examples

Slow, but convergence is sure



Each “jump” may go potentially everywhere, but it is relative to just one training sample

Fast, but convergence is not sure

Mini-Batch gradient descent

Compromise between stochastic and batch gradient descent: takes the advantage of both.

Performs an update for every mini-batch of b training examples
where $1 < b \ll m$ (m is the total number of training samples)

Usually $b \in [2,100]$

In this way it reduces the variance of the parameter updates, leading to more stable convergence (more robust), and allows the use of vectorization libraries rather than computing each step separately (more efficient)

Mini-Batch gradient descent

```
b = 50;
while (not convergence){
    # iteration over training samples
    while(i < m){
        # iteration over theta parameters
        for(j from 1 to n){
            update = 0;
            # iteration over the b samples
            for( z from i to i + b ){
                update = update + (h(x[z]) - y[z]) x[z];
            }
            theta[j] = theta[j] – alpha * update / b;
        }
        i = i + b;
    }
}
```

Gradient descent algorithm stopping conditions

- **Max iteration:** the algorithm stops after a fixed number of iterations
- **Absolute tolerance:** the algorithm stops when a fixed value of the cost function is reached
- **Relative tolerance:** the algorithm stops when the decreasing of the cost function w.r.t. the previous step is below a fixed rate.
- **Gradient Norm tolerance:** the algorithm stops when the norm of the gradient is lower than a fixed value

Linear regression

Features and Polynomial regression

Adding features

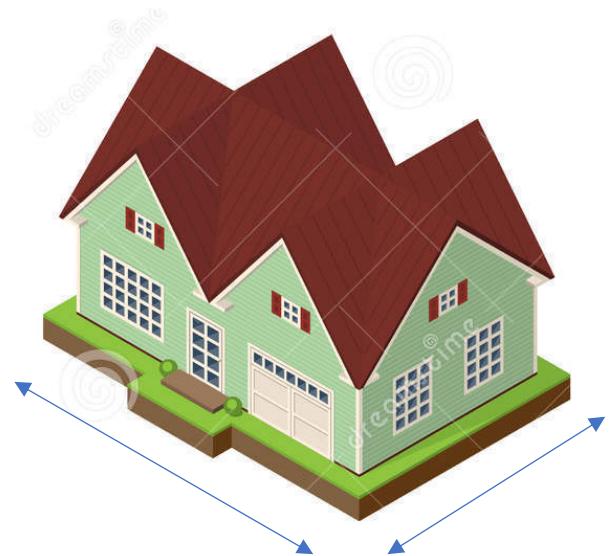
$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 \cdot front^{(i)} + \theta_2 \cdot side^{(i)}$$

Front and side information are not informative but
 Area could be:

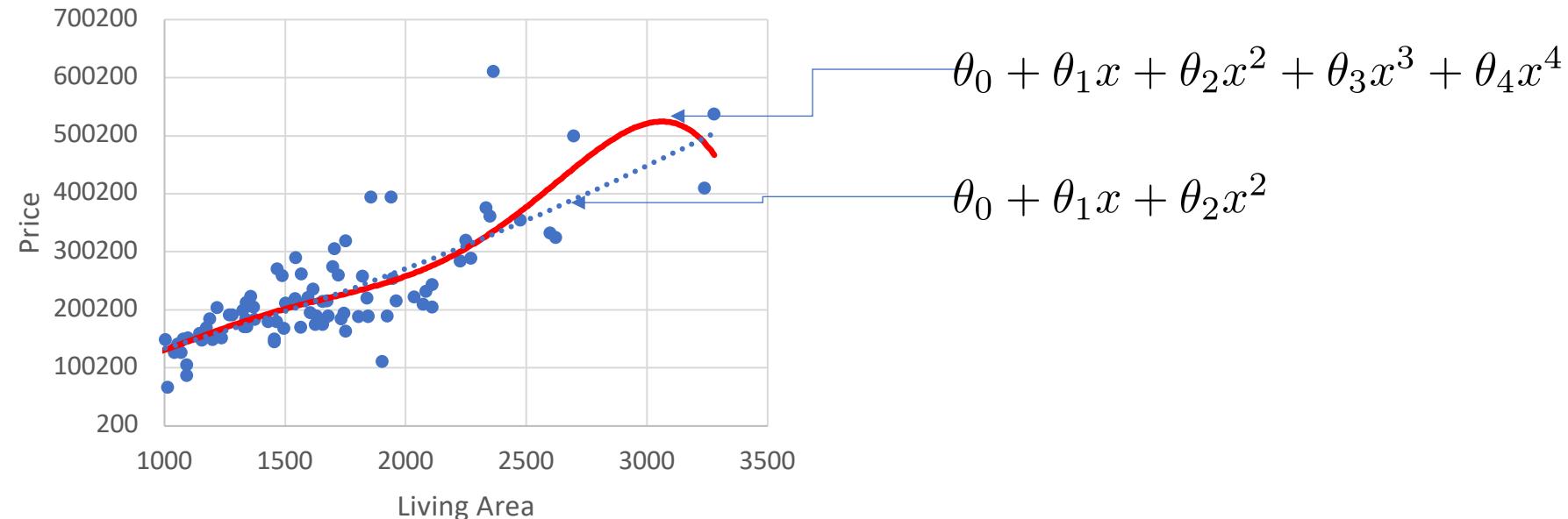
$$\text{Area} = \text{front} * \text{side}$$

We could inject Area or substitute the former features:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 \cdot area^{(i)}$$



Polynomial regression



$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)} + \theta_3 \cdot x_3^{(i)}$$

$$= \theta_0 + \theta_1(\text{area}^{(i)}) + \theta_2(\text{area}^{(i)})^2 + \theta_3(\text{area}^{(i)})^3$$

$$x_1^{(i)} = (\text{area})$$

$$x_2^{(i)} = (\text{area})^2$$

$$x_3^{(i)} = (\text{area})^3$$

Linear regression

Gradient descent for multiple variables

Univariate Linear regression

Hypothesis:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

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

Goal:

$$\underset{\theta_0, \theta_1}{\text{minimize}}(J(\theta_0, \theta_1))$$

Multivariate Linear regression

Hypothesis:

$$h_{\theta}(\mathbf{x}^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \dots + \theta_n x_n^{(i)}$$

Parameters:

$$\theta_0, \theta_1, \theta_2, \dots, \theta_n$$

Cost Function:

$$J(\theta) = J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Gradient descent:

```
repeat {
     $\theta_j := \theta_j - \alpha J(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$ 
}
```

(simultaneous updates of parameters)

Linear regression

Gradient descent: feature scaling

Feature scaling

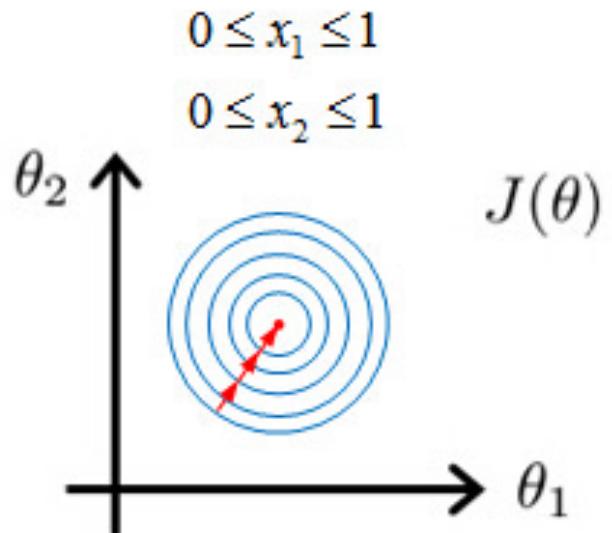
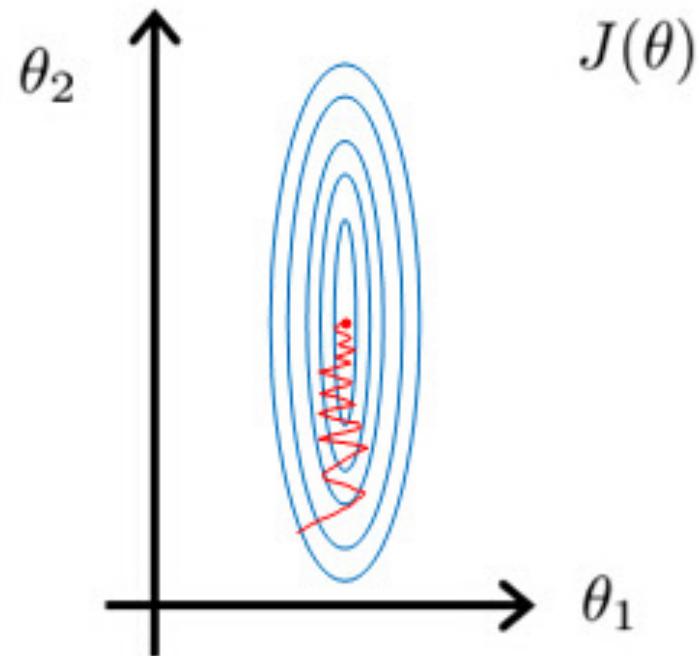
E.g. x_1 = Living Area (0-2110)

x_2 = number of bedrooms (1-5)

Idea:

$$x_1 = \frac{\text{area}(\text{feet}^2)}{2110}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Min-max normalization

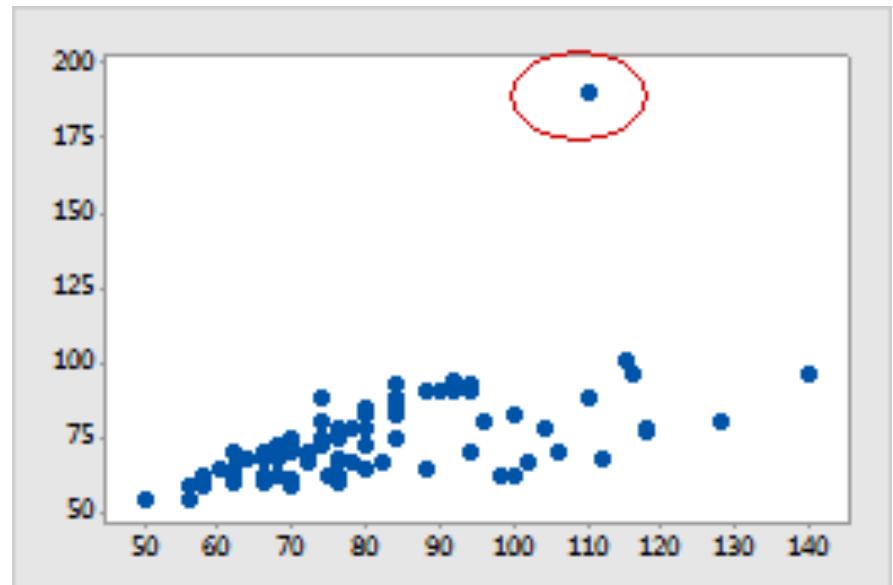
Forces the values in the interval [a,b].

This approach is negatively influenced by outliers.

You have to set up min and max values and in the prediction phase you could meet a new input greater than max.

$$x = \frac{x - \min}{\max - \min} \cdot (b - a) + a$$

Ex: $x=(1,2,3) \rightarrow [0,1] \quad x=((x-1)/(3-1)) * 1 \rightarrow x=(0,0.5,1)$



Z-score normalization

It produces a zero mean distribution through subtraction of the mean and dividing by the standard deviation.

This approach is less influenced by outliers and do not ask for min and max setting.

$$x = \frac{x - \text{mean}(x)}{\text{std_dev}(x)}$$

Linear regression with one variable

Normal equations

A vectorial perspective

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \vdots \\ \mathbf{x}^{(i)\top} \\ \vdots \\ \mathbf{x}^{(m)\top} \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$J(\theta) = \frac{1}{2}(\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}).$$

$$\nabla J(\theta) = \mathbf{X}^T \mathbf{X}\theta - \mathbf{X}^T \mathbf{y} = 0$$

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

Drawbacks

- Matrix Inverse (computationally expensive $O(n^3)$)
- Non invertible matrix:
 - Linear dependent features
 - Training samples are not enough
 - There are too many features

Proof of Normal equations derivations

The original cost function is $J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2$.

It is trivial to move to $J(\theta) = \frac{1}{2} (\mathbf{X}\theta - \mathbf{y})^T (\mathbf{X}\theta - \mathbf{y})$.

Because the latter can be explicitly re-written as

$$\mathbf{X}\theta - \mathbf{y} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h_\theta(\mathbf{x}^{(1)}) - y^{(1)} \\ h_\theta(\mathbf{x}^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(\mathbf{x}^{(m)}) - y^{(m)} \end{bmatrix}$$

Proof of N.E. (cont.d)

It is easy to obtain the squared variant:

$$\begin{aligned}
 & (\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) \\
 &= [(h_\theta(\mathbf{x}^{(1)}) - y^{(1)}) \quad \dots \quad (h_\theta(\mathbf{x}^{(m)}) - y^{(m)})] \\
 &\quad \left[\begin{array}{c} h_\theta(\mathbf{x}^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(\mathbf{x}^{(m)}) - y^{(m)} \end{array} \right] \\
 &= \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2.
 \end{aligned}$$

Proof of N.E. (cont.d)

We want to find the minimum, the point in which the gradient has null value:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_k} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_k} = \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_k^{(i)}$$

$$= [(h_\theta(\mathbf{x}^{(1)}) - y^{(1)}) \quad \dots \quad (h_\theta(\mathbf{x}^{(m)}) - y^{(m)})] \begin{bmatrix} x_k^{(1)} \\ \vdots \\ x_k^{(m)} \end{bmatrix}$$

Proof of N.E. (cont.d)

We can complete the gradient equation and re-write it as a matrix product.

$$\nabla J(\theta) = \begin{bmatrix} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \\ \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})x_1^{(i)} \\ \vdots \\ \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})x_n^{(i)} \end{bmatrix}$$

$$\begin{aligned} \nabla J(\theta) &= \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} h_\theta(\mathbf{x}^{(1)}) - y^{(1)} \\ h_\theta(\mathbf{x}^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(\mathbf{x}^{(m)}) - y^{(m)} \end{bmatrix} \\ &= \mathbf{X}^T(\mathbf{X}\theta - \mathbf{y}) \\ &= \mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y} \end{aligned}$$

$$\begin{aligned} \nabla J(\theta) &= \mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y} = 0 \\ \hat{\theta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

Linear regression with one variable

Probabilistic interpretation of linear regression

Proof Probabilistic interpretation

Toolbox:

$(\mathbf{x}^{(i)}, y^{(i)})$ m training set instances

$h(\mathbf{x}) = \theta^T \mathbf{x}$ hypothesis that approximates the relation between features vectors and output

Output can be computed as a predicted value **plus** an **error**

$$y^{(i)} = \theta^T \mathbf{x}^{(i)} + e^{(i)}. \quad (1)$$

Proof Probabilistic interpretation (cont.d)

Let us assume samples are independent so the errors, that we can model as an exponential random variable, and it is assumed that the variables show all a normal distribution, with 0 mean and variance σ^2

$$p(e^{(i)}) = \mathcal{N}(0, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(e^{(i)})^2}{2\sigma^2}} \quad i = 1 \dots m. \quad (2)$$

Proof Probabilistic interpretation (cont.d)

Now we fix the expressiveness of a model (the hypothesis) and so the parameters. This implies that the probability that the input produces the output has the same probability of the error. Substituting (1) in (2) we get:

$$e^{(i)} = y^{(i)} - \theta^T \mathbf{x}^{(i)}.$$

$$p(y^{(i)} | \mathbf{x}^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2}} \quad i = 1 \dots m.$$

Proof Probabilistic interpretation (cont.d)

We want to find a function (of the parameters) that, given the input, returns the output for each training case. This probability is named «Likelihood»:

$$L(\theta) = L(\theta; \mathbf{X}; \mathbf{y}) = p(\mathbf{y}|\mathbf{X}; \theta)$$

If the independence assumption is given (as we did before) the probability of the joint event is:

$$L(\theta) = p(\mathbf{y}|X; \theta) = \prod_{i=1}^m p(y^{(i)}|\mathbf{x}^{(i)}; \theta) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2}}$$

Proof Probabilistic interpretation (cont.d)

We look for the parameters vector that maximizes the Likelihood function.

We look for the thetas that has the maximum probability to return the output given the input.

This is named «maximum likelihood criterion»:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} L(\theta)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} L(\theta) = \underset{\theta}{\operatorname{argmax}} \log L(\theta) = \underset{\theta}{\operatorname{argmax}} l(\theta)$$

with $\log L(\theta) = l(\theta)$

Proof Probabilistic interpretation (cont.d)

Logarithm operator let us to transform the productory in a summatory

$$\begin{aligned}
 l(\theta) &= \log \prod_{i=1}^m p(y^{(i)} | \mathbf{x}^{(i)}; \theta) \\
 &= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2}} \\
 &= \sum_{i=1}^m \left(\log \frac{1}{\sqrt{2\pi}\sigma} + \log e^{-\frac{(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2}} \right) \\
 &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \sum_{i=1}^m \frac{(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2}{2\sigma^2} \\
 &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2
 \end{aligned}$$

Proof Probabilistic interpretation (cont.d)

Moving back to the maximization problem:

$$\begin{aligned}
 \max_{\theta} l(\theta) &= \max_{\theta} \left(m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2 \right) \\
 &= \max_{\theta} \left(-\frac{1}{2\sigma^2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2 \right) \\
 &= \max_{\theta} \left(-\frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2 \right) \\
 &= \min_{\theta} \left(\frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2 \right).
 \end{aligned}$$

Proof Probabilistic interpretation (cont.d)

We can compare the latter with the error in (1):

$$\min_{\theta} \left(\frac{1}{2m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2 \right) = \min_{\theta} \left(\frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2 \right)$$

These problems are identical but a regularization costant that has no influence in the minimization problem.

Solving the least squares problem in the linear regression means solving the problem of finding the best parameters that produce the output given the input

Logistic Regression

Classification

Classification problem

- Email: Spam/NotSpam?
- Online Transactions: Fraudulent (Yes/No)?
- Tumor: Malignant/Benign?

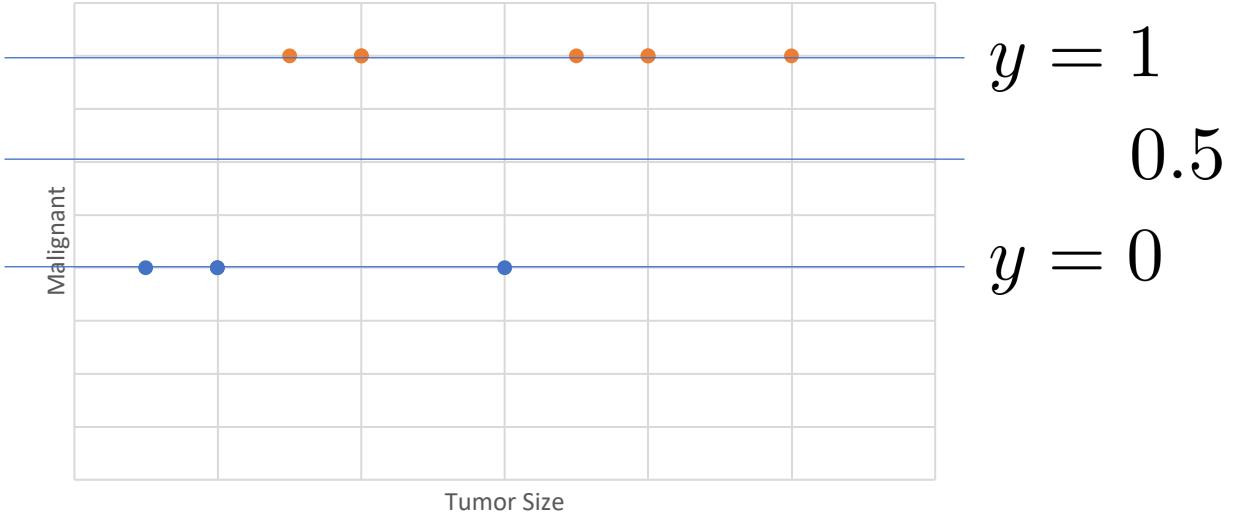
$$y \in \{0, 1\}$$

0: “Negative Class” (e.g., benign tumor)
1: “Positive Class” (e.g., malignant tumor)

$$y \in \{0, 1, 2, 3, 4, 5\}$$

Multiple classes

Classifier threshold



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict «y=1»

If $h_\theta(x) \leq 0.5$, predict «y=0»

Bounding output

Classification: $y = 0$ or 1

$h_\theta(x)$ can be > 1 or < 0

In the Logistic regression we want the hypothesis output to be bound: $0 \leq h_\theta(x) \leq 1$

Logistic Regression

Hypothesis Representation

Logistic regression model

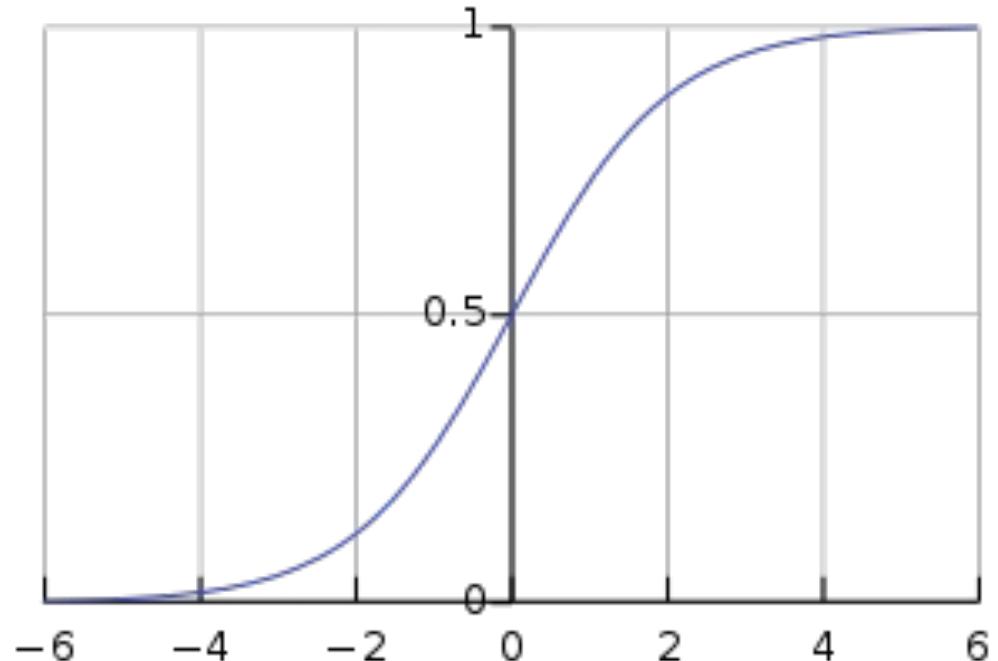
Sigmoid (Logistic) function as hypothesis:

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

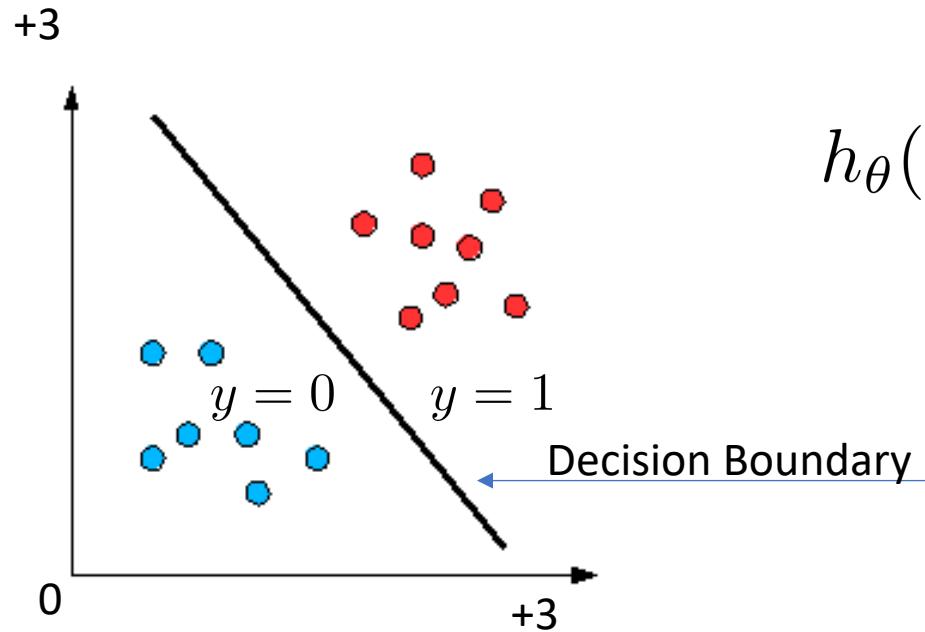
$$= g(z)$$

$$= \frac{1}{1 + e^{-z}}$$

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



Decision boundary intuition

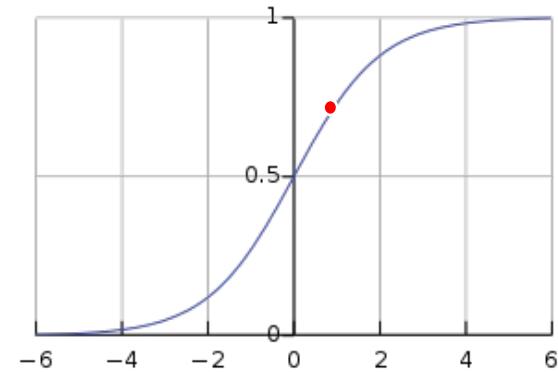


$$h_{\theta}(x) = g(\underbrace{\theta_0}_{-3} + \underbrace{\theta_1}_{+1}x_1 + \underbrace{\theta_2}_{+1}x_2)$$

Predict $y = 1$ if $-3 + x_1 + x_2 \geq 0$

Probabilistic Interpretation

$h_\theta(x)$ = estimated probability that $y=1$ on input x



If $h_\theta(x) = 0.7$, it is estimated a probability of 0.7 that the tumor is malignant. It represents the probability that y is 1, given x , parametrized by theta.

$$h_\theta(x) = p(y = 1|x; \theta)$$

$$y = 0 \quad \text{or} \quad 1$$

$$p(y = 0|x; \theta) + p(y = 1|x; \theta) = 1$$

$$p(y = 0|x; \theta) = 1 - p(y = 1|x; \theta)$$

Cost Function

As before, we want the solution of the minimization problem of the Cost function to be equal to the solution of the hypothesis determination problem using the criterion of maximum likelihood.

We can recap the formula, under the same i.i.d. assumptions (the training samples are independent and with identical distribution). In this case we have only two possible outputs, 0 and 1, so we are facing a Bernoulli distribution.

$$L(\theta) = L(\theta; X; \mathbf{y}) = p(\mathbf{y}|X; \theta) = \prod_{i=1}^m p(y^{(i)}|\mathbf{x}^{(i)}; \theta). \quad (1)$$

Cost Function (cont.d)

$$L(\theta) = L(\theta; X; \mathbf{y}) = p(\mathbf{y}|X; \theta) = \prod_{i=1}^m p(y^{(i)}|\mathbf{x}^{(i)}; \theta). \quad (1)$$

As seen before

$$\begin{cases} p(y^{(i)} = 1|\mathbf{x}^{(i)}; \theta) = h_\theta(\mathbf{x}^{(i)}) \\ p(y^{(i)} = 0|\mathbf{x}^{(i)}; \theta) = 1 - h_\theta(\mathbf{x}^{(i)}) \end{cases}$$

And we can express it in a more compact form:

$$p(y^{(i)}|\mathbf{x}^{(i)}; \theta) = h_\theta(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(\mathbf{x}^{(i)}))^{1-y^{(i)}}.$$

We can substitute the latter in (1):

$$L(\theta) = \prod_{i=1}^m h_\theta(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(\mathbf{x}^{(i)}))^{1-y^{(i)}}$$

Cost Function (cont.d)

Now we can move to logarithmic form:

$$\begin{aligned}
 l(\theta) &= \log \prod_{i=1}^m h_\theta(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(\mathbf{x}^{(i)}))^{1-y^{(i)}} \\
 &= \sum_{i=1}^m \left(y^{(i)} \log h_\theta(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(\mathbf{x}^{(i)})) \right)
 \end{aligned}$$

We can move from maximization to the minimization problem

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)} \log h_\theta(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(\mathbf{x}^{(i)})) \right).$$

Cross Entropy Error Function

Finally we can set the whole problem as a parameters fitting problem

$$\hat{\theta} = \operatorname{argmin}_\theta J(\theta).$$

Errors

The logistic function cannot be easily studied analytically, but we can get an intuition using the error contribution:

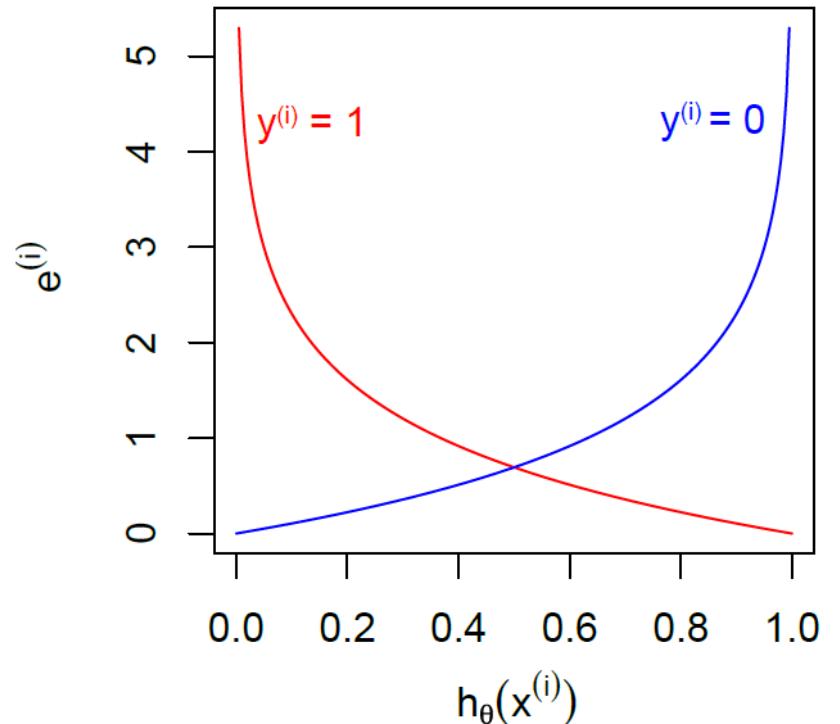
$$e^{(i)} = -y^{(i)} \log h_{\theta}(\mathbf{x}^{(i)}) - (1 - y^{(i)}) \log (1 - h_{\theta}(\mathbf{x}^{(i)}))$$

If $y^{(i)} = 1$ the error has value $e^{(i)} = -\log h_{\theta}(\mathbf{x}^{(i)})$

- if $h(x)$ is low the error is high: $h_{\theta}(\mathbf{x}^{(i)}) = 0 \Rightarrow e^{(i)} \rightarrow \infty$
- if $h(x)$ is high the error is low: $h_{\theta}(\mathbf{x}^{(i)}) = 1 \Rightarrow e^{(i)} \rightarrow 0$

If $y^{(i)} = 0$ the error has value $e^{(i)} = -\log (1 - h_{\theta}(\mathbf{x}^{(i)}))$

- if $h(x)$ is low the error is low: $h_{\theta}(\mathbf{x}^{(i)}) = 0 \Rightarrow e^{(i)} \rightarrow 0$
- if $h(x)$ is high the error is high: $h_{\theta}(\mathbf{x}^{(i)}) = 1 \Rightarrow e^{(i)} \rightarrow \infty$



Logistic Regression

Gradient descent

Gradient descent

Goal: $\hat{\theta} = \operatorname{argmin}_{\theta} J(\theta)$.

Weights update: $\theta_k = \theta_k - \alpha \frac{\partial J(\theta)}{\partial \theta_k}$

We can derive the Logistic function

$$\begin{aligned} g'(z) &= \frac{\partial}{\partial z} \left(\frac{1}{1 + e^{-z}} \right) = \frac{e^{-z}}{(1 + e^{-z})^2} \\ &= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right) = g(z)(1 - g(z)) \end{aligned}$$

Weight update derivation

From $g'(z) = g(z)(1 - g(z))$

We can compute the derivative of $h_\theta(x)$

$$\frac{\partial h_\theta(\mathbf{x})}{\partial \theta_k} = h_\theta(\mathbf{x})(1 - h_\theta(\mathbf{x})) \frac{\partial(\theta^T \mathbf{x})}{\partial \theta_k}$$

In linear regression gradient descent we saw that $\frac{\partial(\theta^T \mathbf{x})}{\partial \theta_k} = x_k$

Hence $\frac{\partial h_\theta(\mathbf{x})}{\partial \theta_k} = h_\theta(\mathbf{x})(1 - h_\theta(\mathbf{x}))x_k$

Weight update derivation (cont.d)

We can compute the the partial derivative of

$$\frac{\partial J(\theta)}{\partial \theta_k} = -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)} \underbrace{\frac{\partial}{\partial \theta_k} (\log h_\theta(\mathbf{x}^{(i)}))}_{+} + (1 - y^{(i)}) \underbrace{\frac{\partial}{\partial \theta_k} (\log (1 - h_\theta(\mathbf{x}^{(i)})))}_{-} \right) \quad (2)$$

We can deal with the two derivative separately

$$\frac{\partial}{\partial \theta_k} \log h_\theta(\mathbf{x}^{(i)}) = \frac{1}{h_\theta(\mathbf{x}^{(i)})} h_\theta(\mathbf{x}^{(i)}) (1 - h_\theta(\mathbf{x}^{(i)})) x_k^{(i)} = (1 - h_\theta(\mathbf{x}^{(i)})) x_k^{(i)}$$

$$\frac{\partial}{\partial \theta_k} \log (1 - h_\theta(\mathbf{x}^{(i)})) = \frac{1}{1 - h_\theta(\mathbf{x}^{(i)})} (-h_\theta(\mathbf{x}^{(i)})) (1 - h_\theta(\mathbf{x}^{(i)})) x_k^{(i)} = -h_\theta(\mathbf{x}^{(i)}) x_k^{(i)}$$

We substitute the latter two terms in (2)

Weight update derivation (cont.d)

And finally we reach the last derivative step:

$$\begin{aligned}
 \frac{\partial J(\theta)}{\partial \theta_k} &= -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)}(1 - h_\theta(\mathbf{x}^{(i)}))x_k^{(i)} + (1 - y^{(i)})(-h_\theta(\mathbf{x}^{(i)}))x_k^{(i)} \right) \\
 &= -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)}(1 - h_\theta(\mathbf{x}^{(i)})) + (1 - y^{(i)})(-h_\theta(\mathbf{x}^{(i)})) \right) x_k^{(i)} \\
 &= \frac{1}{m} \sum_{i=1}^m \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) x_k^{(i)}
 \end{aligned}$$

Logistic Regression

Multi-class classification
One vs all

Multi-class classification

Email tagging: Work, Friends, Family, Hobby

Medical diagrams: Not ill, Cold, Flu

Weather: Sunny, Cloudy, Rain, Snow

Multi-class classification



Email tagging: Work, Friends, Family, Hobby

$$y = [\ 1 \ , \ 2 \ , \ 3 \ , \ 4 \]$$



Medical diagrams: Not ill, Cold, Flu

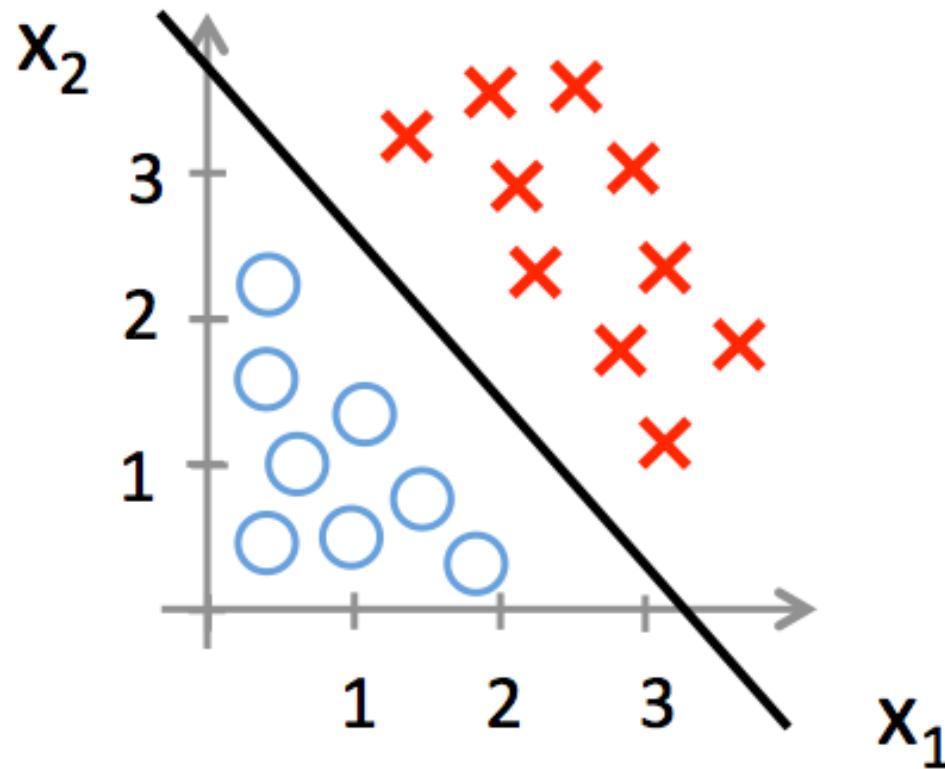
$$y = [\ 1 \ , \ 2 \ , \ 3 \]$$



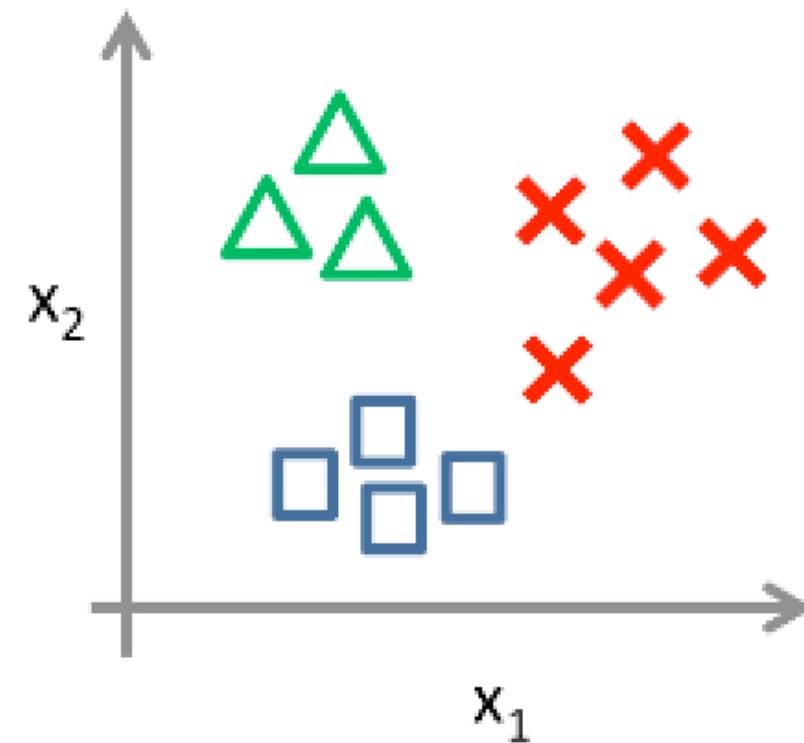
Weather: Sunny, Cloudy, Rain, Snow

$$y = [\ 1 \ , \ 2 \ , \ 3 \ , \ 4 \]$$

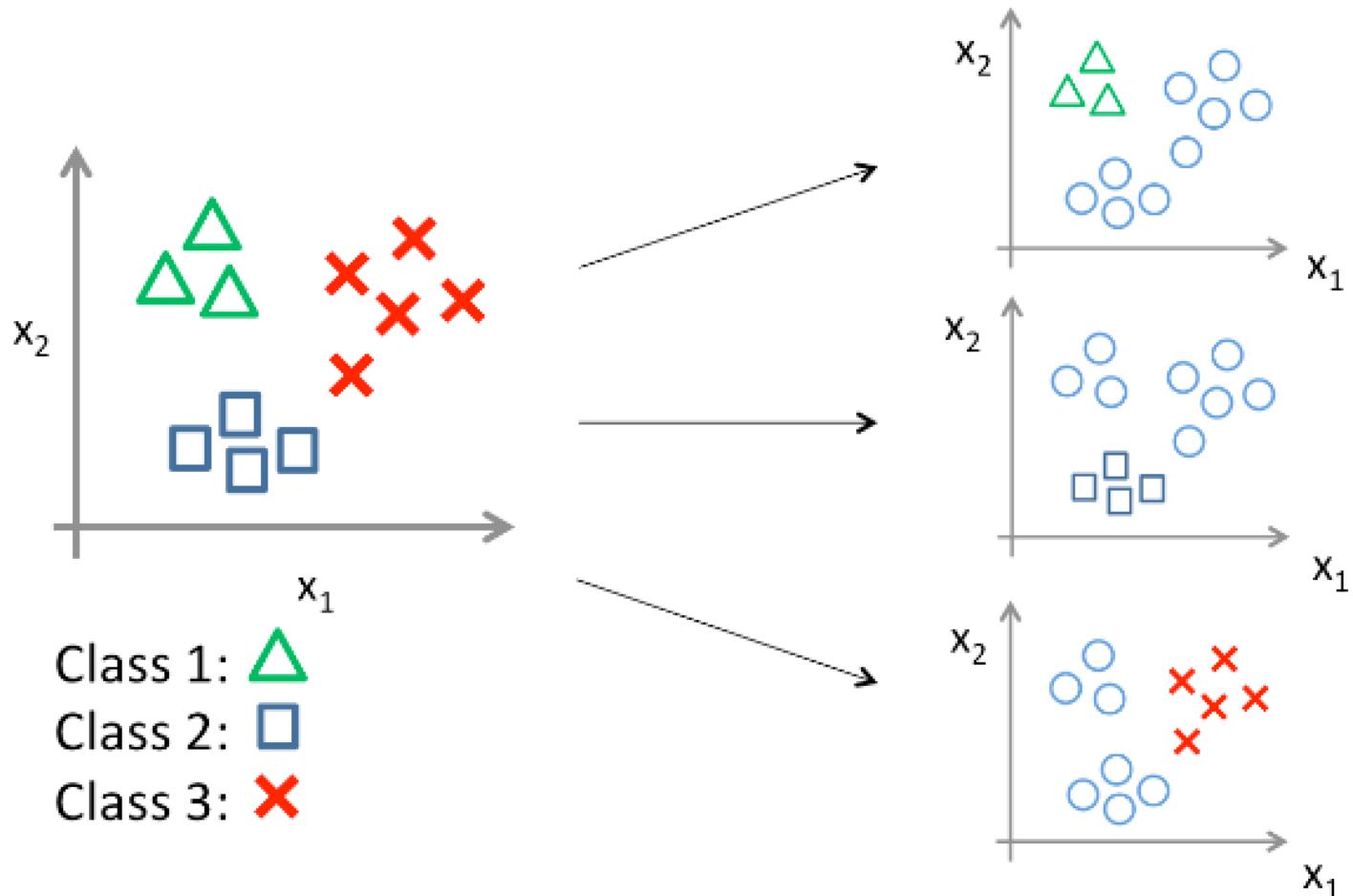
Binary classification



Multi-class classification



One vs all

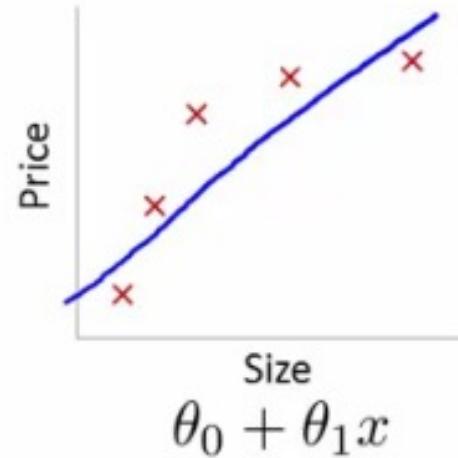


$$h_{\theta}^{(i)}(x) = p(y = i|x; \theta) \quad (i = 1, 2, 3)$$

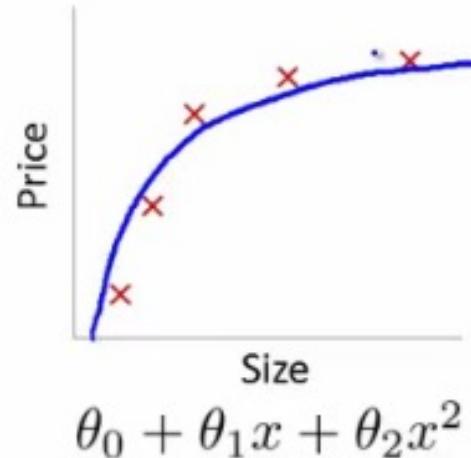
Fitting

Just Fit

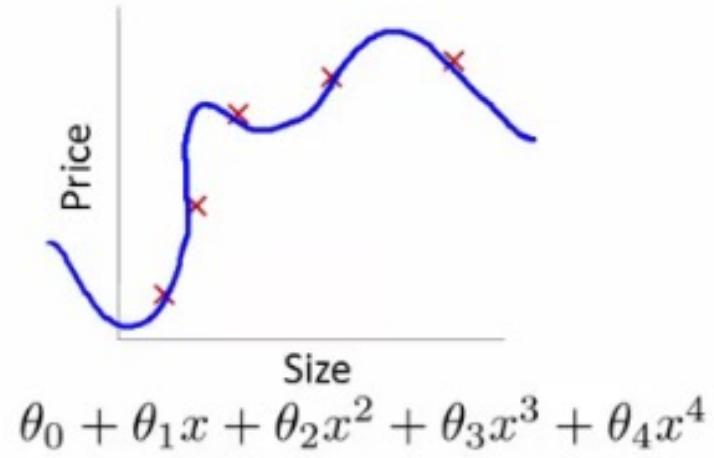
Fitting - regression



High bias
(underfit)

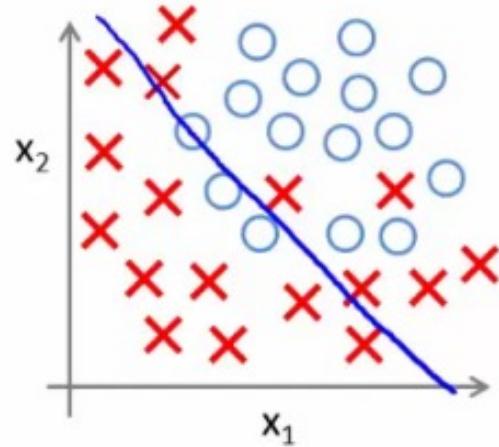


"Just right"



High variance
(overfit)

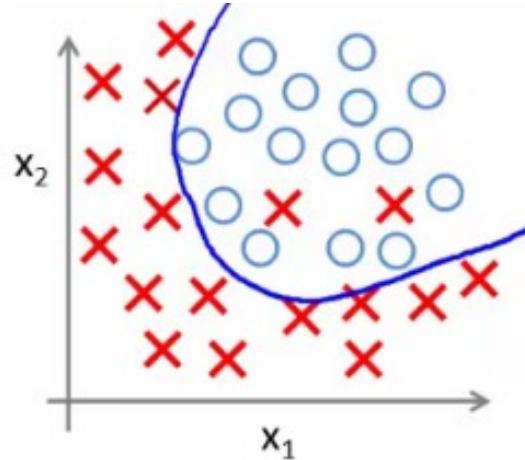
Fitting - classification



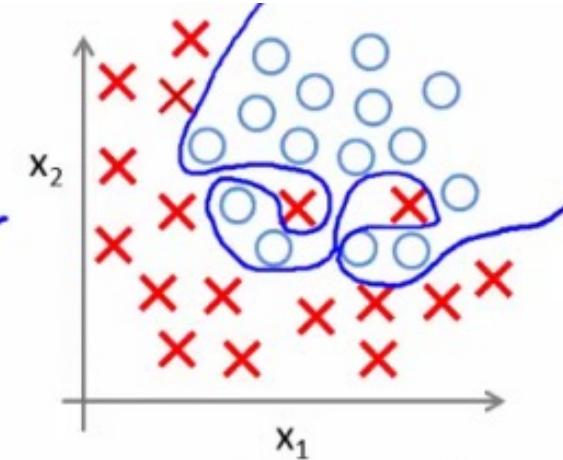
$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

(g = sigmoid function)

UNDERFITTING
(high bias)



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$



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

OVERRFITTING
(high variance)

Overfitting and Underfitting

We want a model able to generalize from its experience (training example). Machine learning is concerned with perform accurately on new, unseen examples/tasks after having experienced a learning data set

Underfitting: the model does not fit the training data, because it is too simple and not able to learn. **It performs bad both on training and test data.**

Overfitting: the **model performs well on the training data but not on unseen data**, because it is too complex and not able to generalize. The model begins to "memorize" training data rather than "learning" to generalize from trend

Expected squared error and its relationship with expected value

- Expected value of a random variable X

$$E[X] = \sum_{i=1}^m p_i \cdot x_i$$

- Let X be the input and Y be the output in a problem we want to model

$$Y = f(X) + e$$

- We assume our hypothesis $h(X)$ to approximate $f(X)$. The expected squared error (a.k.a. Mean Squared Error) is then

$$Err(f(X)) = E[(Y - h(X))^2]$$

Bias and Variance trade-off

We assume to have a number of *Dataset* D_i , each of size m, sampled from the original data distribution. They are subsets of the corresponding random variable

We fix the model and we train it each time with a different D_i . We get a corresponding number of hypothesis $h^{(D_i)}(x)$.

Each training sample is in the form $\langle \mathbf{x}^{(i)}, y^{(i)} \rangle$, where y is the sum of the ground truth and a gaussian error with zero mean and variance σ_e :

$$y^{(i)} = f(\mathbf{x}^{(i)}) + e^{(i)}$$

Each hypothesis $h^{(D_i)}(x)$ will be affected by an error in predicting values w.r.t. the ground truth.

We can model this error in the form of a Mean Squared Error (MSE):

$$MSE\left(h^{(D_i)}(x)\right) = E_x \left[(h^{(D_i)}(x) - f(x))^2 \right]$$

Bias and Variance trade-off (cont.d)

We want to estimate the expectation of the error w.r.t. all the possible D_i .

In details we want to compute the MSE of each hypothesis and then we compute the overall mean.

We can compute the expected value of the mean as the expectation of the MSE over all the datasets.

This expectation is named Generalization Error (GER):

$$\begin{aligned}
 GER &= E_D[MSE] \\
 &= E_D \left[E_x \left[\left(h^{(D_i)}(x) - f(x) \right)^2 \right] \right] \\
 &= E_x \left[E_D \left[\left(h^{(D)}(x^{(i)}) - f(x^{(i)}) \right)^2 \right] \right]
 \end{aligned}$$

The last step can be performed because of the linearity of expectation operator

Bias and Variance trade-off (cont.d)

Focus on the term $E_D \left[\left(h^{(D)}(x^{(i)}) - f(x^{(i)}) \right)^2 \right]$

We can define a new quantity, the best estimation of $f(x^{(i)})$, by feeding each hypothesis with the same training sample $x^{(i)}$, and then we compute the mean of all the values:

$$\bar{h}(x^{(i)}) = E_D \left[h^{(D)}(x^{(i)}) \right]$$

We can now perform a sum zero operation by adding and subtracting the latter from the initial term:

$$E_D \left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) + \bar{h}(x^{(i)}) - f(x^{(i)}) \right)^2 \right]$$

We can solve the power and exploit the linearity of E:

$$E_D \left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) \right)^2 \right] + E_D \left[\left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right)^2 \right] + E_D \left[2 \left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) \right) \left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right) \right]$$

Bias and Variance trade-off (cont.d)

$$E_D \left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) \right)^2 \right] + E_D \left[\left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right)^2 \right] + E_D \left[2 \left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) \right) \left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right) \right]$$

Let us focus on the first term:

$$E_D \left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) \right)^2 \right] = Var \left(h^{(D)}(x^{(i)}) \right)$$

It matches the variance definition.

The second term:

$$E_D \left[\left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right)^2 \right] = (\bar{h}(x^{(i)}) - f(x^{(i)}))^2$$

Matches the squared bias

Bias and Variance trade-off (cont.d)

The third term:

$$\begin{aligned}
 & 2 \left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right) E_D \left[h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)}) \right] \\
 &= 2 \left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right) \left(E_D \left[h^{(D)}(x^{(i)}) \right] - E_D \left[\bar{h}(x^{(i)}) \right] \right) \\
 &= 2 \left(\bar{h}(x^{(i)}) - f(x^{(i)}) \right) \left(\bar{h}(x^{(i)}) - \bar{h}(x^{(i)}) \right) \\
 &= 0
 \end{aligned}$$

We can move back to the MSE:

$$MSE(h^{(D)}(x^{(i)})) = Bias^2 \left(h^{(D)}(x^{(i)}) \right) + Var \left(h^{(D)}(x^{(i)}) \right)$$

Bias and Variance

Bias is the systematic deviation of the estimator. It represents the mean of the error of the predicted values:

$$Bias(h(X), f(X)) = (E[h(X)] - f(X))^2$$

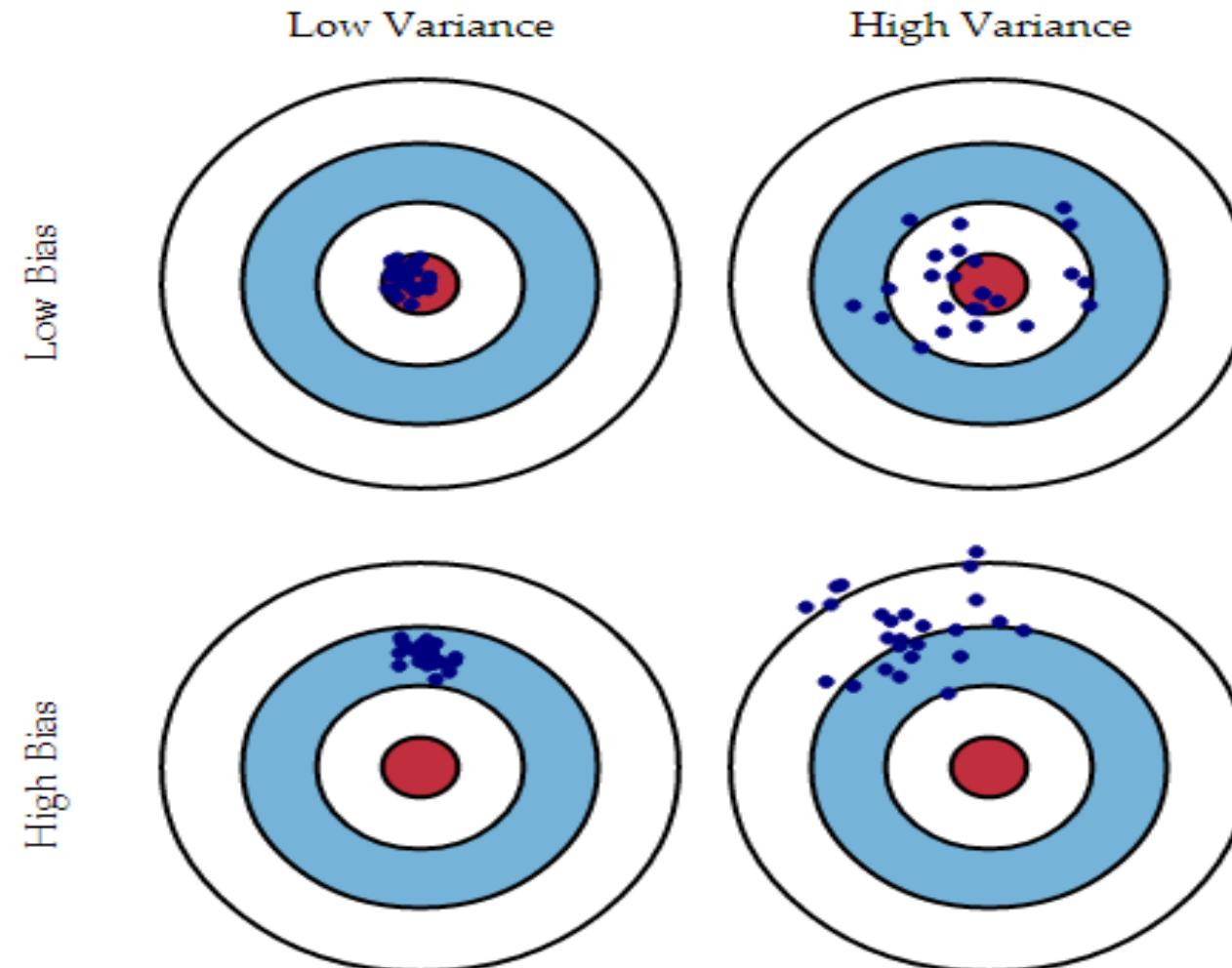
Variance represents the mean of the variations of the predicted values with respect to the mean of the predicted values.

The variance is about the stability of the model in response to new training examples.

It gives us an idea of the mean of the variations of the errors of the predicted values w.r.t. Y :

$$Var(h(X)) = E(h(X) - E[h(X)])^2$$

Bias and Variance - Intuition



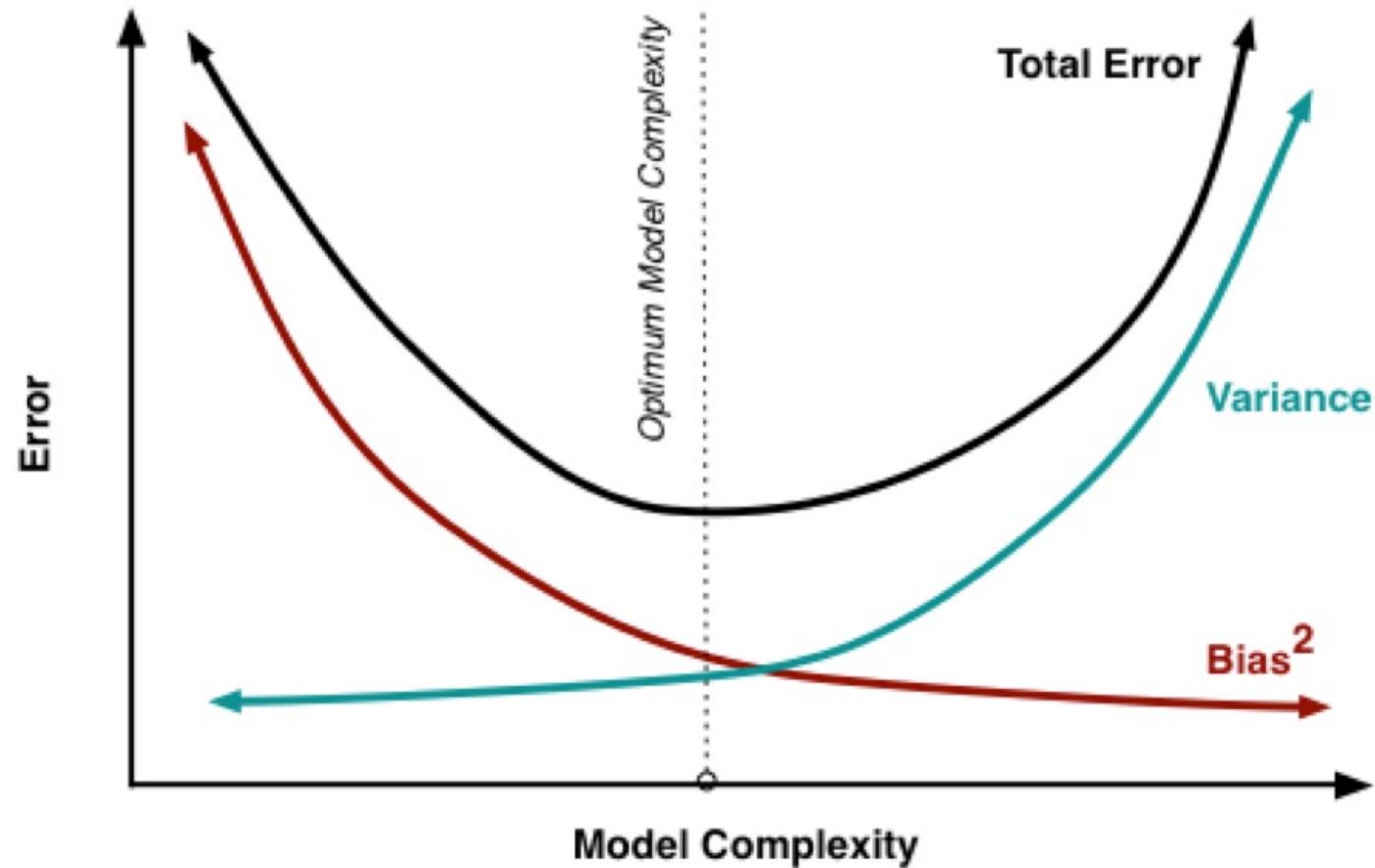
Bias and Variance trade-off (cont.d)

In order to reduce the Generalization Error, we should reduce the bias or the variance:

- Bias represents how much the best estimation is able to get close to the ground truth. The high bias for a specific model denotes a too-simple model that is not able to predict, whatever Dataset you are training on.
- Variance represents how much a single hypothesis can be different from the best estimation. The high variance means that the same model, trained with different datasets, generates very different hypotheses. When the hypotheses are very complex, they are prone to overfit and they are not able to generalize.

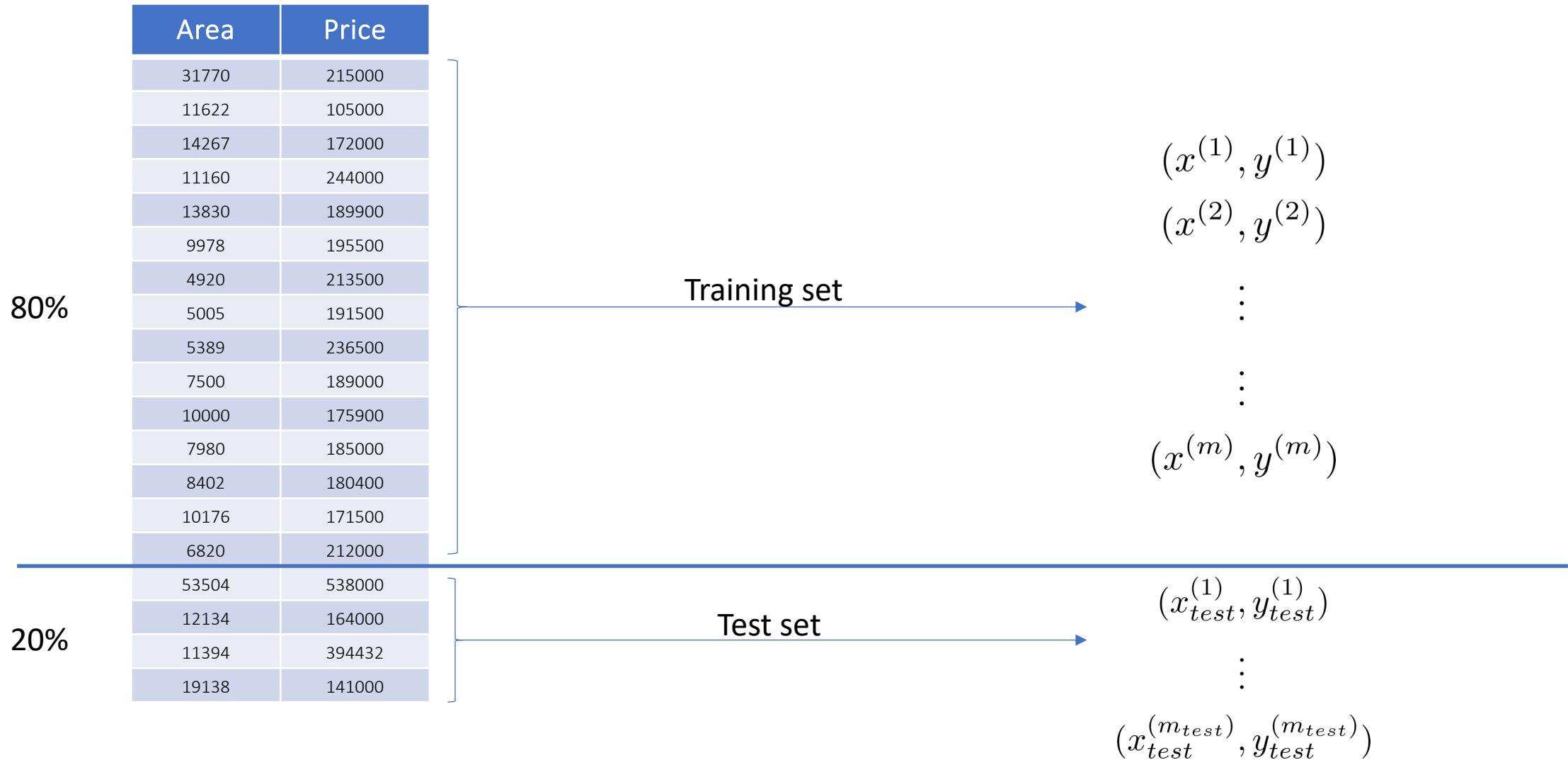
The optimum would be to maintain low bias and variance, but, a model given, the bias and the variance behave at the opposite w.r.t. the complexity of the model.

Bias and Variance w.r.t. complexity

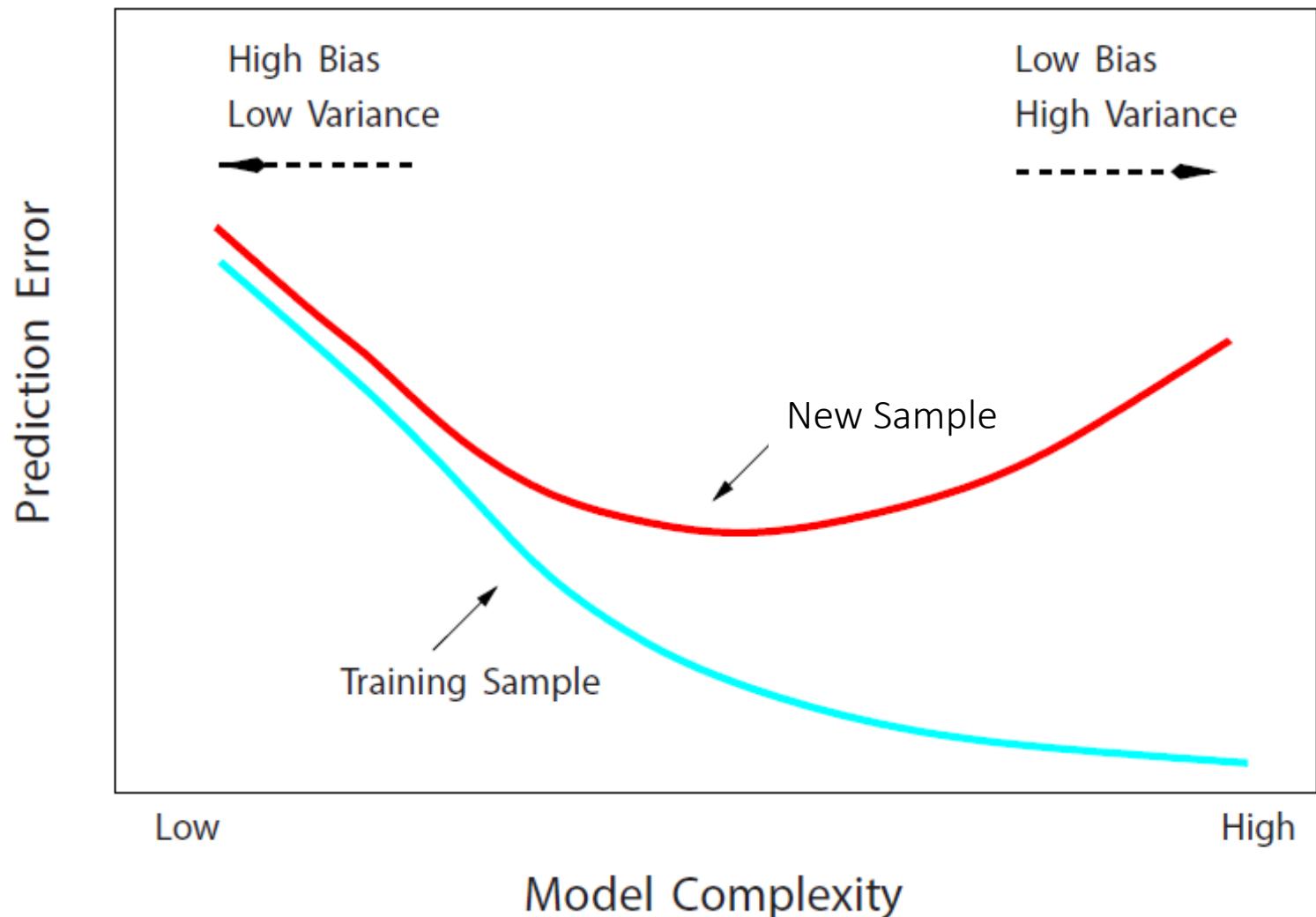


The best trade-off is choosing the model complexity that show the minimum sum between bias and variance

Evaluating our hypothesis



Training set and test set w.r.t. complexity



Fitting

Regularization

Addressing overfitting:

Options:

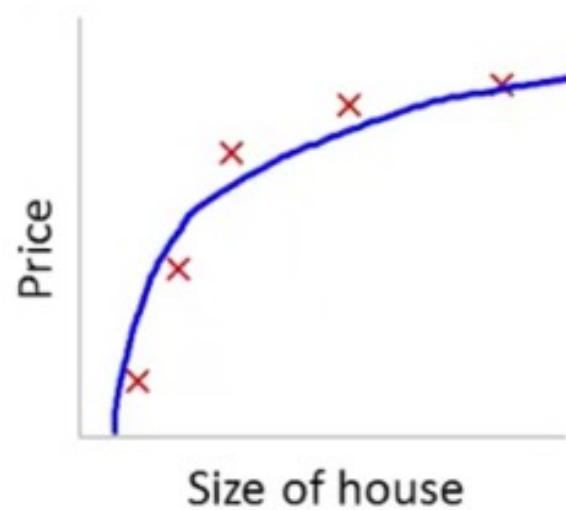
1. Reduce number of features

- Manually select which features to keep
- Model selection algorithm

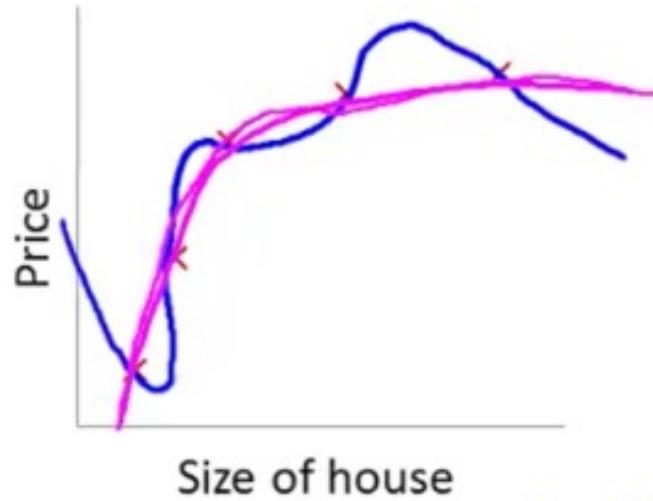
2. Regularization

- Keep all the features, but reduce magnitude/values of the parameters
- Works well when we have a lot of features, each of which contributes a bit to predicting y

Regularization Intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

We want to reduce the contribution of the high degree terms of the polynomial to make the curve smoother. If we added an heavy term to the high degree parameters we are imposing the minimization function to make them really small:

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

$$\theta_3 \approx 0 \quad \theta_4 \approx 0$$

Regularization

If we have small values for the parameters leads to have Simple hypothesis that is less prone to overfitting

$$\begin{aligned}
 J(\theta) &= \frac{1}{2m} \left(\sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2 \right) \\
 \downarrow & \\
 J(\theta) &= \frac{1}{2m} \left(\sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2 + \lambda \|\theta\|_2^2 \right) \quad \text{Squared L2-Norm} \\
 \downarrow & \\
 J(\theta) &= \frac{1}{2m} \left(\sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right) \quad \theta_0 \text{ not included}
 \end{aligned}$$

Lambda is the regularization parameter that controls the influence of regularization w.r.t. the hypothesis:

The bigger is, the smaller will be the theta and smoother will be the curve

Regularization for regression

In regularized linear regression, we choose to minimize

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

What if lambda is set to an extremely large value?

- Algorithm works fine; setting to be very large can't hurt it.
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit well even training data).
- Gradient descent will fail to converge.

Regularization for regression

In regularized linear regression, we choose to minimize

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

What if lambda is set to an extremely large value?

- Algorithm works fine; setting to be very large can't hurt it.
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

Fitting

Regularized linear regression

Regularization for linear regression

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

We want to derive the cost function to modify the update rule in the gradient descent:

Repeat until convergence {

$$\begin{cases} \theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)} \\ \theta_j = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad j = 1, 2, \dots, n \end{cases}$$

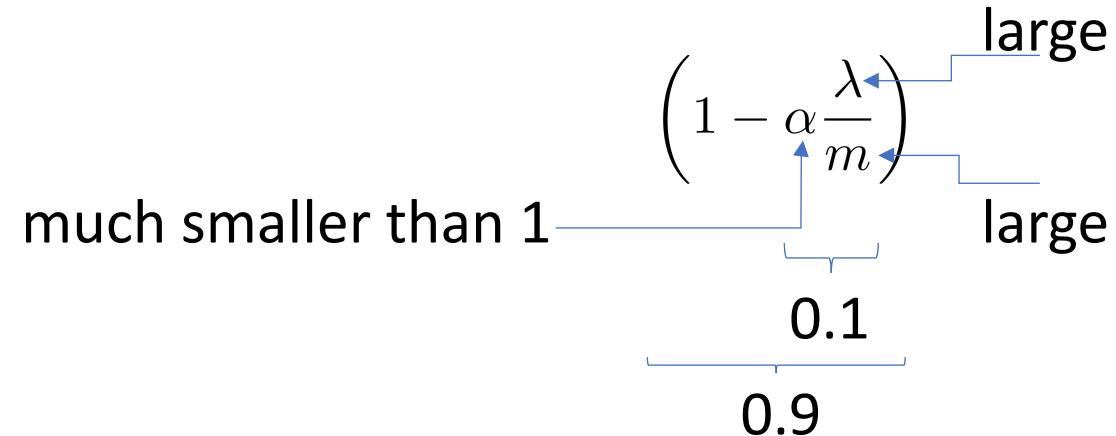
}

Regularization for linear regression

The update function can be re-written as

$$\theta_j = \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \quad j = 1, 2, \dots, n$$

It is worth to note that the second part is same as before the regularization but we update the theta moving from a ratio of the old theta smaller than 1:



Fitting

Regularized logistic regression

Regularization for logistic regression

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

We want to derive the cost function to modify the update rule in the gradient descent:

Repeat until convergence {

$$\begin{cases} \theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) x_0^{(i)} \\ \theta_j = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m \left(h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad j = 1, 2, \dots, n \end{cases}$$

}

Fitting

Regularization with L1 norm

Regularization with L1-norm

It is possible to regularize the cost function using L1 norm:

$$\ell^1 = \sum_{j=1}^n |\theta_j|$$

But L1-norm is no more derivable and standard gradient descent cannot be used.

Why should we consider (in some very specific cases) to use it?

Let us focus on the minimization of the norms:

$$\begin{aligned} \text{Norm } \ell^2 : \quad & \min \sum_{i=1}^m (\dots)^2 \\ \text{s.t.} \quad & \|\theta\|_2^2 < t \end{aligned}$$

$$\begin{aligned} \text{Norm } \ell^1 : \quad & \min \sum_{i=1}^m (\dots) \\ \text{s.t.} \quad & \|\theta\|_1 < t \end{aligned}$$

Limiting our analysis to two parameters

$$\begin{aligned} \text{Norm } \ell^2 : \quad & \min \sum_{i=1}^m (\dots)^2 \\ \text{s.t.} \quad & \theta_1^2 + \theta_2^2 < t \end{aligned}$$

$$\begin{aligned} \text{Norm } \ell^1 : \quad & \min \sum_{i=1}^m (\dots) \\ \text{s.t.} \quad & |\theta_1| + |\theta_2| < t \end{aligned}$$

Regularization with L1-norm

$\sum(\dots)^2$ denotes a parabolic curve equation;

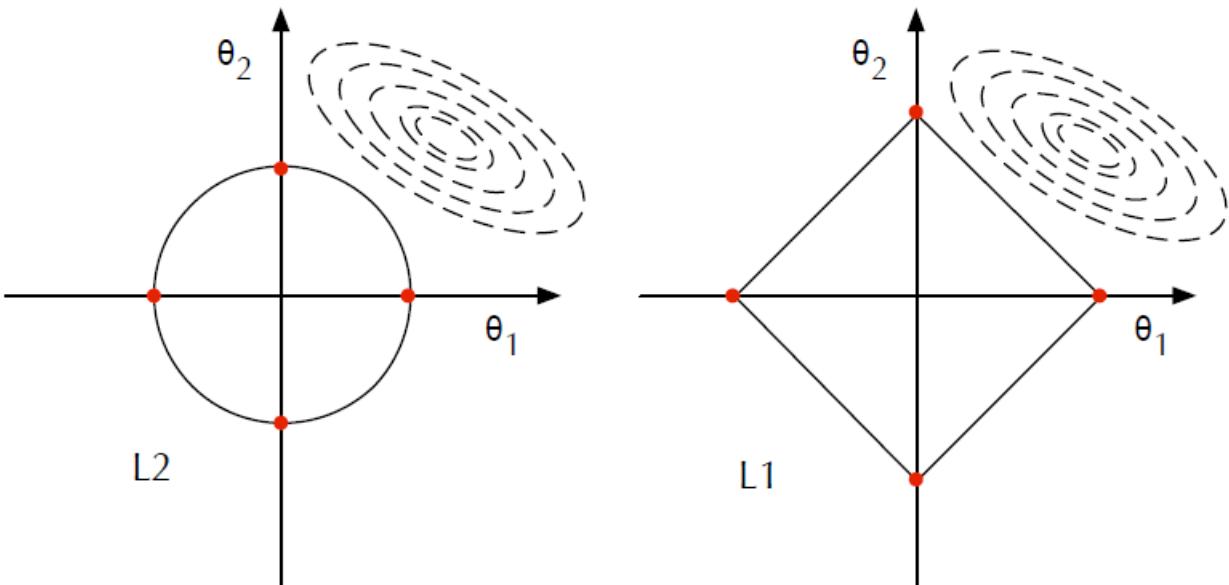
$\theta_1^2 + \theta_2^2 < t$ denotes the internal area of a circle;

$|\theta_1| + |\theta_2| < t$ denotes the internal area of a rhombus.

During the minimization we want to minimize that parabolic curve and lie inside the circle or the rhombus.

It can be proved that it is easier, in the L1 case, to reach the intercept with the axis and hence to set that parameter to zero.

This lead to a simple but effective feature selection.

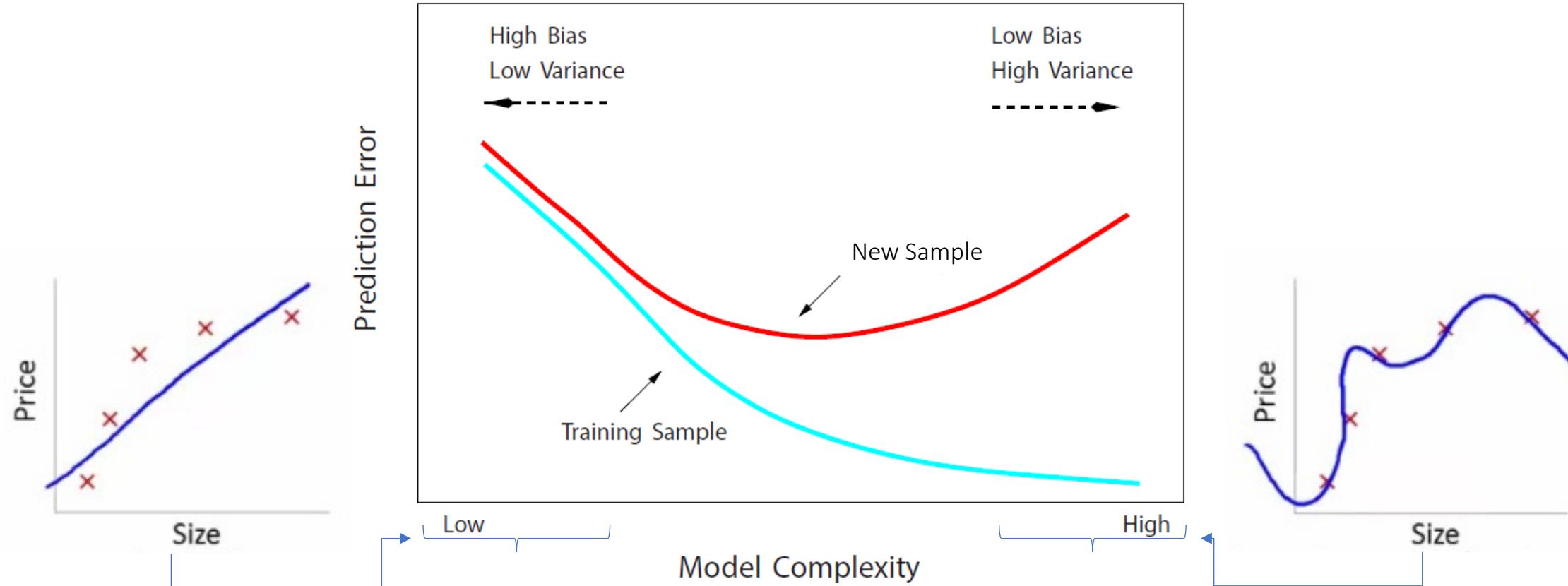


Fitting

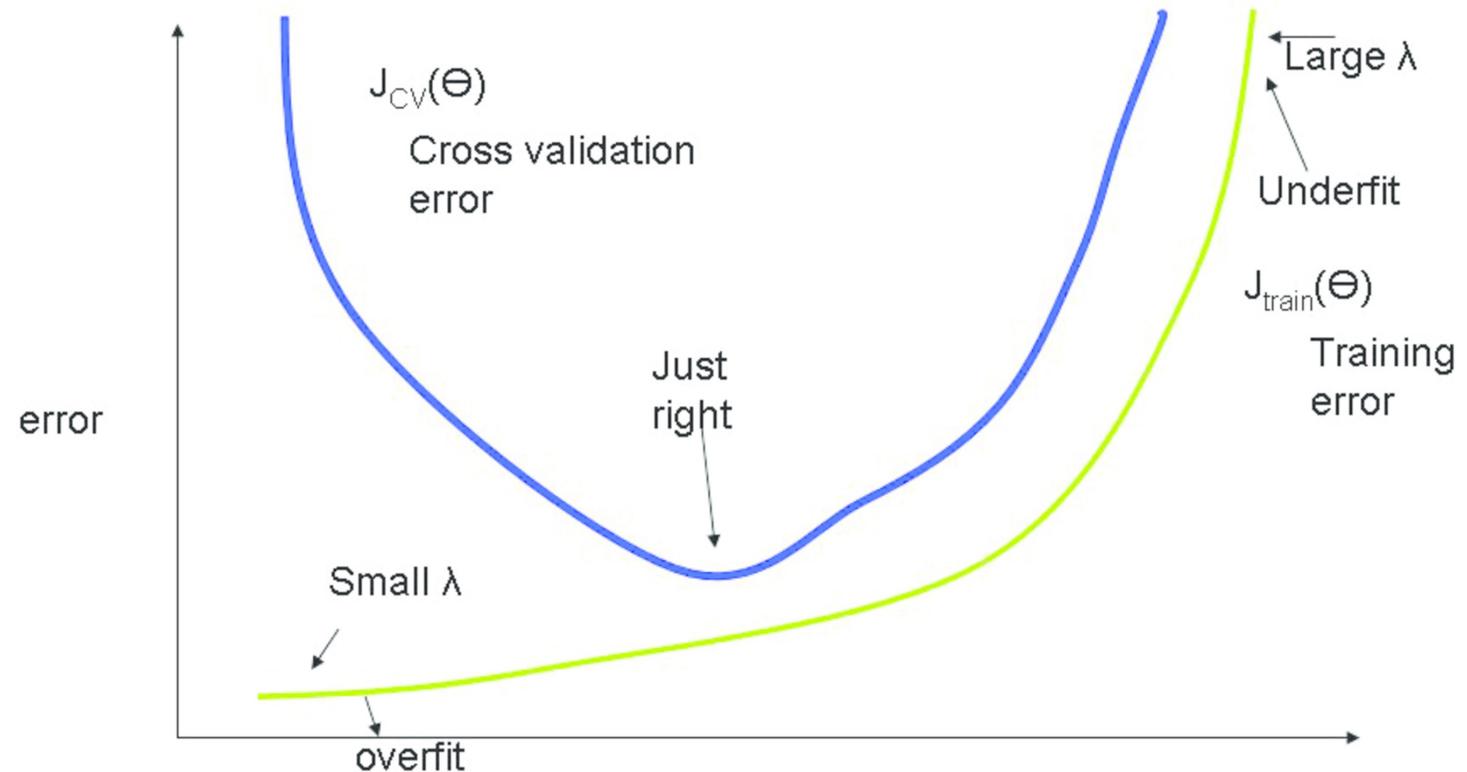
Regularization vs Bias/Variance

Training set and test set w.r.t. complexity

Let us recap the Bias variance behavior w.r.t. the complexity of the model



Training set and new samples w.r.t. regularization



How to build a ML System

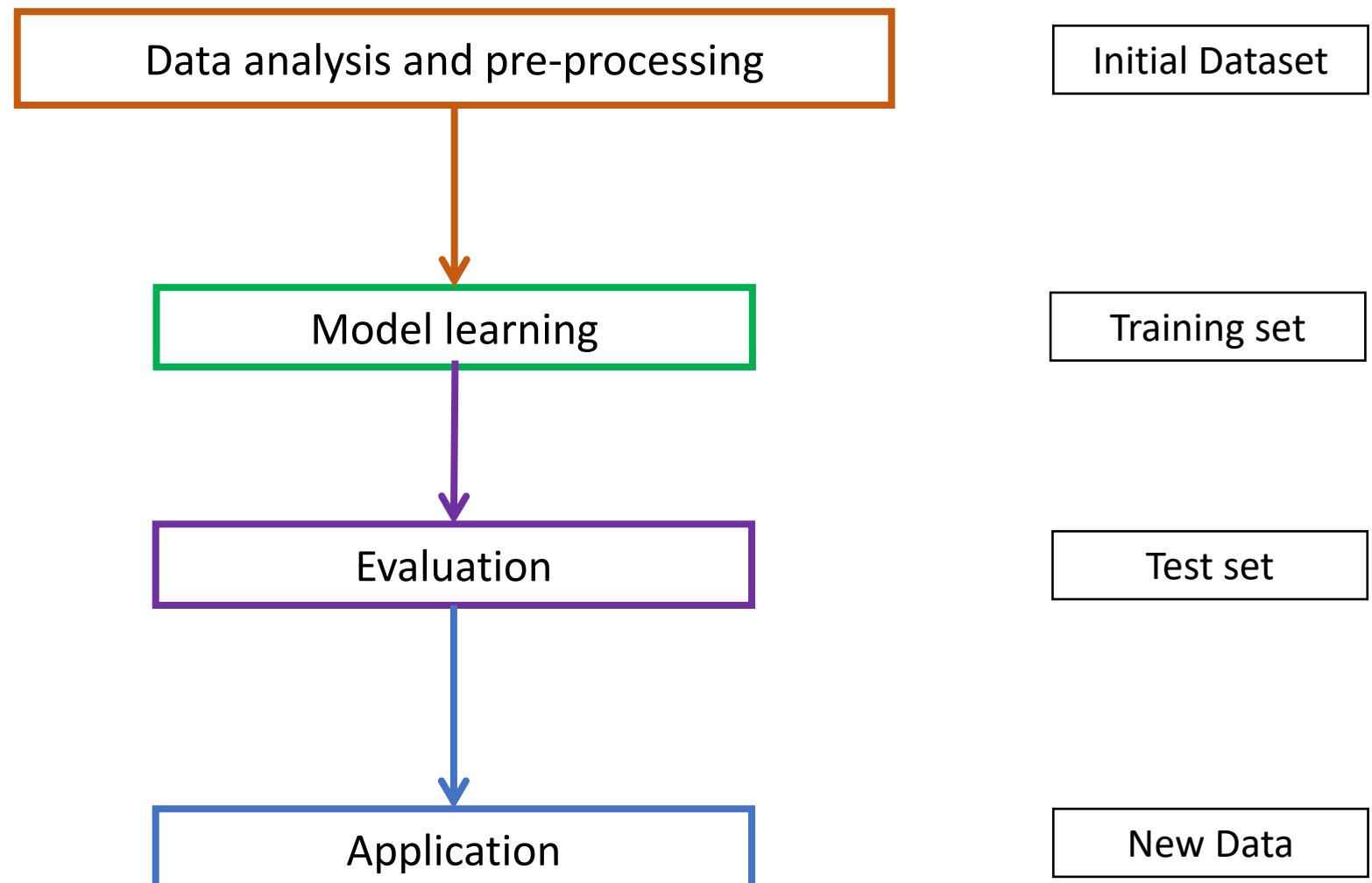
Overview

Machine Learning System

Learning grounds on three parts: **representation + optimization + evaluation**

- Representation: identify the hypothesis space of the learner and decide which features to use to represent data
- Optimization: choose the method to train the learner (e.g. Gradient descent, greedy search);
- Evaluation: choose an evaluation function (scoring/objective function) to distinguish a good from a bad learner.
- Goal: generalize well from seen examples (training set) on unseen examples.

ML system life cycle (for starters)



How to build a ML System

What we do first

Data analysis and pre-processing

Real word data are dirty:

- Incomplete: the value of some attributes is missing or interesting attributes are completely missing.
- Inaccurate: data contain wrong values deriving from inaccurate or partial observations

GIGO: garbage in – garbage out

An accurate data analysis and cleaning is required. This pre-processing phase takes generally a lot of time.

Data analysis and pre-processing

Pre-processing:

- Cleaning of data: outliers removing, noise removing, duplicates removing
- Changing data:
 - Discretize
 - Aggregate
 - Normalization and re-scaling
- Creating new attributes

Outlier removal example (boxplot)

Quartiles, deciles, and percentiles divide the data set into equal parts

Fractiles	Summary	Symbols
Quartiles	Divide a data set into four equal parts.	Q1 , Q 2, Q3
Deciles	Divide a data set into ten equal parts.	D1 , D2 , D3 ...D9
Percentiles	Divide a data set into one hundred equal parts.	P1, P2, P3 ... P99

QUARTILES:

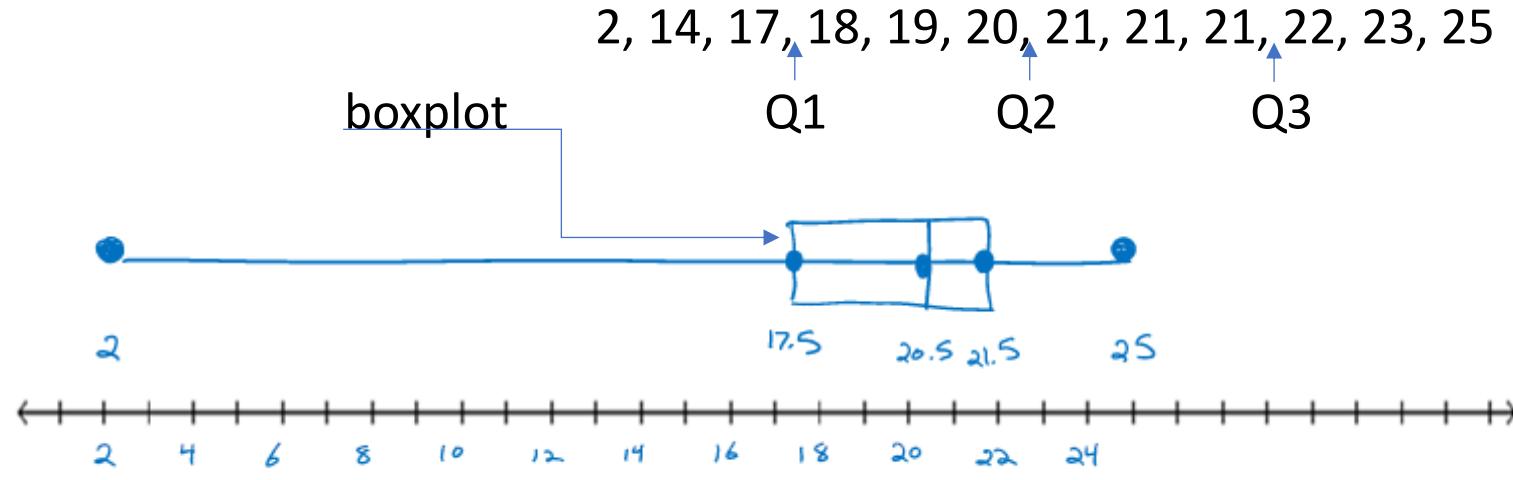
Q2 is equal to the median of the data set.

Q1 is the median of the values that are below Q2 .

Q3 is the median of the values that are above Q2 .

Outlier removal example (boxplot)

Let us see an example:



$$Q_2 = \frac{20 + 21}{2} = 20.5$$

$$Q_1 = \frac{17 + 18}{2} = 17.5$$

$$Q_3 = \frac{21 + 22}{2} = 21.5$$

The interquartile range is defined as $IQR = Q_3 - Q_1$, it tells us the spread of the middle 50% of the data. (in this case $IQR = 21.5 - 17.5 = 4$)

The interquartile range can be used to identify outliers in a data set. If a data value is less than $Q_1 - 1.5 * IQR$ or greater than $Q_3 + 1.5 * IQR$, it is considered an outlier.

In this case, the lower fence is $17.5 - 1.5 * 4 = 11.5$ hence the value "2" < 11.5 (It is an outlier)

Normalization (recap)

min-max normalization:

$$x = ((x - \text{min}) / (\text{max} - \text{min})) * (b - a) + a$$

z-score normalization:

$$x = (x - \text{mean}(x)) / (\text{dev_std}(x))$$

Feature Selection

Selection of relevant features for the learning task.

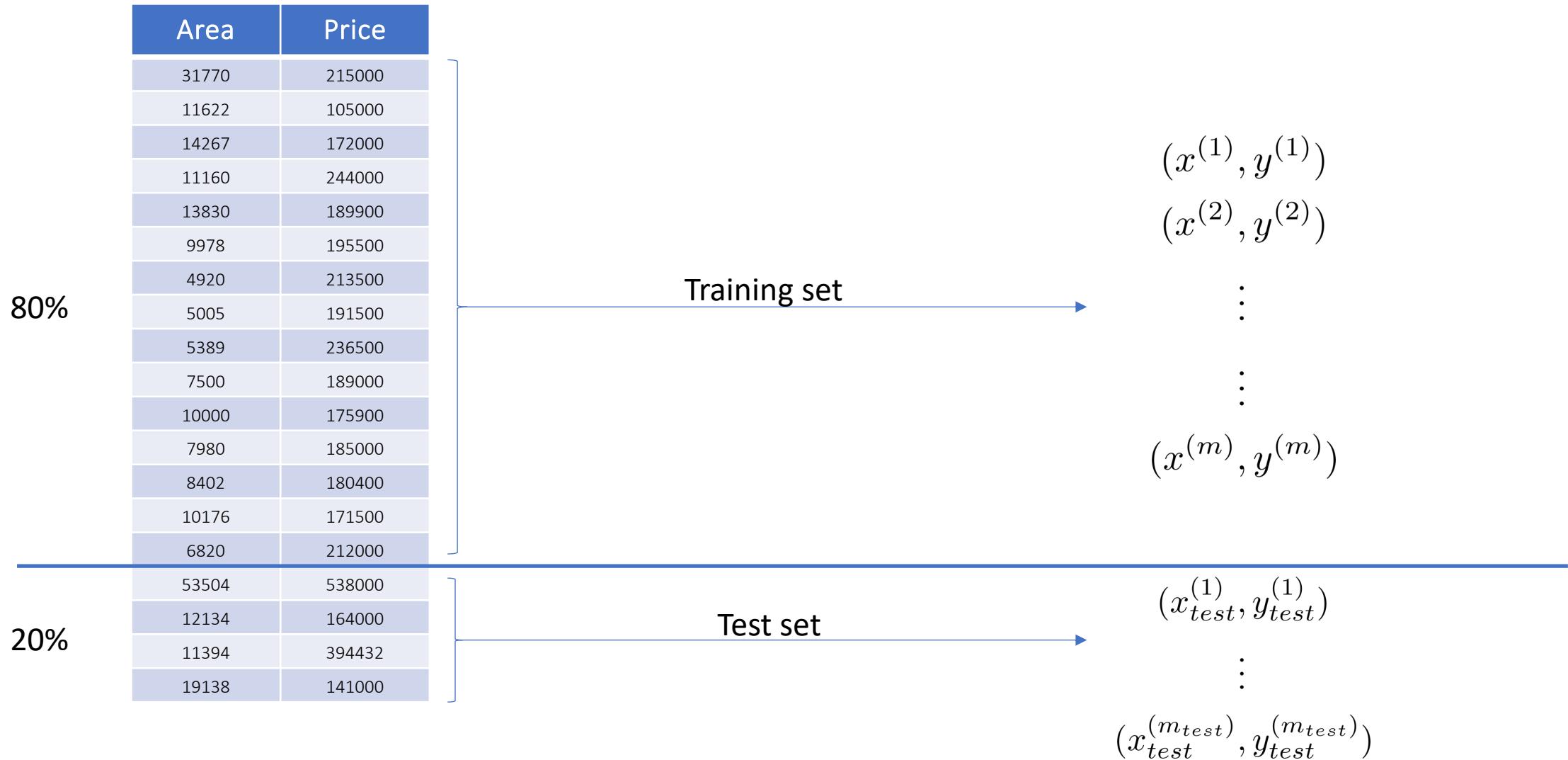
Sometimes there are many features, some of them could be redundant or not really important

- Domain experts
- Filter: measure the importance of each feature to discriminate across the classes. Es. Information Gain, Entropy, Mutual Information
- Wrappers: iterative method to find a good subset of features using a subset
- Dimensionality reduction (PCA, SVD)

How to build a ML System

How to evaluate the hypothesis

Evaluating our hypothesis



Training/testing procedure for linear regression

- Learn parameters θ from training data (minimizing training error $J(\theta)$)
- Compute the test set error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Training/testing procedure for logistic regression

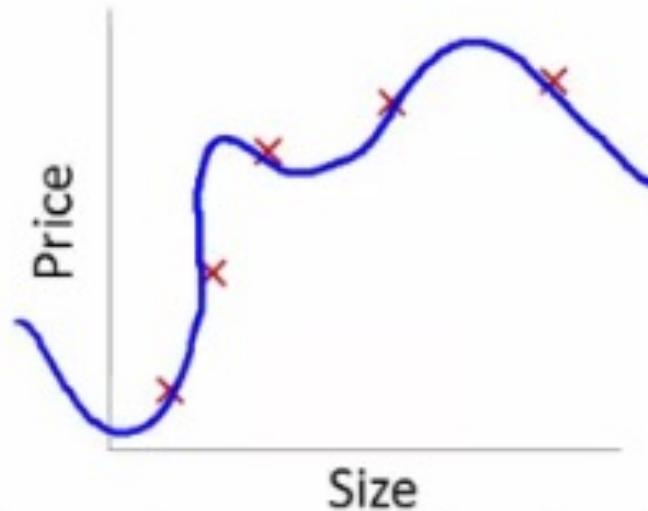
- Learn parameters θ from training data (minimizing training error $J(\theta)$)
- Compute the test set error:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m_{test}} \left(y^{(i)} \log h_\theta(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(\mathbf{x}^{(i)})) \right).$$

The generalization error in practice

- The overfitting example

Once we trained our model to overfit, we know that the error on the training set will be very low but our algorithm does not generalize well.



We can suppose that the actual generalization error is higher than the one we measured on the training data.

This is the reason why we need to evaluate the hypothesis on a portion of data NEVER USED BEFORE.

Model selection

Imagine now that we want to compare different hypothesis in which we change the degree of the polynomial.

$$d1 \quad 1. \quad h_{\theta}(x) = \theta_0 + \theta_1 x \rightarrow \theta^{(1)} \rightarrow J_{test}(\theta^{(1)})$$

$$d2 \quad 2. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{test}(\theta^{(2)})$$

$$d3 \quad 3. \quad h_{\theta}(x) = \theta_0 + \dots + \theta_3 x^3 \rightarrow \theta^{(3)} \rightarrow J_{test}(\theta^{(3)})$$

$$\vdots \quad \vdots$$

$$d10 \quad 10. \quad h_{\theta}(x) = \theta_0 + \dots + \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow J_{test}(\theta^{(10)})$$

Imagine the 5 degree polynomial show the lowest cost function $J_{test}(\theta^{(5)})$.

Does our model generalize well?

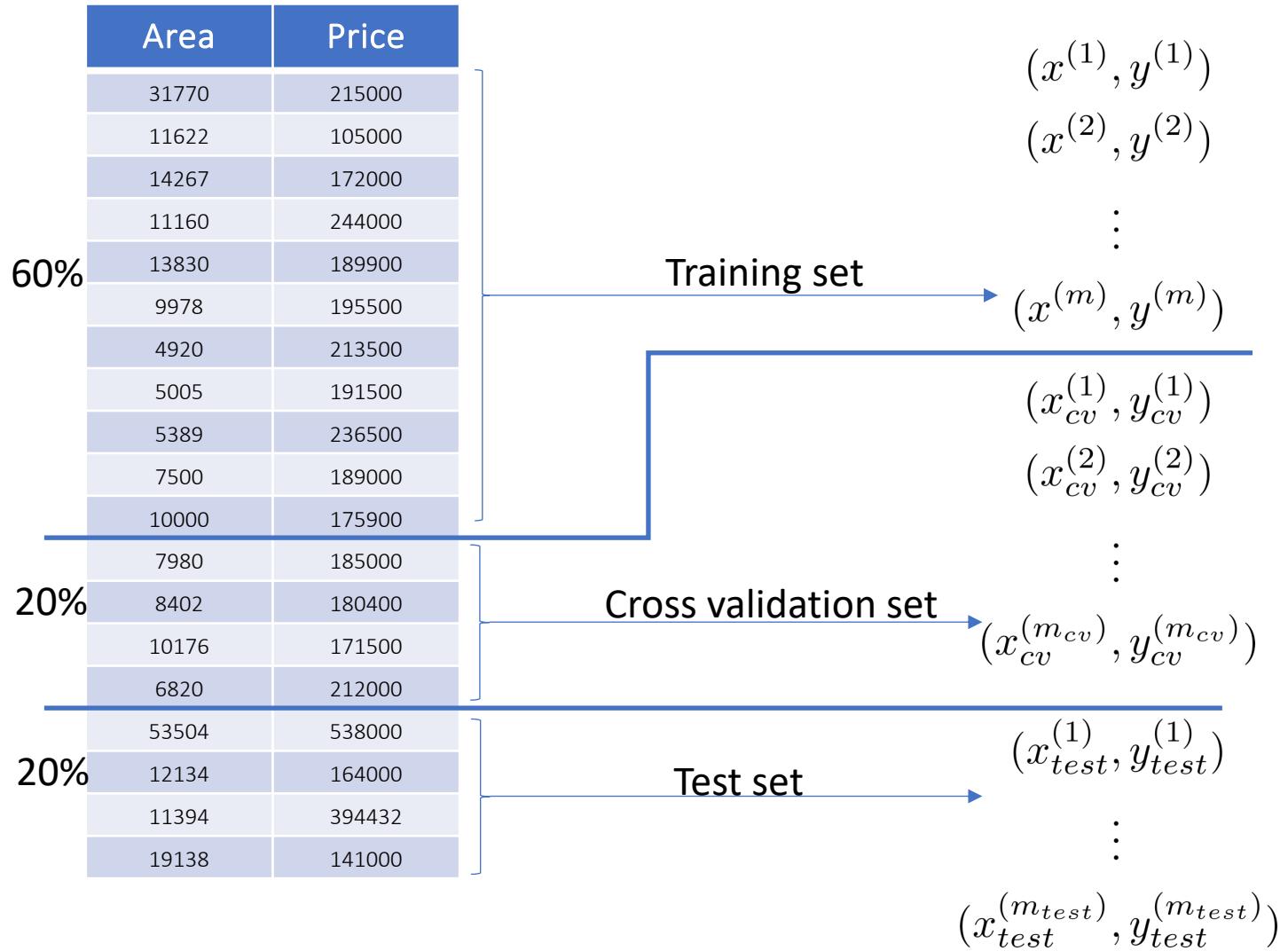
The hypothesis we chose is the one that guarantees the best performance on that specific test set (and for this reason is likely to be an optimistic estimation of the generalization error).

NOW WE NEED ANOTHER TEST SET...

Model Selection: What is the best model?

How can we find it?

Cross Validation
 Using a part of the
 Training Set as
 Validation Set



Data splitting

Given a dataset, we can split it in three sets:

- **Training set**, to train the models.
- **Validation set**, to choose the best model. It is also useful for the feature selection and set the hyper parameters.
- **Test set**, to evaluate the best model.

Model Selection (recap)

Can we choose the model that gives the best performance on the Training Set? NO!

- We risk overfitting: the model will not be able to generalize.
- The error on the training data is an underestimation of the generalization error.

Model Selection (recap)

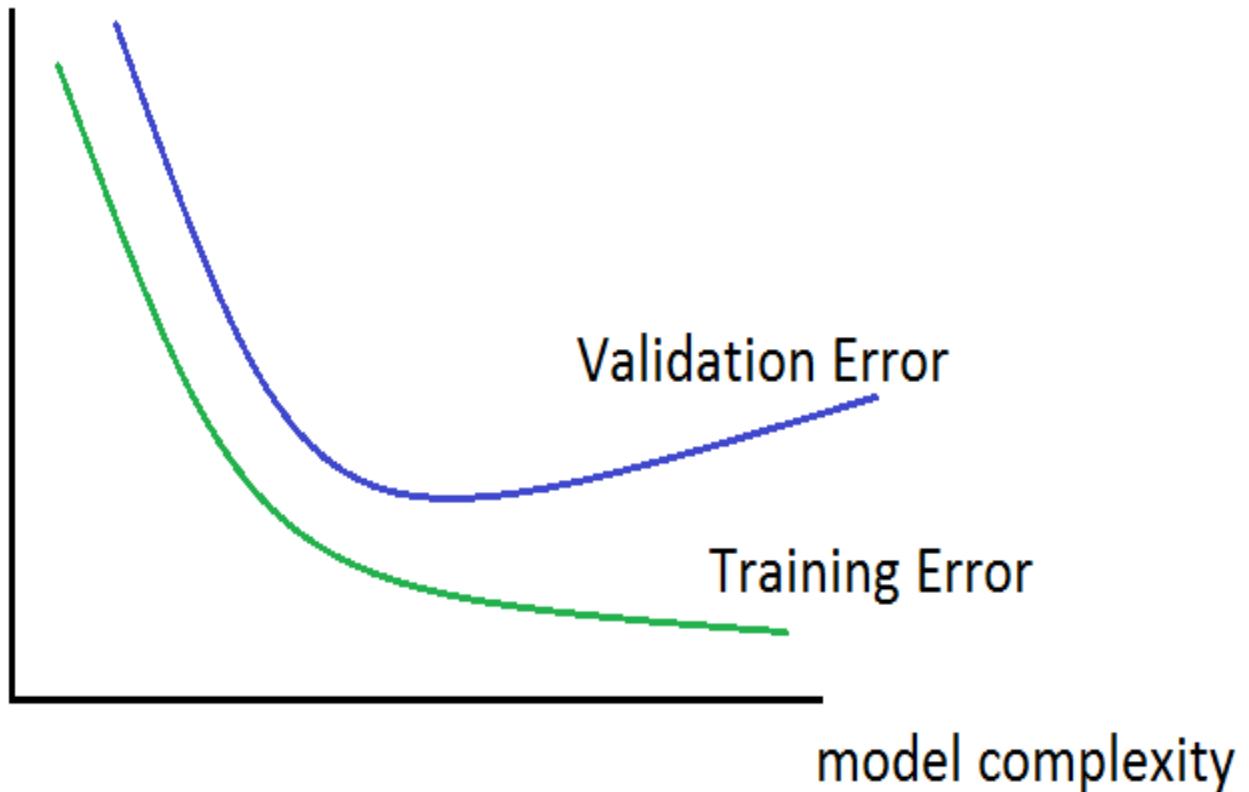
Can we choose the model that gives the best performance on the Validation Set? YES!

- We must train the model on the training set and then validate it on the validation set.
- In this way, we evaluate the generalization error on unseen examples.

Cross Validation

- Usually, the validation set is built by a 1/3 of training set (**hold-out cross validation**) or we can use a **k-folds cross validation**.
- Once we choose a model, it can be trained again on the whole training set
- Cross validation can also be used to find the hyper-parameters. For example, the regularization factor.
- Or it can be used for the selection of a subset of relevant features.

Cross Validation



Model selection

Imagine now that we want to compare different hypothesis in which we change the degree of the polynomial.

$$d1 \quad 1. \quad h_{\theta}(x) = \theta_0 + \theta_1 x \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$$

$$d2 \quad 2. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$$

$$d3 \quad 3. \quad h_{\theta}(x) = \theta_0 + \dots + \theta_3 x^3 \rightarrow \theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})$$

$$\vdots \quad \vdots$$

$$d10 \quad 10. \quad h_{\theta}(x) = \theta_0 + \dots + \theta_{10} x^{10} \rightarrow \theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})$$

Imagine the 5 degree polynomial show the lowest cost function for the cross validation set.

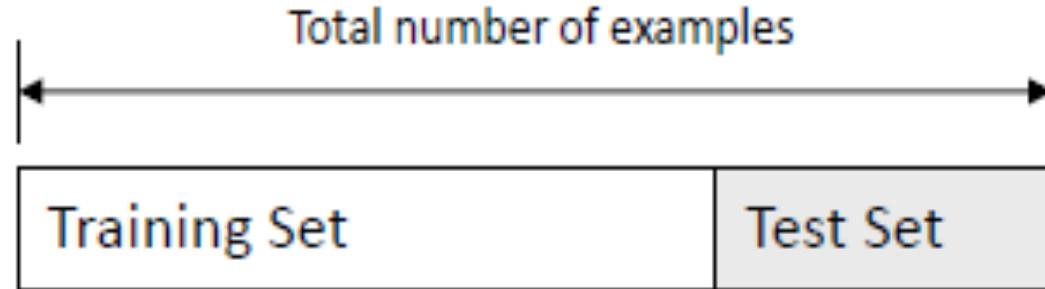
Now we can evaluate the generalization error of the 5 degree polynomial

on the test set data ($J_{test}(\theta^{(5)})$)

Model evaluation

- To finally evaluate the model, we need a Test Set: data not in Training and Validation sets.
- Evaluating the model on a test set is useful to see if the model really generalize well enough
- It is not correct to use the test set for choosing a model for an algorithm. You must use the test set only when the model is chosen and you want to evaluate it
- As we have seen before, there are two ways to split a dataset in Training and Test sets:
 - Holdout
 - K-fold cross validation

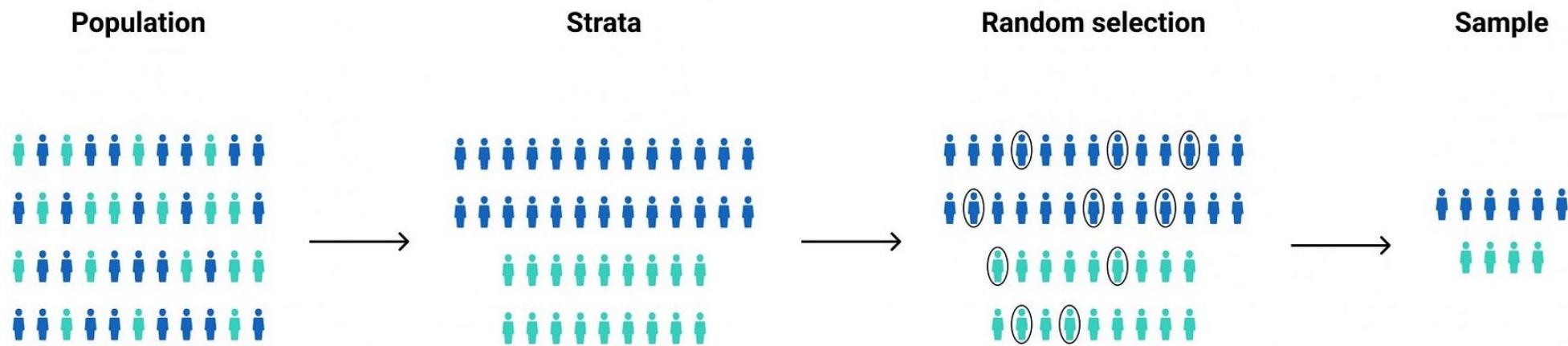
Holdout



Drawback : The choice of training and test sets can influence training and test. For instance, a class is not represented equally in the two sets

Solution: We can use the ***stratification in order to better distribute*** the classes on the two sets

Stratification



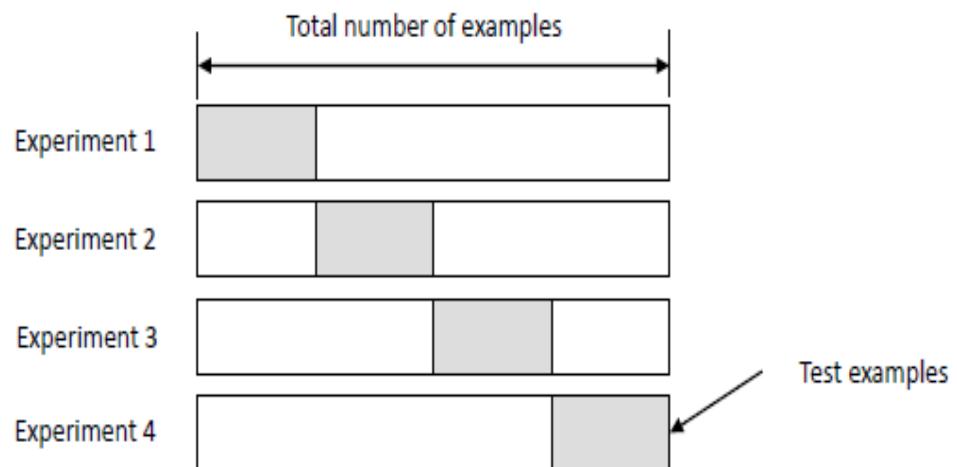
Limitations of using a single training/test partition

- We may not have enough data to make sufficiently large training and test sets
 - a larger test set gives us more reliable estimate of accuracy (i.e., a lower variance estimate)
 - but... a larger training set will be more representative of how much data we actually have for learning process
- A single training set doesn't tell us how sensitive accuracy is to a particular training sample

K-folds Cross validation

The original dataset is randomly partitioned into k equal sized subsets (folds). Of the k folds, a single fold is retained as the test set, and the other folds are used together as training set. The cross-validation process is then repeated k times, with each of the k folds used exactly once as test set. The k results from the folds can then be averaged to produce a single estimation.

10-folds cross validations is usually used



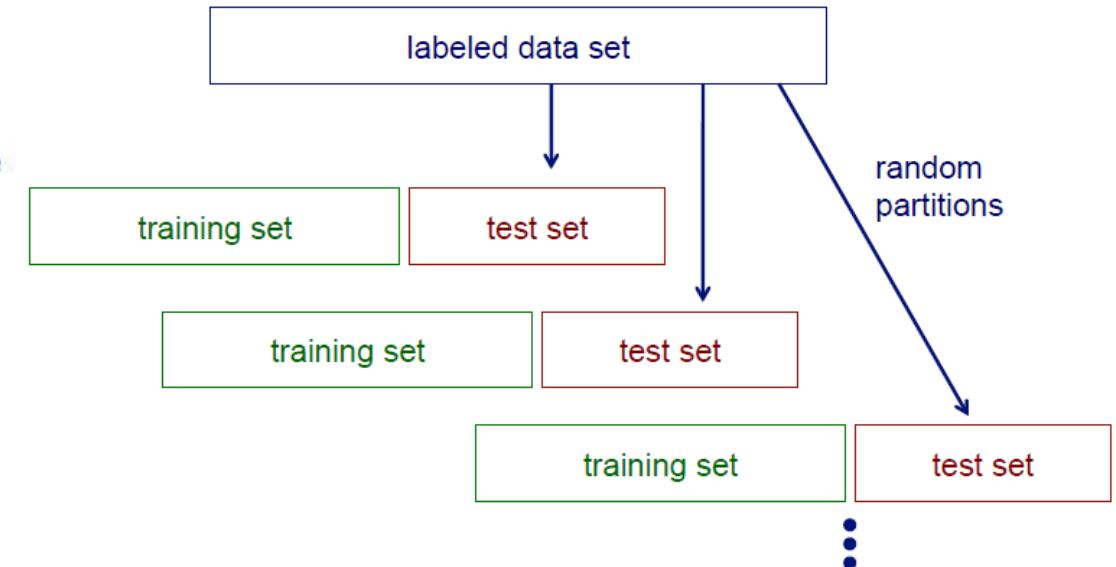
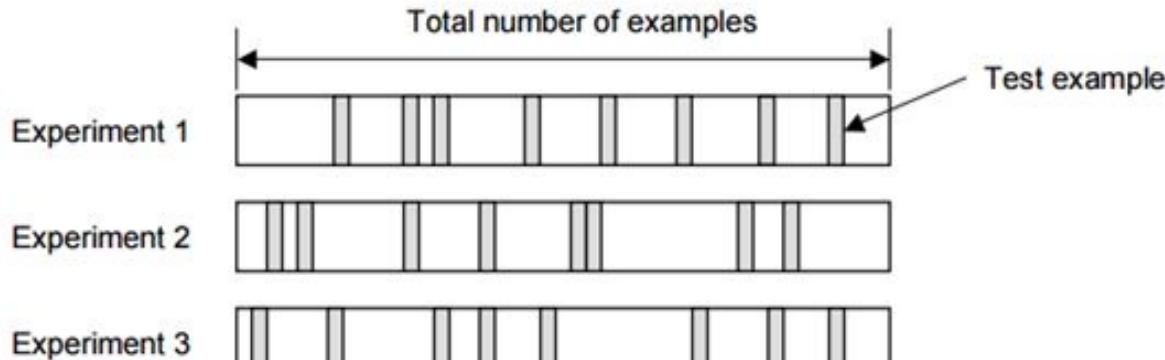
Advantage: all observations are used for training and testing, and each observation is used for testing exactly once.

Random subsampling

We can address the second issue by repeatedly randomly partitioning the available data into training and set sets.

In details random subsampling performs K data splits of the dataset

- Each split randomly selects fixed number of examples **without replacement**
- For each data split we retrain the classifier from scratch with the training examples and estimate the Errors with the test samples

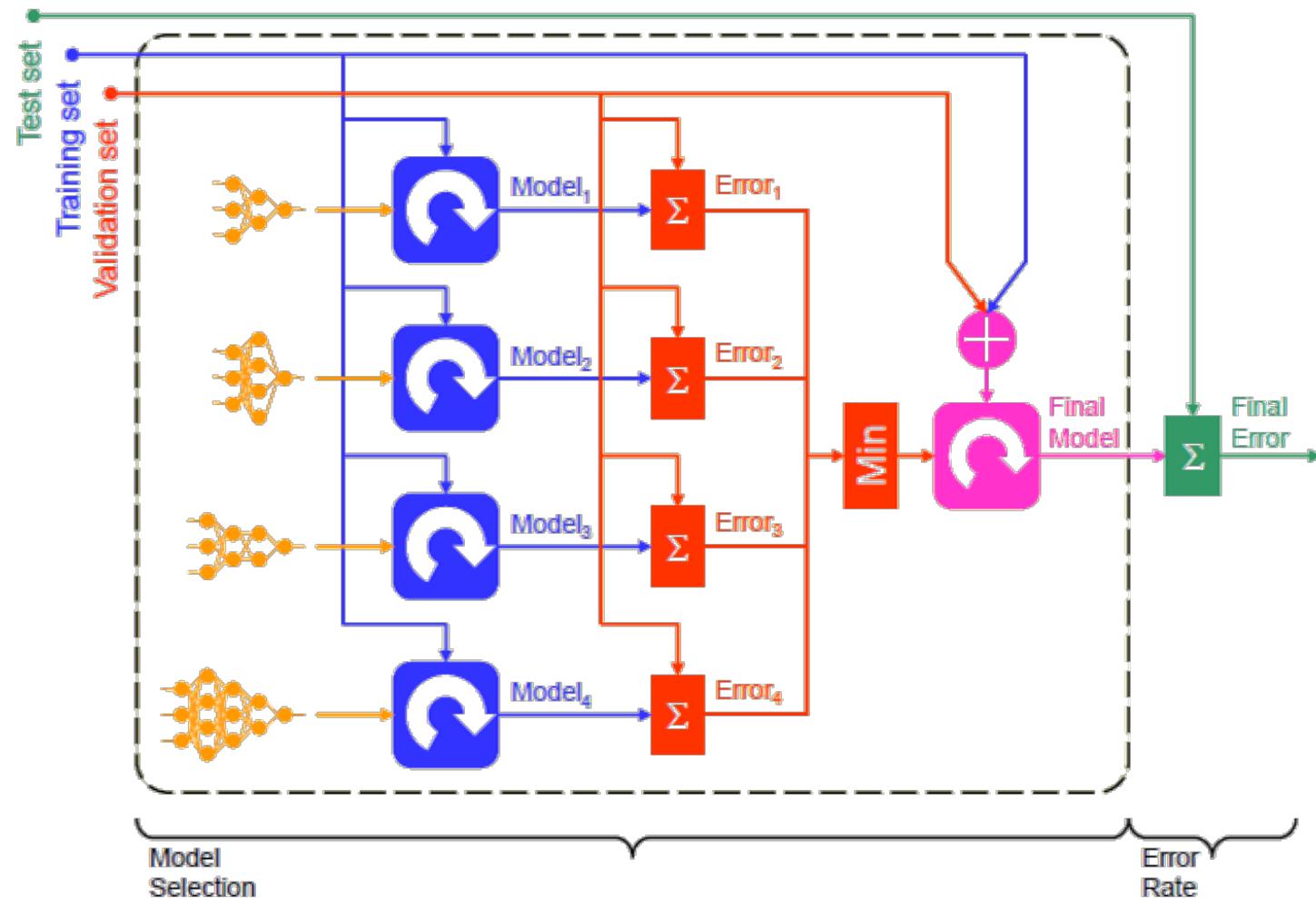


Error estimate on K-folds

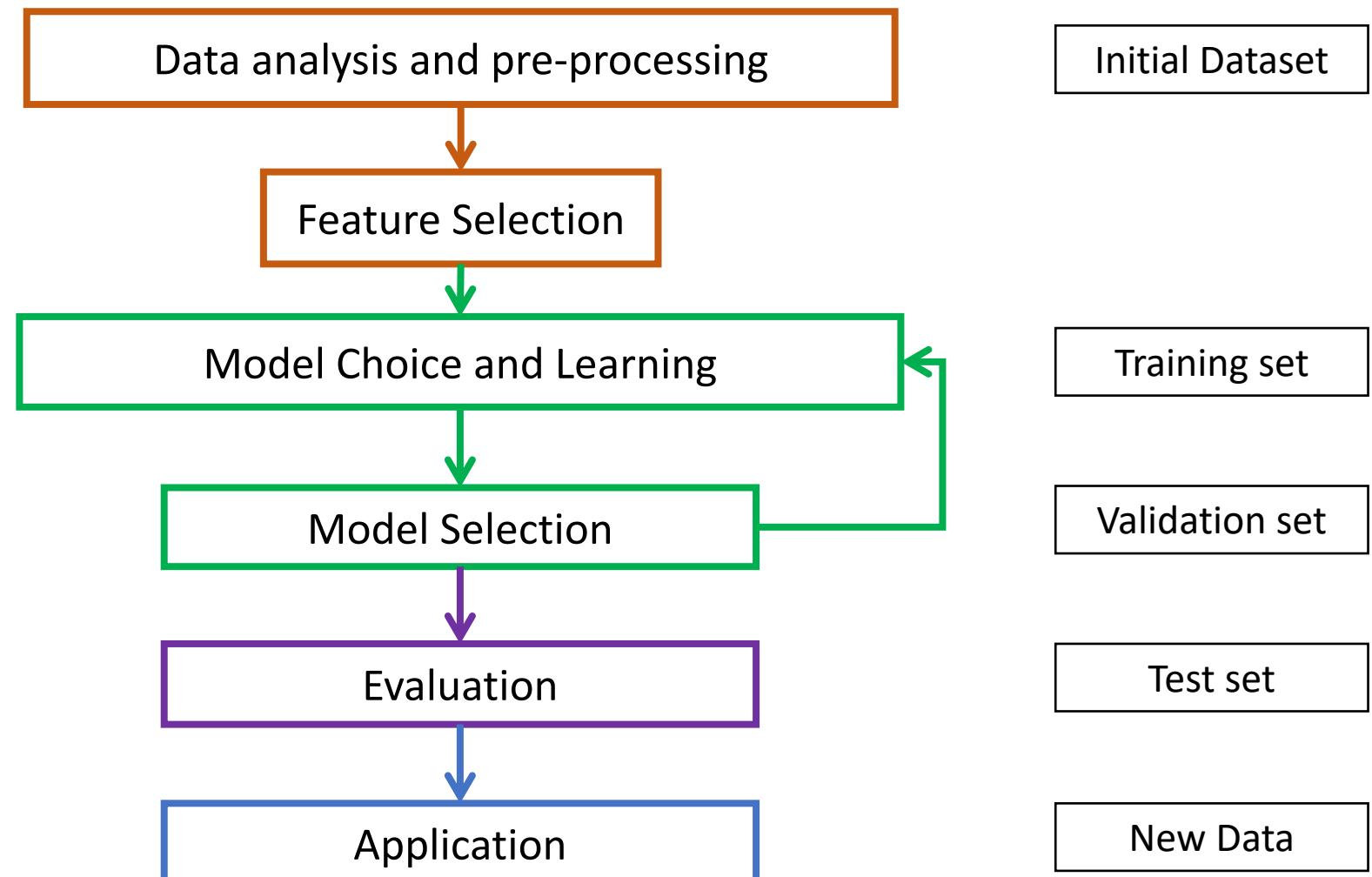
Case i	Train on					Test on	Error
Case1		F2	F3	F4	F5	F1	1.5
Case2	F1		F3	F4	F5	F2	0.5
Case3	F1	F2		F4	F5	F3	0.3
Case4	F1	F2	F3		F5	F4	0.9
Case5	F1	F2	F3	F4		F5	1.1

$$Error = \frac{1}{k} \sum_{i=1}^k Error_i$$

Cross-Validation overview



ML system life cycle (revisited)



How to build a ML System

How to choose the next hypothesis

Debugging a learning algorithm:

Suppose you have implemented a regularized linear regression to predict housing prices.

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

When you test your hypothesis on a new set of houses, you find that your regressor performs very bad with very large errors. What should you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features
- Try decreasing λ
- Try increasing λ

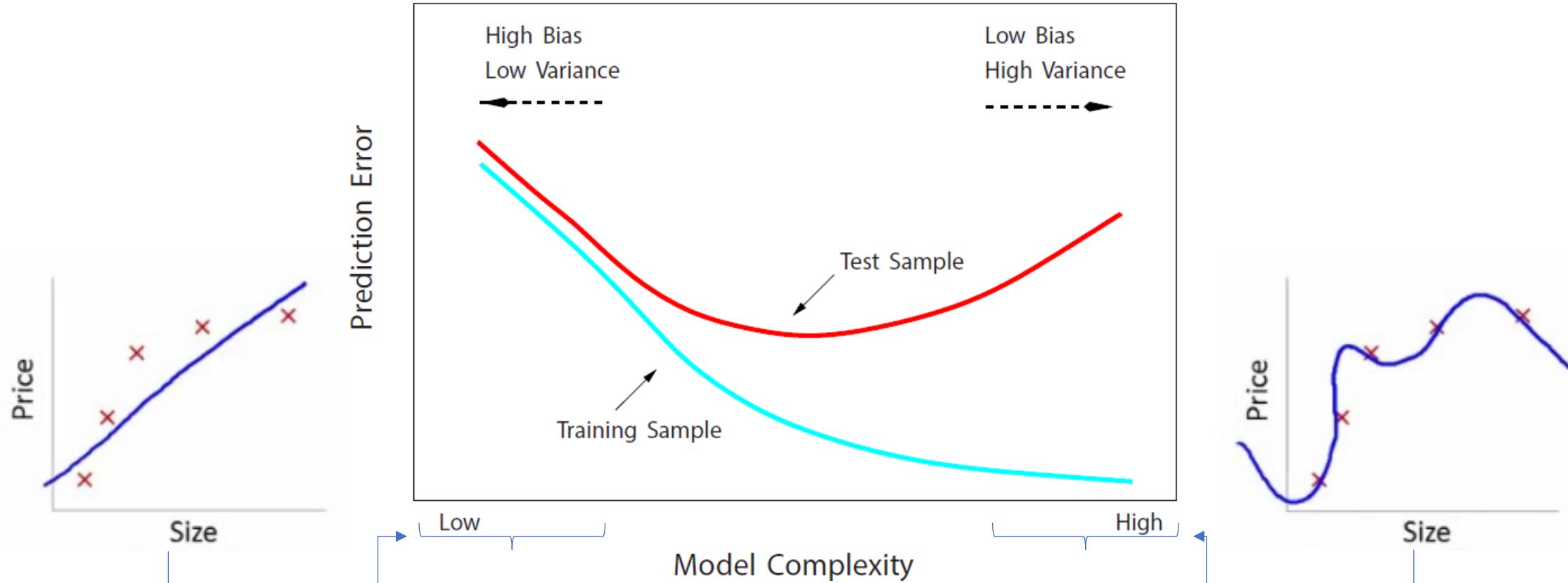
Machine learning diagnostic

Diagnostic: A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

Diagnostics can take time to implement, but doing so can be a very good use of your time.

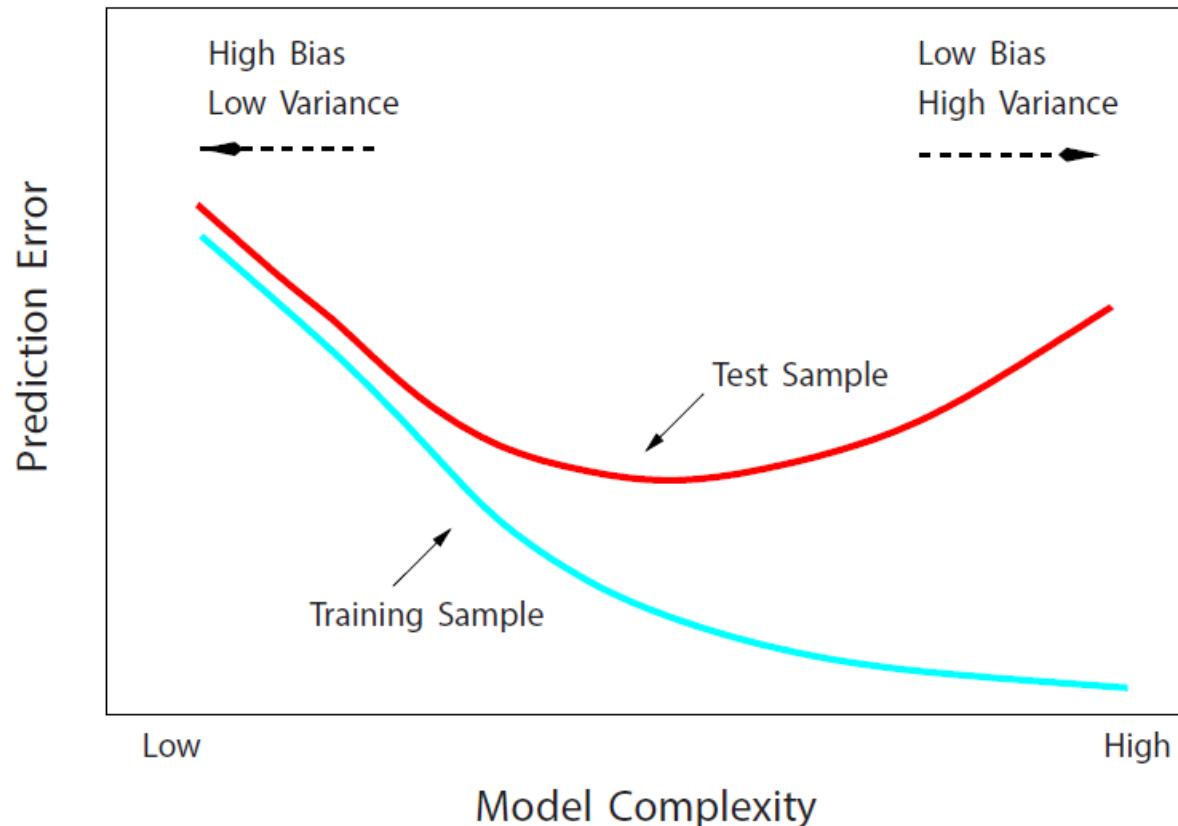
Diagnosing bias/variance

Let us recap the Bias variance behavior w.r.t. the complexity of the model



Diagnosing bias/variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high) Is it a bias problem or a variance problem?



Bias (underfit):

$$J_{train}(\theta) \text{ will be high}$$

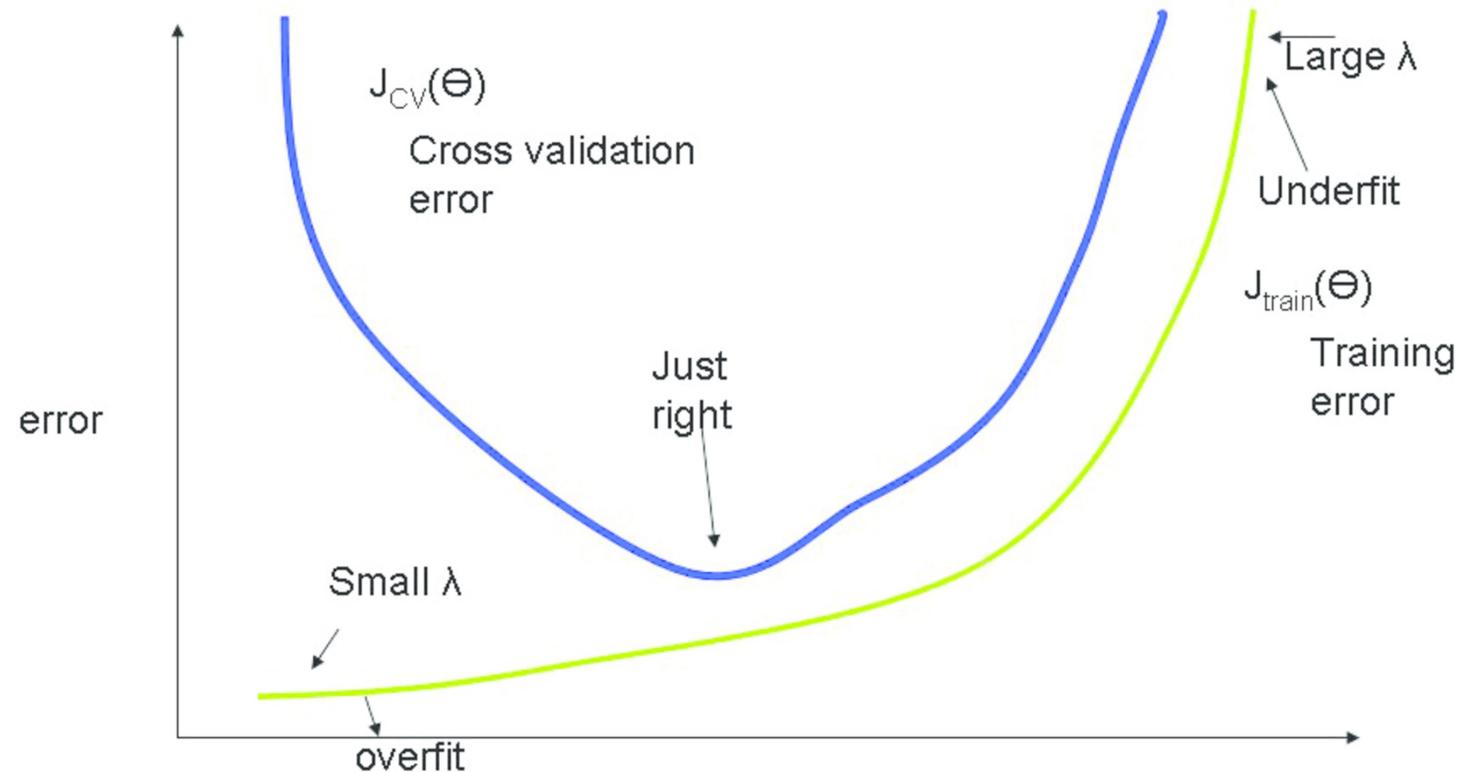
$$J_{cv}(\theta) \approx J_{test}(\theta)$$

Variance (overfit):

$$J_{train}(\theta) \text{ will be low}$$

$$J_{cv}(\theta) \gg J_{train}(\theta)$$

Diagnosing Lambda (regularization)



Learning curves

Learning curve is a good technique

- To sanity-check a model
- To improve performance

A learning curve is a plot where we have two functions of m (m is a set size):

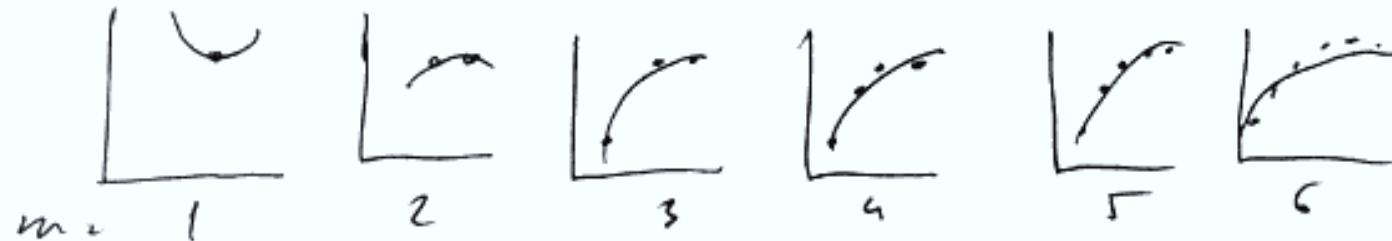
- training set error $J_{train}(\theta)$
- the cross-validation error $J_{cv}(\theta)$

We can artificially reduce our training set size.

We start from $m=1$, then $m=2$ and so on

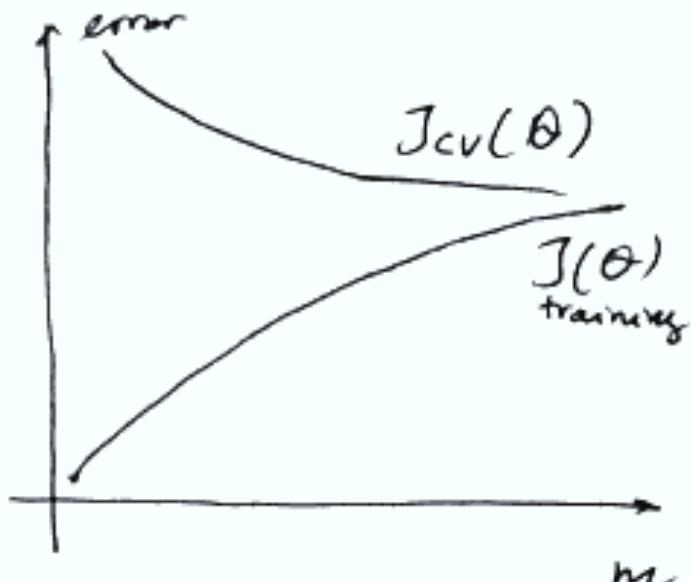
Learning curves (cont.d)

Suppose we have the following model: $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$



For each m we compute $J_{train}(\theta)$ and $J_{cv}(\theta)$ and plot the values.

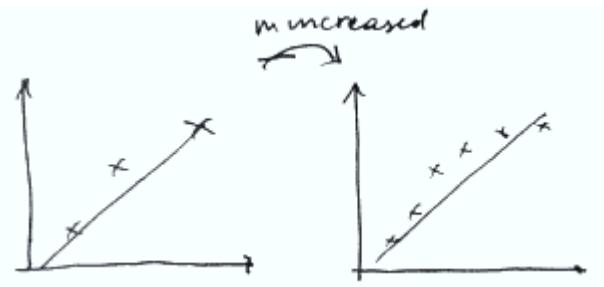
This is the learning curve of the model:



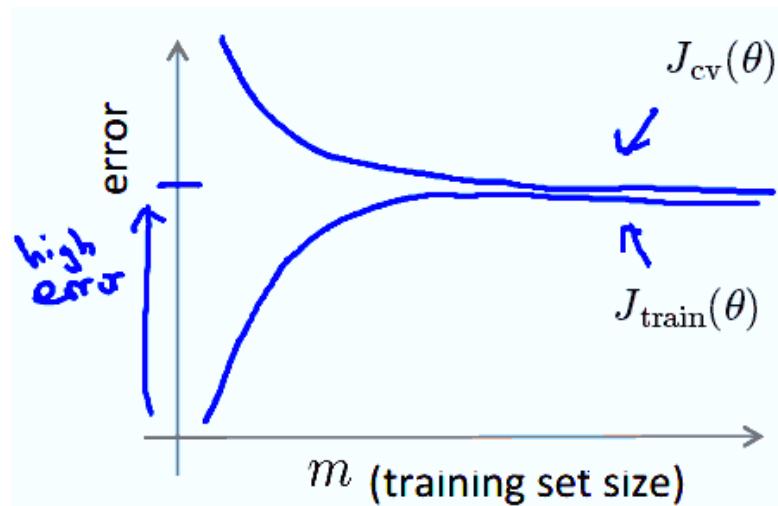
Diagnose High Bias (Underfitting)

Suppose we want to fit a straight line to our data: $h_{\theta}(x) = \theta_0 + \theta_1 x$

As m increases, we have almost the same line:



If we draw the learning curves, we'll have



We see that as m grows
 $J_{cv}(\theta) \rightarrow J_{train}(\theta)$

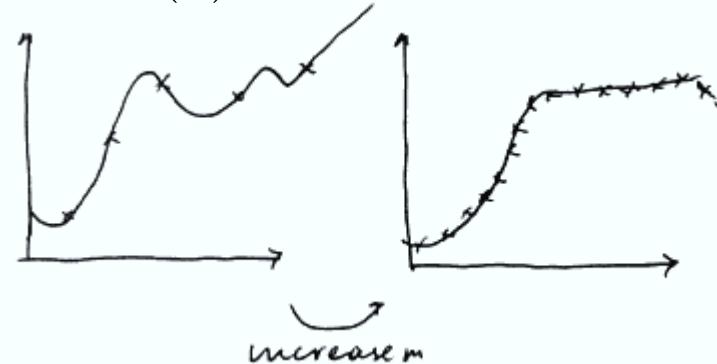
and both errors are high

If learning algorithm is suffering from high bias,
getting more examples will not help

Diagnose High Variance (Overfitting)

Now suppose we have a model

with polynomial of very high order: $h_\theta(x) = \theta_0 + \theta_1x + \theta_2x^2 + \dots + \theta_{100}x^{100}$



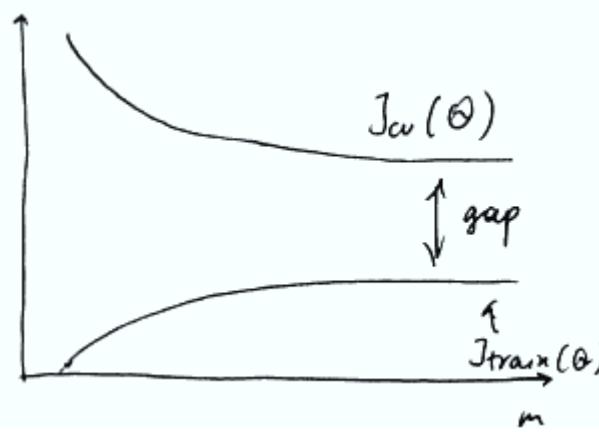
at the beginning we very much overfit, as we increase m , we are still able to fit the data well.

So, we can see that as m increases, $J_{train}(\theta)$ increases.

(we have more and more data - so it's harder and harder to fit $h_\theta(x)$),
but it increases very slowly.

On the other hand, $J_{cv}(\theta)$ decreases, but also very very slowly and
there's a huge gap between these 2 to fill that gap we need many many
more training examples.

If a learning algorithm is suffering from high variance (i.e. it overfits),
getting more data is likely to help



Debugging a learning algorithm (revisited):

Suppose you have implemented a regularized linear regression to predict housing prices.

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

When you test your hypothesis on a new set of houses, you find that your regressor performs very bad with very large errors. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features → fixes high bias
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

How to build a ML System

How to measure if a ML System works well

Evaluation metrics

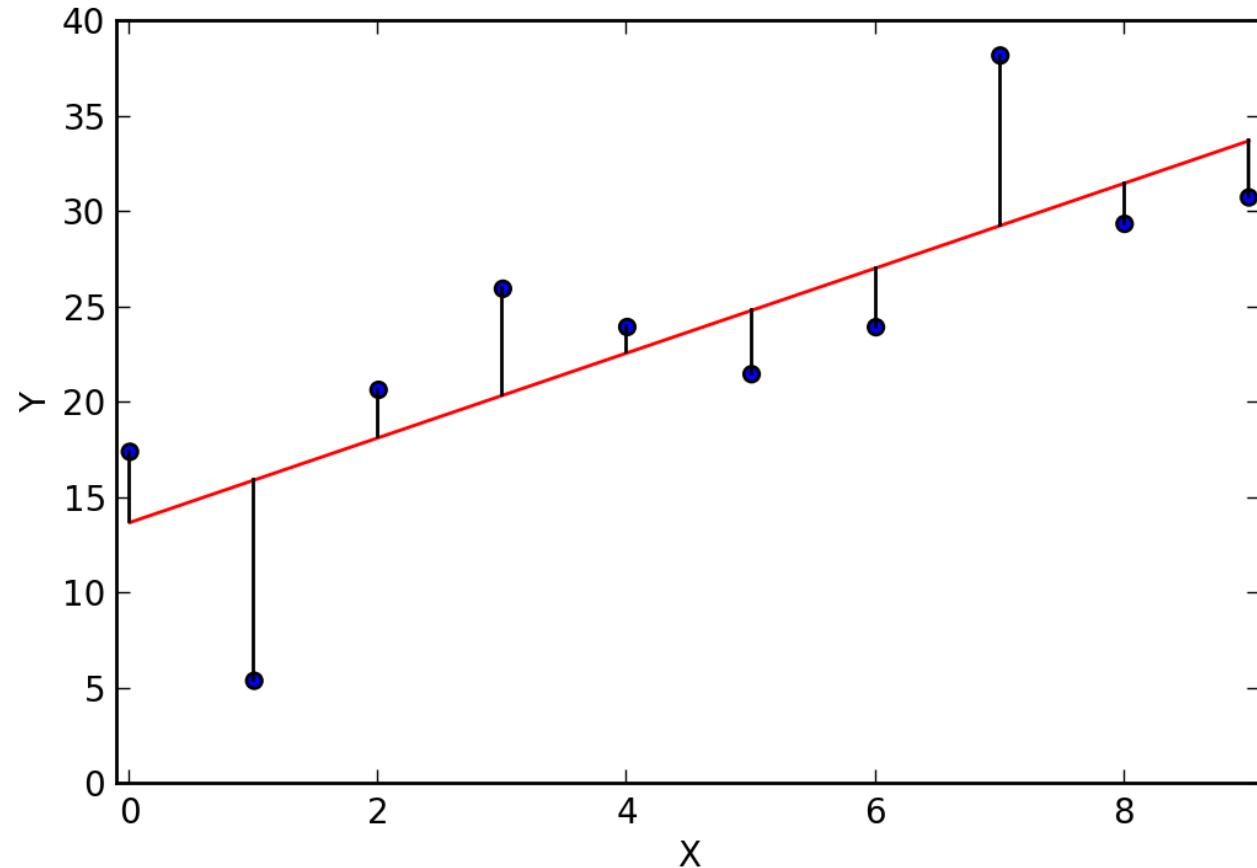
Regression

- MAE, MSE, RMSE

Classification

- Accuracy, Precision/Recall, ROC

Regression error



Evaluation metrics for Regression

We should consider the residuals: difference between values predicted and actual value:

Mean Absolute Error (MAE)

$$MAE = \frac{\sum_{i=1}^m (y_i^* - y_i)}{m}$$

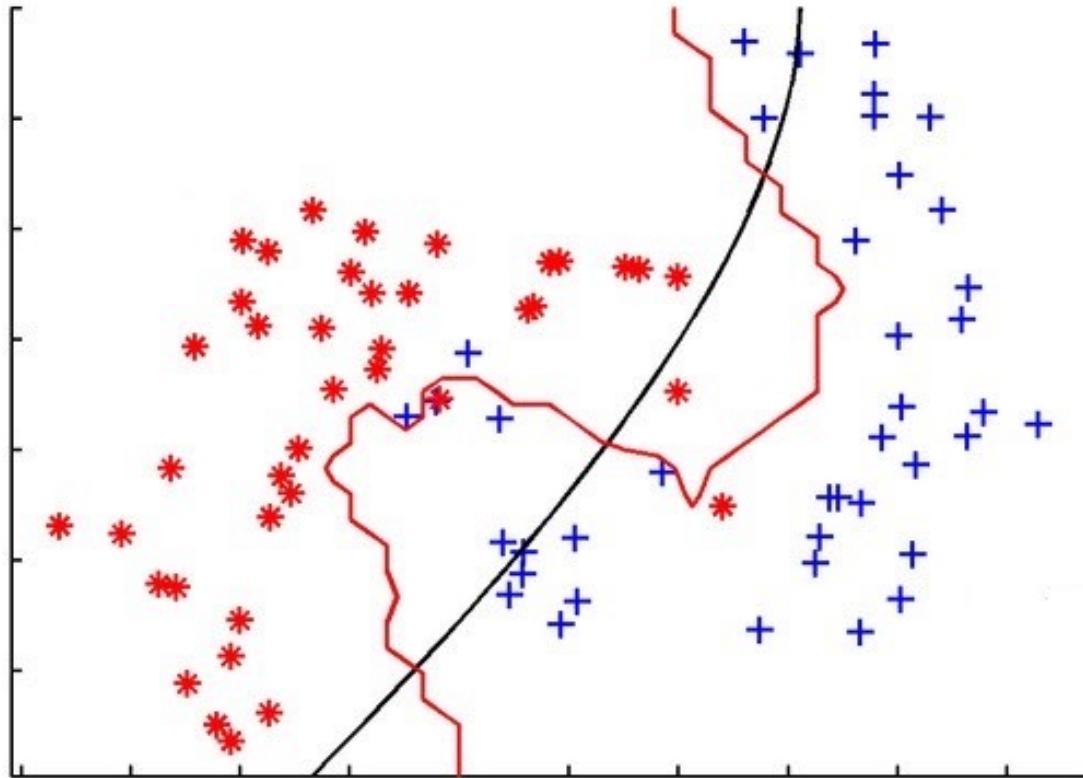
Mean Squared Error (MSE)

$$MSE = \frac{\sum_{i=1}^m (y_i^* - y_i)^2}{m}$$

Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{\sum_{i=1}^m (y_i^* - y_i)^2}{m}}$$

Classification error



Confusion matrix

		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

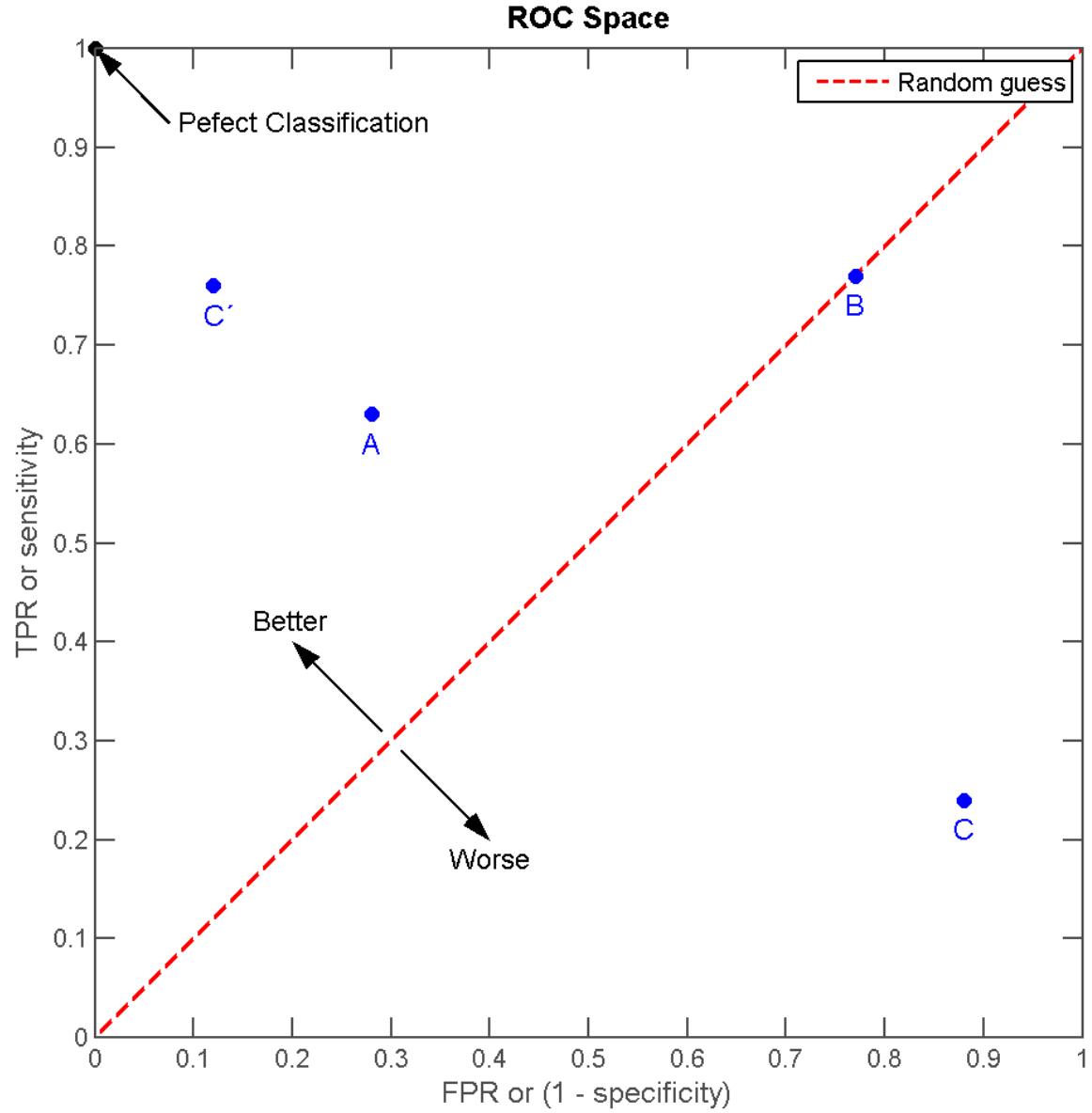
Confusion matrix

	Accuracy	$(TP+TN)/(TP+TN+FP+FN)$
	Precision	$TP/(TP+FP)$
Recall or Sensitivity or True Positive Rate		$TP/(TP+FN)$
Specificity or True Negative Rate		$TN/(FP+TN)$
Error rate = 1- accuracy		$(FP+FN)/(TP+TN+FP+FN)$
F-measure		$2*precision*recall/(precision+recall)$
False Positive Rate=1-specificity		$FP/(TN+FP)$

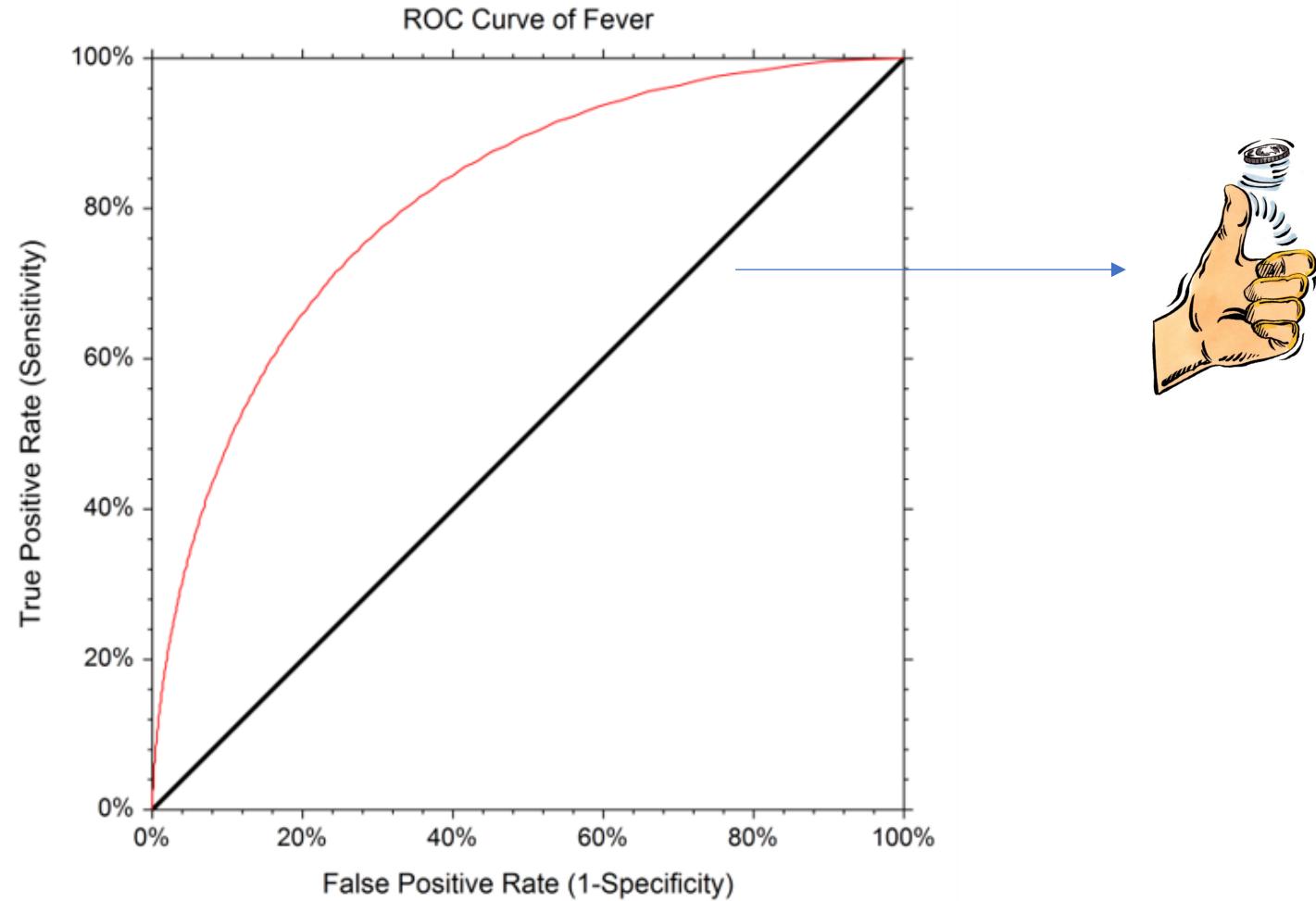
Receiver Operating Characteristic (ROC) Space

A		B		
TP=63	FP=28	TP=77	FP=77	91 154
FN=37	TN=72	FN=23	TN=23	109 46
100	100	100	100	200 200
TPR = 0.63		TPR = 0.77		
FPR = 0.28		FPR = 0.77		
PPV = 0.69		PPV = 0.50		
F1 = 0.66		F1 = 0.61		
ACC = 0.68		ACC = 0.50		

C		C'		
TP=24	FP=88	TP=76	FP=12	112 88
FN=76	TN=12	FN=24	TN=88	88 112
100	100	100	100	200 200
TPR = 0.24		TPR = 0.76		
FPR = 0.88		FPR = 0.12		
PPV = 0.21		PPV = 0.86		
F1 = 0.23		F1 = 0.81		
ACC = 0.18		ACC = 0.82		



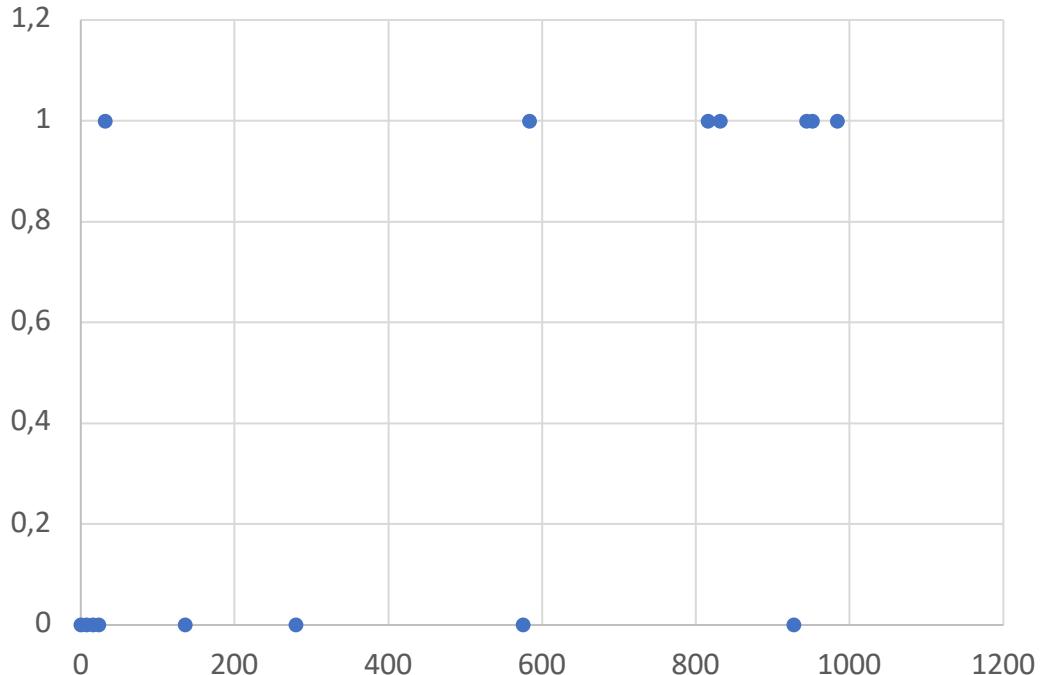
Receiver Operating Characteristic (ROC) curve



ROC curve example

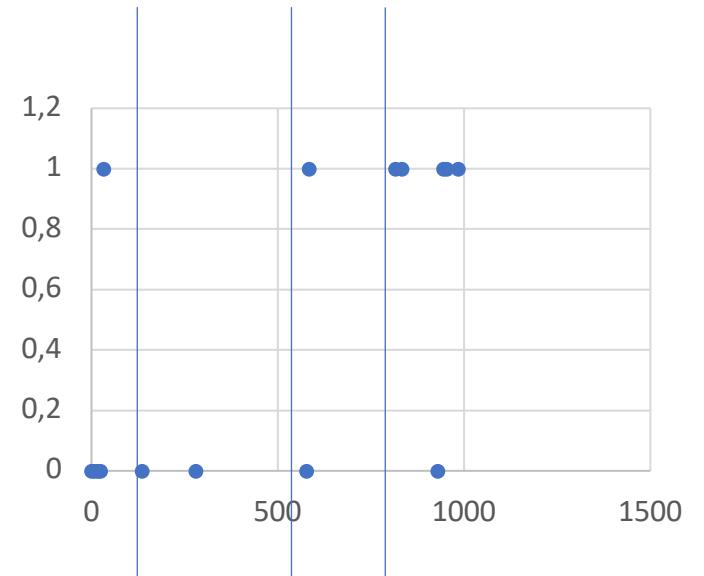
Year	Observed event (1) or non-event(0)	Forecast Probability (FP)
1994	1	0.984
1995	1	0.952
1984	1	0.944
1981	0	0.928
1985	1	0.832
1986	1	0.816
1988	1	0.584
1982	0	0.576
1991	0	0.28
1987	0	0.136
1989	1	0.032
1992	0	0.024
1990	0	0.016
1983	0	0.008
1993	0	0

Data describes March-May precipitation over North-East Brazil for 1981-1995
Arranged in decreasing probability



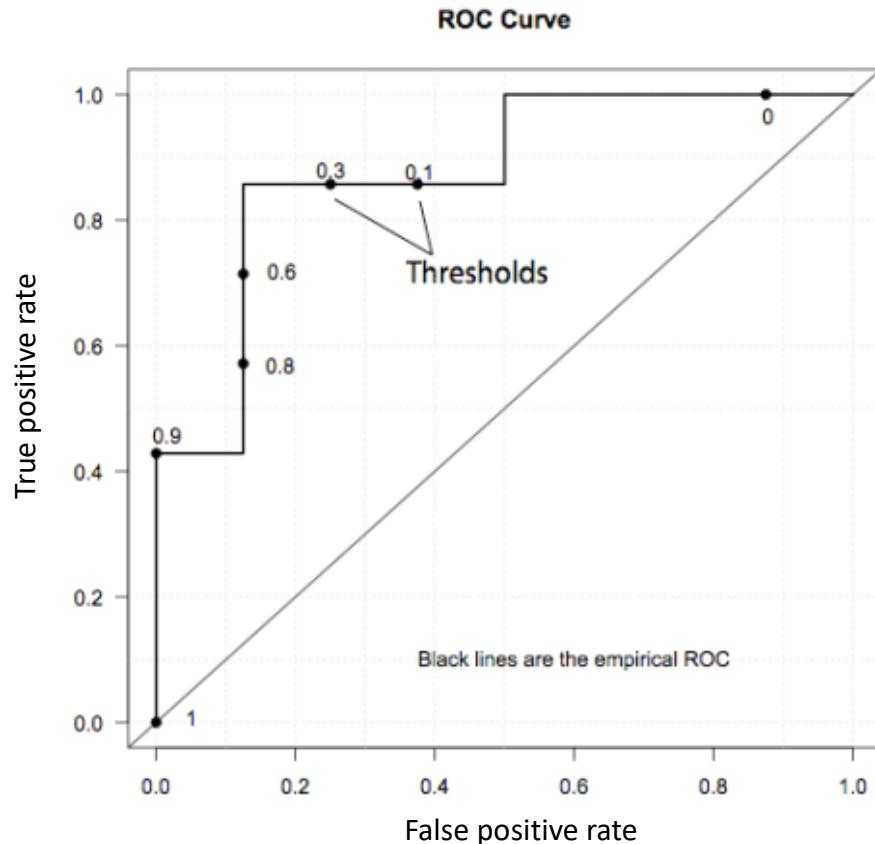
ROC curve example (cont.d)

	Year	Observed event (1) or non-event(0)	Forecast Probability	T=0.1	T=0.5	T=0.8
Correct positive	1994	1	0.984	1	1	1
	1995	1	0.952	1	1	1
	1984	1	0.944	1	1	1
	1981	0	0.928	1	1	1
	1985	1	0.832	1	1	1
	1986	1	0.816	1	1	1
	1988	1	0.584	1	1	0
	1982	0	0.576	1	1	0
	1991	0	0.28	1	0	0
	1987	0	0.136	1	0	0
Correct negative	1989	1	0.032	0	0	0
	1992	0	0.024	0	0	0
	1990	0	0.016	0	0	0
	1983	0	0.008	0	0	0
	1993	0	0	0	0	0



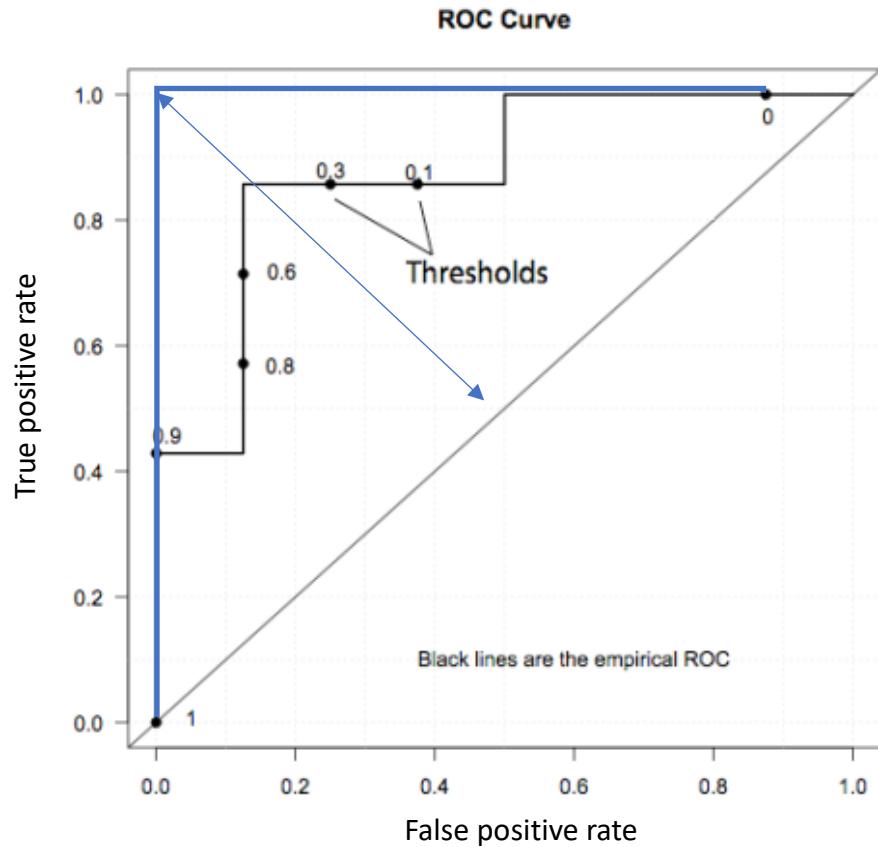
ROC curve example (cont.d)

We can plot the points of the curve for each sample:



ROC curve example (cont.d)

Optimal ROC curve



How to build a ML System

How to measure if the
results are significant

Comparing systems using a paired t test

- The paired sample t -test is a statistical procedure used to determine whether the mean difference between two sets of observations is zero.
- In a paired sample t -test, each subject or entity is measured twice, resulting in *pairs* of observations.
- In our case, pairs of observations are, e.g., accuracy values computed by two systems varying the threshold

$$\mathbf{y}^a = \{y_1^a, y_2^a, \dots, y_n^a\}$$

$$\mathbf{y}^b = \{y_1^b, y_2^b, \dots, y_n^b\}$$

Comparing systems using a paired t test (cont'd)

Null Hypothesis: the 2 learning systems have the same accuracy

Alternative Hypothesis: one of the systems is more accurate than the other

Under the null hypothesis, all observable differences are explained by random variation.

Hypothesis Test:

- use paired t-test to determine the probability p that the mean difference supports the null hypothesis
- if p is sufficiently small (typically < 0.05) then reject the null hypothesis

Comparing systems using a paired t test (cont.d)

$$\vec{y}^a = \{y_1^a, y_2^a, \dots, y_n^a\}$$

$$\vec{y}^b = \{y_1^b, y_2^b, \dots, y_n^b\}$$

$$\vec{\delta} = \{y_1^a - y_1^b, y_2^a - y_2^b, \dots, y_n^a - y_n^b\}$$

1. calculate the sample mean

$$\bar{\delta} = \frac{1}{n} \sum_{i=1}^n \delta_i$$

2. calculate the t statistic

$$t = \frac{\bar{\delta}}{\sqrt{\frac{1}{n(n-1)} \sum_{i=1}^n (\delta_i - \bar{\delta})^2}}$$

3. determine the corresponding p-value, by looking up t in the Student's t-distribution with n-1 degrees of freedom

Comparing systems using a paired t test (cont.d)

$$p = 2 \cdot \Pr(T > |t|)$$
 Two-tailed

$$p = \Pr(T > t)$$
 upper-tailed

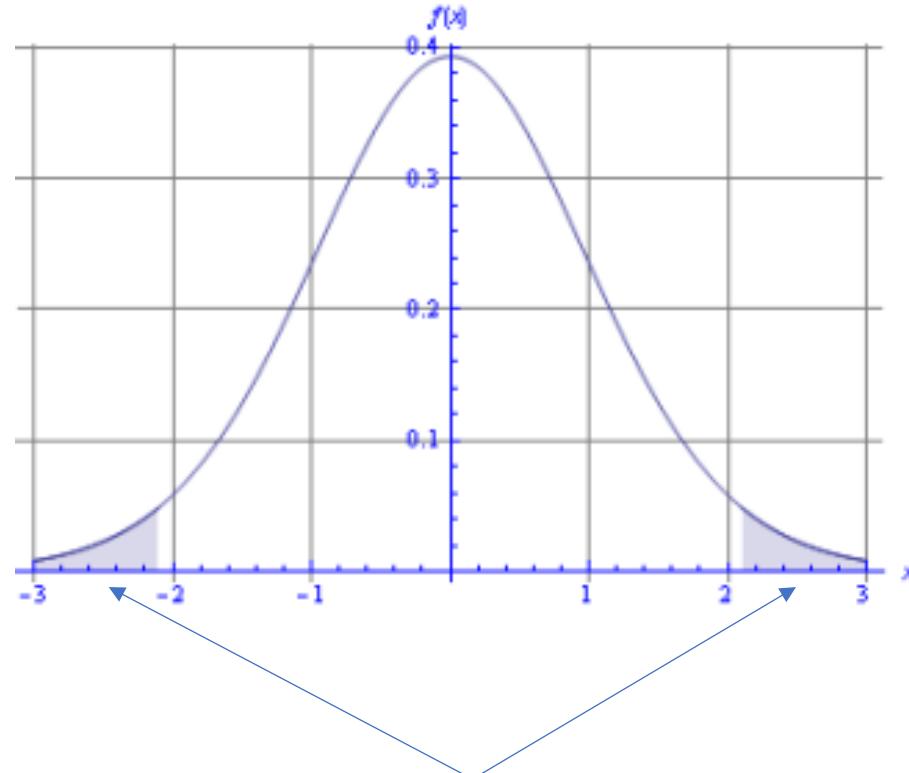
$$p = \Pr(T < t)$$
 lower-tailed

Comparing systems using a paired t test (cont.d)

- A two-tailed test asks the question: is the accuracy of the two systems different?
- A one-tailed test asks the question: is system A better than system B?

You have to choose the right one for your case

Comparing systems using a paired t test (cont.d)



for a two-tailed test, the p -value represents the probability mass in these two regions

The null distribution of our t statistic looks like this

The p -value indicates how far out in a tail our t statistic is

If the p -value is sufficiently small, we reject the null hypothesis, since it is unlikely we'd get such a t by chance

Coefficient of determination - R²

R² is a measure of the proportion of the variance in the dependent variable that is **predictable** from the independent variable.

$$R^2 = 1 - \frac{RSS}{TSS}$$

$$RSS = \sum_{i=1}^n (y^{(i)} - h(x^{(i)}))^2 \quad \text{Residual Sum of Squares}$$

$$TSS = \sum_{i=1}^n (y^{(i)} - \bar{y})^2 \quad \text{Total Sum of Squares}$$

Coefficient of determination - R^2 (cont'd)

R^2 values are in the range [0...1]

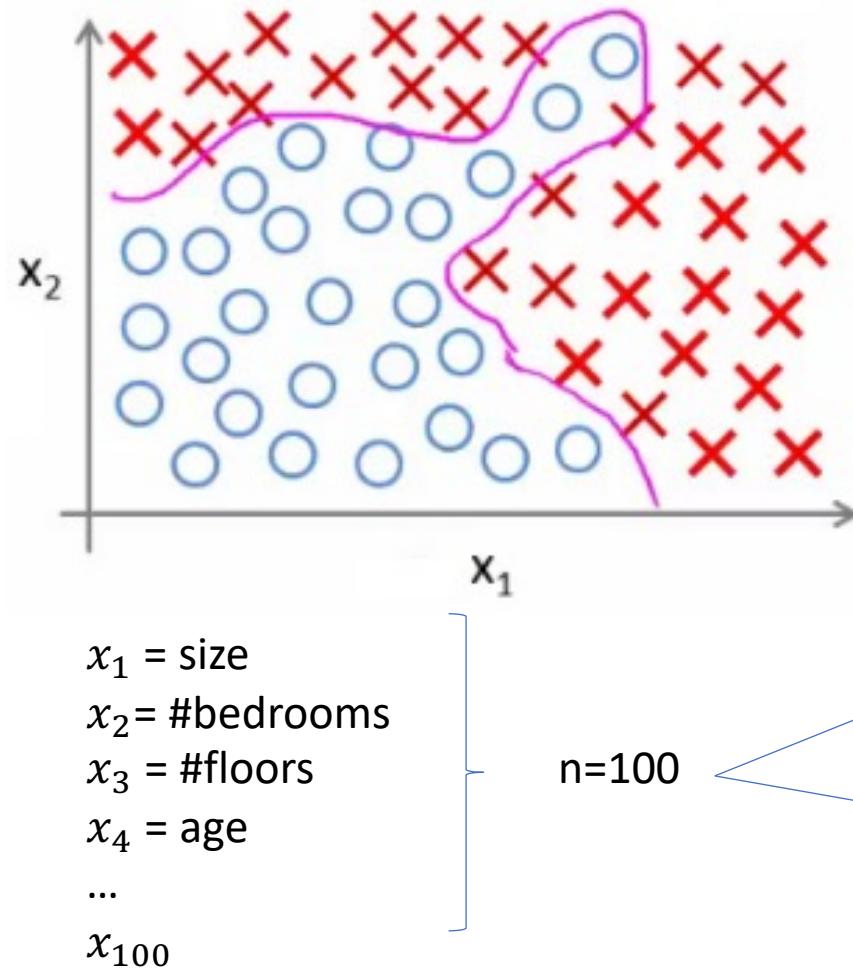
The closer the value to 1 the better the data fit the model

It provides a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

Neural Networks

Non-linear hypotheses

Non-linear Classification

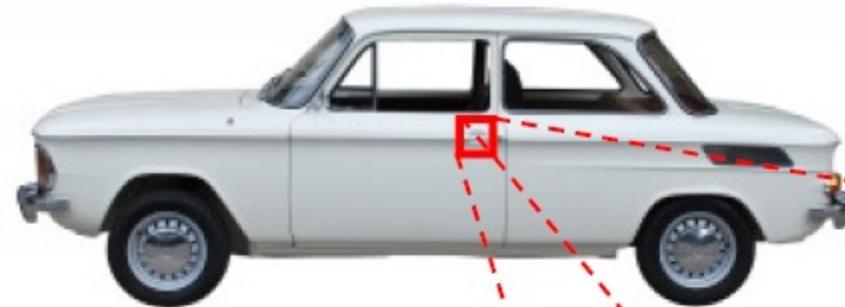


$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \theta_7 x_1 x_2^3 + \dots)$$

Arrows point from the text to the corresponding terms in the polynomial equation:

- $x_1^2, x_2^2, x_3^2, \dots$
- $x_1 x_2, x_1 x_3, x_1 x_4, \dots, x_1 x_{100}$
- $\approx 5000 \text{ features } \mathcal{O}(n^2)$
- $x_1^3, x_2^3, x_3^3, \dots$
- $x_1^2, x_2^2, x_3^2, \dots$
- $x_1^2 x_2, x_1^2 x_3, x_1^2 x_4, \dots, x_1^2 x_{100}$
- $x_1 x_2, x_1 x_3, x_1 x_4, \dots, x_1 x_{100}$
- $\approx 170000 \text{ features } \mathcal{O}(n^2)$

How many features do we usually deal with?

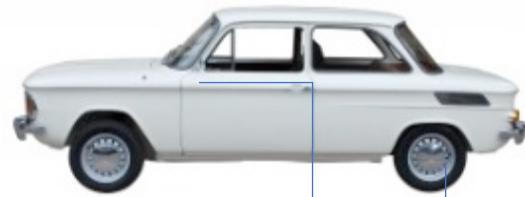
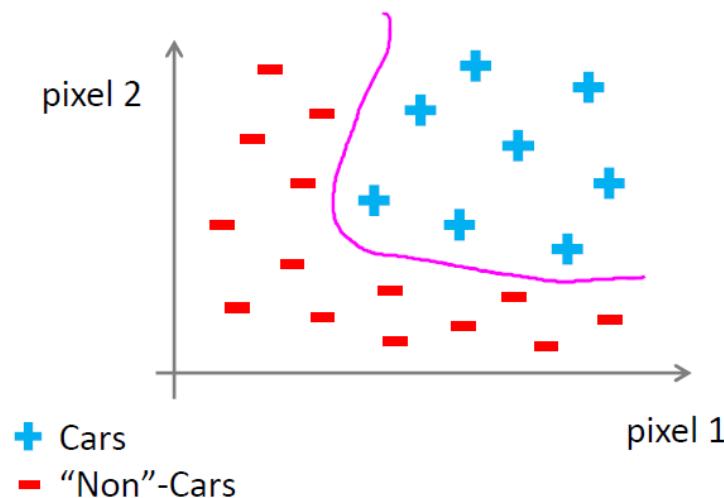


But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

How many features do we usually deal with? (cont.d)

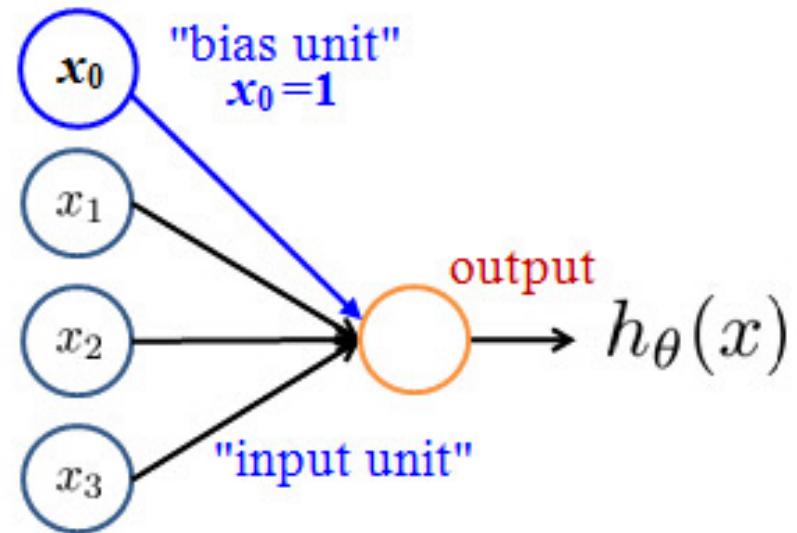
Simple 50×50 pixels images imply 2500 features



$$\mathbf{x} = \begin{bmatrix} \text{pixel } 1 & \text{intensity} \\ \text{pixel } 2 & \text{intensity} \\ \vdots \\ \text{pixel } 2500 & \text{intensity} \end{bmatrix}$$

If we consider the quadratic features we get $O(2.500^2) = O(6 \cdot 10^6)$ features

Neuron model



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

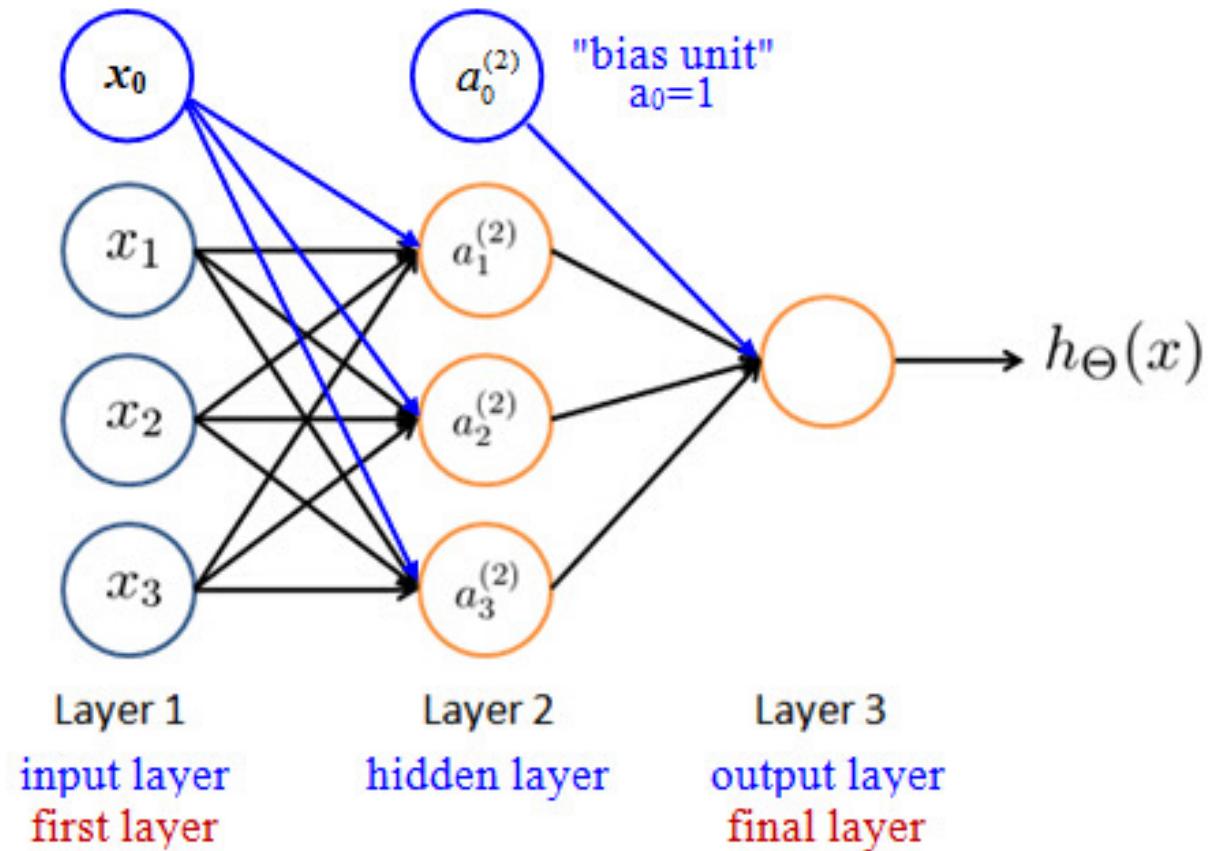
weights

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

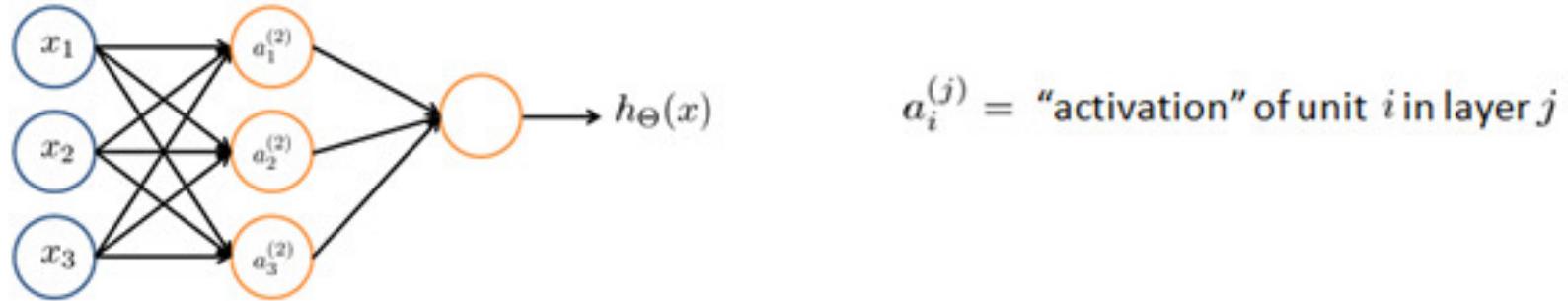
$$g(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid (logistic) activation function.

Neural Network



Neural Network (cont.d)



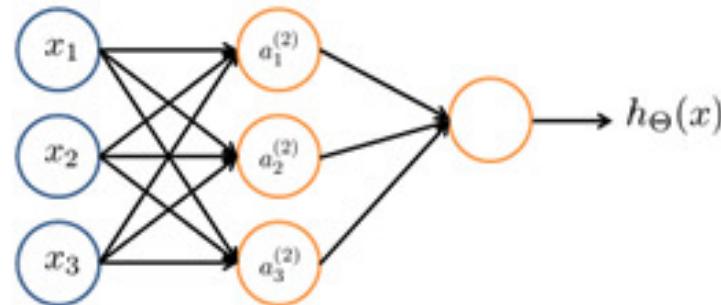
$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0 + \Theta_{11}^{(2)}a_1 + \Theta_{12}^{(2)}a_2 + \Theta_{13}^{(2)}a_3)$$

Neural Network (cont.d)



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling
function mapping from layer j to
layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

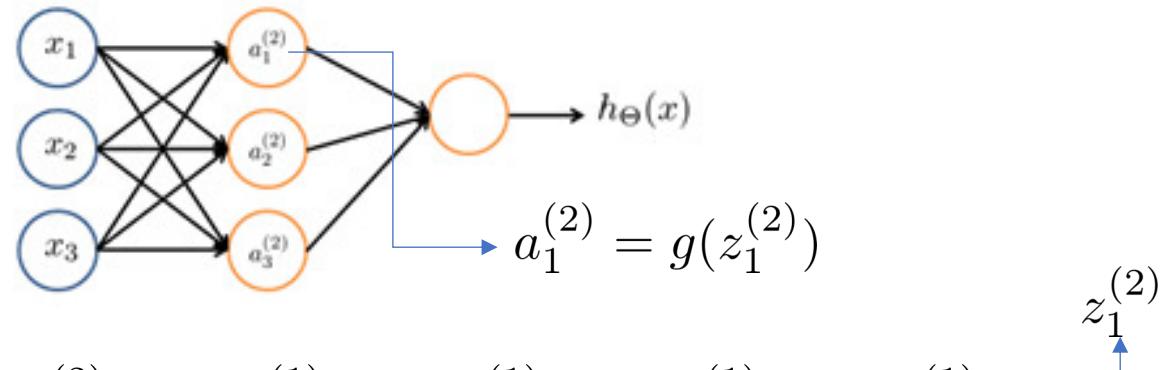
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0 + \Theta_{11}^{(2)} a_1 + \Theta_{12}^{(2)} a_2 + \Theta_{13}^{(2)} a_3)$$

$$\Theta^{(1)} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix}$$

$$\Theta^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix}$$

If the network has s_j units in layer j , s_{j+1} units in layer $j + 1$ then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0 + \Theta_{11}^{(2)}a_1 + \Theta_{12}^{(2)}a_2 + \Theta_{13}^{(2)}a_3)$$

$$\Theta^{(1)} = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix}_{3 \times 4} \quad \Theta^{(2)} = \begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{bmatrix}_{1 \times 4}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)}x$$

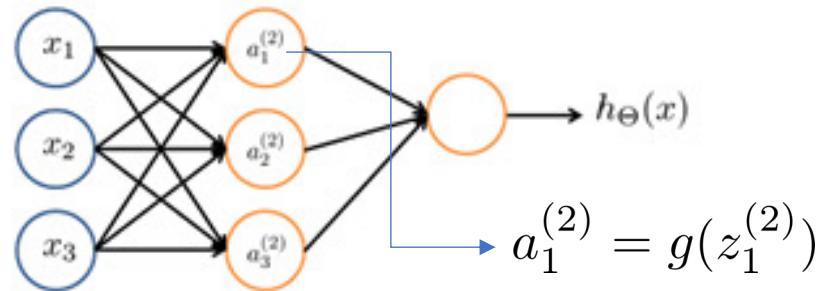
$$a^{(2)} = g(z^{(2)})$$

$$Add \quad a_0^{(2)} = 1$$

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$

$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(z_1^{(2)})$$

$$a_2^{(2)} = g(z_2^{(2)})$$

$$a_3^{(2)} = g(z_3^{(2)})$$

$$h_{\Theta}(x) = a_1^{(3)} = g(z_1^{(3)})$$

$$\Theta^{(1)} = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix}_{3 \times 4} \quad \Theta^{(2)} = \begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{bmatrix}_{1 \times 4}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \quad \mathbf{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$Add \quad a_0^{(2)} = 1$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_{\Theta}(x) = a^{(3)} = g(z^{(3)})$$

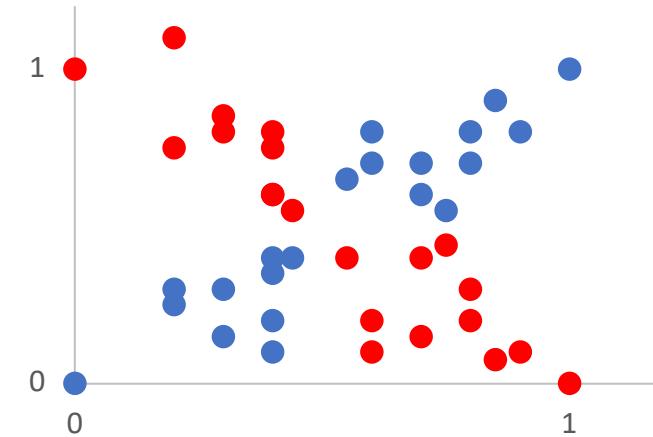
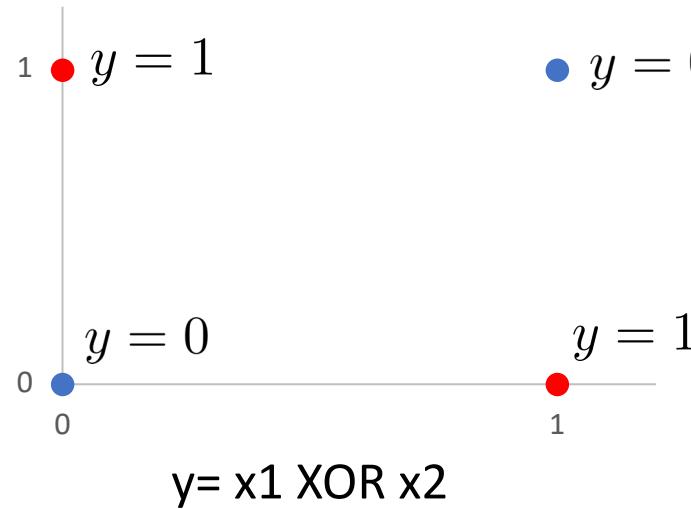
Neural Networks

Intuition

Non-linear classification example:XOR

Linear classifiers are not able to solve it

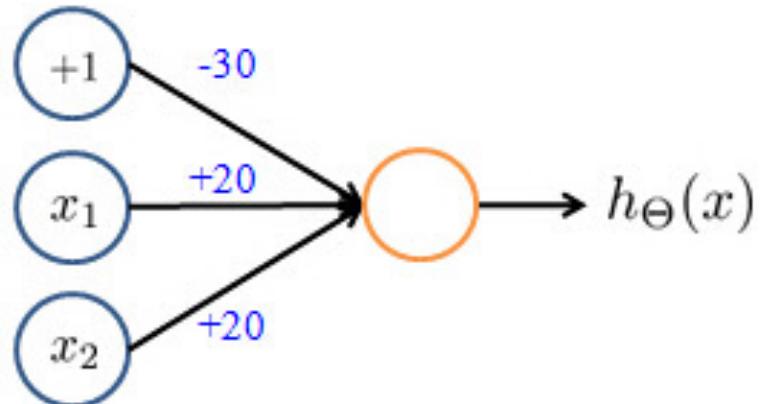
x_1 and x_2 are binary (0 or 1)



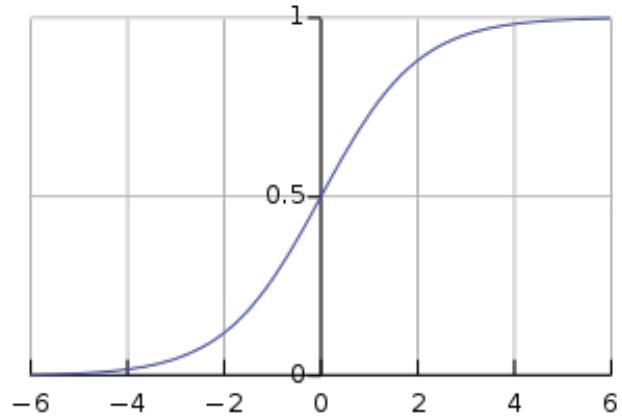
NN example: AND function

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{AND} x_2$$



$$h_{\Theta}(x) = g(-30 + 20x_1 + 20x_2)$$

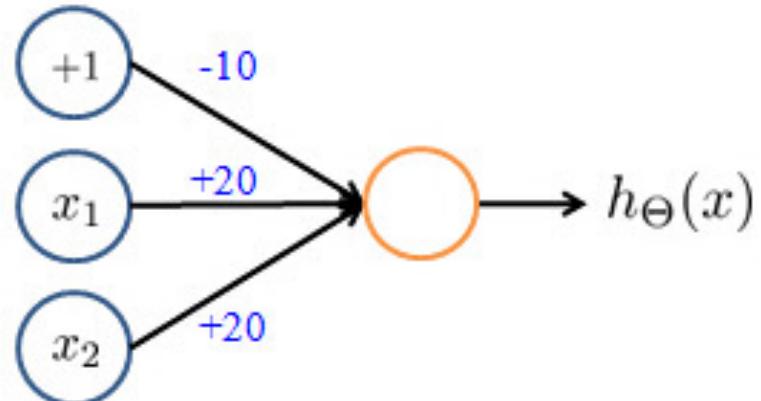


x_1	x_2	$h_{\Theta}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(+10) \approx 1$

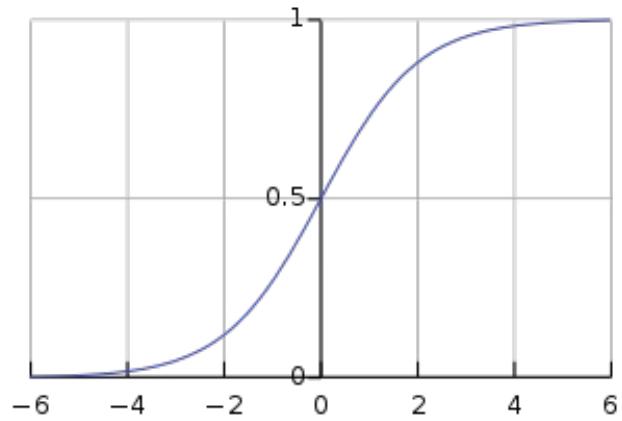
NN example: OR function

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{OR} x_2$$



$$h_{\Theta}(x) = g(-10 + 20x_1 + 20x_2)$$

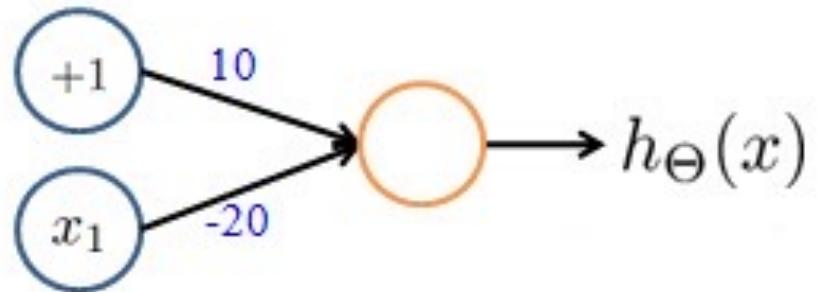


x_1	x_2	$h_{\Theta}(x)$
0	0	$g(-10) \approx 0$
0	1	$g(+10) \approx 1$
1	0	$g(+10) \approx 1$
1	1	$g(+30) \approx 1$

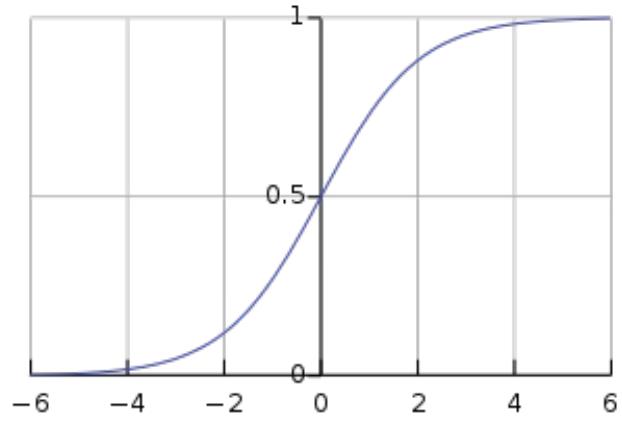
NN example: NOT function

$$x_1 \in \{0, 1\}$$

$$y = \text{NOT } x_1$$



$$h_{\Theta}(x) = g(10 - 20x_1)$$

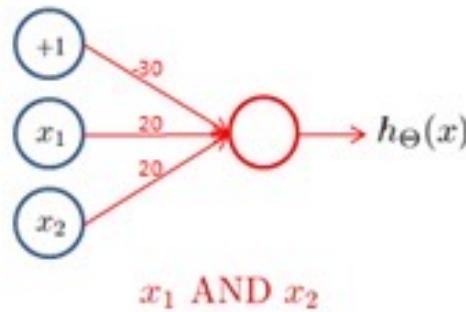


x_1	$h_{\Theta}(x)$
0	$g(+10) \approx 1$
1	$g(-10) \approx 0$

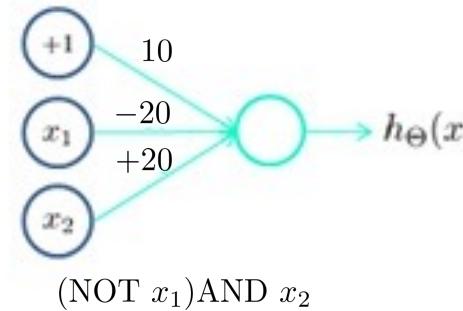
NN example: XOR function

$$x_1 \in \{0, 1\}$$

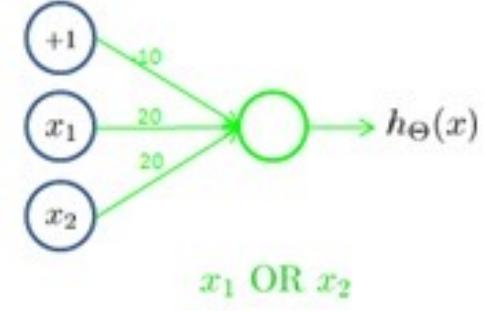
$$y = x_1 \text{XOR } x_2$$



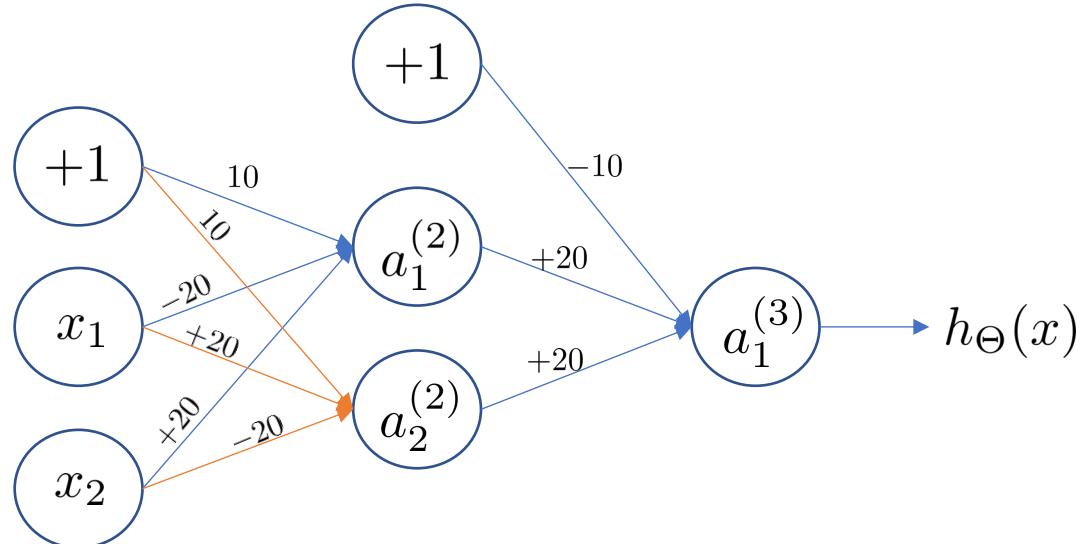
$x_1 \text{ AND } x_2$



(NOT x_1) AND x_2



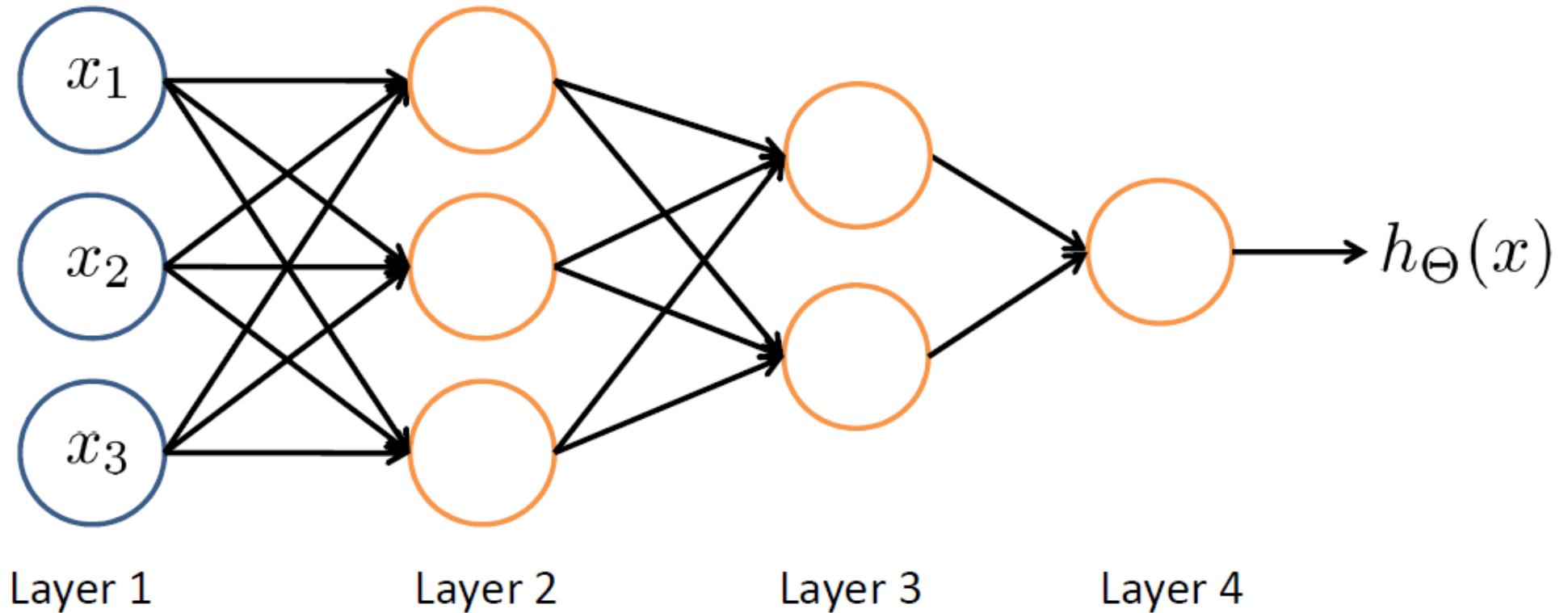
$x_1 \text{ OR } x_2$



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_{\Theta}(x)$
0	0	0	0	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

$$x_1 \text{XOR } x_2 = (x_1 \text{ AND } (\text{NOT } x_2)) \text{OR } (x_2 \text{ AND } \text{NOT } x_1)$$

Neural Networks



Neural Networks for multi-class classification

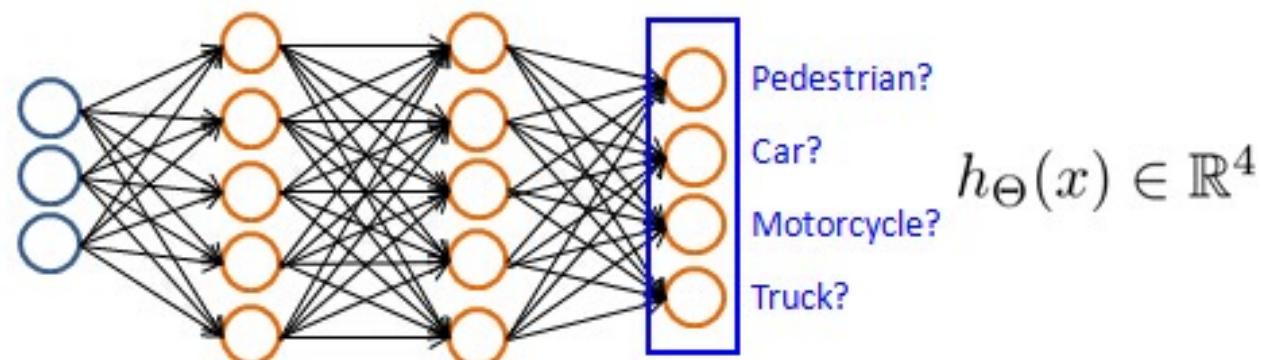


Pedestrian

Car

Motorcycle

Truck



Want $h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.
 when pedestrian when car when motorcycle

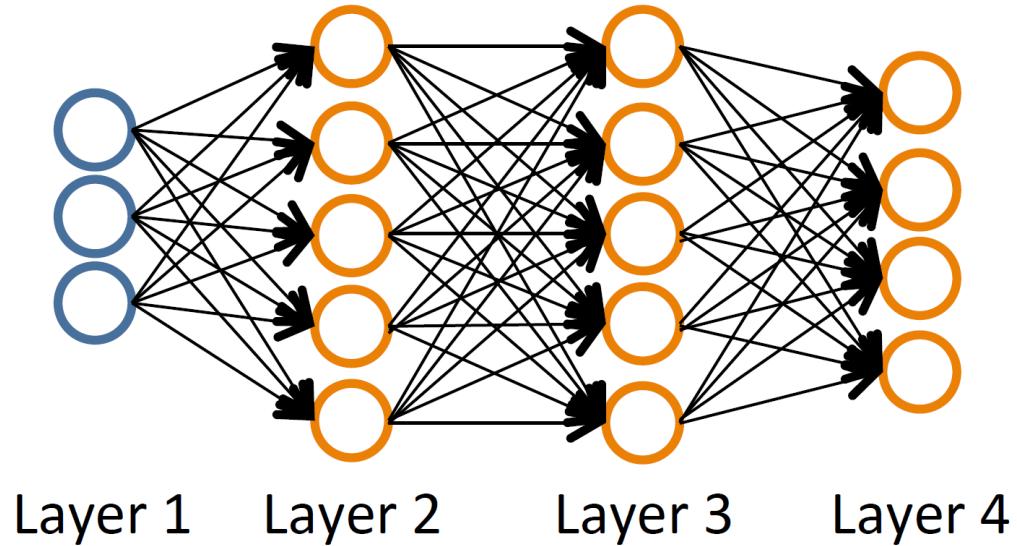
Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$
 pedestrian car motorcycle truck

Neural Networks

Cost function

Neural Network for classification task



$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

L = total number of layers in network

s_l = number of units (but the bias unit) in layer l

Binary classification

$$y = 0 \text{ or } 1$$

1 output unit

Multi-class classification (k classes)

$$y \in \mathbb{R}^K = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

K output units

Cost function for Classification

Logistic regression

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

Neural Network

$$h_\Theta(x) \in \mathbb{R}^K \quad \text{with } (h_\Theta(x))_k = k^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l+1} (\Theta_{ji}^{(l)})^2$$

Cost function for Regression

Linear regression

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

Neural Network

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^m (h_\Theta(\mathbf{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \theta_{ji}^2$$

Neural Networks

Backpropagation algorithm

Gradient computation for Classification

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l+1} (\Theta_{ji}^{(l)})^2$$

To minimize $J(\Theta)$: $\min_{\Theta} J(\Theta)$

We need to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation for Regression

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \theta_{ji}^2$$

To minimize $J(\Theta)$: $\min_{\Theta} J(\Theta)$

We need to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

Given the training example (x, y) :

Forward propagation:

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

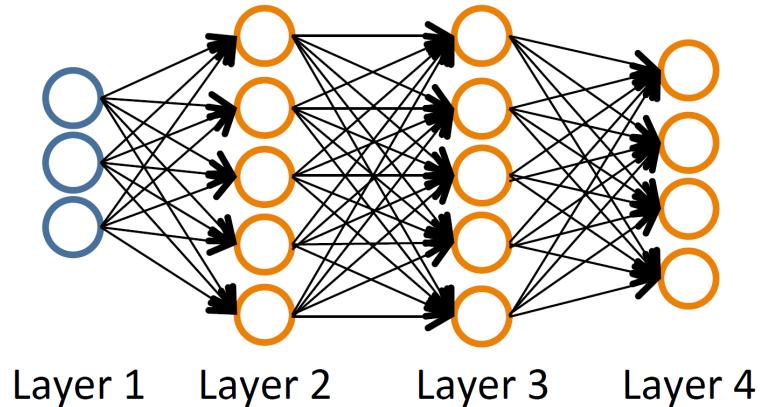
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$

$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = h_{\Theta}(x) = g(z^{(4)})$$



Gradient computation – Regression Loss

$$\begin{aligned}
 \frac{\partial J}{\partial \Theta^{(3)}} &= \frac{\partial \left(\frac{1}{2m} \sum_{i=1}^m (h_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2 \right)}{\partial \Theta^{(3)}} \\
 &= (a^{(4)} - y) \frac{\partial g(z^{(4)})}{\partial \Theta^{(3)}} \\
 &= (a^{(4)} - y) g'(z^{(4)}) \frac{\partial z^{(4)}}{\partial \Theta^{(3)}} \text{ with } z^{(4)} = \Theta^{(3)} a^{(3)} \\
 &= \delta^{(4)} a^{(3)}
 \end{aligned}$$

$a^{(1)} = x$
 $z^{(2)} = \Theta^{(1)} a^{(1)}$
 $a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$
 $z^{(3)} = \Theta^{(2)} a^{(2)}$
 $a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$
 $z^{(4)} = \Theta^{(3)} a^{(3)}$
 $a^{(4)} = h_\Theta(x) = g(z^{(4)})$

Gradient computation – Regression Loss

$$\begin{aligned}
 \frac{\partial J}{\partial \Theta^{(2)}} &= \delta^{(4)} \frac{\partial z^{(4)}}{\partial \Theta^{(2)}} \\
 &= \delta^{(4)} \frac{\partial \Theta^{(3)} a^{(3)}}{\partial \Theta^{(2)}} \\
 &= \Theta^{(3)} \delta^{(4)} \frac{\partial a^{(3)}}{\partial \Theta^{(2)}} \\
 &= \Theta^{(3)} \delta^{(4)} \frac{\partial g(z^{(3)})}{\partial \Theta^{(2)}} \text{ with } a^{(3)} = g(z^{(3)}) \\
 &= \Theta^{(3)} \delta^{(4)} g'(z^{(3)}) \frac{\partial z^{(3)}}{\partial \Theta^{(2)}} \text{ with } z^{(3)} = \Theta^{(2)} a^{(2)} \\
 &= \delta^{(3)} a^{(2)}
 \end{aligned}$$

$$\begin{aligned}
 a^{(1)} &= x \\
 z^{(2)} &= \Theta^{(1)} a^{(1)} \\
 a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
 z^{(3)} &= \Theta^{(2)} a^{(2)} \\
 a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
 z^{(4)} &= \Theta^{(3)} a^{(3)} \\
 a^{(4)} &= h_\Theta(x) = g(z^{(4)})
 \end{aligned}$$

Gradient computation – Regression Loss

$$\begin{aligned}
 \frac{\partial J}{\partial \Theta^{(1)}} &= \delta^{(3)} \frac{\partial z^{(3)}}{\partial \Theta^{(1)}} \\
 &= \Theta^{(2)} \delta^{(3)} g'(z^{(2)}) \frac{\partial z^{(2)}}{\partial \Theta^{(1)}} \\
 &= \delta^{(2)} a^{(1)}
 \end{aligned}$$

$$\begin{aligned}
 a^{(1)} &= x \\
 z^{(2)} &= \Theta^{(1)} a^{(1)} \\
 a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
 z^{(3)} &= \Theta^{(2)} a^{(2)} \\
 a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
 z^{(4)} &= \Theta^{(3)} a^{(3)} \\
 a^{(4)} &= h_{\Theta}(x) = g(z^{(4)})
 \end{aligned}$$

Gradient computation – Regression Loss

Focus on the last error contribution:

$$\delta_j^{(4)} \propto a_j^{(4)} - y_j \quad \xrightarrow{h_{\Theta}(x)}$$

In this example lambda will be considered 0

Get an idea of the gradient:

$$\frac{\partial}{\partial \theta_k} J(\theta) = \left(h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_k^{(i)}$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\begin{aligned} a^{(1)} &= x \\ z^{(2)} &= \Theta^{(1)} a^{(1)} \\ a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\ z^{(3)} &= \Theta^{(2)} a^{(2)} \\ a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\ z^{(4)} &= \Theta^{(3)} a^{(3)} \\ a^{(4)} &= h_{\Theta}(x) = g(z^{(4)}) \end{aligned}$$

Logistic regression gradient

Gradient computation – Regression Loss

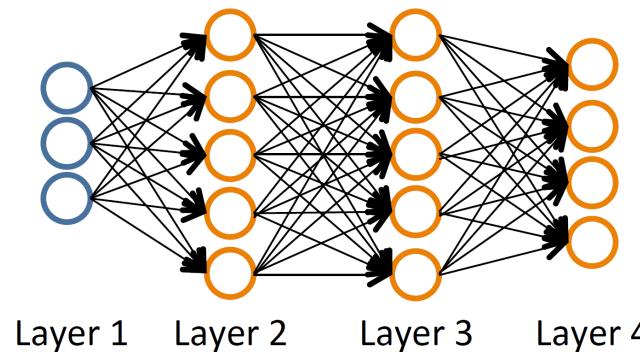
Intuition: $\delta_j^{(l)}$ = «error» of node j in layer l.

For each output unit (layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j \quad \delta^{(4)} = a^{(4)} - y$$

$$\begin{aligned} \delta^{(3)} &= (\Theta^{(3)})^T \delta^{(4)} \cdot * g'(z^{(3)}) \rightarrow a^{(3)} \cdot * (1 - a^{(3)}) \\ \delta^{(2)} &= (\Theta^{(2)})^T \delta^{(3)} \cdot * g'(z^{(2)}) \rightarrow a^{(2)} \cdot * (1 - a^{(2)}) \end{aligned}$$

$a^{(1)} = x$
$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$
$z^{(4)} = \Theta^{(3)} a^{(3)}$
$a^{(4)} = h_{\Theta}(x) = g(z^{(4)})$



Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

For $i=1$ to m

- Set $a^{(1)} = x^{(1)}$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$
- Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad \text{if } j \neq 0$$

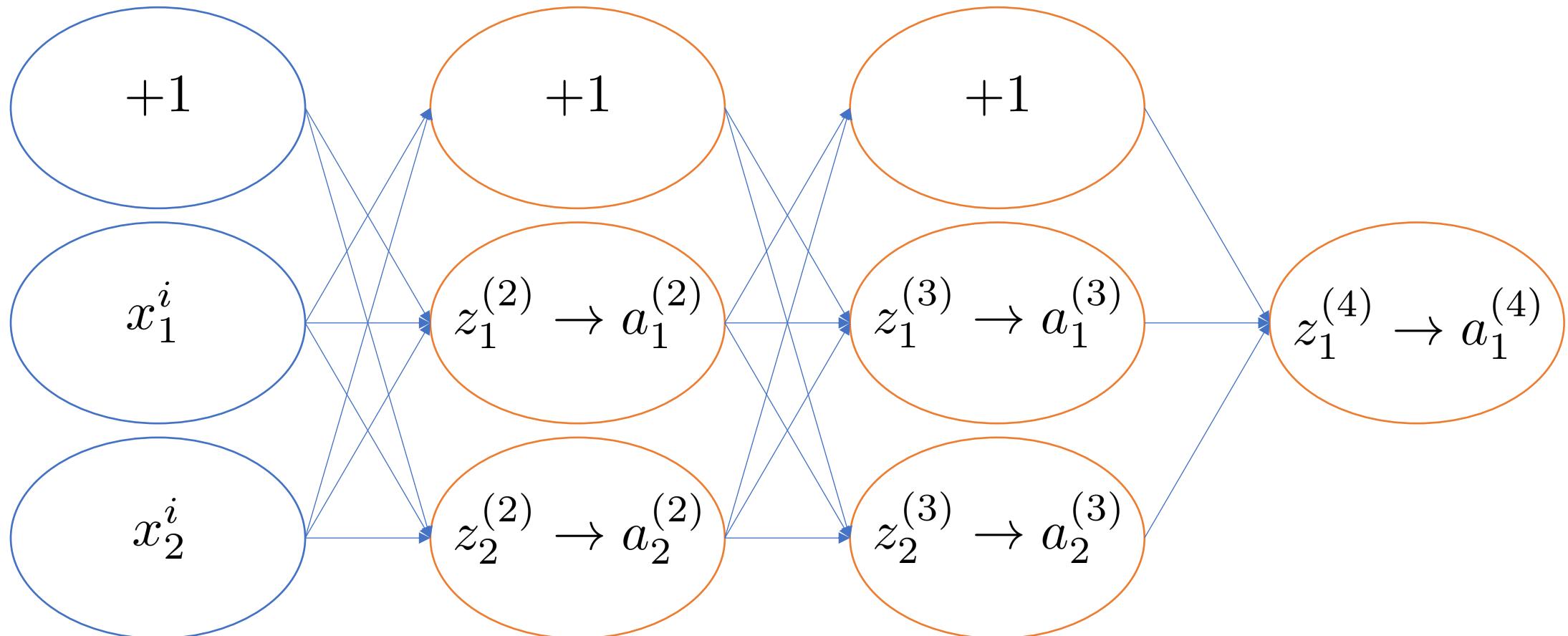
$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

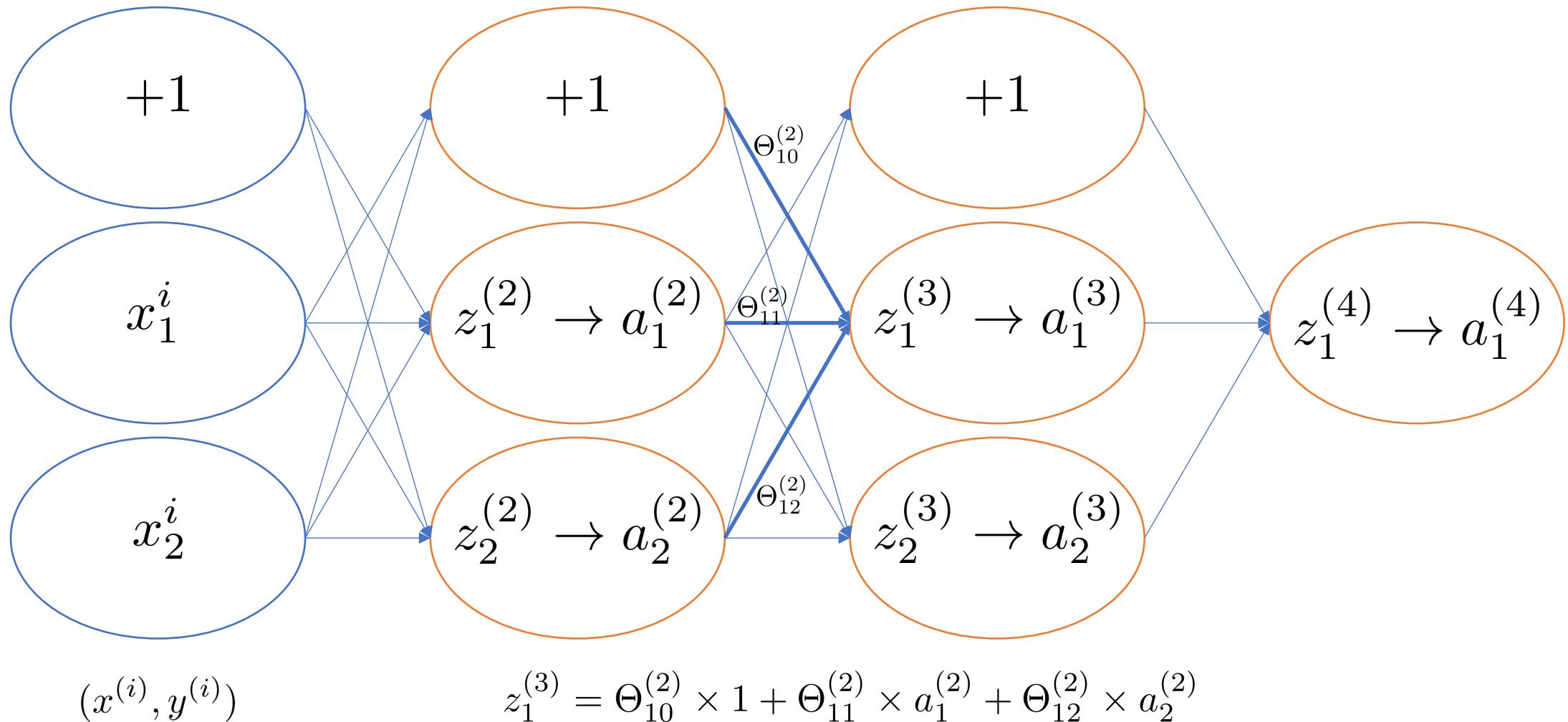
Neural Networks

Backpropagation intuition

Forward Propagation

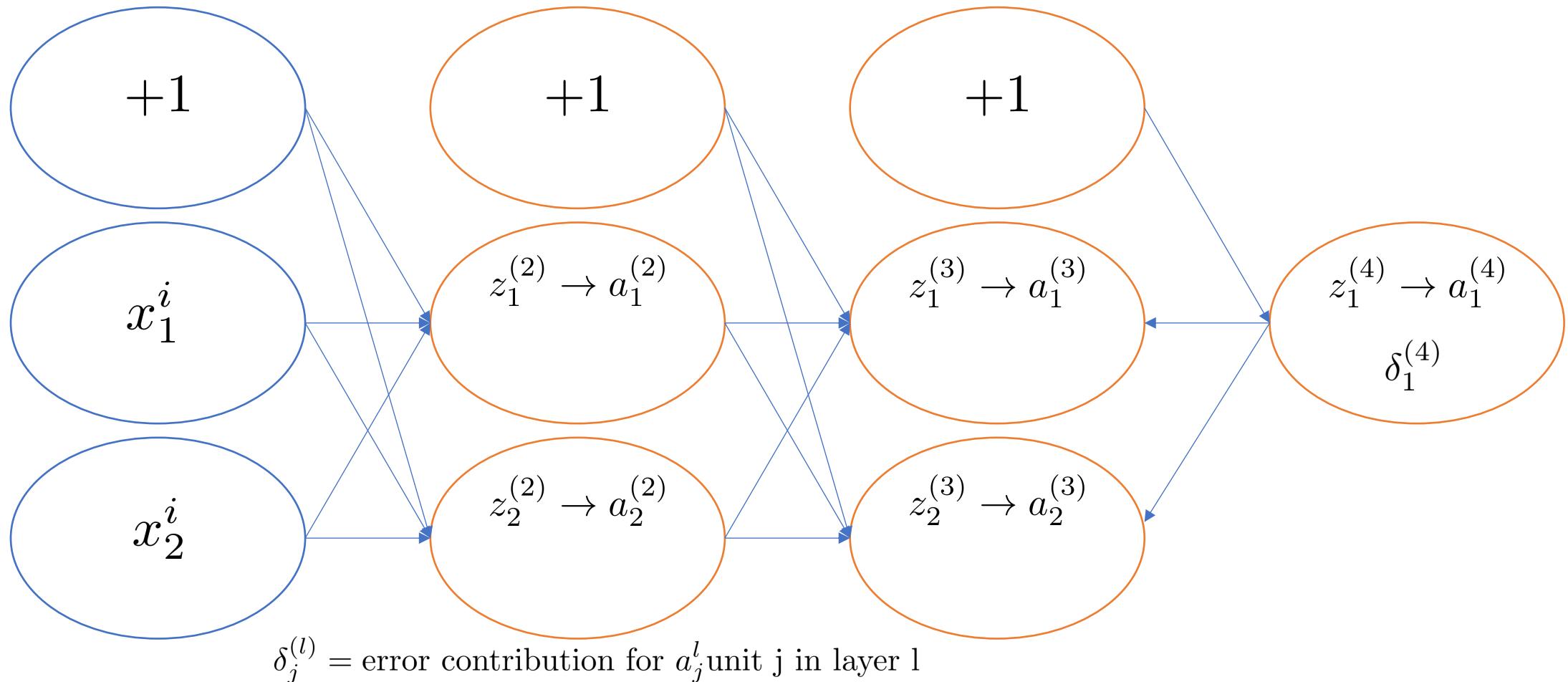


Forward Propagation



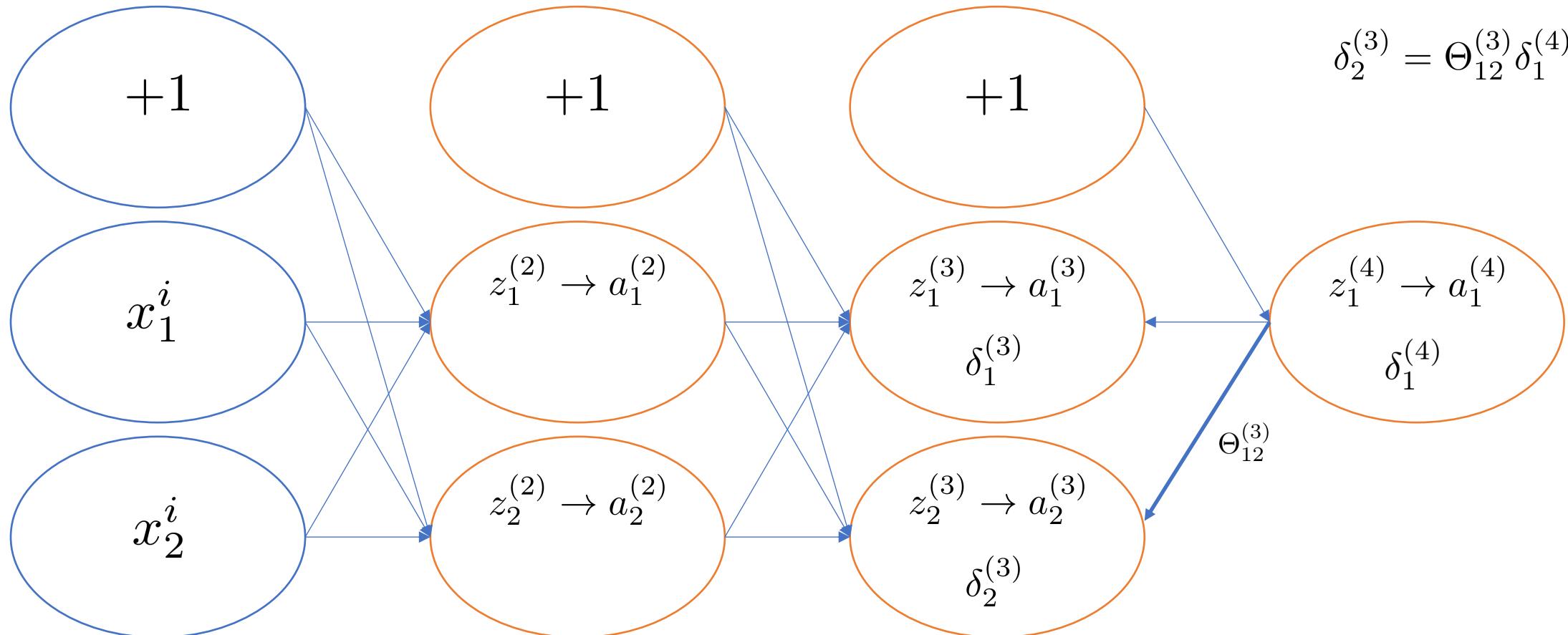
Backward Propagation

$$\delta_1^{(4)} = y^{(i)} - a_1^{(i)}$$



Formally $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(i)$ for $j \geq 0$ with $\text{cost}(i) = y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)$

Backward Propagation

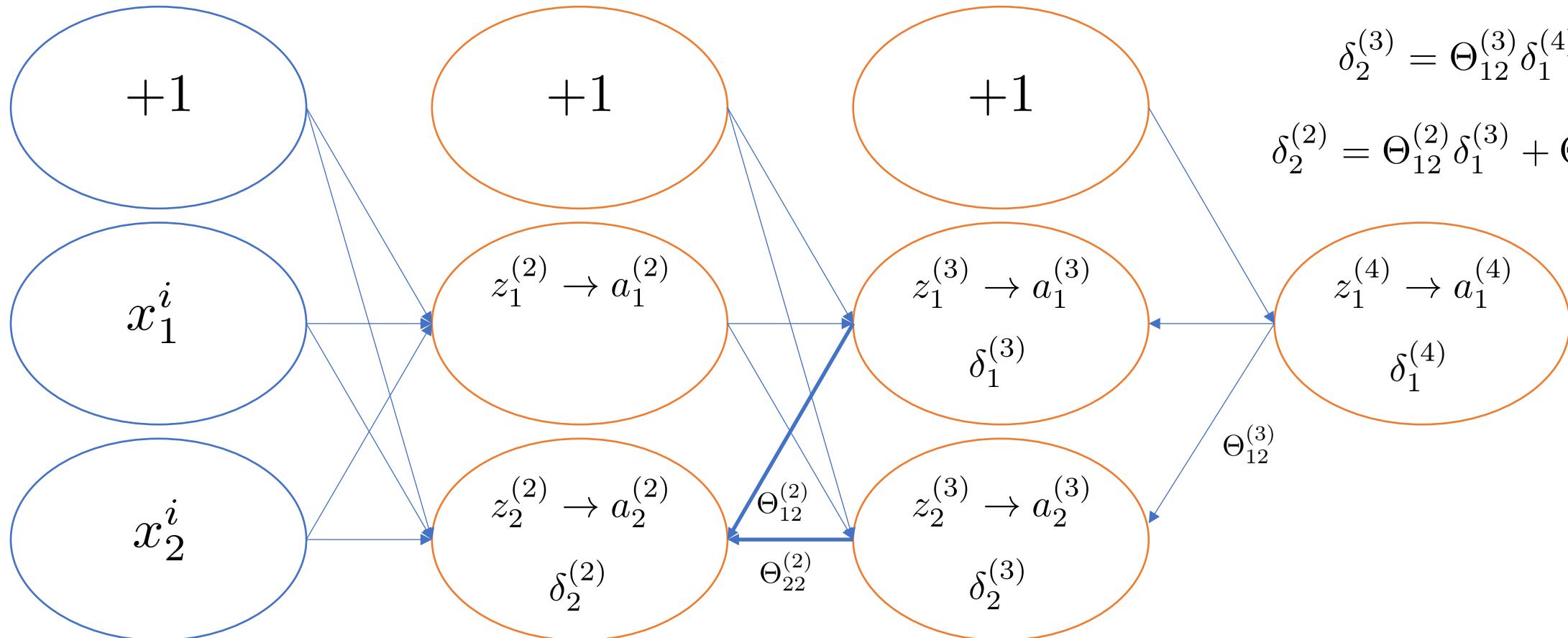


$$\delta_1^{(4)} = y^{(i)} - a_1^{(i)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)} \delta_1^{(4)}$$

$$\Theta_{12}^{(3)}$$

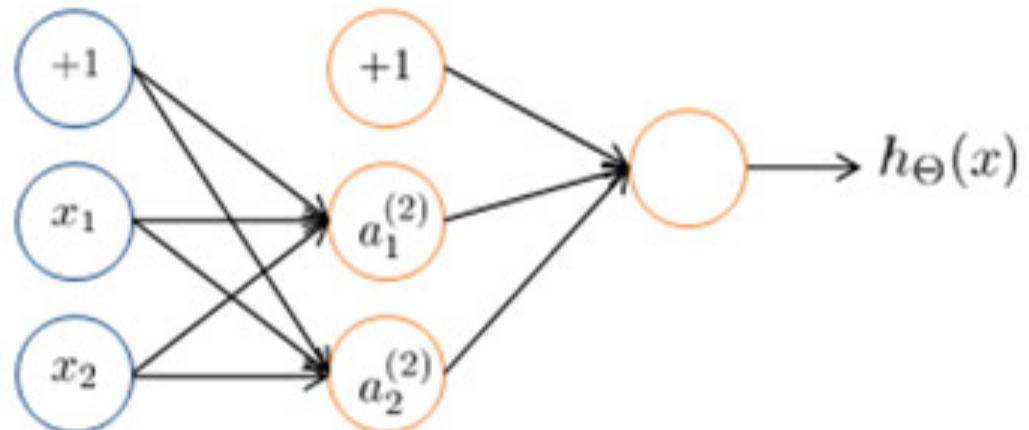
Backward Propagation



Neural Networks

Random initialization
and
Activation Functions

Zero initialization



$$\Theta_{ij}^{(l)} = 0 \text{ for all } i, j, l.$$

Can we set the initial values of Theta to zero?

What happens?

$$a_1^2 = a_2^2$$

$$\delta_1^2 = \delta_2^2$$

**After each update
all parameters are identical !!**

$$\frac{\partial}{\partial \Theta_{01}^1} J\Theta = \frac{\partial}{\partial \Theta_{02}^1} J\Theta$$

$$\Theta_{01}^1 = \Theta_{02}^1$$

Random initialization

In order to break the symmetry problem

One simple solution can be to initialize the theta parameters to really small random values:

$$\Theta_{ij}^{(l)} \in [-\epsilon, \epsilon]$$

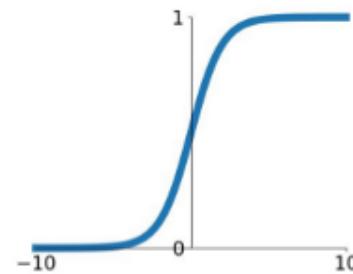
Random initialization (a bit more...)

- To prevent the gradients of the network's activations from vanishing or exploding, we will stick to the following rules of thumb:
 - The mean of the activations should be zero.
 - The variance of the activations should stay the same across every layer.
- Possible solution: select values with respect to a probability distribution
 - Standard Normal
 - Uniform
 - Xavier (this is the most effective)

Activation Functions

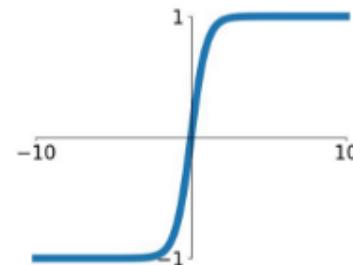
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



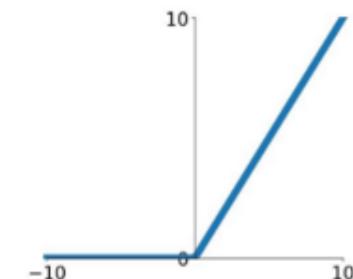
tanh

$$\tanh(x)$$

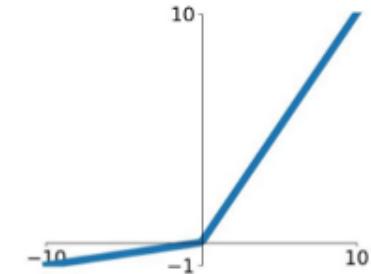


ReLU

$$\max(0, x)$$



Leaky ReLU
 $\max(0.1x, x)$

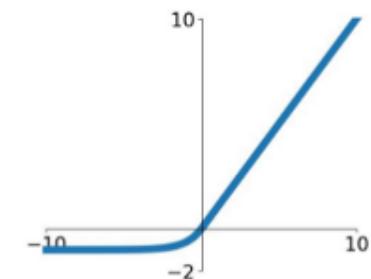


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

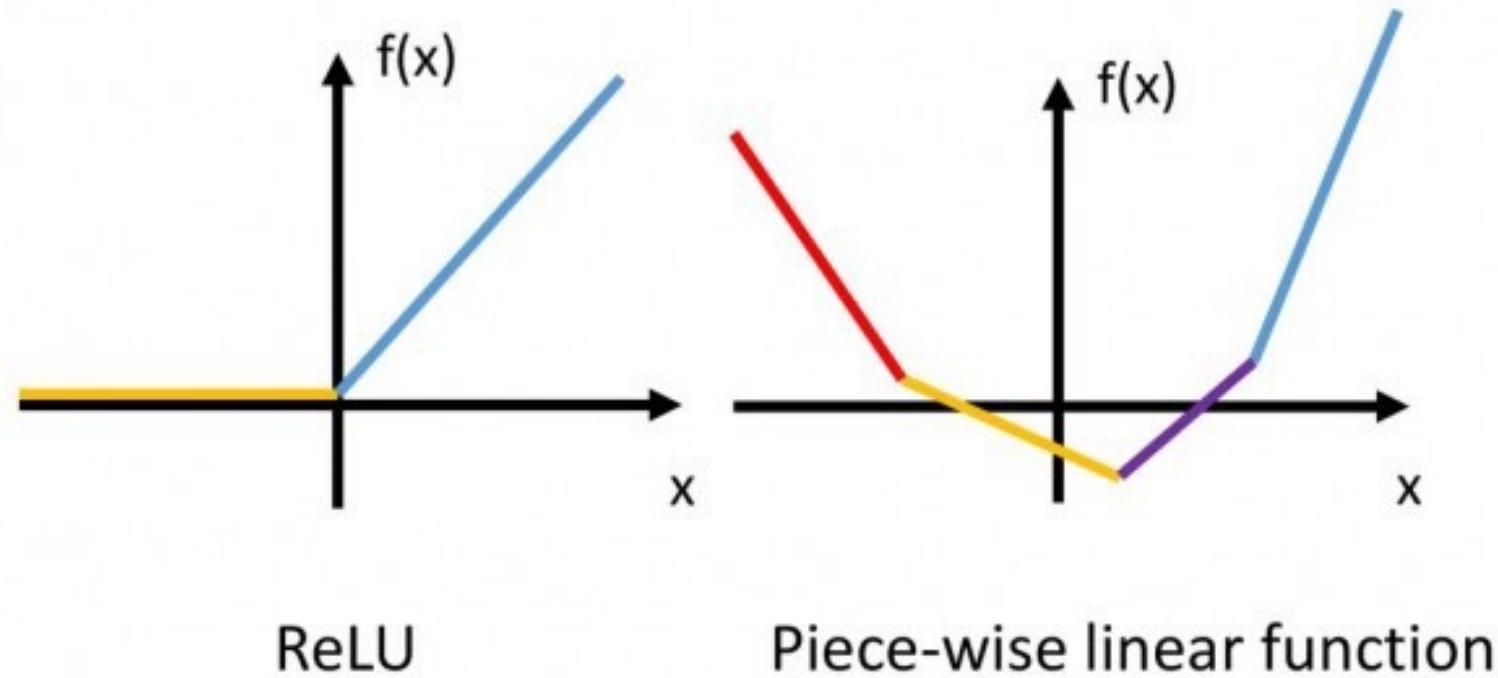
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



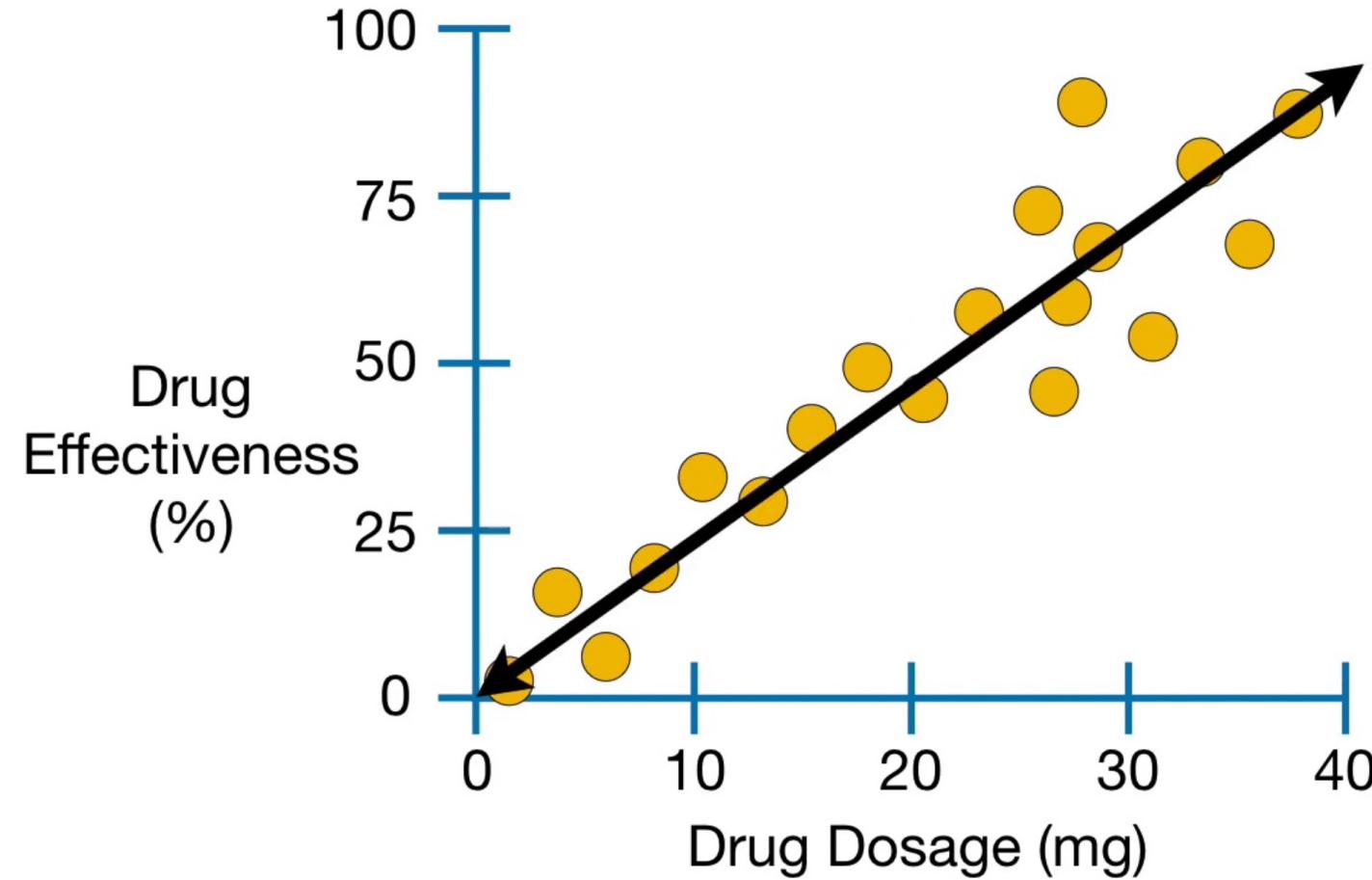
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

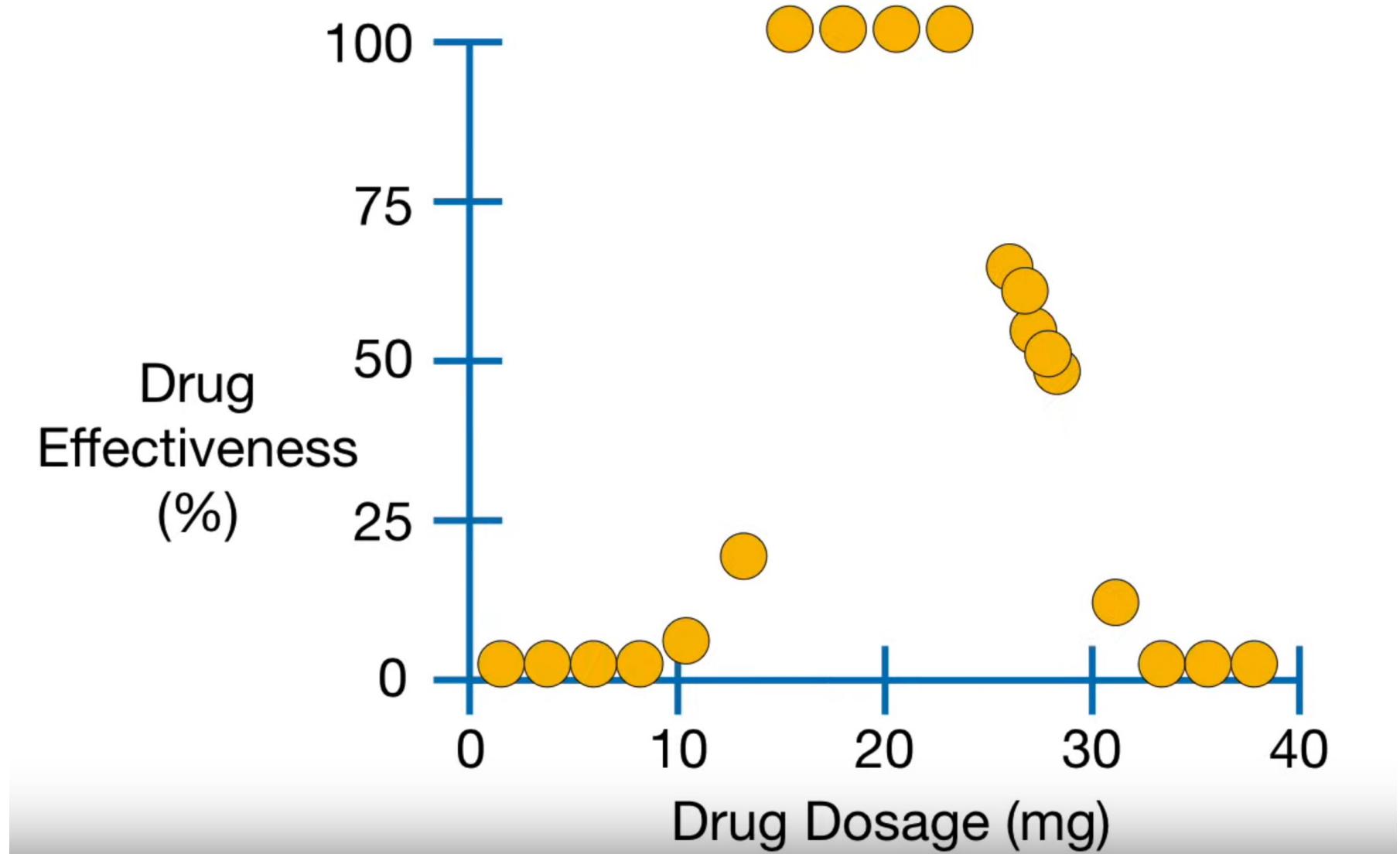


Regression Tree

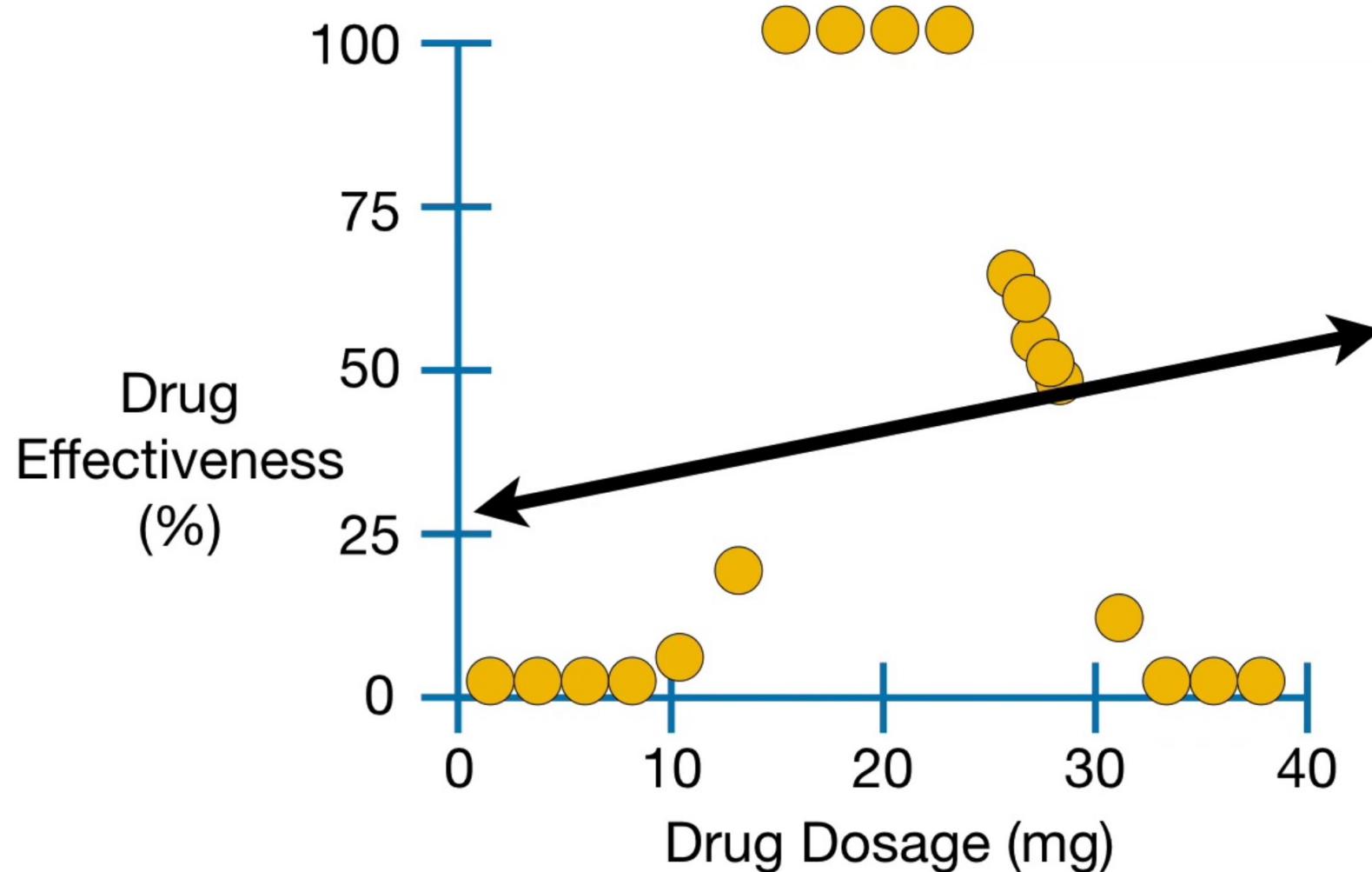
Linear Regression model



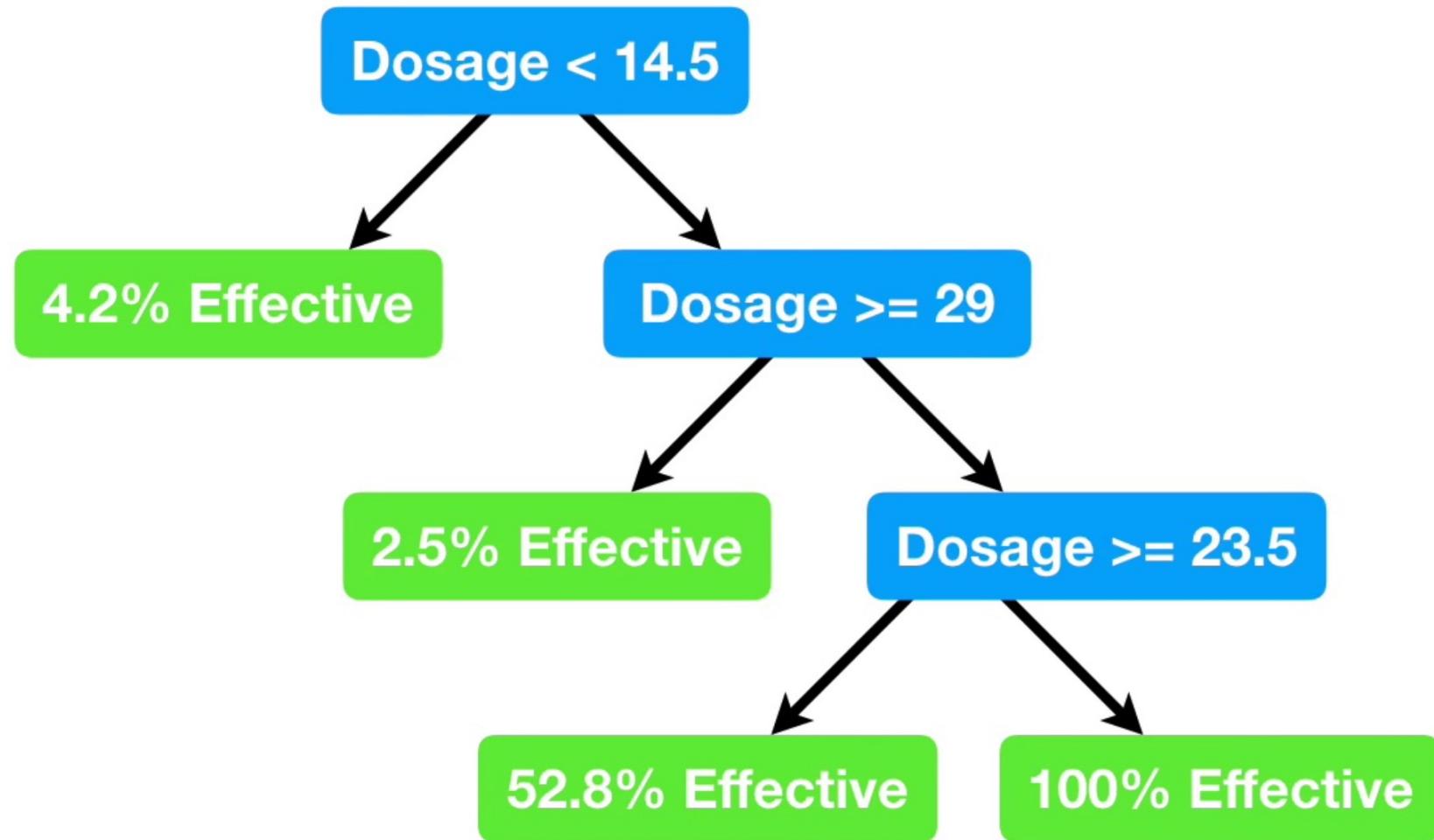
Limits



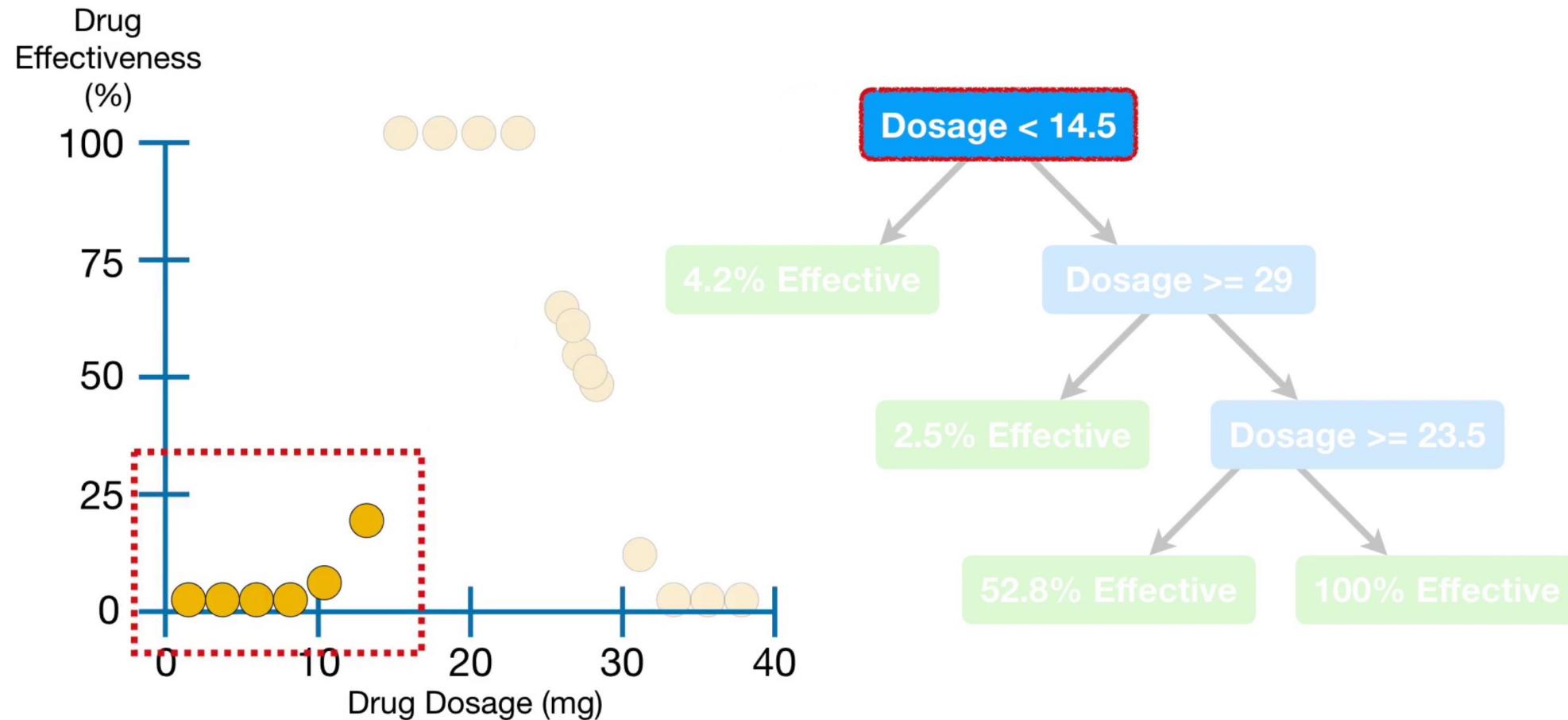
Limits



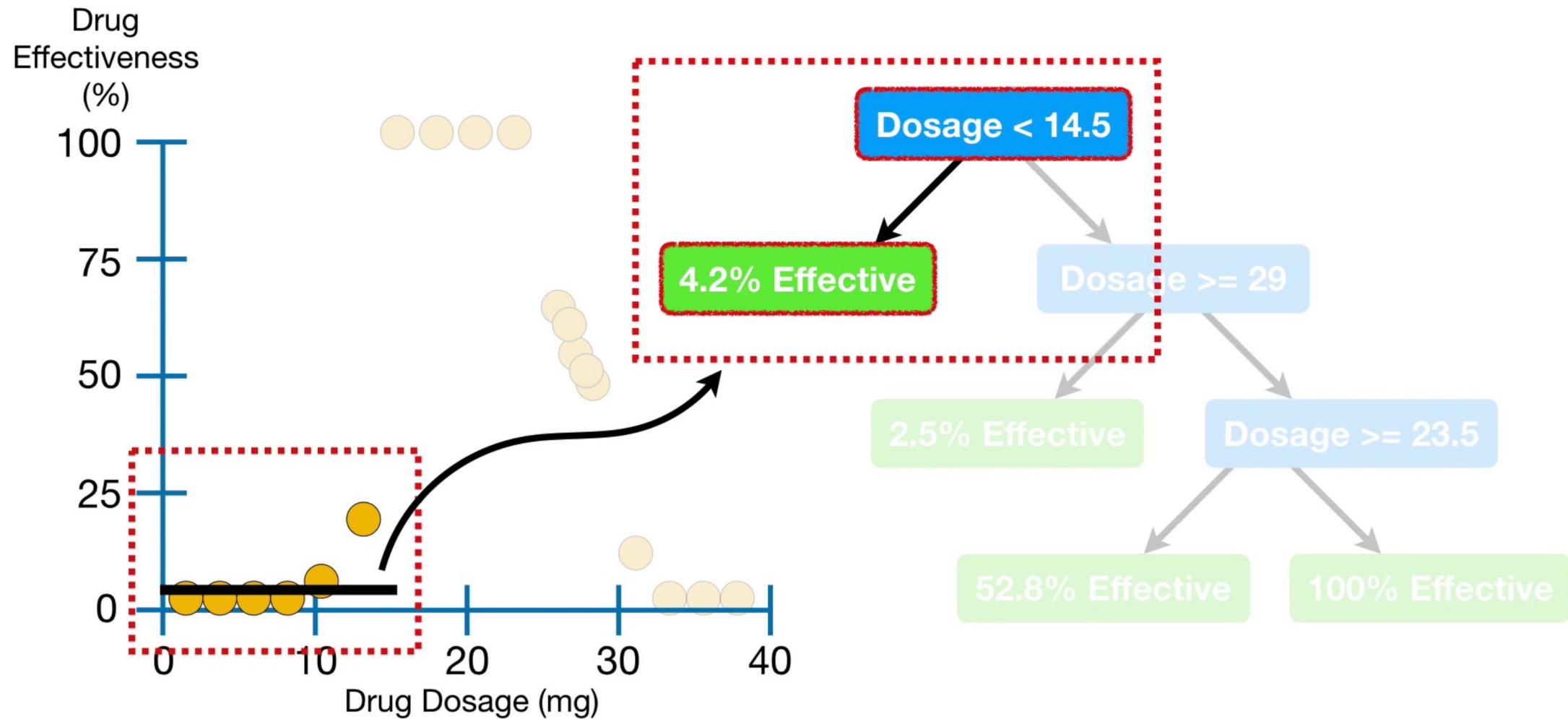
Regression Tree



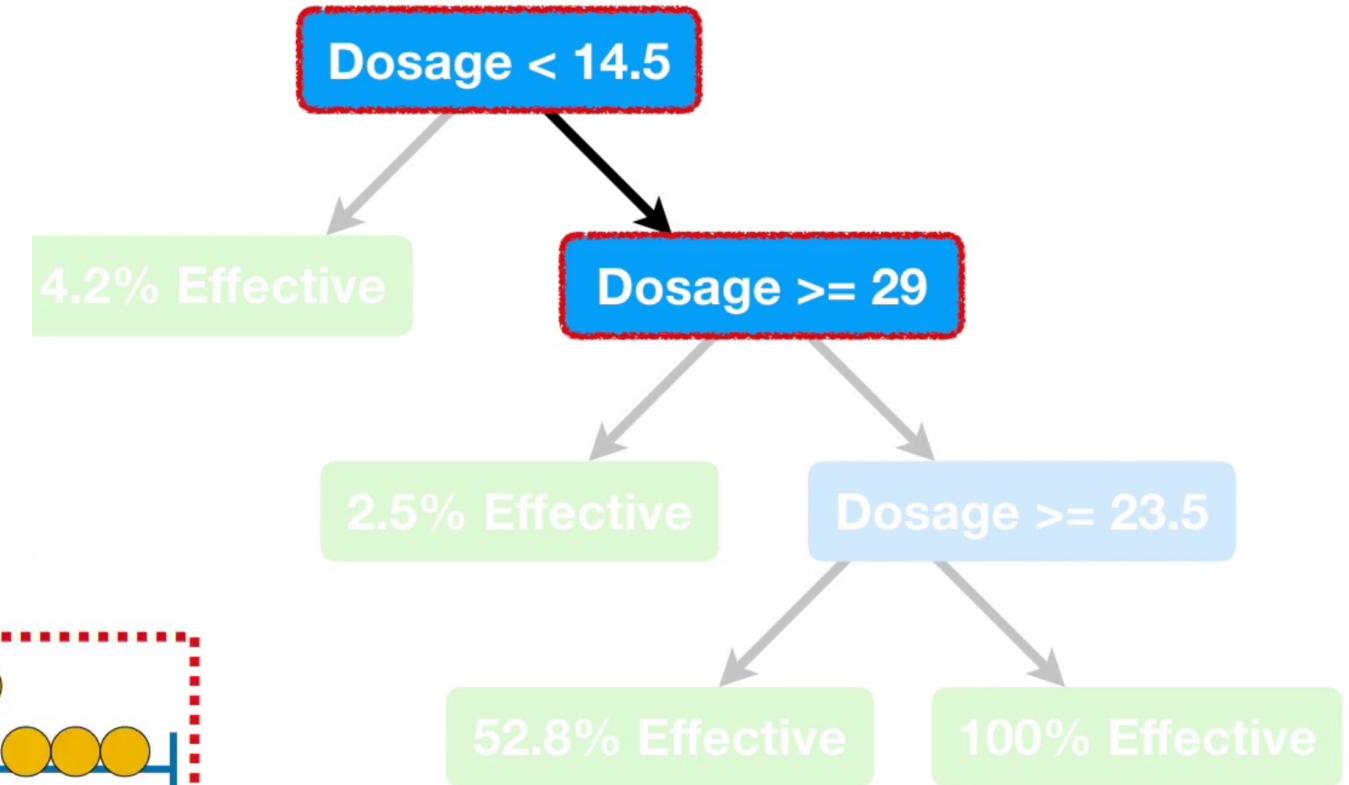
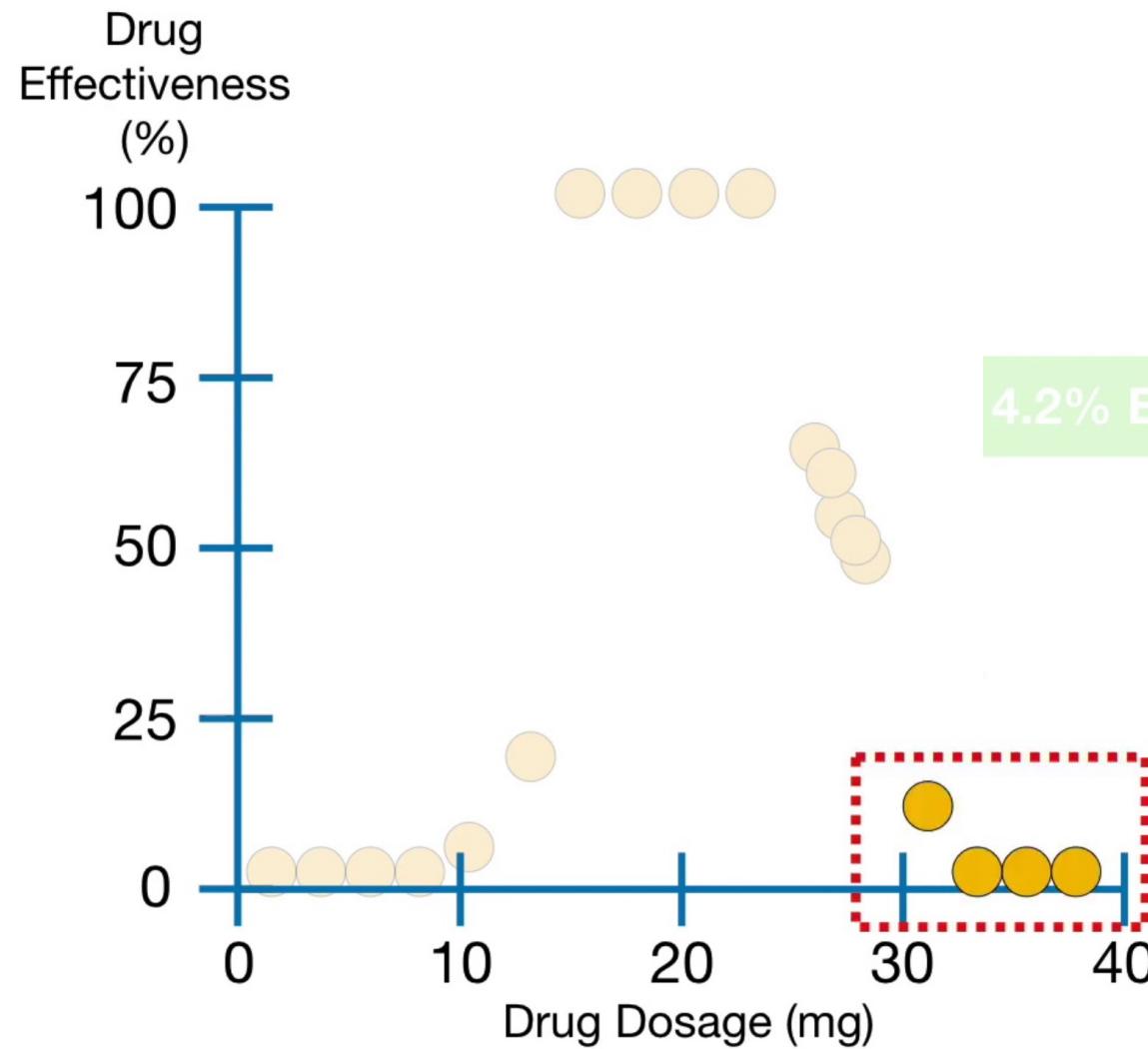
Decision nodes



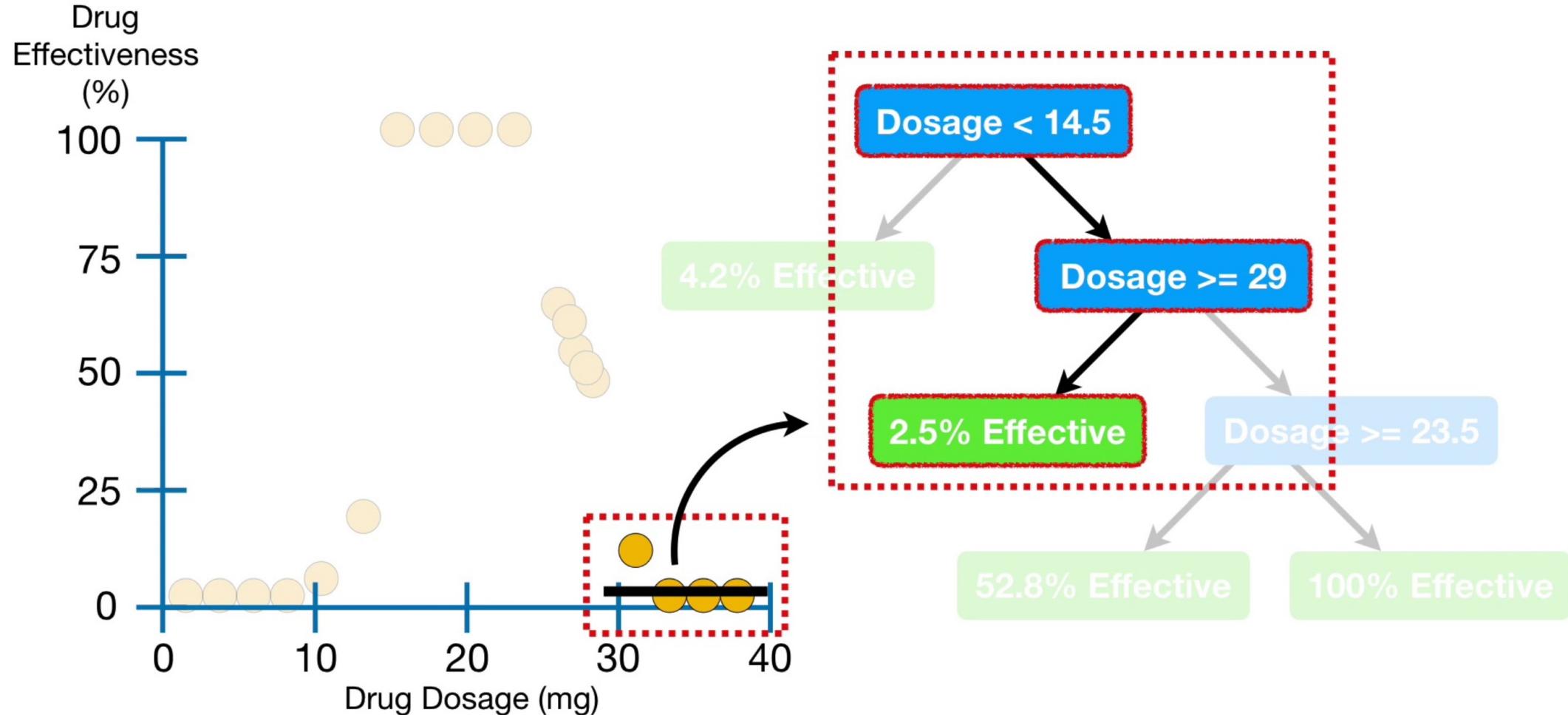
Leaves nodes



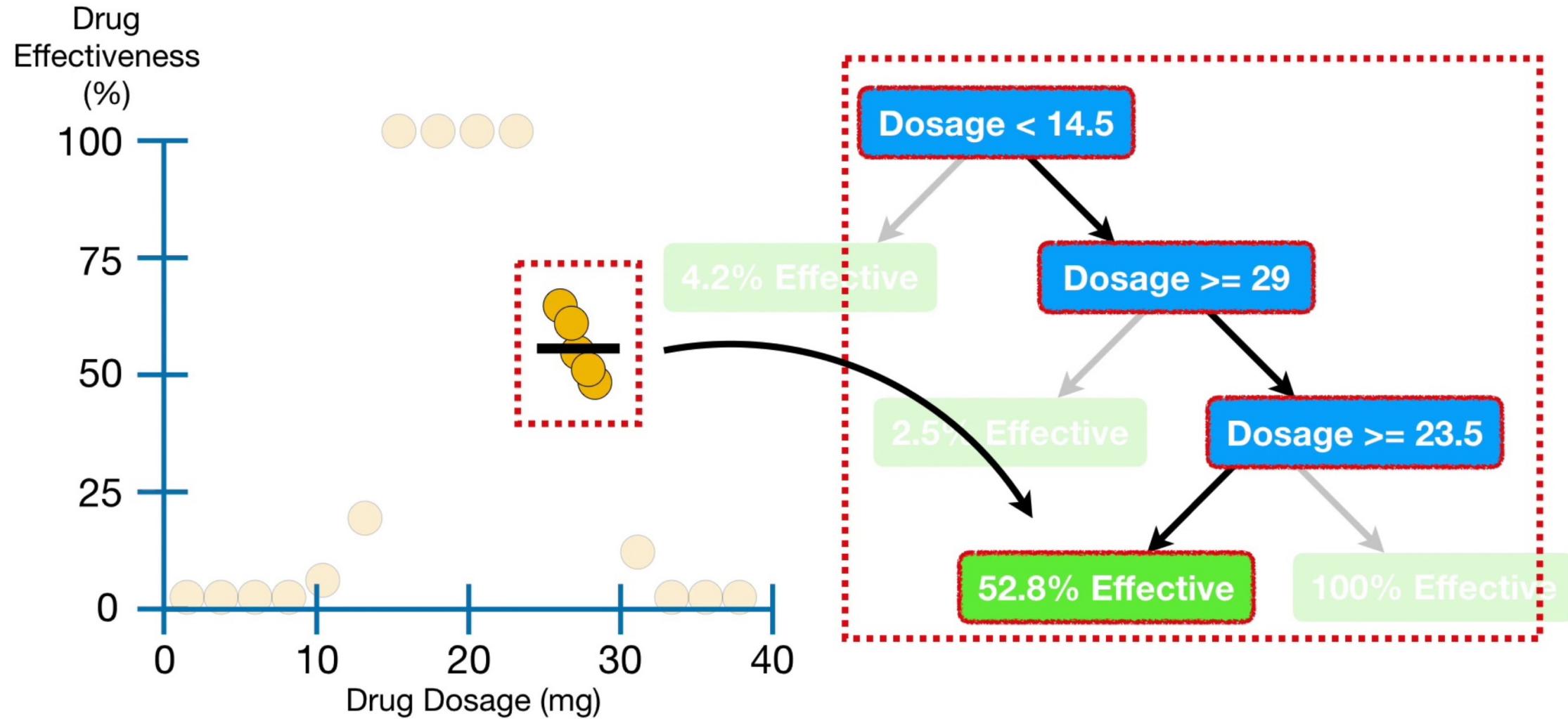
Next nodes



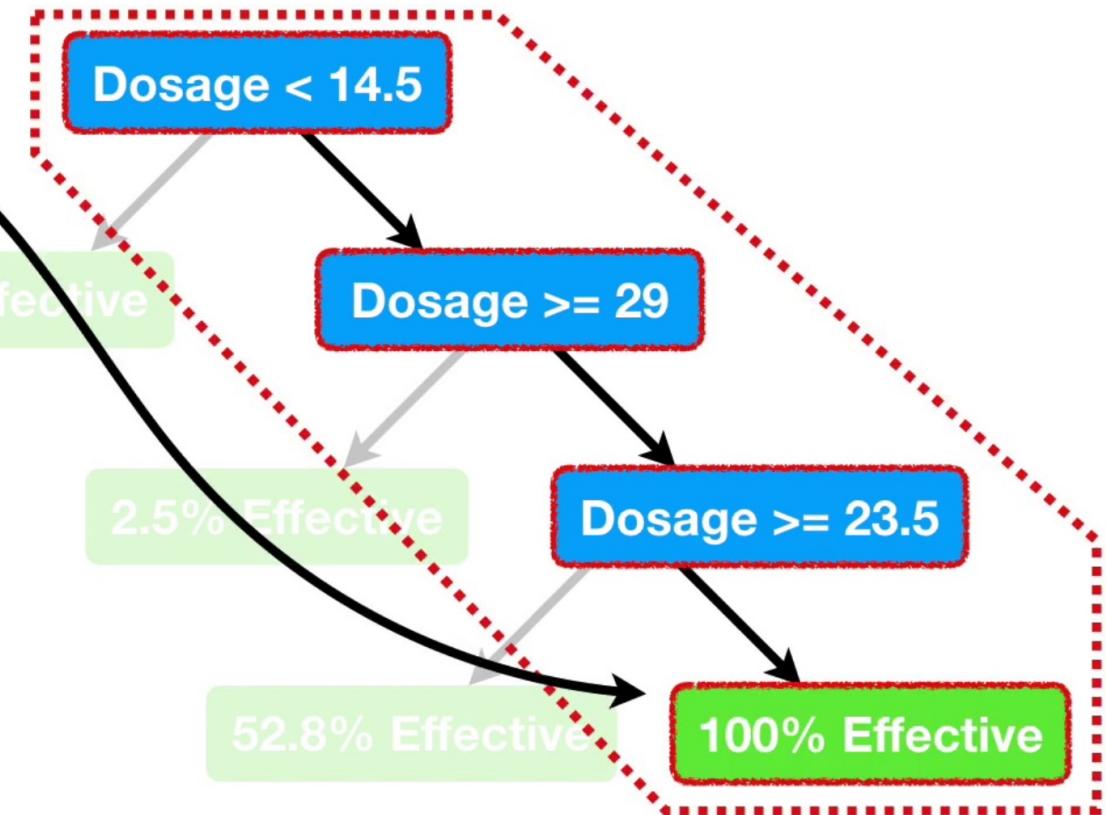
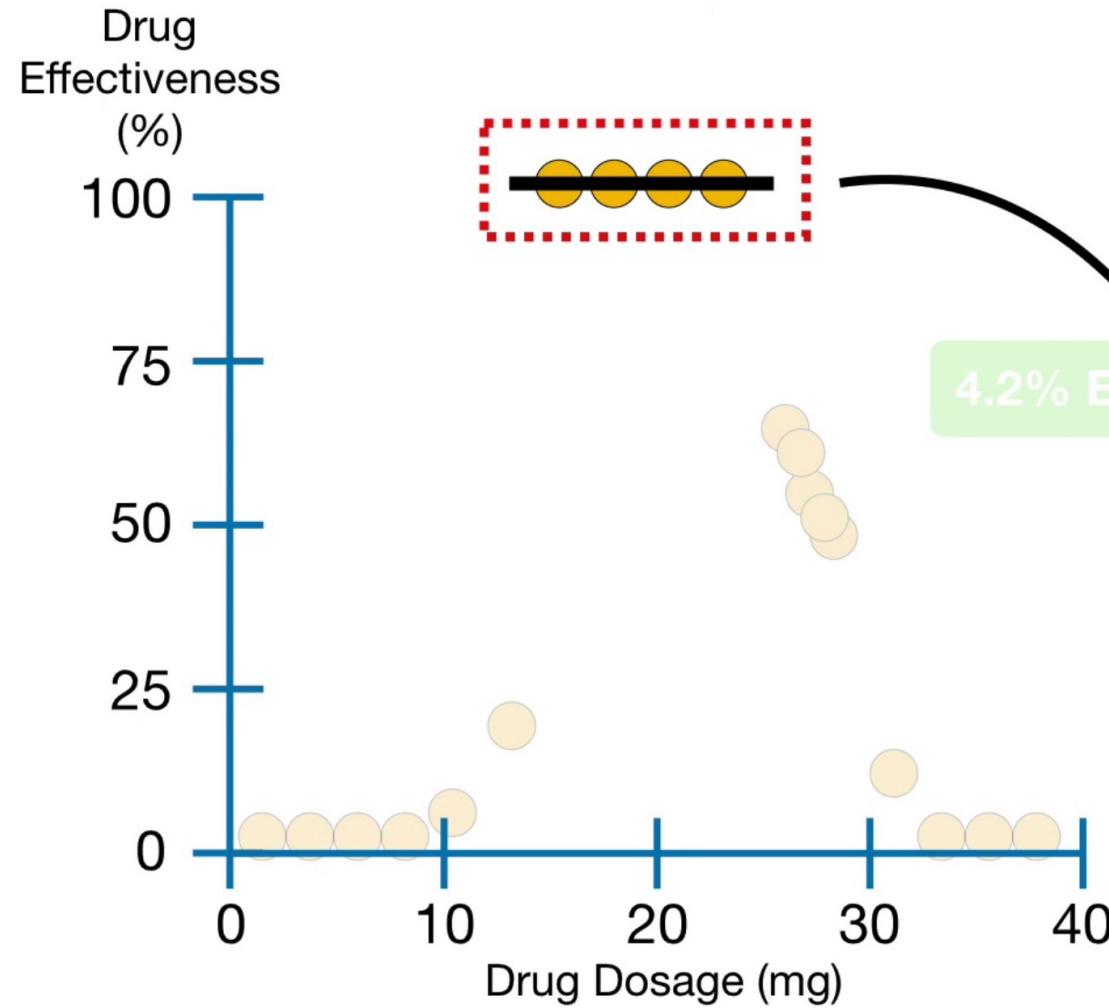
Next Nodes



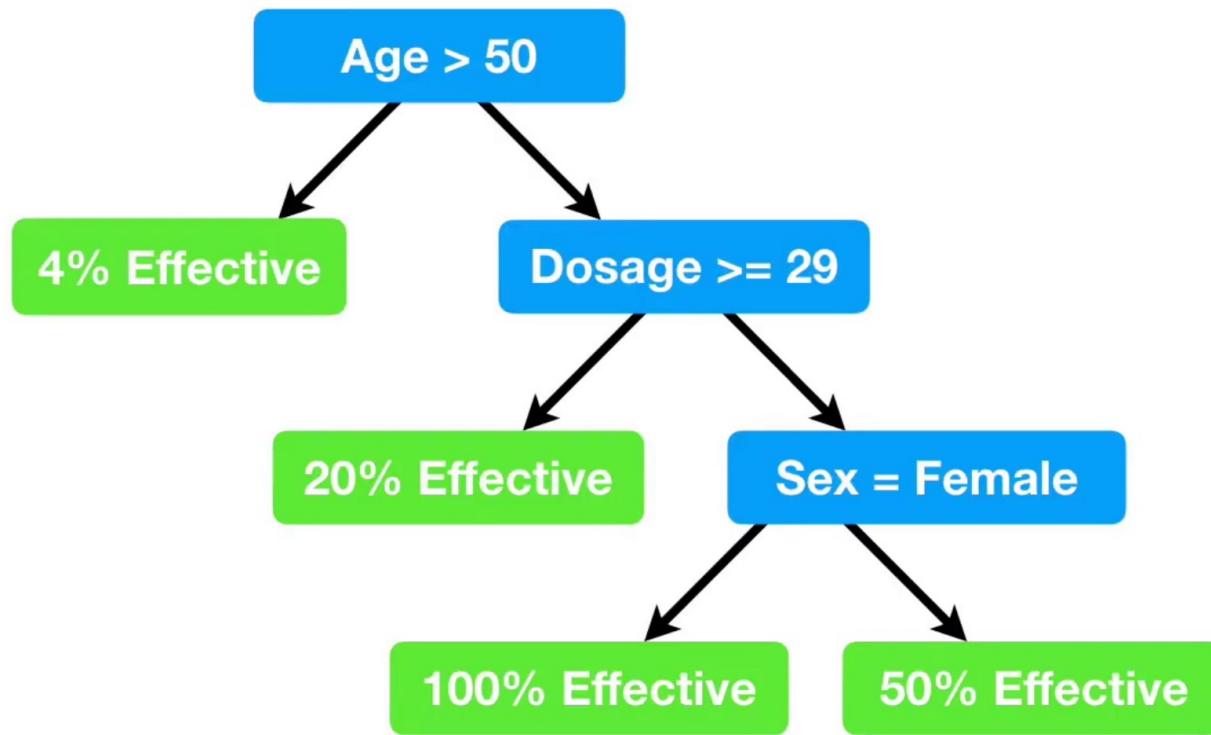
Next Nodes



Next Nodes

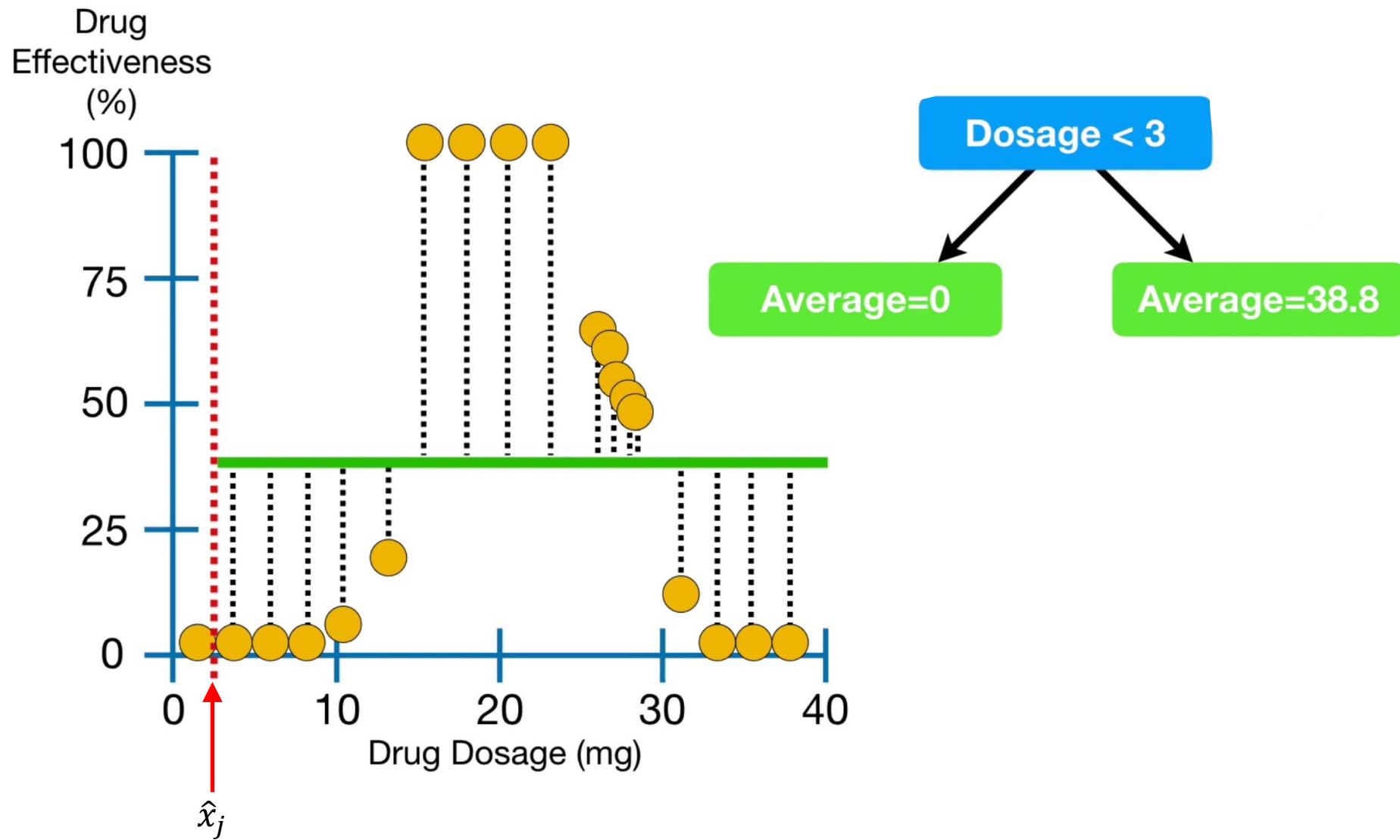


Multiple Features



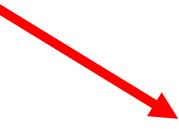
Dosage	Age	Sex	Etc.	Drug Effect.
10	25	Female	...	98
20	73	Male	...	0
35	54	Female	...	100
5	12	Male	...	44
etc...	etc...	etc...	etc...	etc...

Building a Regression Tree



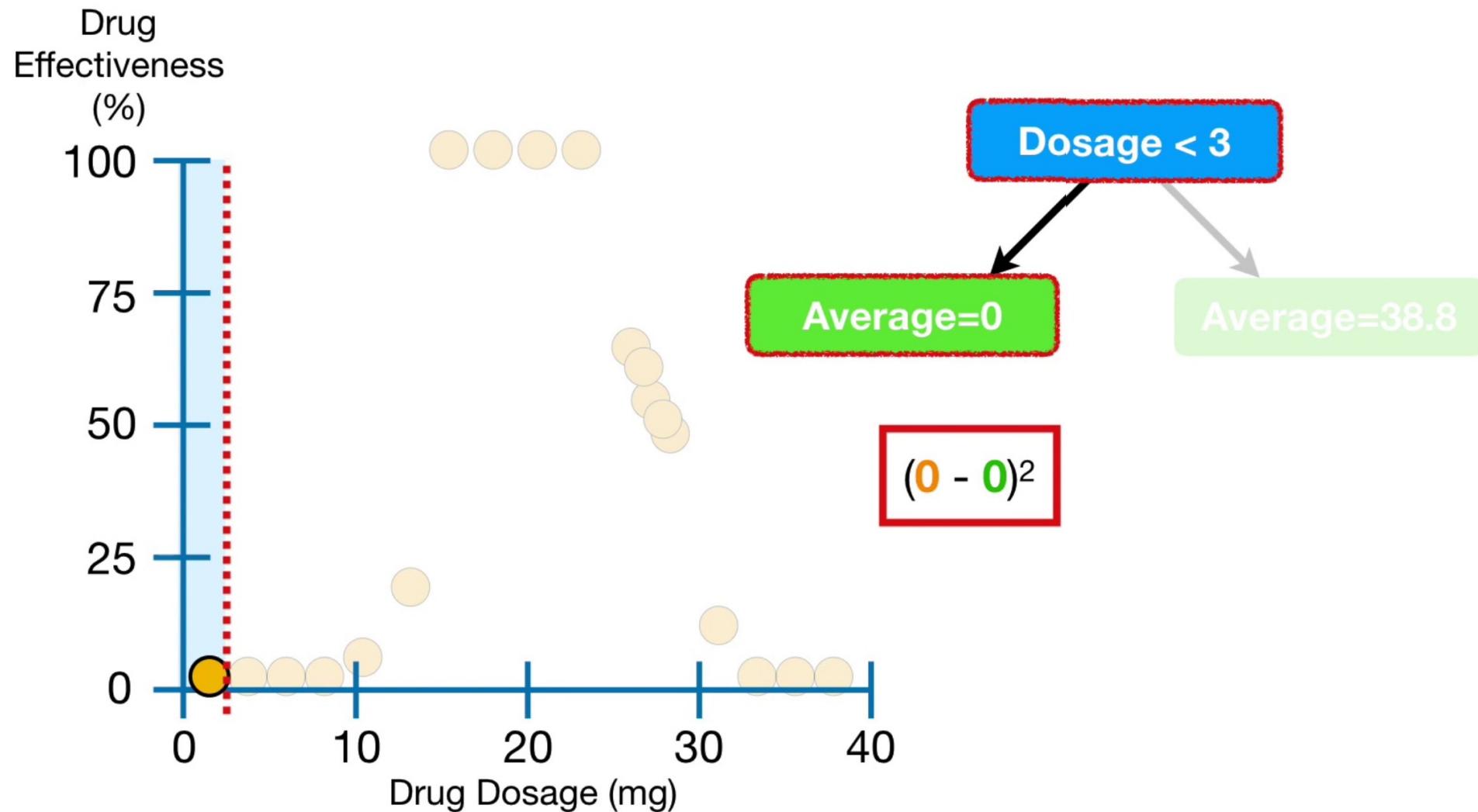
Hypotesis and Residual Sum of Squares (RSS)

Average value

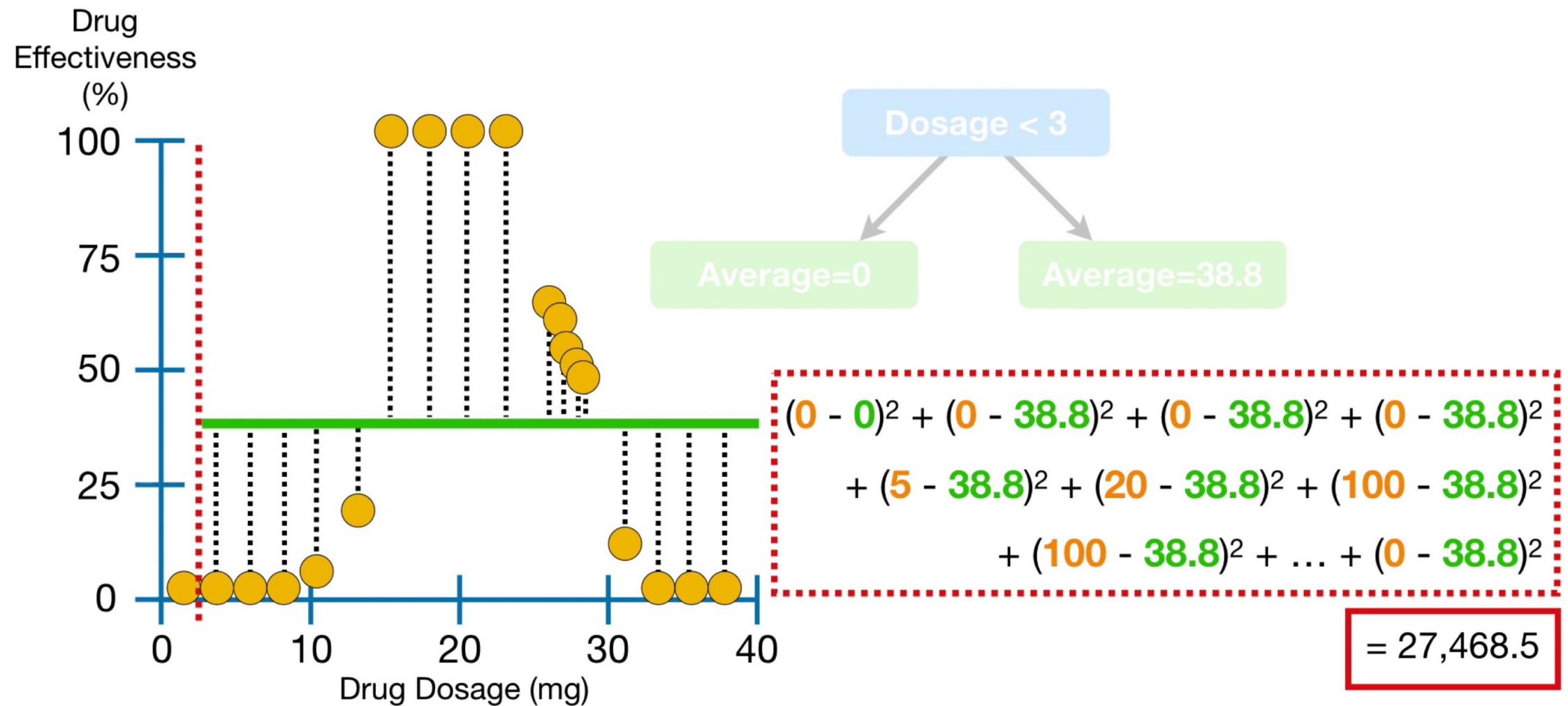

$$h(x^{(i)}) = \sum_{\hat{x}_{j-1} < x^{(i)} < \hat{x}_j} \frac{y^{(i)}}{|\{x^{(i)} : \hat{x}_{j-1} < x^{(i)} < \hat{x}_j\}|}$$

$$RSS = \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

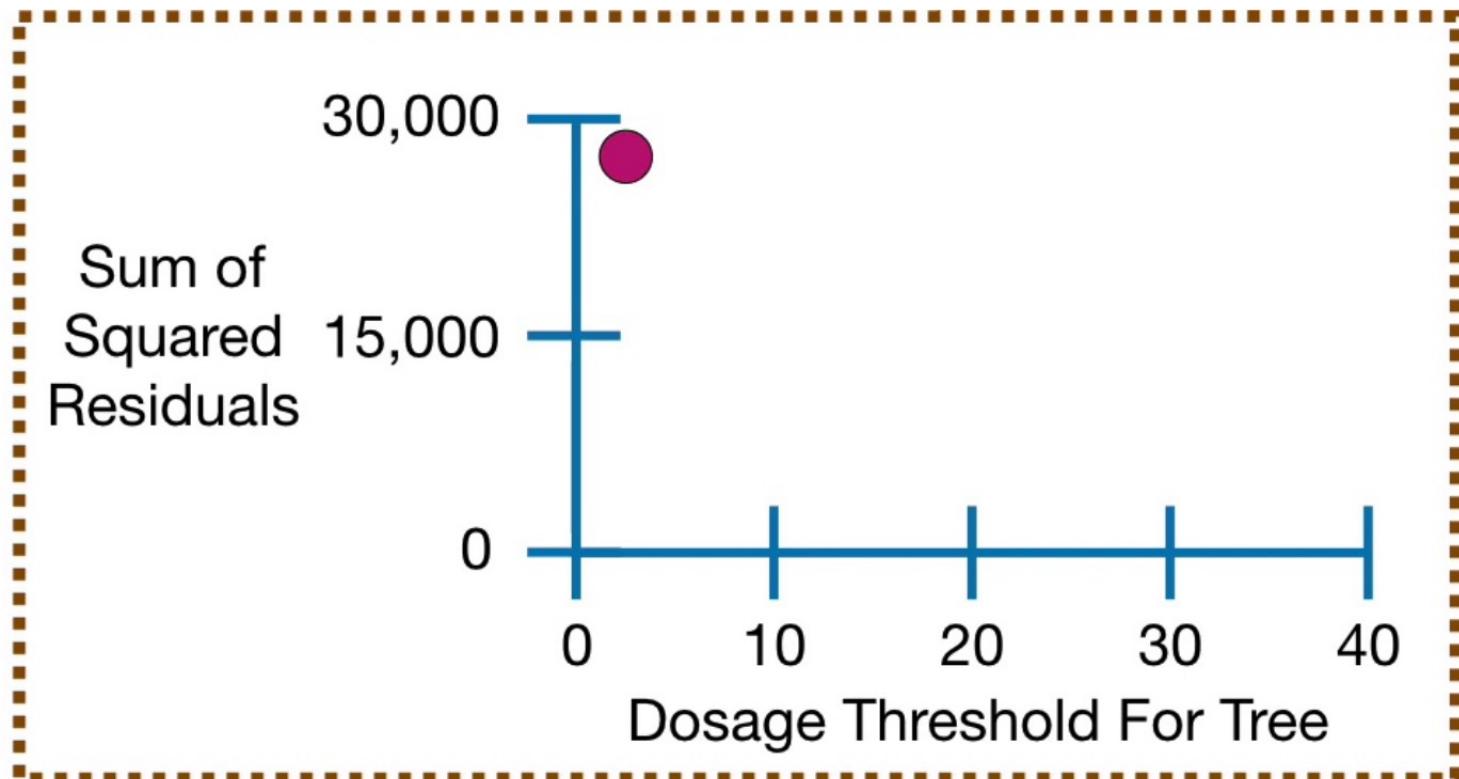
RSS



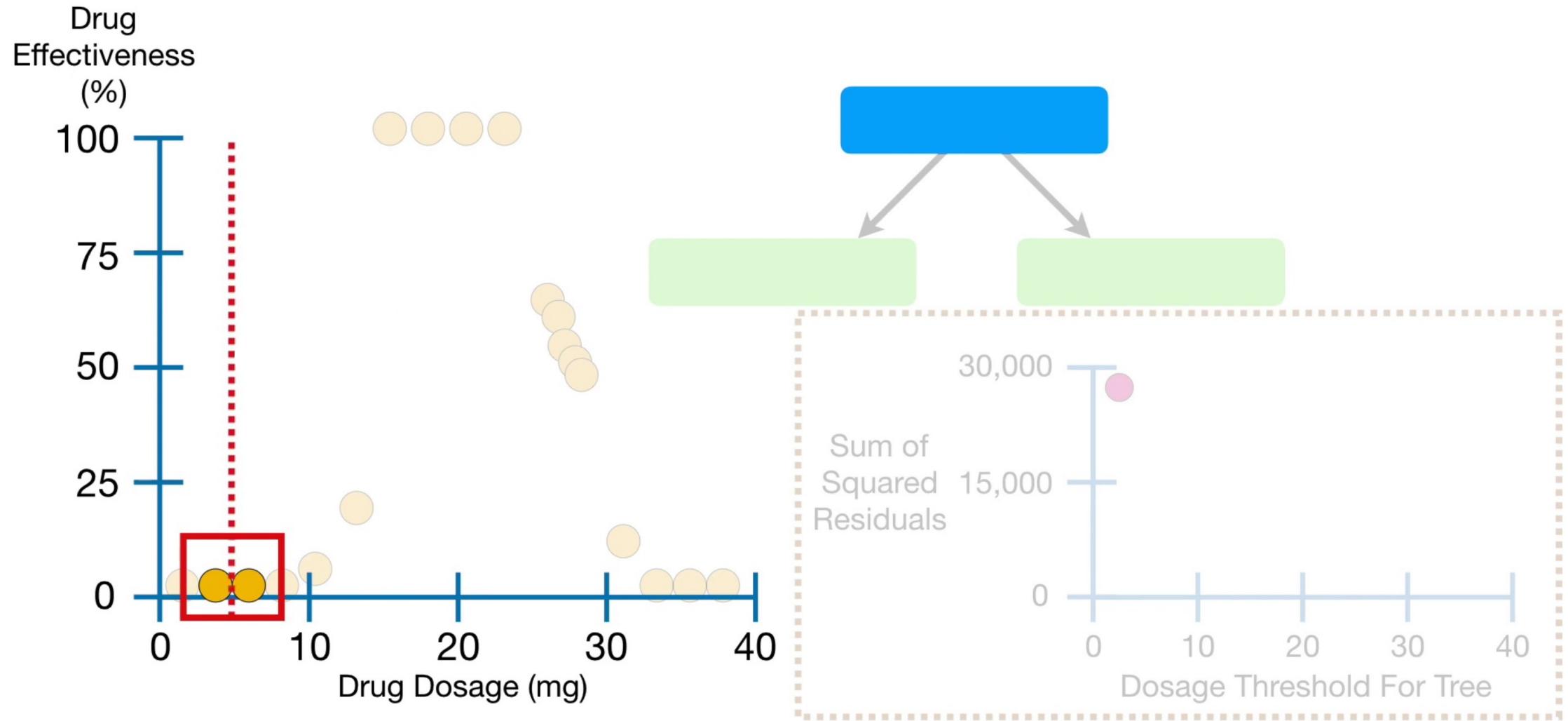
RSS



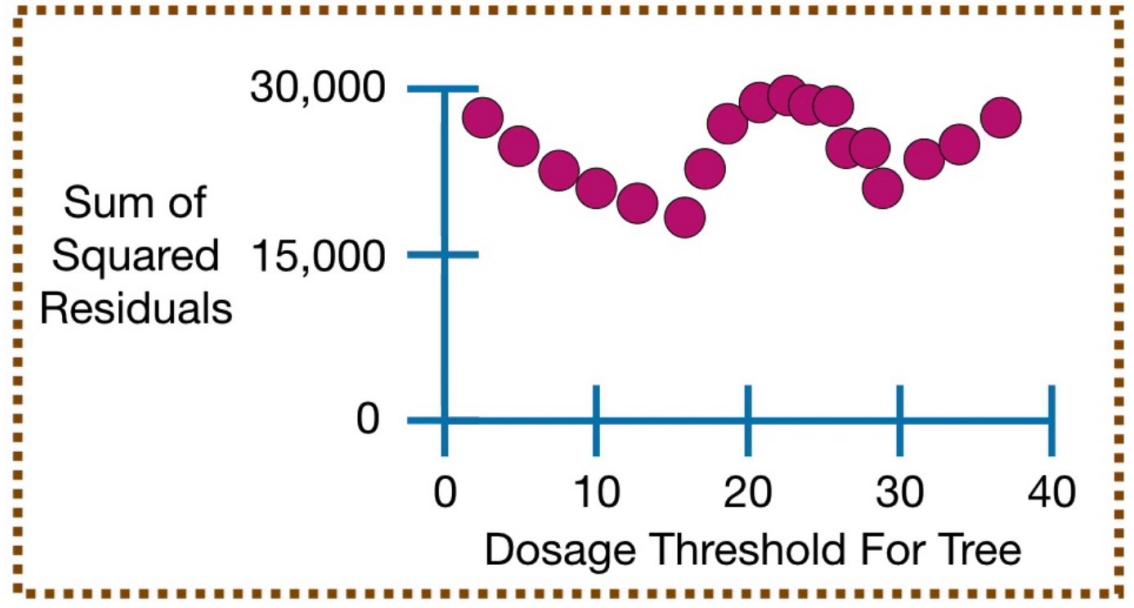
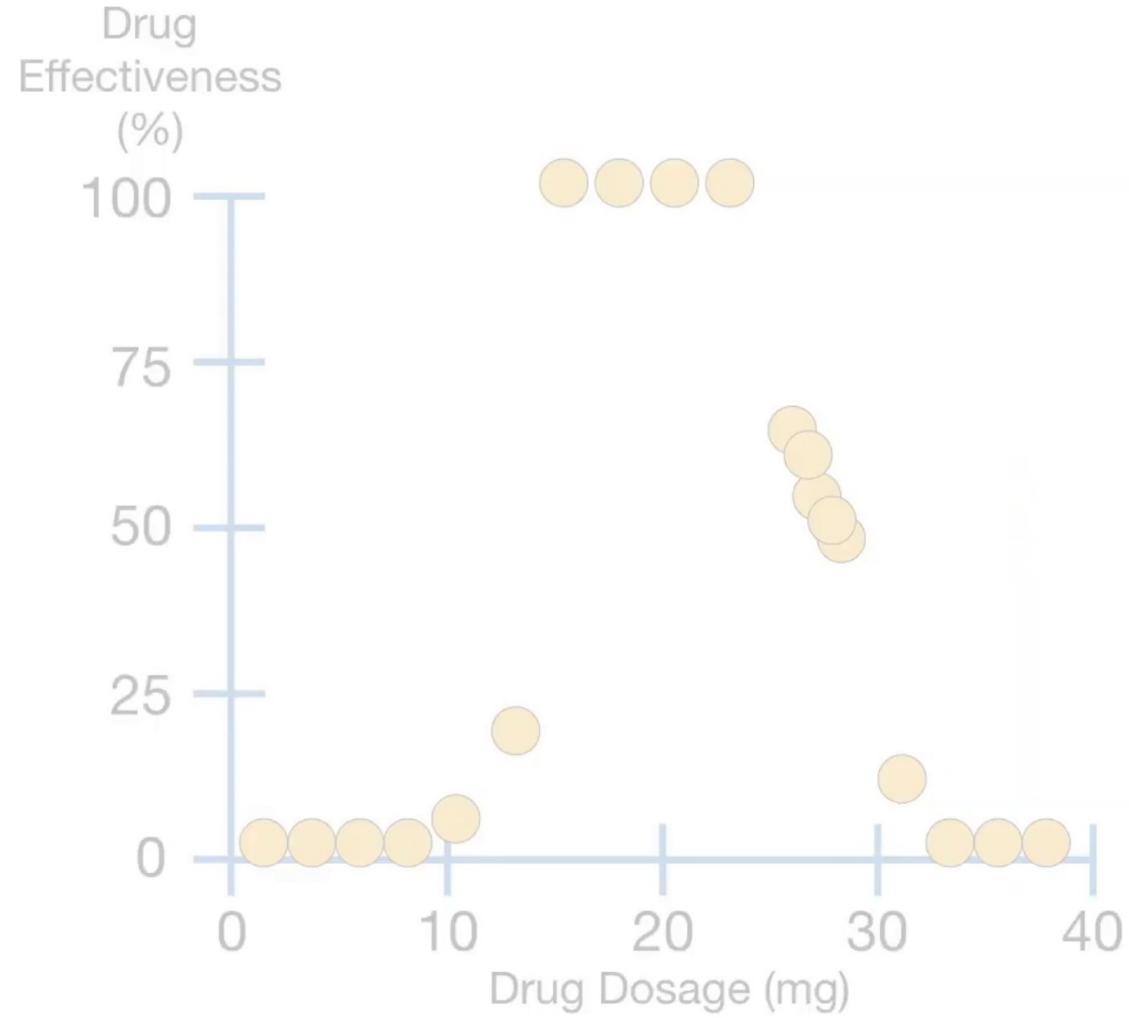
RSS plot



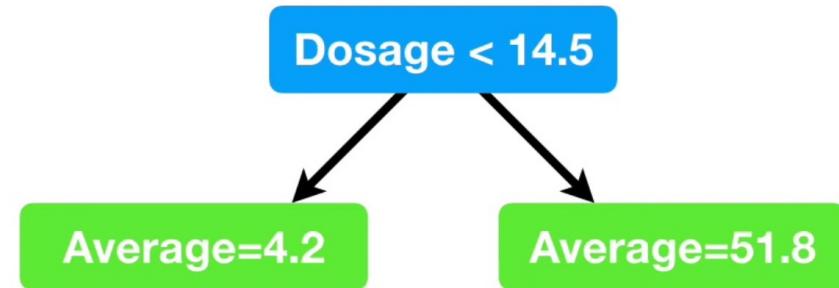
Building a Regression Tree



Building a Regression Tree

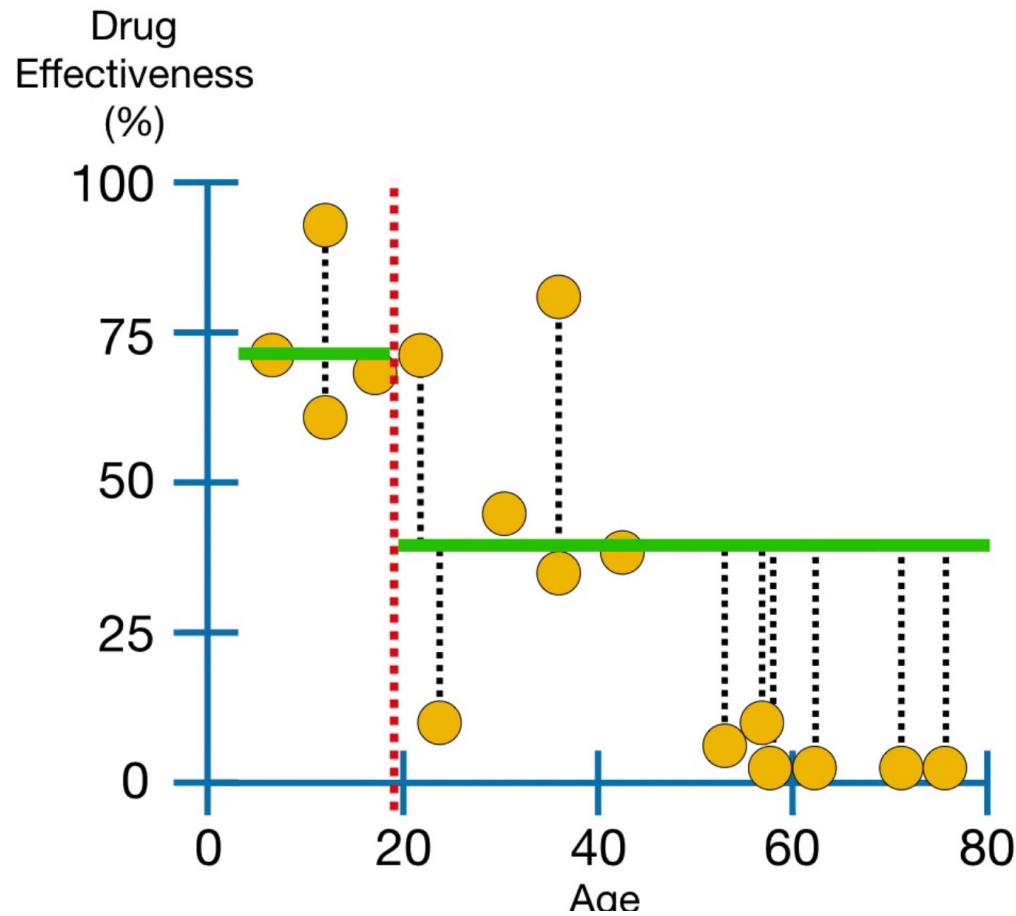


Multiple Features



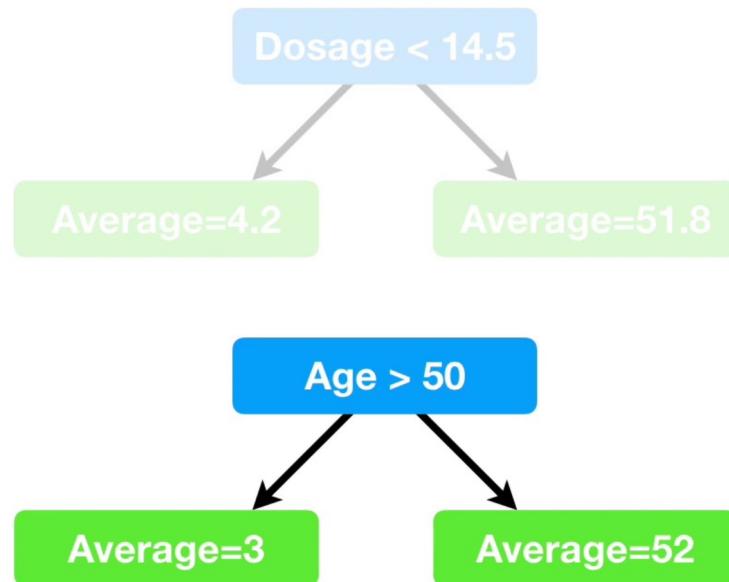
Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc...	etc...	etc...	etc...

Multiple Features



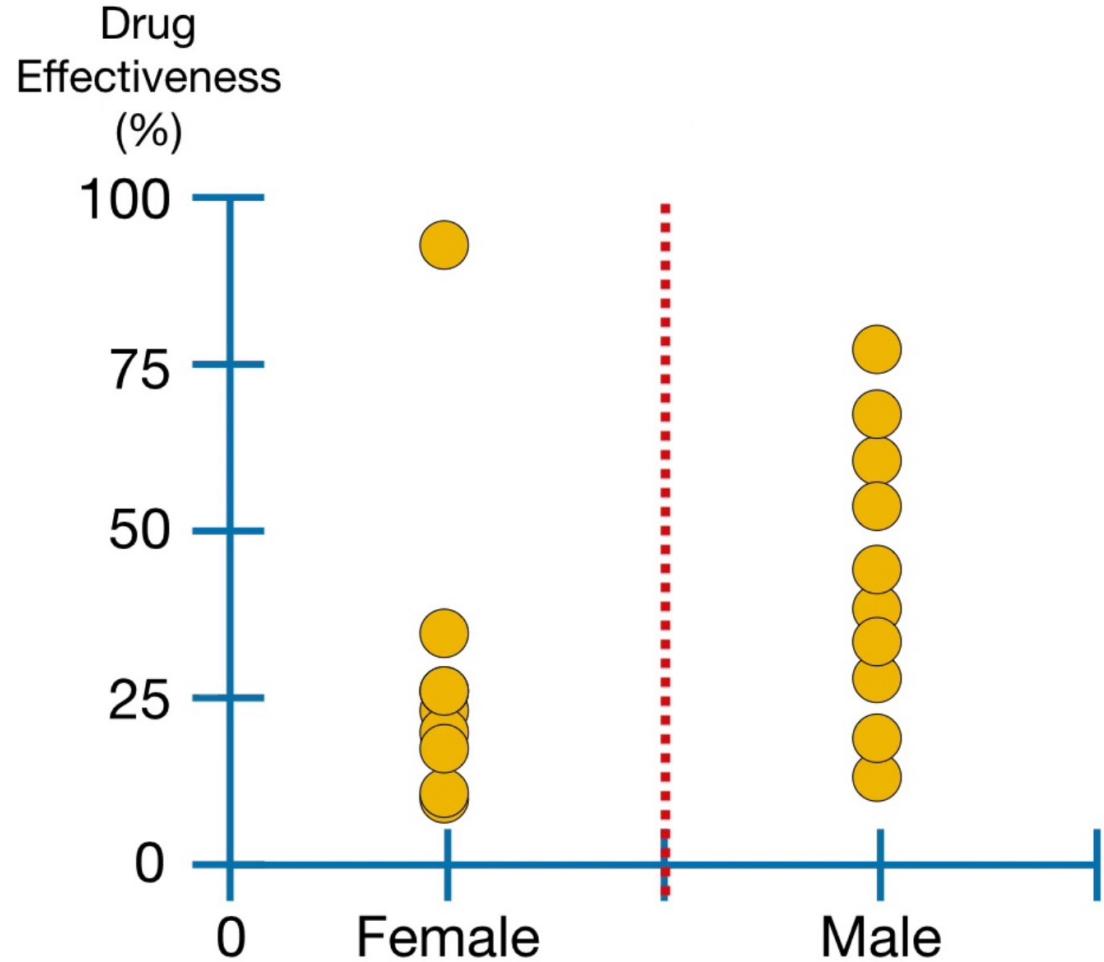
Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc...	etc...	etc...	etc...

Multiple Features



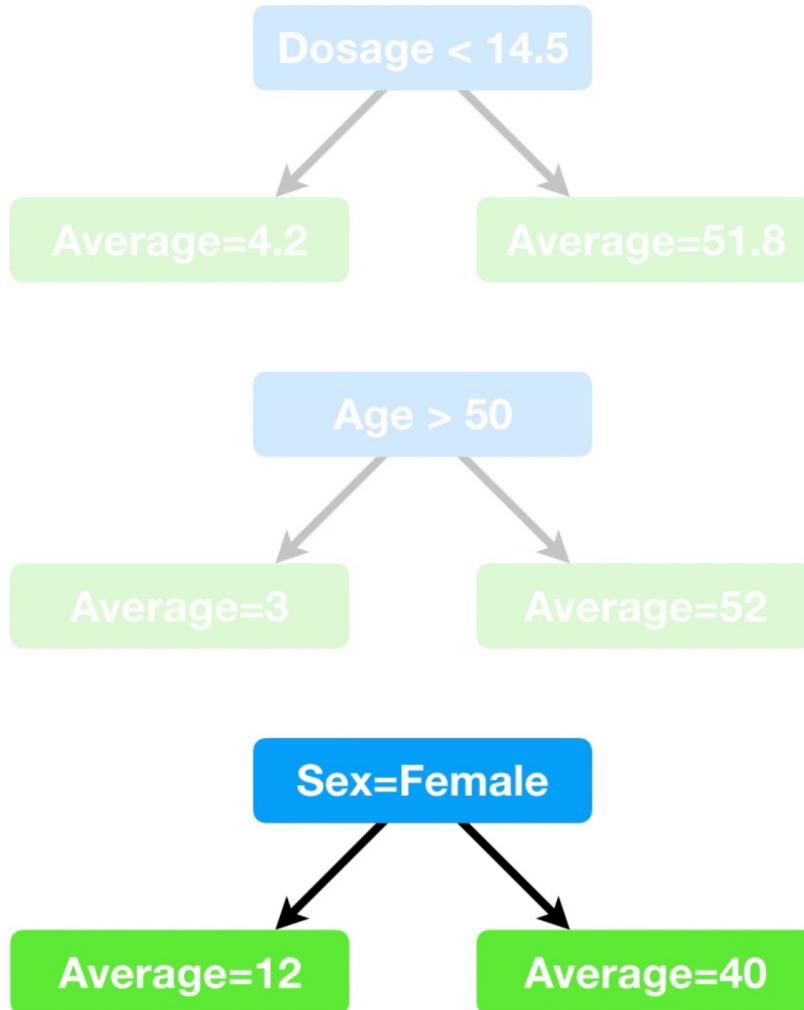
Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc...	etc...	etc...	etc...

Multiple Features



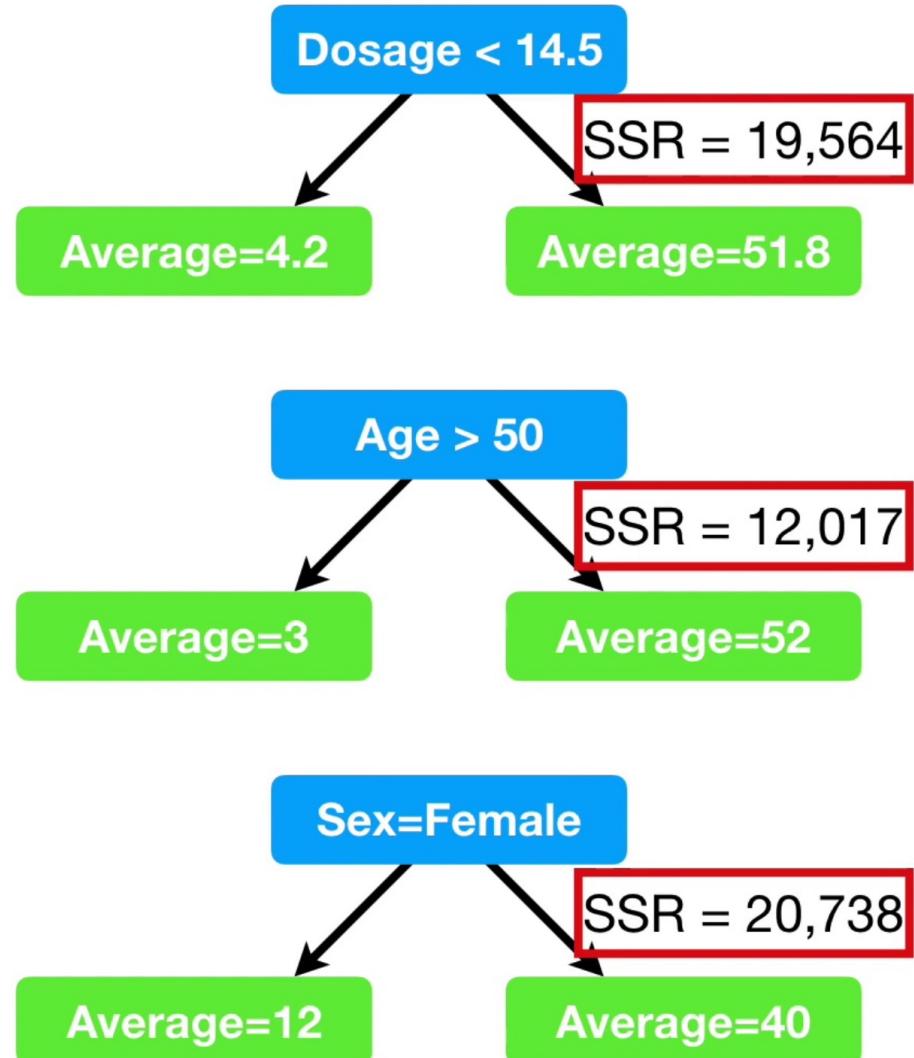
Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc...	etc...	etc...	etc...

Multiple Features

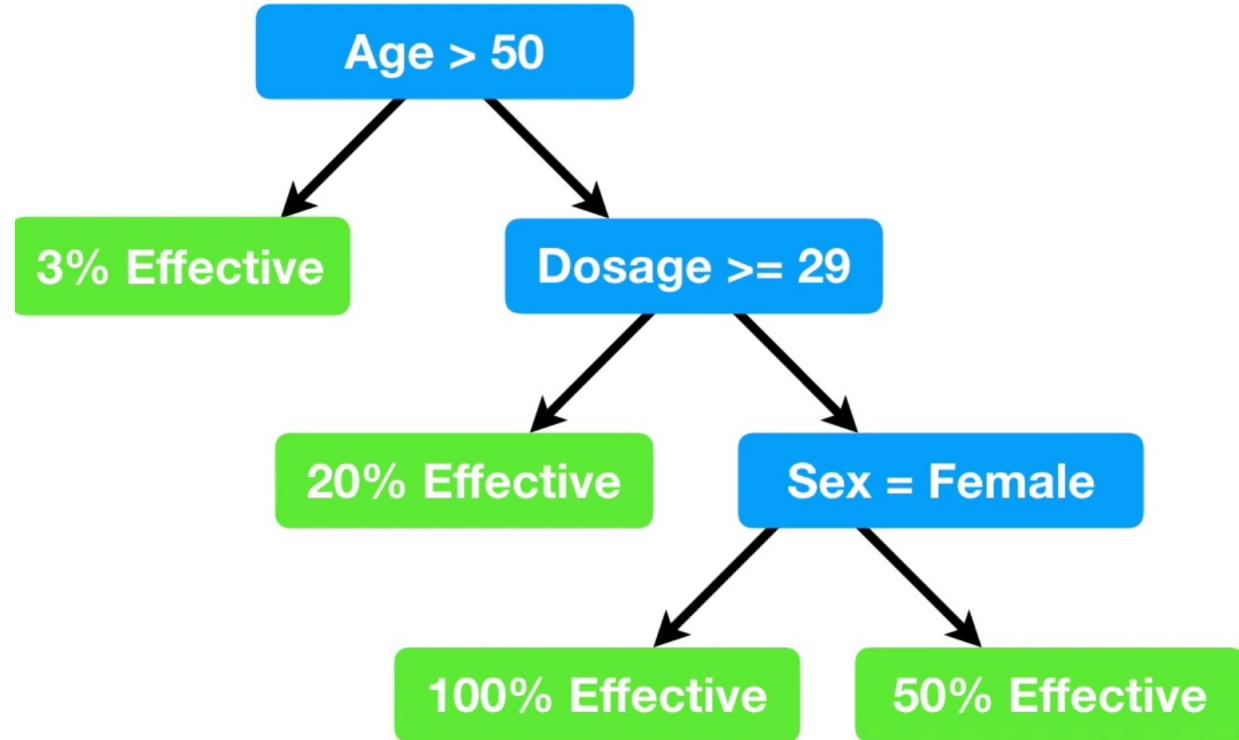


Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc...	etc...	etc...	etc...

Multiple Features



Multiple Features



Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc...	etc...	etc...	etc...

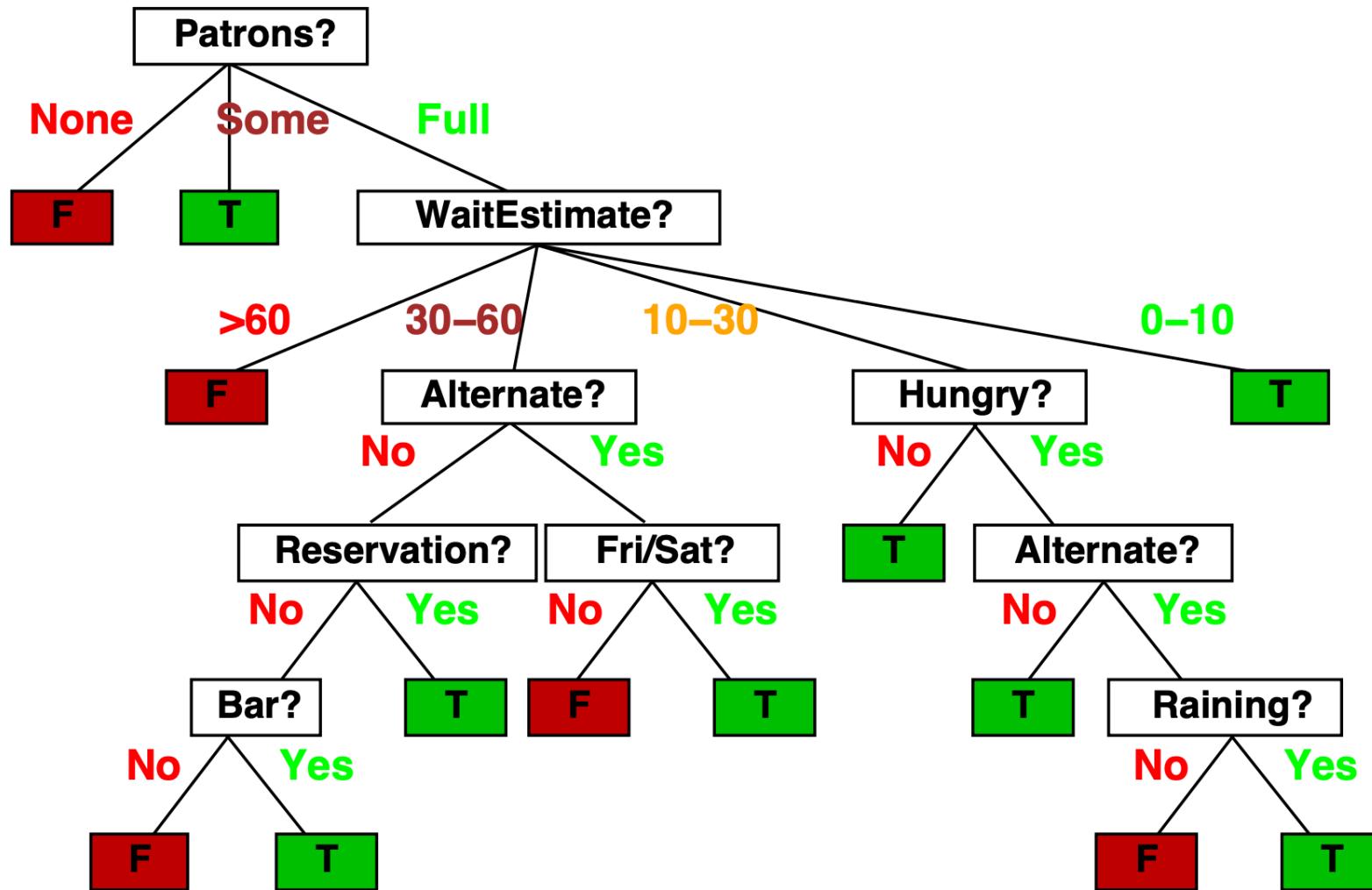
Classification Tree

Classification vs Regression

Example	Attributes										Target WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Dosage	Age	Sex	Etc.	Drug Effect.
10	25	Female	...	98
20	73	Male	...	0
35	54	Female	...	100
5	12	Male	...	44
etc...	etc...	etc...	etc...	etc...

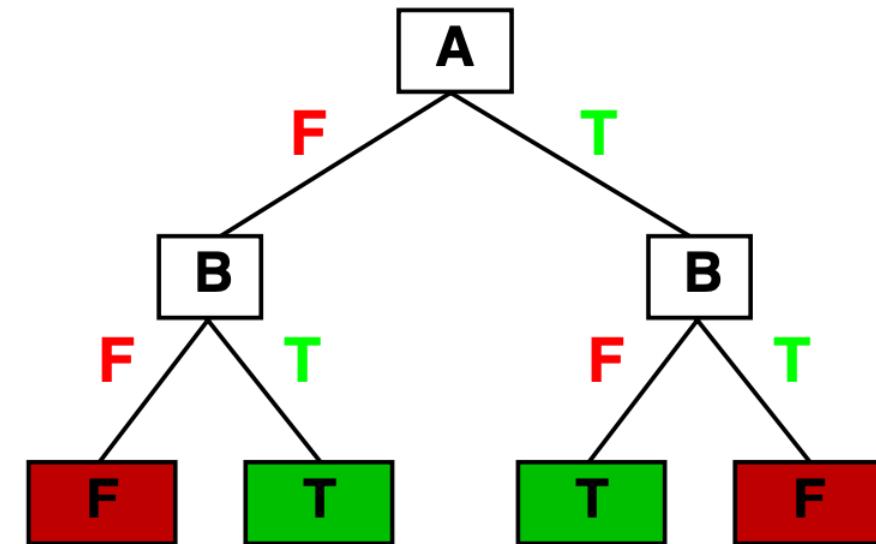
Decision Tree



Expressiveness

Decision trees can express any function of the input attributes.
E.g., for Boolean functions, truth table row → path to leaf:

A	B	$A \text{ xor } B$
F	F	F
F	T	T
T	F	T
T	T	F

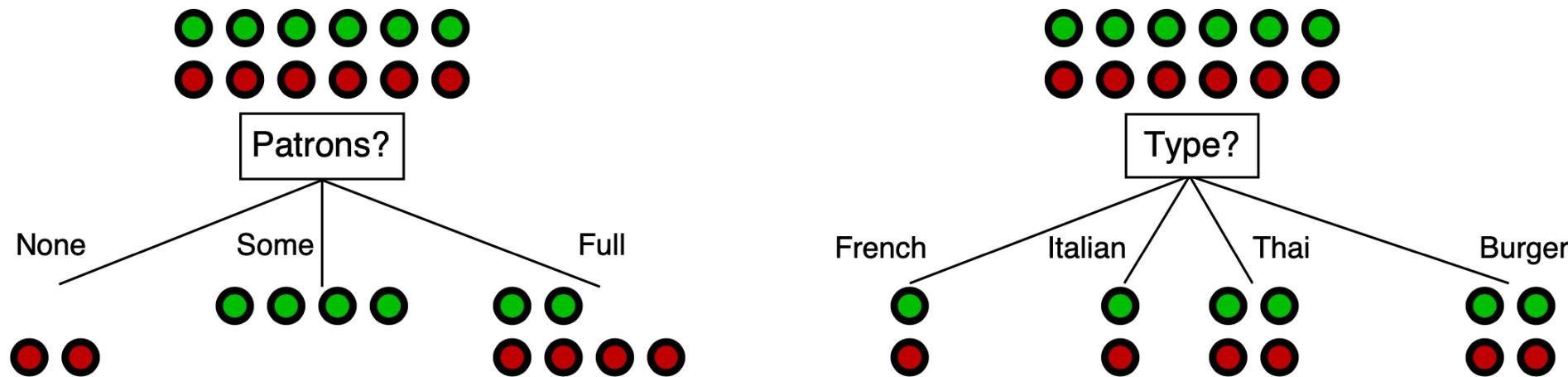


Hypothesis Space

- The number of possible decision trees with n attributes is equal to the number of possible Boolean functions with n attributes
- Given n attributes we have a truth table with 2^n rows
- Given a truth table we have 2^{2^n} possible Boolean functions

Selecting a feature

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”



Patrons? is a better choice—gives **information** about the classification

Entropy = Informativeness

Given n events we need $\log_2 n$ bits to represent them

$$\log_2 n = -\log_2 \frac{1}{n}$$

$-\log_2 \frac{1}{n}$ is the information needed to represent the events

$$-\log_2 \frac{1}{n} = -\sum_{(n \text{ events})} \frac{1}{n} \cdot \log_2 \frac{1}{n}$$

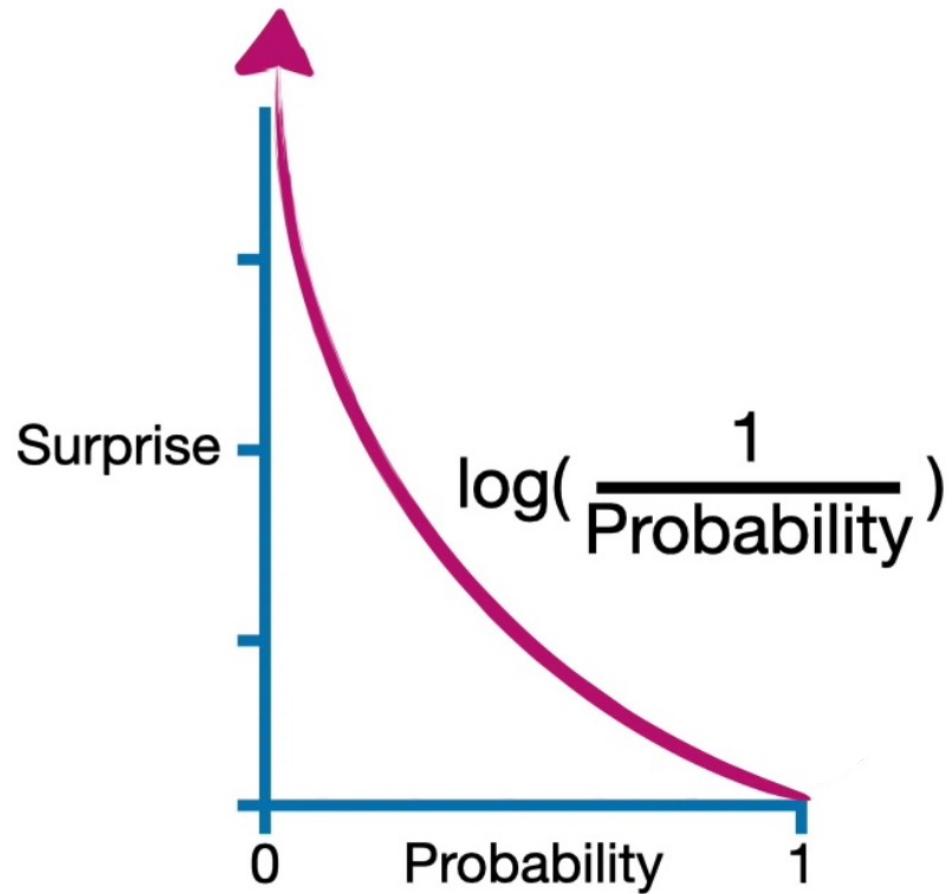
$\frac{1}{n}$ is the probability of the event in case we have a uniform distribution

Entropy = Informativeness

In the most general case, with a generic probability p_i for each event i

$$-\sum_{(n \text{ events})} \frac{1}{n} \cdot \log_2 \frac{1}{n} \curvearrowright -\sum_{i=1}^n p_i \cdot \log_2 p_i$$

Entropy and Surprise



Entropy = Expected Surprise

Information Gain (1/2)

$$H\left(\frac{t}{t+f}, \frac{f}{t+f}\right) = -\frac{t}{t+f} \cdot \log_2 \frac{t}{t+f} - \frac{f}{t+f} \cdot \log_2 \frac{f}{t+f}$$

Example	Attributes										Target <i>WillWait</i>
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	
X_1	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	\$\$\$	<i>F</i>	<i>T</i>	<i>French</i>	<i>0–10</i>	<i>T</i>
X_2	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	\$	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30–60</i>	<i>F</i>
X_3	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	\$	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0–10</i>	<i>T</i>
X_4	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	\$	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10–30</i>	<i>T</i>
X_5	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	\$\$\$	<i>F</i>	<i>T</i>	<i>French</i>	<i>>60</i>	<i>F</i>
X_6	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	\$\$	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0–10</i>	<i>T</i>
X_7	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	\$	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0–10</i>	<i>F</i>
X_8	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	\$\$	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0–10</i>	<i>T</i>
X_9	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	\$	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>>60</i>	<i>F</i>
X_{10}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	\$\$\$	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10–30</i>	<i>F</i>
X_{11}	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	\$	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0–10</i>	<i>F</i>
X_{12}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	\$	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30–60</i>	<i>T</i>

Information Gain (2/2)

$$R(Some) = \frac{t_i + f_i}{t + f} \cdot I\left(\frac{t_i}{t_i + f_i}, \frac{f_i}{t_i + f_i}\right)$$

$$R(Full) = \frac{t_i + f_i}{t + f} \cdot I\left(\frac{t_i}{t_i + f_i}, \frac{f_i}{t_i + f_i}\right)$$

$$R(None) = \frac{t_i + f_i}{t + f} \cdot I\left(\frac{t_i}{t_i + f_i}, \frac{f_i}{t_i + f_i}\right)$$

$$R(A) = \sum_{i=1}^v \frac{t_i + f_i}{t + f} \cdot I\left(\frac{t_i}{t_i + f_i}, \frac{f_i}{t_i + f_i}\right)$$

$$IG(A) = H\left(\frac{t}{t + f}, \frac{f}{t + f}\right) - R(A)$$

Example	Attributes											Target WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est		
X ₁	T	F	F	T	Some	\$\$\$	F	T	French	0–10		T
X ₂	T	F	F	T	Full	\$	F	F	Thai	30–60		F
X ₃	F	T	F	F	Some	\$	F	F	Burger	0–10		T
X ₄	T	F	T	T	Full	\$	F	F	Thai	10–30		T
X ₅	T	F	T	F	Full	\$\$\$	F	T	French	>60		F
X ₆	F	T	F	T	Some	\$\$	T	T	Italian	0–10		T
X ₇	F	T	F	F	None	\$	T	F	Burger	0–10		F
X ₈	F	F	F	T	Some	\$\$	T	T	Thai	0–10		T
X ₉	F	T	T	F	Full	\$	T	F	Burger	>60		F
X ₁₀	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30		F
X ₁₁	F	F	F	F	None	\$	F	F	Thai	0–10		F
X ₁₂	T	T	T	T	Full	\$	F	F	Burger	30–60		T

Building a Classification Tree

$$H\left(\frac{t}{t+f}, \frac{f}{t+f}\right) = -\frac{6}{12} \cdot \log_2 \frac{6}{12} - \frac{6}{12} \cdot \log_2 \frac{6}{12} = 1$$

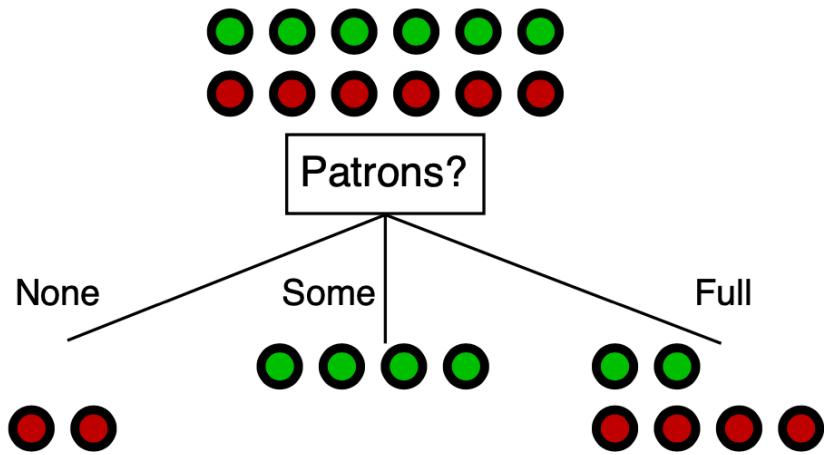
$$IG(Patrons) = 1 - \left[\frac{2}{12} \cdot H(0,1) + \frac{4}{12} \cdot H(1,0) + \frac{6}{12} \cdot H\left(\frac{2}{6}, \frac{4}{6}\right) \right] \approx 0,541$$

$$IG(Type) = 1 - \left[\frac{2}{12} \cdot H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} \cdot H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} \cdot H\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} \cdot H\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0$$

Select the attribute with the highest gain

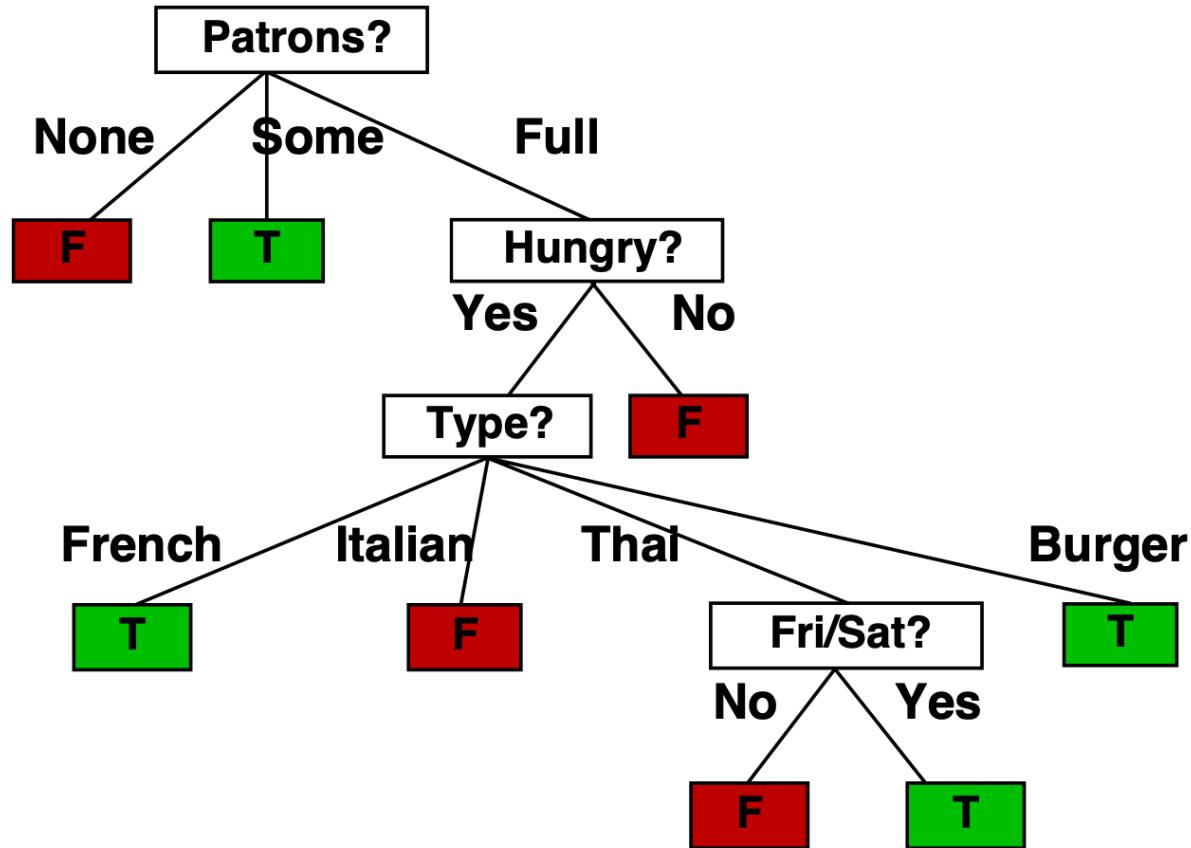
Example	Attributes											Target <i>WillWait</i>
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>		
<i>X</i> ₁	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	\$\$\$	<i>F</i>	<i>T</i>	<i>French</i>	<i>0-10</i>	<i>T</i>	
<i>X</i> ₂	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	\$	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30-60</i>	<i>F</i>	
<i>X</i> ₃	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	\$	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>T</i>	
<i>X</i> ₄	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	\$	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10-30</i>	<i>T</i>	
<i>X</i> ₅	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	\$\$\$	<i>F</i>	<i>T</i>	<i>French</i>	<i>>60</i>	<i>F</i>	
<i>X</i> ₆	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	\$\$	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0-10</i>	<i>T</i>	
<i>X</i> ₇	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	\$	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>F</i>	
<i>X</i> ₈	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	\$\$	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0-10</i>	<i>T</i>	
<i>X</i> ₉	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	\$	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>>60</i>	<i>F</i>	
<i>X</i> ₁₀	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	\$\$\$	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10-30</i>	<i>F</i>	
<i>X</i> ₁₁	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	\$	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0-10</i>	<i>F</i>	
<i>X</i> ₁₂	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	\$	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30-60</i>	<i>T</i>	

Building a Classification Tree



Example	Attributes										Target WillWait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0-10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30-60	T

Building a Classification Tree



Classification and Regression Trees... a bit more

Categorical and Numerical values
in Classification trees

Categorical and Numerical values in Classification trees (1/4)

Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Sort rows



Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Categorical and Numerical values in Classification trees (2/4)

Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Compute average for all adjacent rows



Age	Loves Cool As Ice
7	No
9.5	No
12	No
15	Yes
18	Yes
26.5	Yes
35	Yes
36.5	Yes
38	Yes
44	No
50	No
66.5	No
83	No

Categorical and Numerical values in Classification trees (3/4)

Age	Loves Cool As Ice
7	No
12	No
18	Yes
35	Yes
38	Yes
50	No
83	No

Compute the Information Gain for all possible binary options.

Consider the value with the highest gain as representative of the feature

$\text{Age} < 9.5$

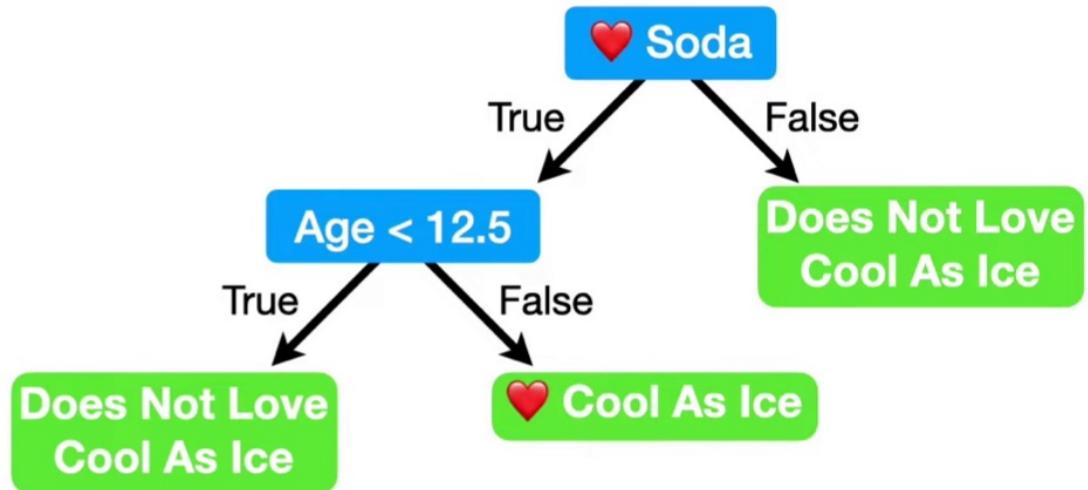
$\text{Age} < 15$

Categorical and Numerical values in Classification trees (4/4)

Loves Popcorn	Loves Soda	Age	Loves Cool As Ice
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Compare the Information Gain for all the features and select the one with the highest value.

Continue until you build the whole tree.



Missing values

Missing categorical values

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	???	Yes
etc...	etc...	etc...	etc...

Missing categorical values

Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
Yes	Yes	No	No
Yes	No	YES	Yes
etc...	etc...	etc...	etc...

Add the most frequent value

Missing categorical values

Find a correlated feature and use it as guideline

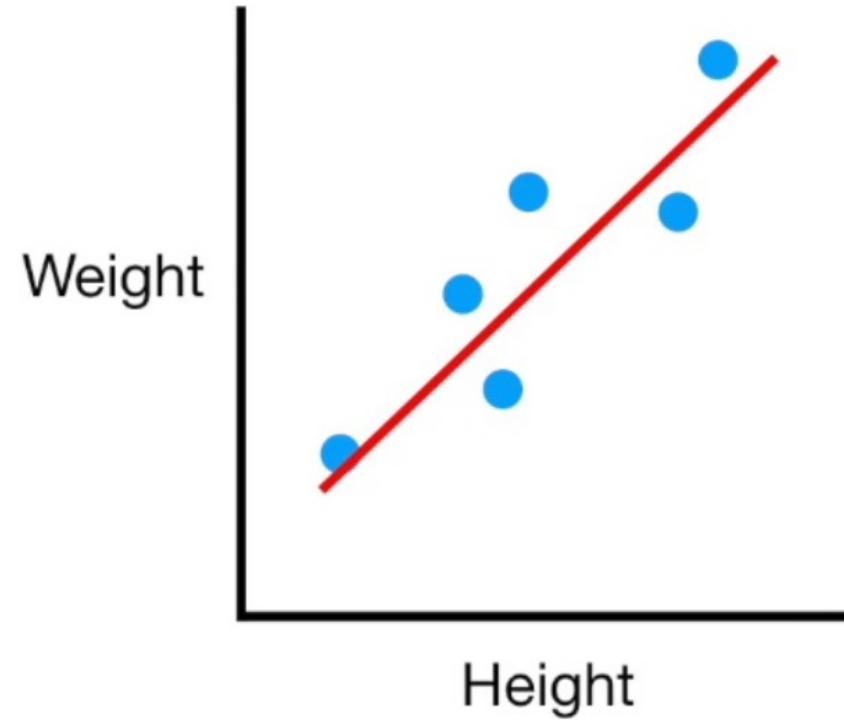
Chest Pain	Good Blood Circulation	Blocked Arteries	Heart Disease
No	No	No	No
Yes	Yes	Yes	Yes
No	Yes	No	No
Yes	No	???	Yes
etc...	etc...	etc...	etc...

Missing continuous values

Height	Good Blood Circulation	Weight	Heart Disease
5'7"	No	155	No
6'	Yes	180	Yes
5'4"	Yes	120	No
5'8"	No	???	Yes
etc...	etc...	etc...	etc...

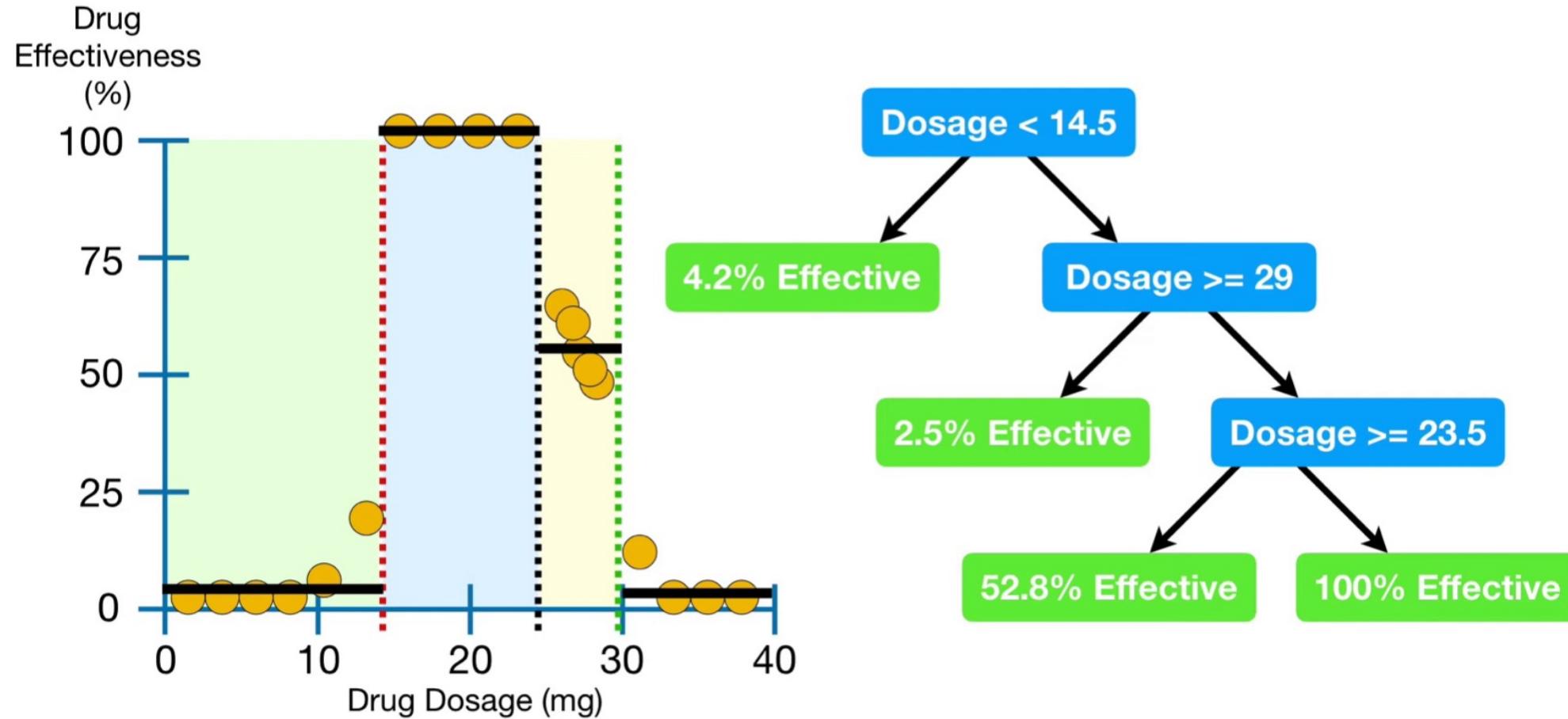
Missing continuous values

Height	Good Blood Circulation	Weight	Heart Disease
5'7"	No	155	No
6'	Yes	180	Yes
5'4"	Yes	120	No
5'8"	No	???	Yes
etc...	etc...	etc...	etc...

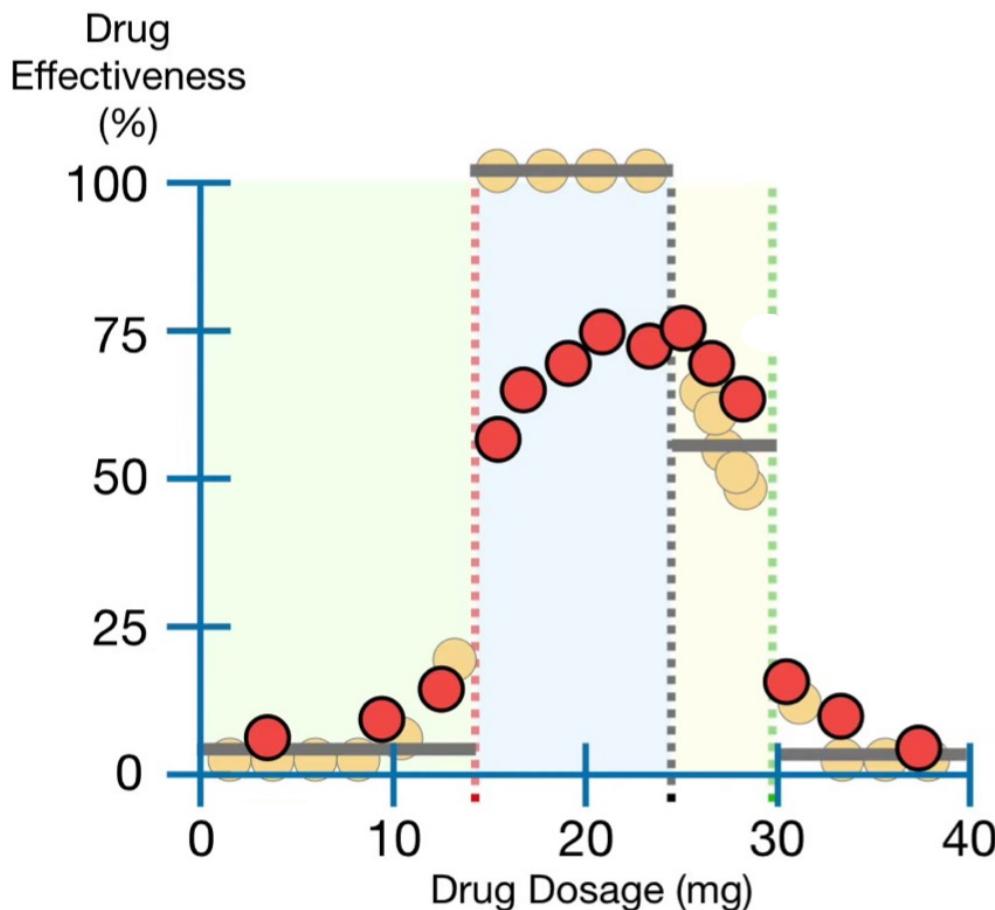


Pruning Regression Trees

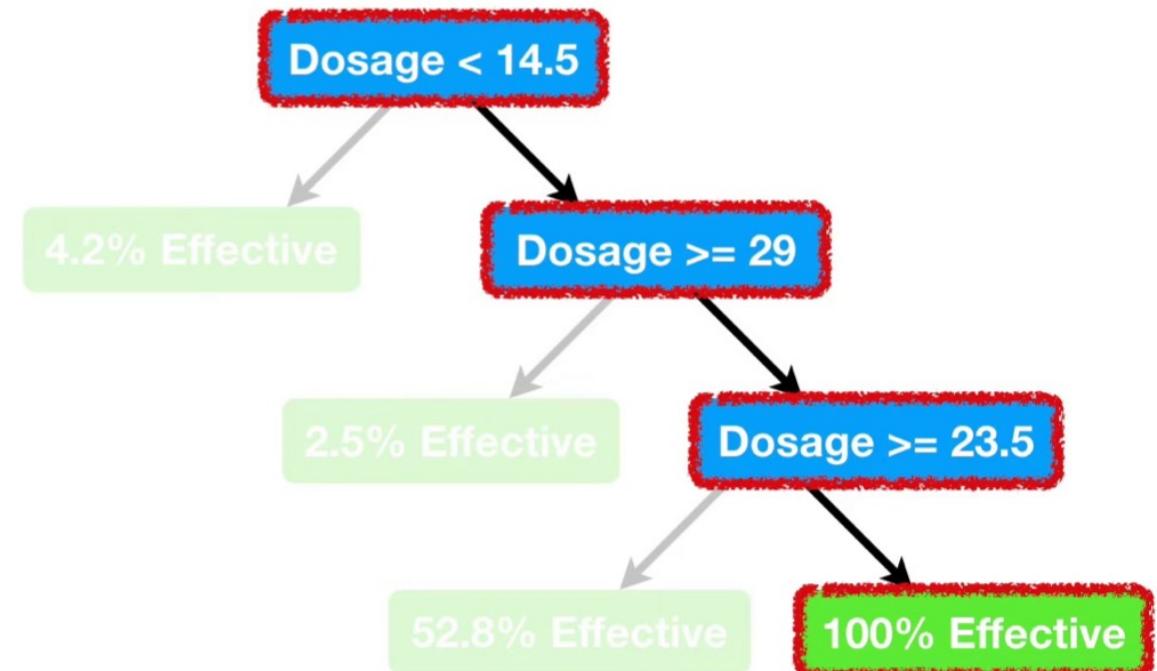
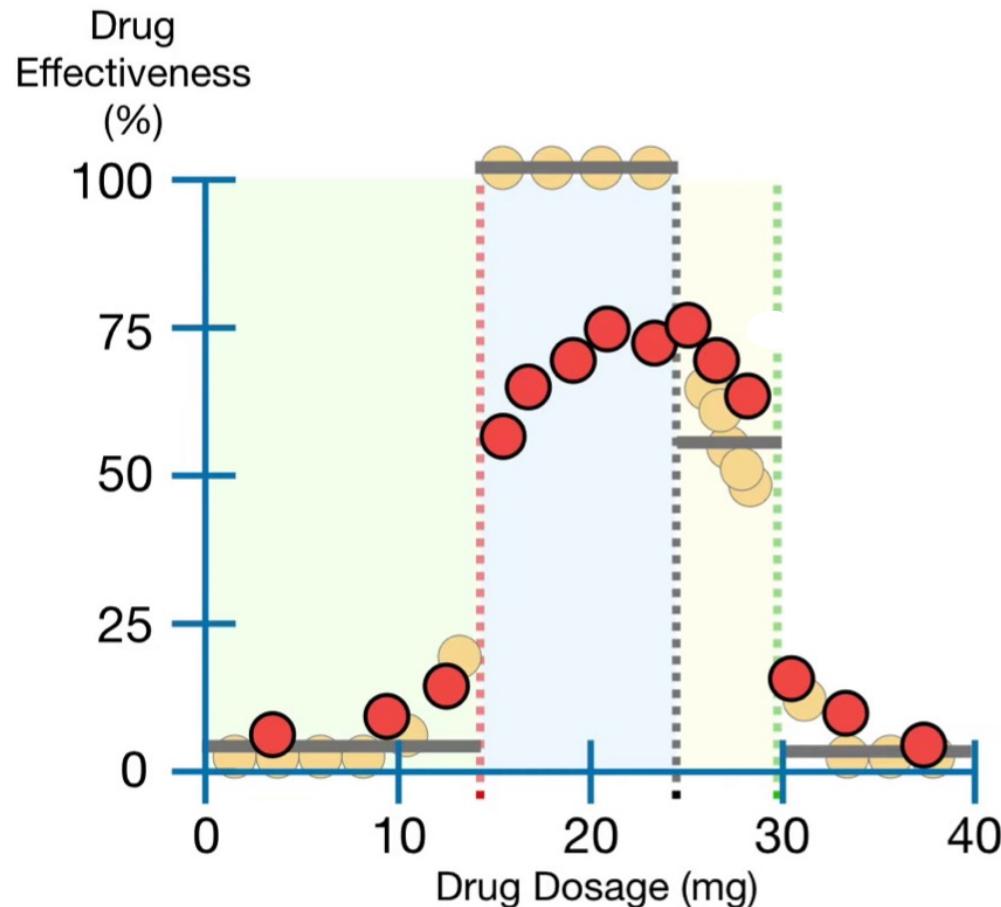
Regression Tree



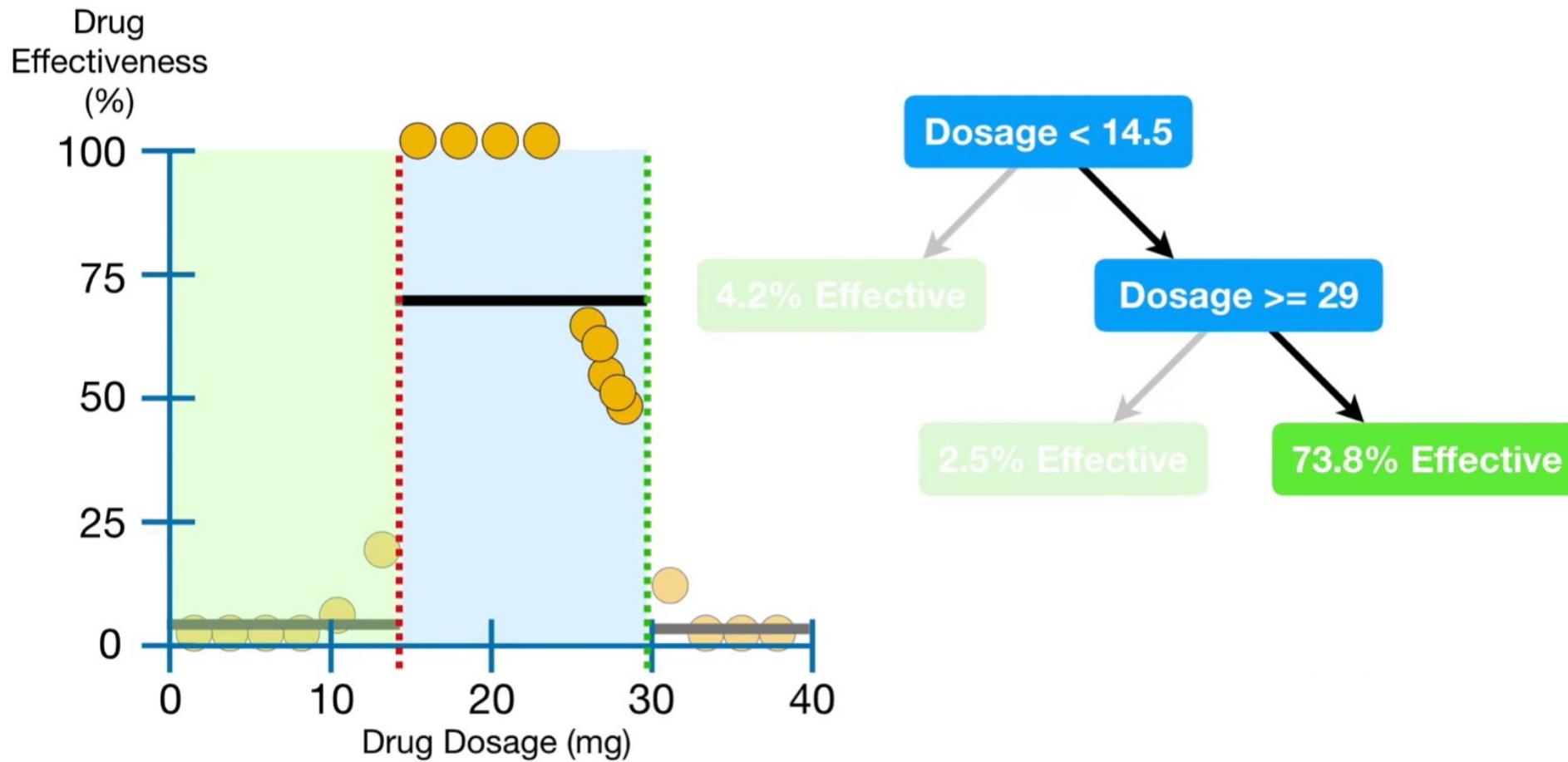
Residual Sum of Squares on the Test Set



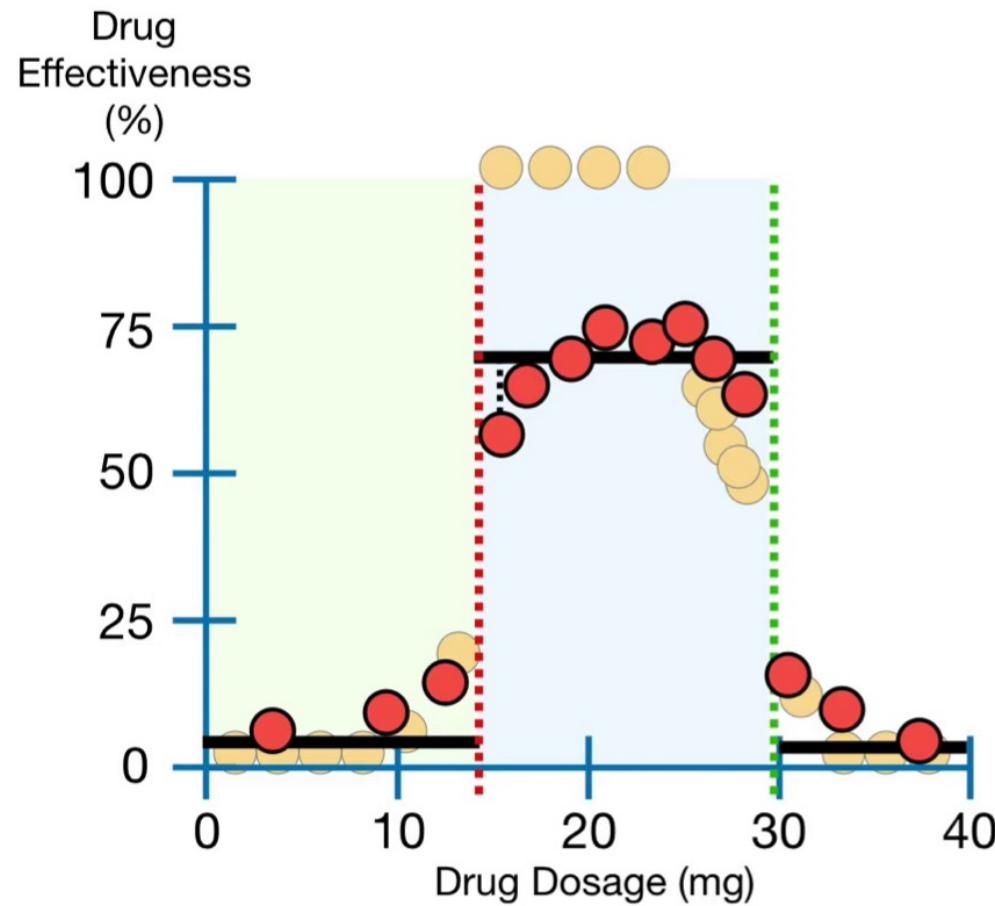
Residual Sum of Squares on the Test Set



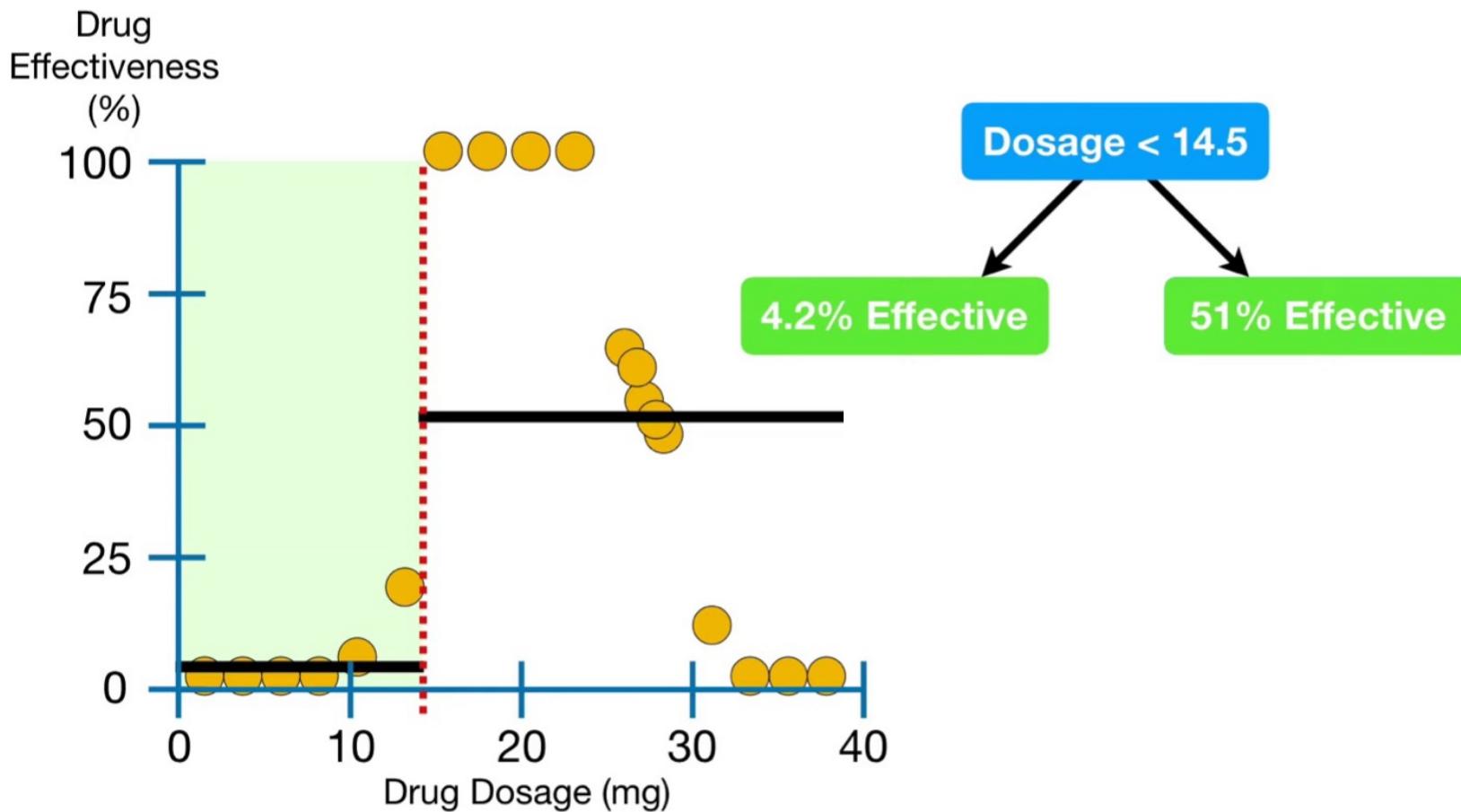
Reduce the precision of the tree



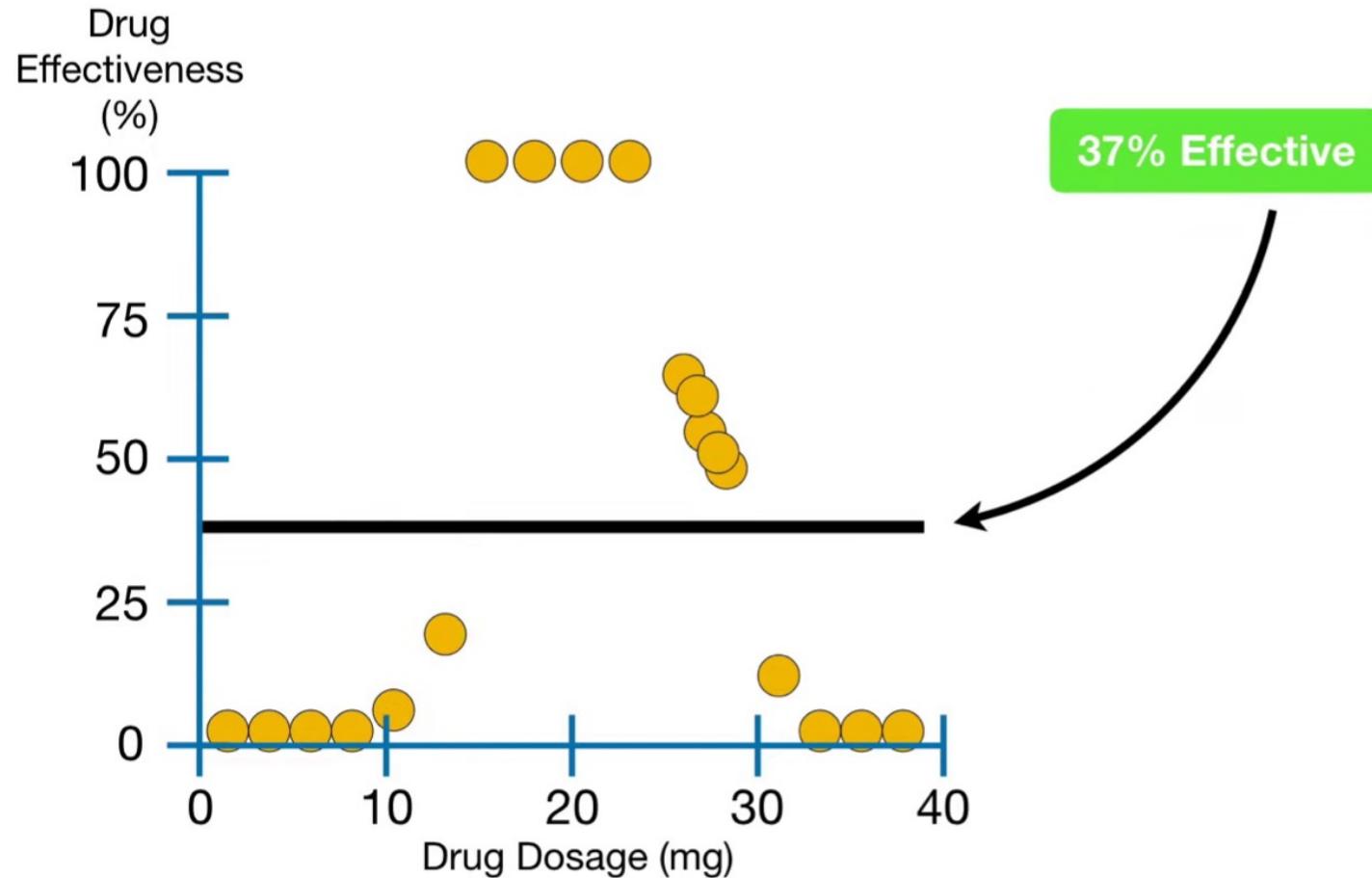
Reduce overfitting



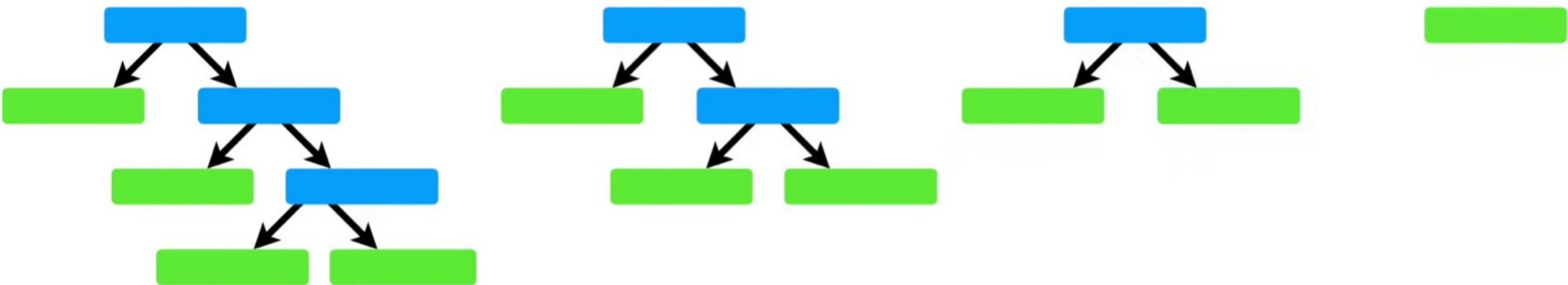
Further pruning



Further pruning

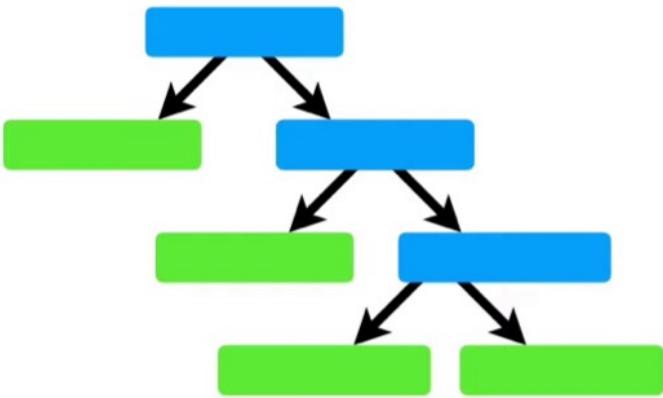


What is the best tree?

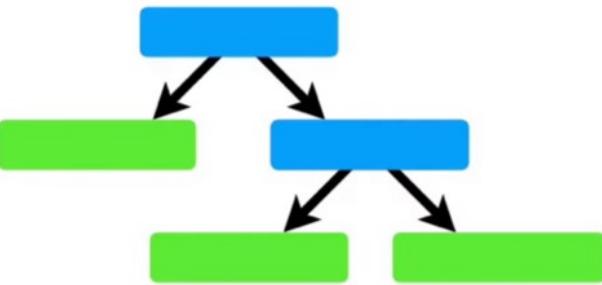


Compute the RSS for each tree

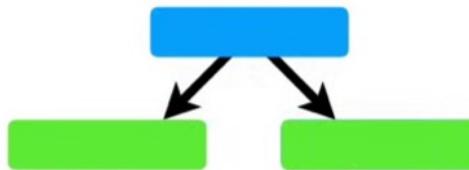
RSS = 543.8



RSS = 5494.8



RSS = 19563.7

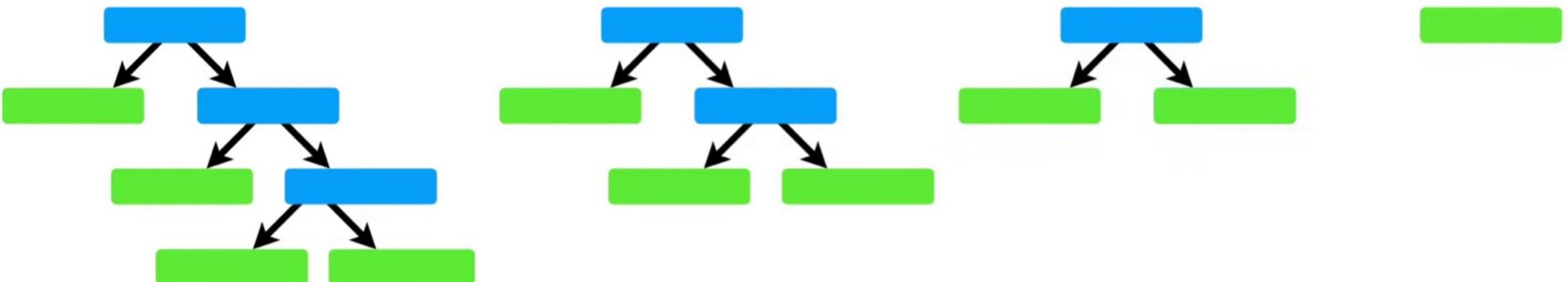


RSS = 28897.2



Weakest Link Pruning

Tree Score = RSS + Tree Complexity Penalty
= RSS + $\alpha \cdot T$

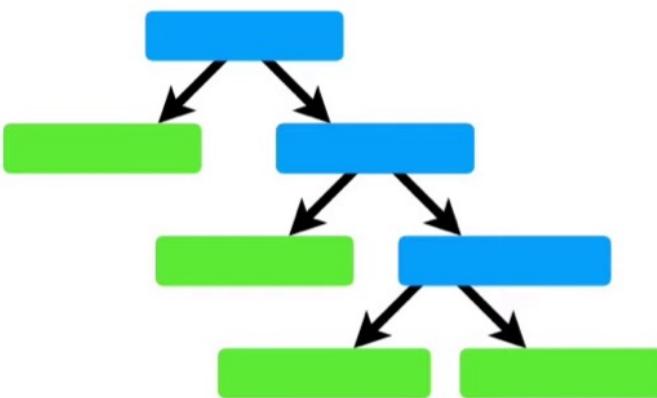


Weakest Link Pruning

$$\text{Tree Score} = \text{RSS} + \alpha \cdot T$$

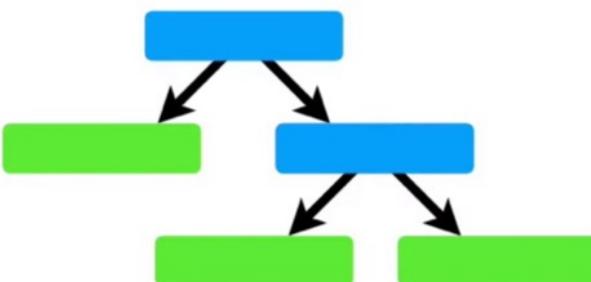
Example with $\alpha = 10.000$

RSS = 543.8



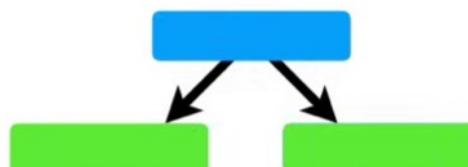
Tree Score = 40,543.8

RSS = 5494.8



Tree Score = 35,494.8

RSS = 19563.7



Tree Score = 39,243.7

RSS = 28897.2

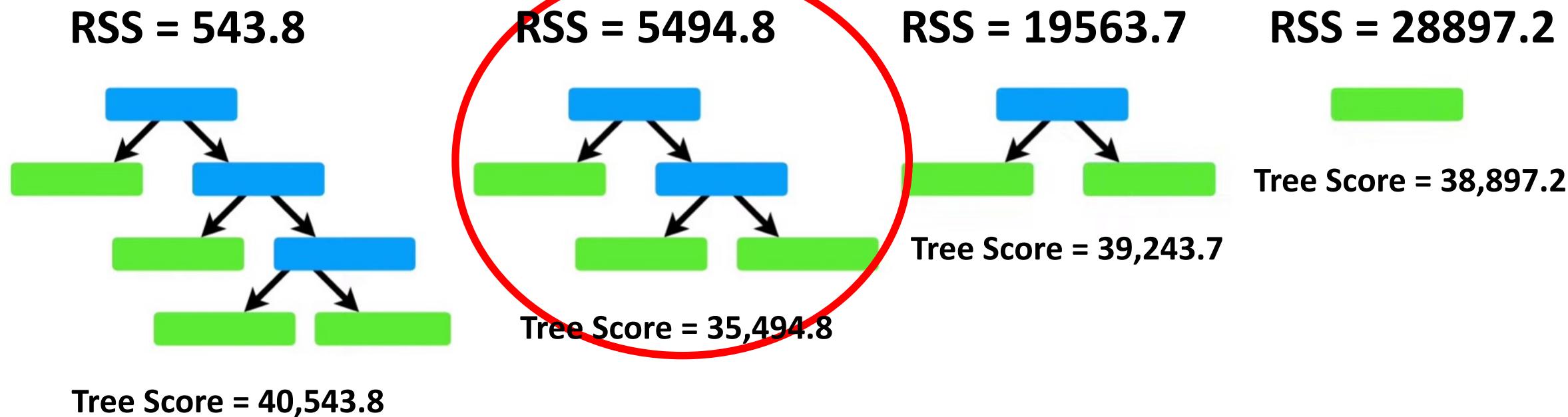


Tree Score = 39,243.7

Weakest Link Pruning

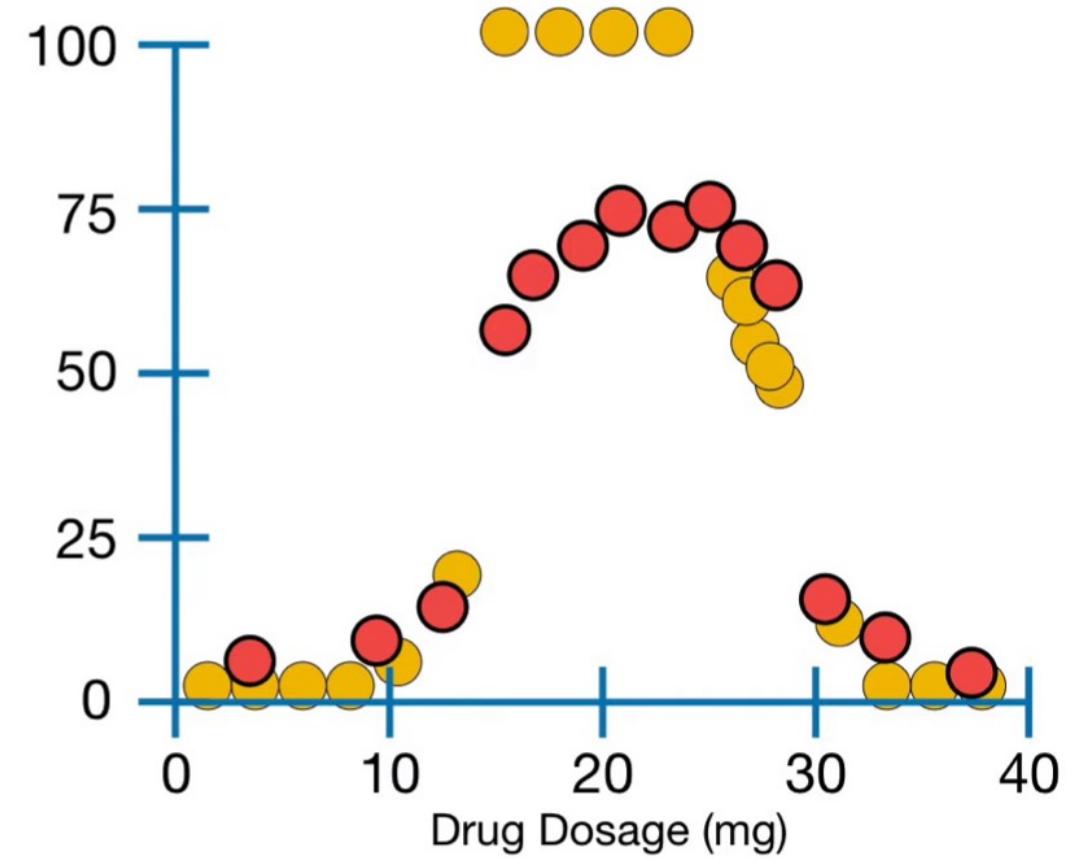
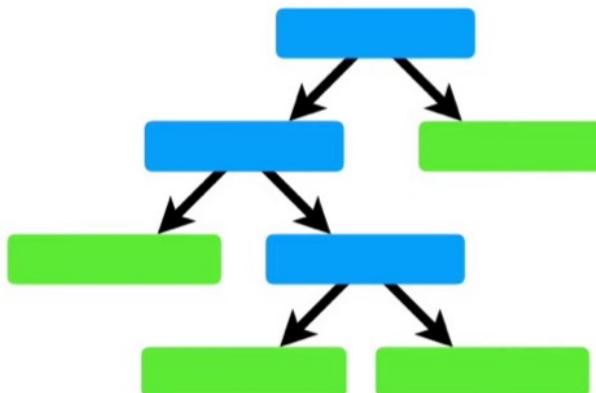
$$\text{Tree Score} = \text{SSR} + \alpha \cdot T$$

Example with $\alpha = 10.000$



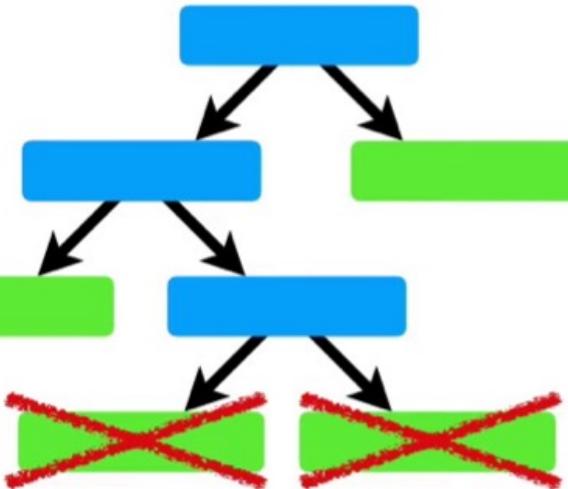
How to evaluate the best α

Build a tree considering all the data

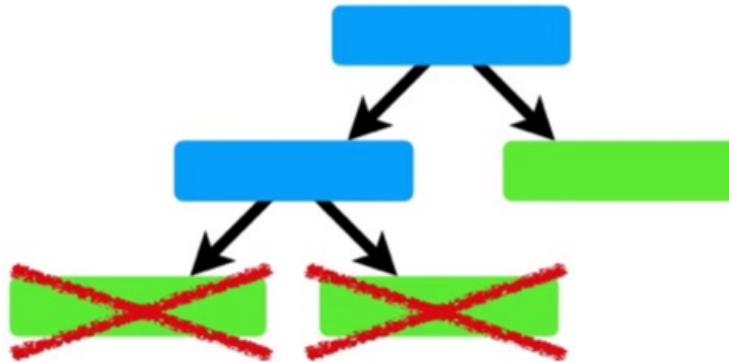


How to evaluate the best α

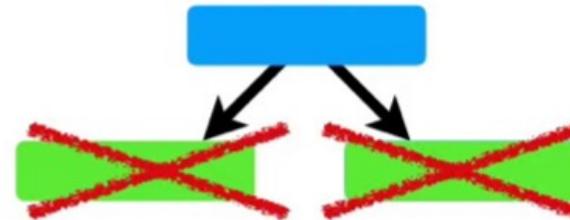
$\alpha = 0$



$\alpha = 10,000$



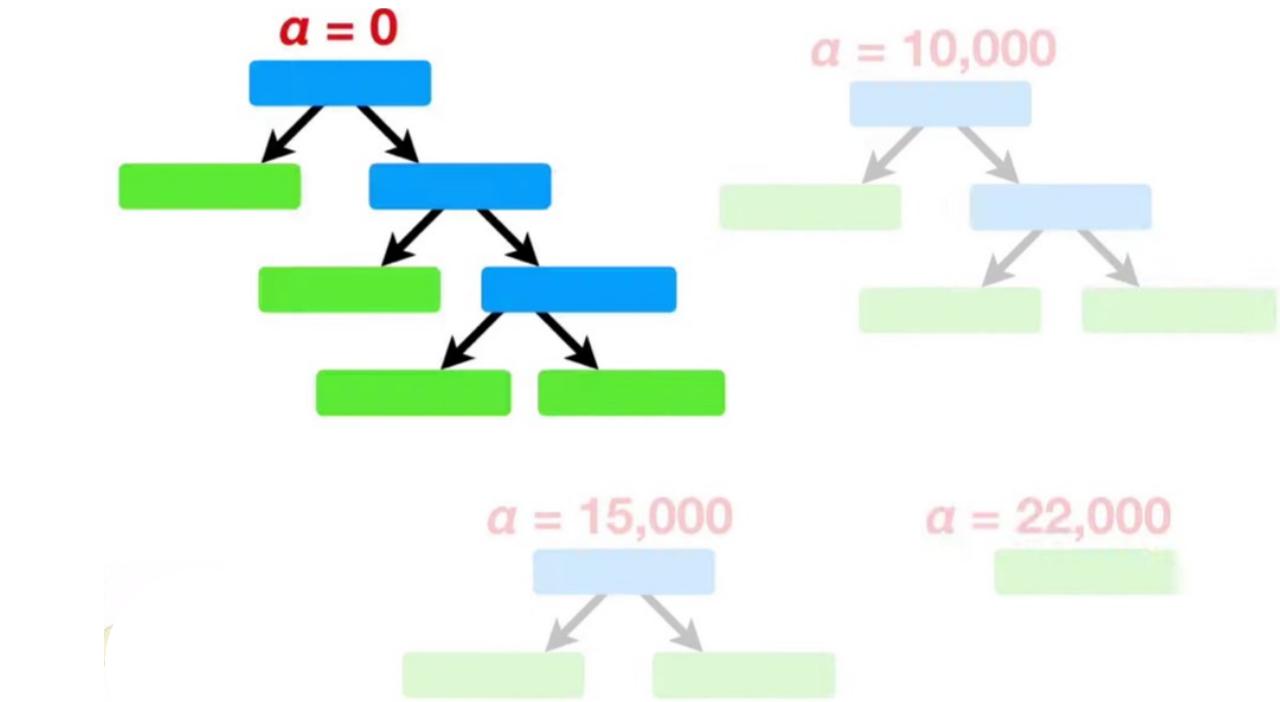
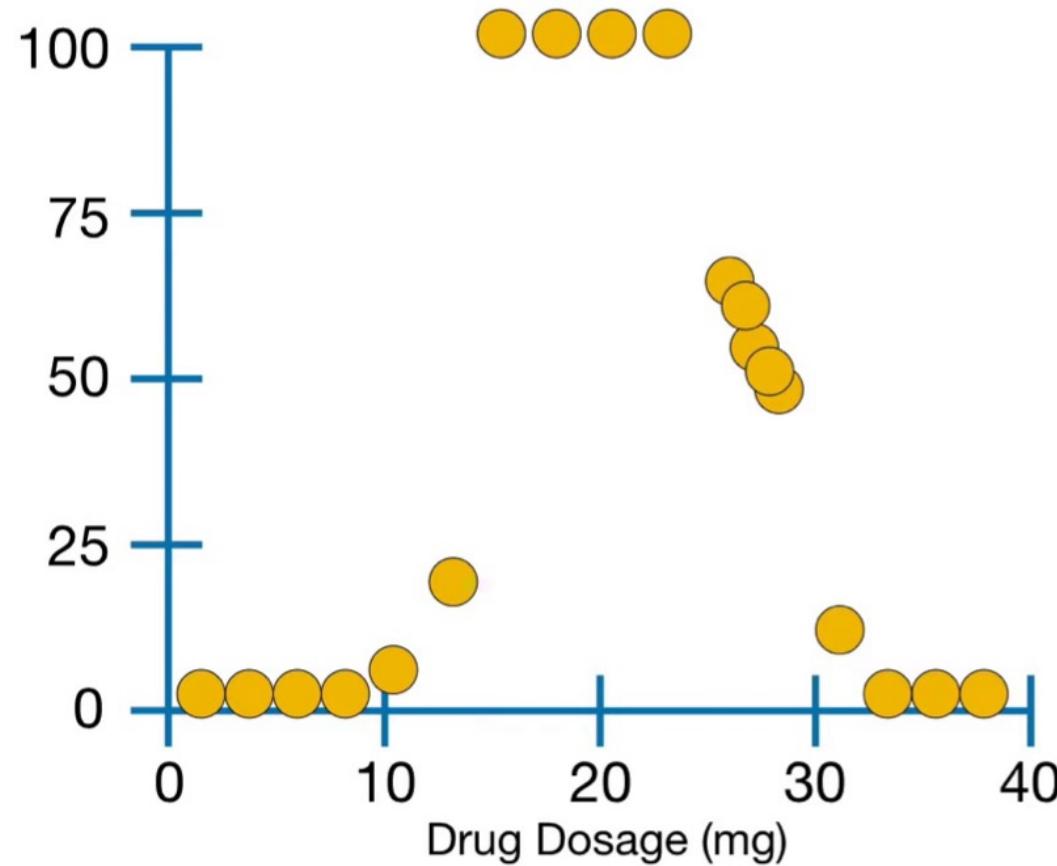
$\alpha = 15,000$



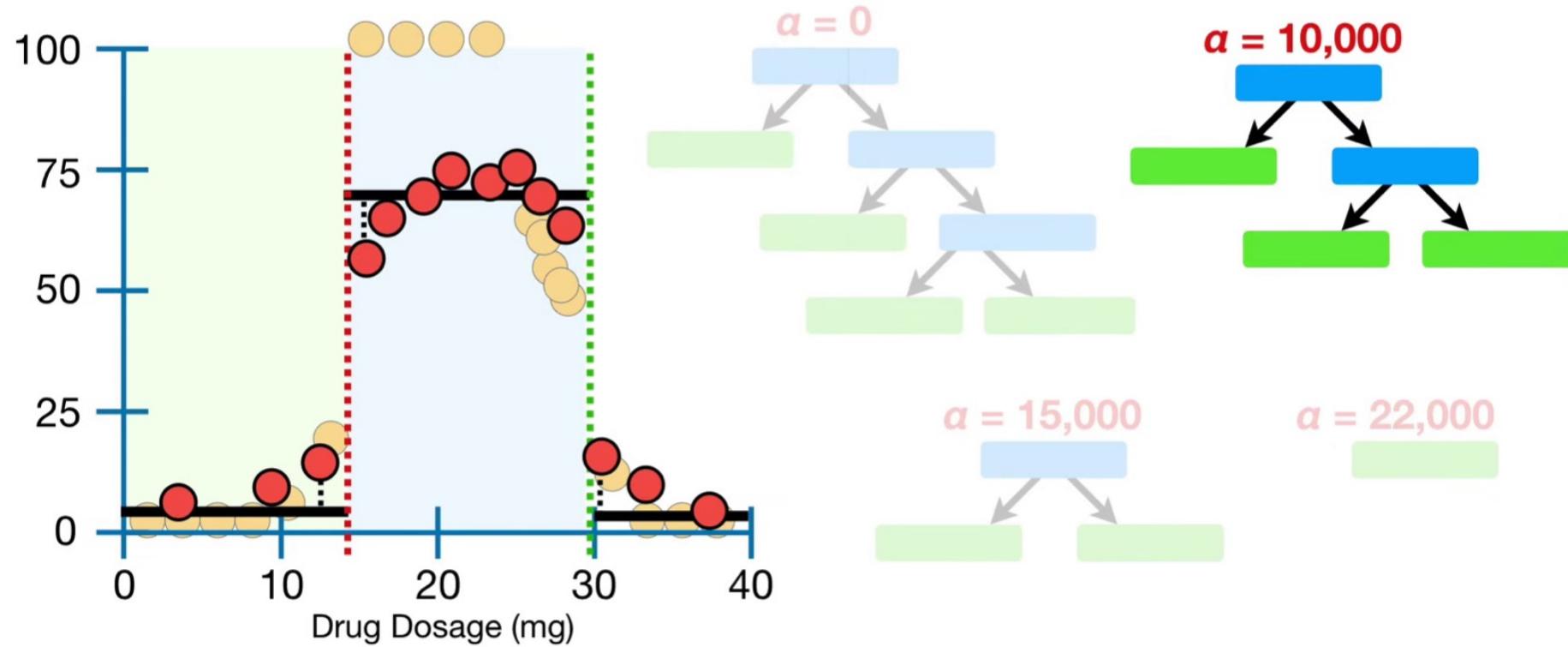
$\alpha = 22,000$



Train the tree again using the Training set only

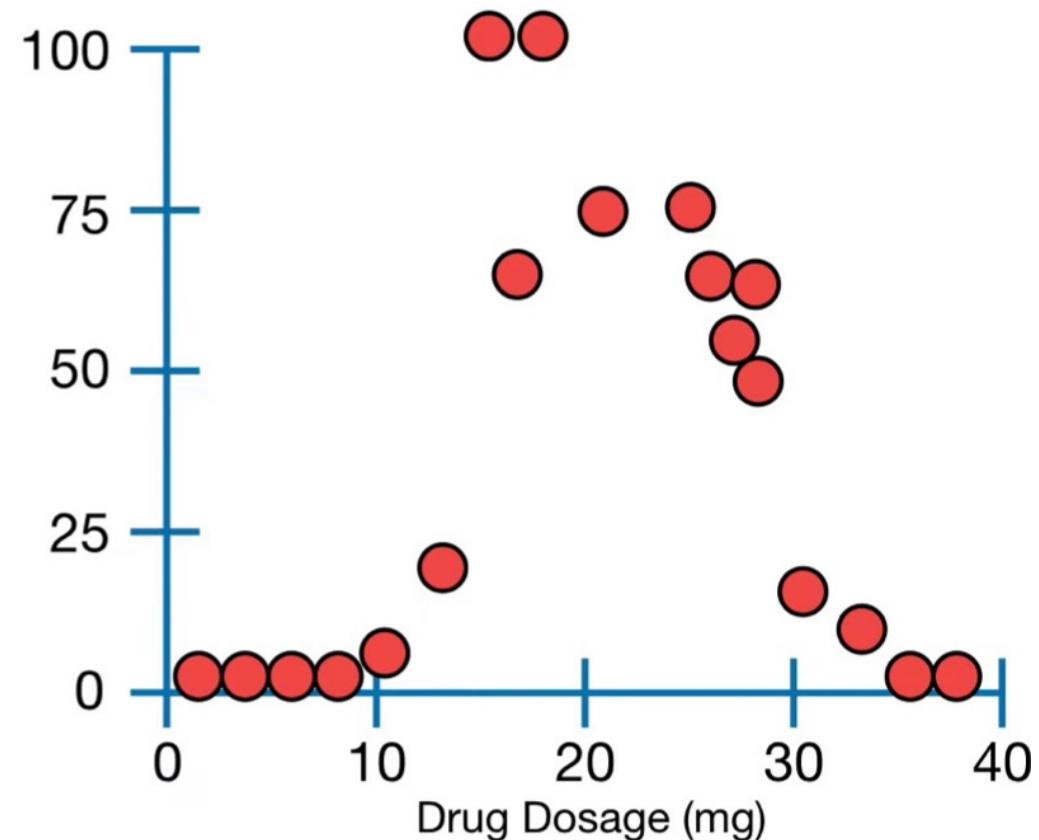
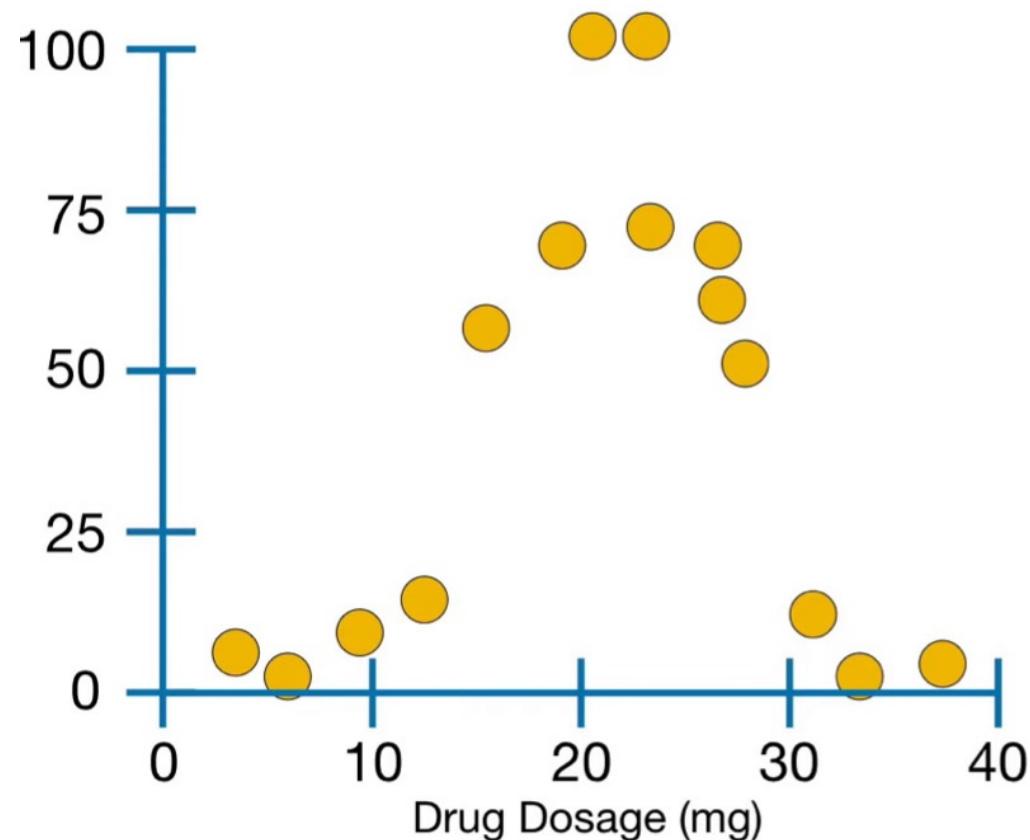


Compute RSS on the Test set for all the trees

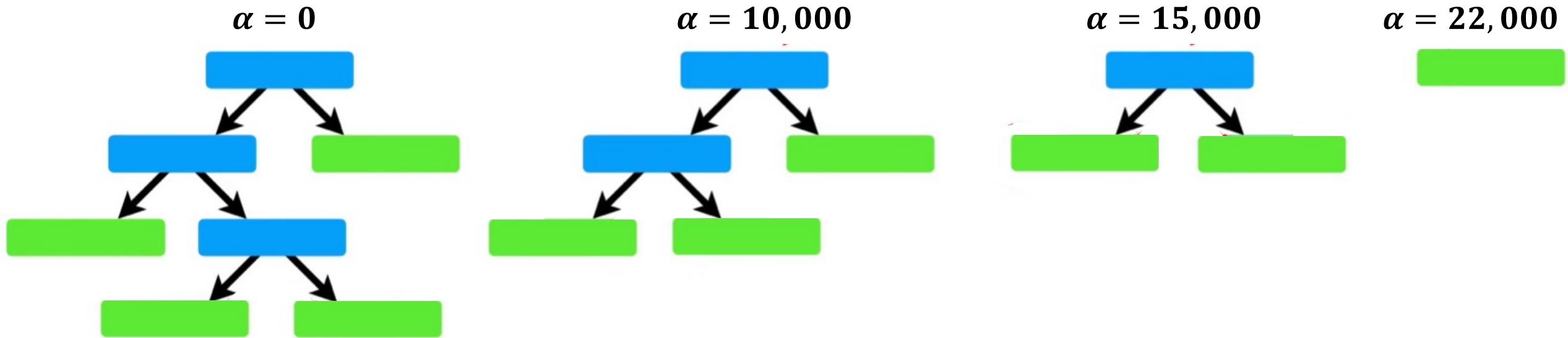


Vote for the one with the lowest RSS

Repeat the process with new Training and Test sets → k-fold Cross-Validation



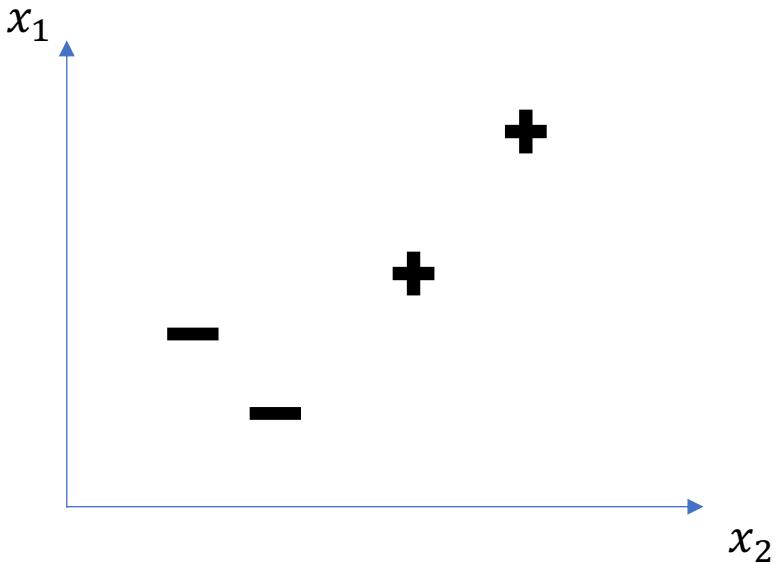
Go back to the trees built on the full dataset
and select the one with the most voted α



Support Vector Machines

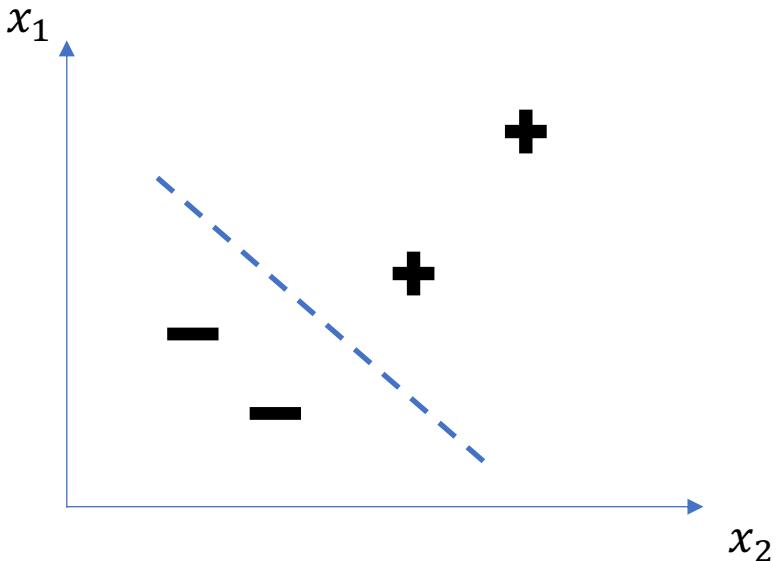
Large Margin Classifier

SVM Intuition



Let x_1 , and x_2 be two features in a data space. Let + and - be samples belonging respectively to positive and negative classes.

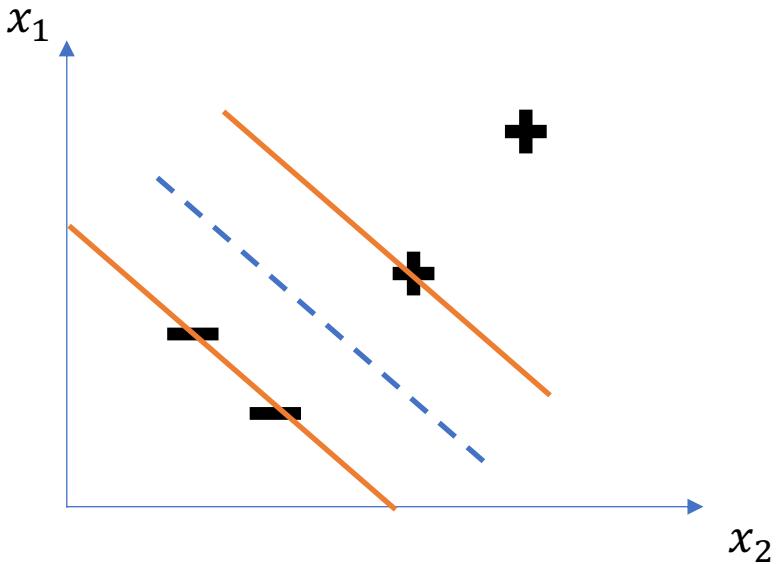
SVM Intuition



Let $\vec{\omega} \cdot \vec{x} + b$ be a separating hyperplane that we want to be «optimal».

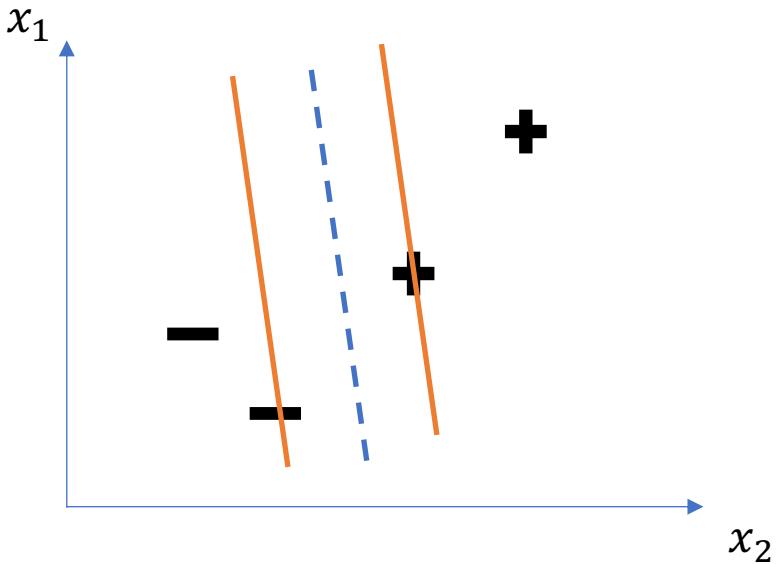
Observation: This is equivalent to $\theta_0 + \theta_1 x_1 + \theta_2 x_2$ we already know

SVM Intuition



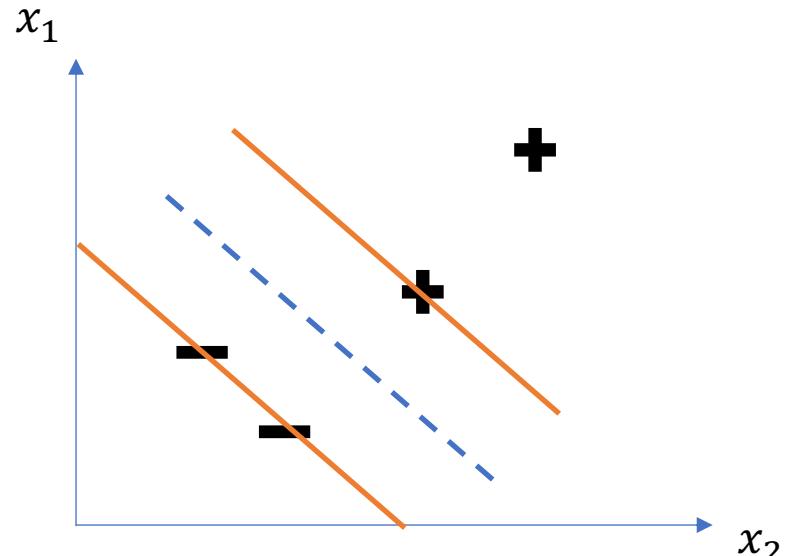
We assume that «optimal» means that we could draw a «street» around the hyperplane **as wide as possible**.

SVM Intuition



Please note that there are infinite feasible hyperplanes. However, we are looking for the only one with the **maximum width**.

Decision rule

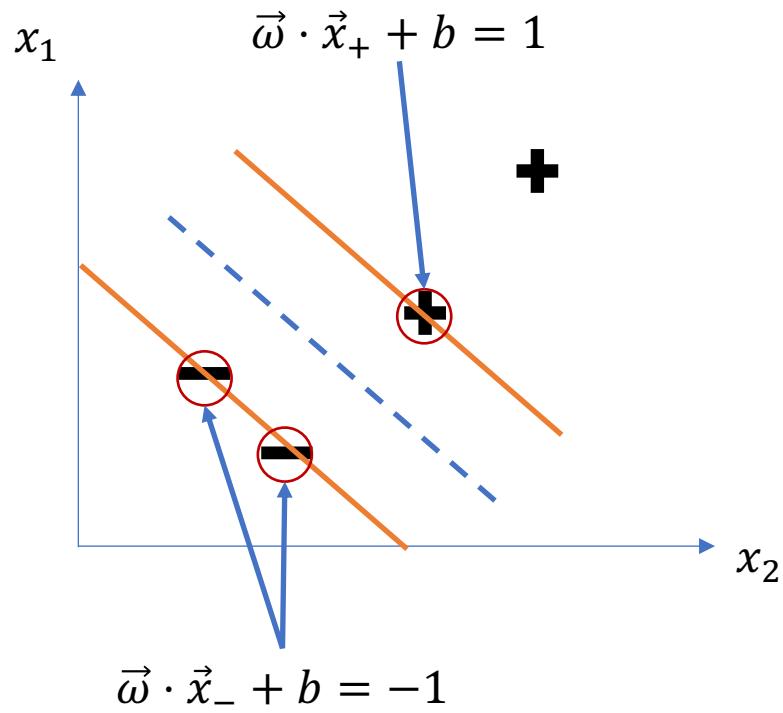


IF $\vec{\omega} \cdot \vec{u} + b \geq 0$ THEN +
DECISION RULE

1

If we set $\vec{\omega} \cdot \vec{x} + b = 0$ we will obtain all the points on the hyperplane. However, if for a certain sample x , $\vec{\omega} \cdot \vec{x} + b > 0$, x will be on the «right side» of the hyperplane, together with positive samples.

Decision rule (cont.d)



$$\vec{\omega} \cdot \vec{x}_+ + b \geq 1 \quad y_i \cdot (\vec{\omega} \cdot \vec{x}_+ + b) \geq 1$$

$$\vec{\omega} \cdot \vec{x}_- + b \leq -1 \quad y_i \cdot (\vec{\omega} \cdot \vec{x}_- + b) \leq 1$$

y_i such that $y_i = +1$ for + samples
 $y_i = -1$ for - samples

$$y_i \cdot (\vec{\omega} \cdot \vec{x}_i + b) - 1 \geq 0$$

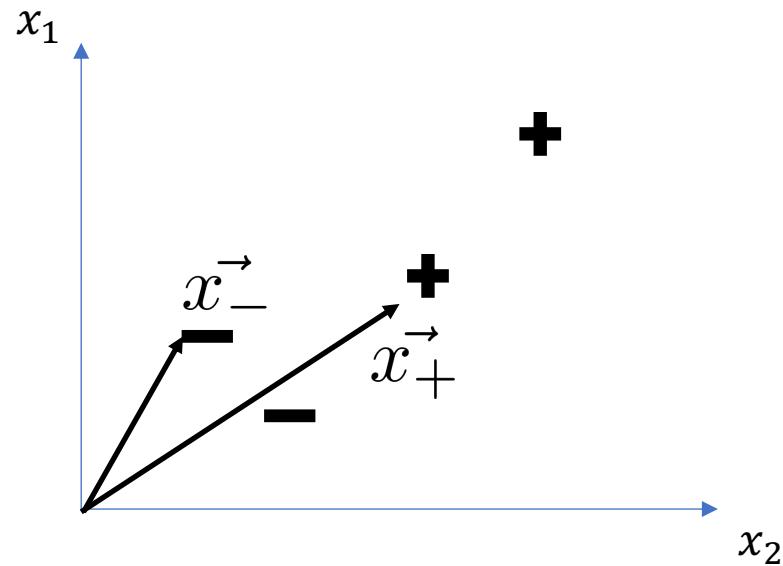
$y_i \cdot (\vec{\omega} \cdot \vec{x}_i + b) = 1$
for x_i on the borders

2

Problem definition

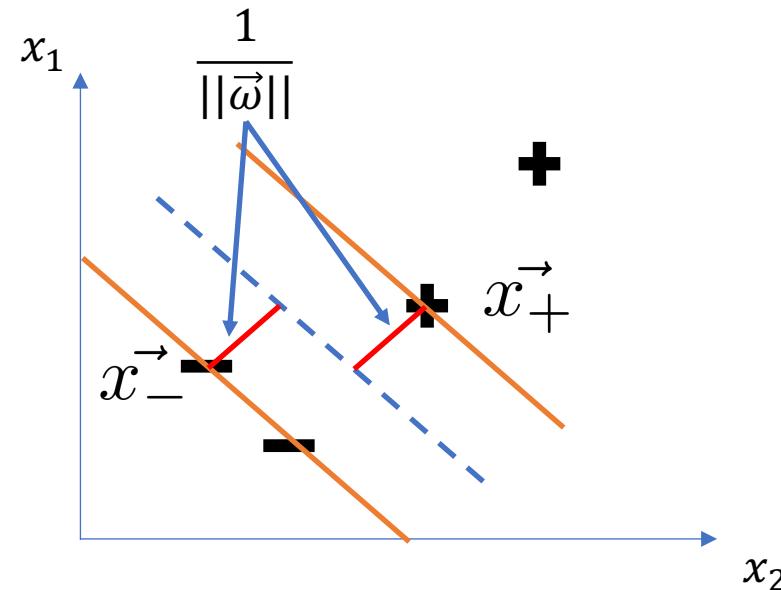
In a real scenario, the optimal hyperplane is unknown.

We want to figure out how to set the line such as the «street» between the positive and negative examples is as wide as possible



Problem definition (cont.d)

We don't know the width of the street but we can compute the difference



$$y_i(\vec{\omega} \cdot \vec{x}_i + b) = 1$$

y_i such that $y_i = +1$ for + samples

$y_i = -1$ for - samples

We know that the distance of a point x_i from a hyperplane is

$$\frac{|(\vec{\omega} \cdot \vec{x}_i) + b|}{\|\vec{\omega}\|}$$

$$\text{width} = \frac{2}{\|\vec{\omega}\|}$$

Problem definition (cont.d)

What we want is to maximize the width of the «street».

In order to do this we can perform a couple of mathematically convenient steps.

$$\max \frac{2}{\|\omega\|}$$

$$\max \frac{1}{\|\omega\|}$$

$$\min \|\omega\|$$

$$\min \frac{1}{2} \|\omega\|^2$$

3

Problem definition (cont.d)

$$\text{minimize} \frac{1}{2} \|\omega\|^2$$

$$s.t. \quad \forall_i \quad y^{(i)} \cdot (\omega^T \cdot x^{(i)} + b) \geq 1$$

Constrained optimization (digression)

Lagrange duality is widely used to solve constrained optimization problems.

Let us define a generic problem with 1 equality constraint:

$$\begin{aligned} & \min f(\omega) \\ & s.t. h(\omega) = 0 \end{aligned}$$

Where $f(\omega)$, and $h(\omega)$ are convex functions. Now we introduce the Lagrangian function $L(\omega, \alpha)$:

$$L(\omega, \alpha) = f(\omega) + \alpha \cdot h(\omega)$$

Where alphas are named Langrangian multipliers.

Constrained optimization (digression)

It can be proved that for the considered problem, there exists a value ω^* which is a global minimum for $f(\omega)$ subject to the following condition:

$$\exists \alpha^* \text{ such that } \frac{\partial L}{\partial \omega}(\omega^*, \alpha^*) = 0$$

$$\frac{\partial L}{\partial \alpha}(\omega^*, \alpha^*) = 0$$

Constrained optimization (digression)

- In case we have an equality constraint $h(\omega)$ and an inequality constraint $g(\omega)$ (e.g. \geq) the corresponding Lagrangian is defined as

$$L(\omega, \alpha, \beta) = f(\omega) + \alpha \cdot g(\omega) + \beta \cdot h(\omega)$$

- Corresponding to the dual problem

$$\text{maximize } L(\omega, \alpha, \beta)$$

$$\text{s.t. } \alpha \geq 0$$

Primal form of the optimization problem

In our case, we have

$$L = \frac{1}{2} \|\vec{\omega}\|^2 - \sum \alpha_i [y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1]$$

With $\alpha_i \geq 0$

Primal problem

$$\frac{\partial L}{\partial \vec{\omega}} = \vec{\omega} - \sum \alpha_i y_i \vec{x}_i = 0$$

$$\vec{\omega} = \sum_i \alpha_i y_i \vec{x}_i$$

4a

This is an excellent result because the vector ω is a linear sum of the samples.

BUT NOT ALL the samples, because some alphas could be zero.

Primal problem (cont.d)

Now we have to derive with respect to b .

$$L = \frac{1}{2} \|\vec{\omega}\|^2 - \sum \alpha_i [y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1]$$

$$\frac{\partial L}{\partial b} = - \sum \alpha_i y_i = 0$$

$$\boxed{\sum_i \alpha_i y_i = 0}$$

4b

Primal problem (cont.d)

If we derive with respect to alpha.

$$L = \frac{1}{2} \|\vec{\omega}\|^2 - \sum \alpha_i [y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1]$$
$$\frac{\partial L}{\partial \alpha} = \boxed{[y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1]} = 0$$

4c

Only the support vectors contribute to the computation of the solution

Dual problem (cont.d)

This is a quadratic optimization problem.

However we can perform some more steps:

$$L = \frac{1}{2} \|\vec{\omega}\|^2 - \sum \alpha_i [y_i(\vec{\omega} \cdot \vec{x}_i + b) - 1]$$

- We can substitute ω in L

$$\vec{\omega} = \sum_i \alpha_i y_i \vec{x}_i$$

$$\sum_i \alpha_i y_i = 0$$

4

$$L = \frac{1}{2} \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$$

Dual problem (cont.d)

$$L = \frac{1}{2} \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \underbrace{\sum_i \alpha_i y_i b}_{0} + \sum_i \alpha_i$$

$$L = \boxed{\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j}$$
5

The optimization only depends on pairs of samples.

Dual problem (cont.d)

- The dual problem is then

$$\text{Maximize } L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j$$

$$\text{s.t. } \alpha_i \geq 0$$

$$\sum_i \alpha_i y_i = 0$$

Dual problem (cont.d)

Now we have computed the optimal values of alpha and the separator hyperplane can be obtained:

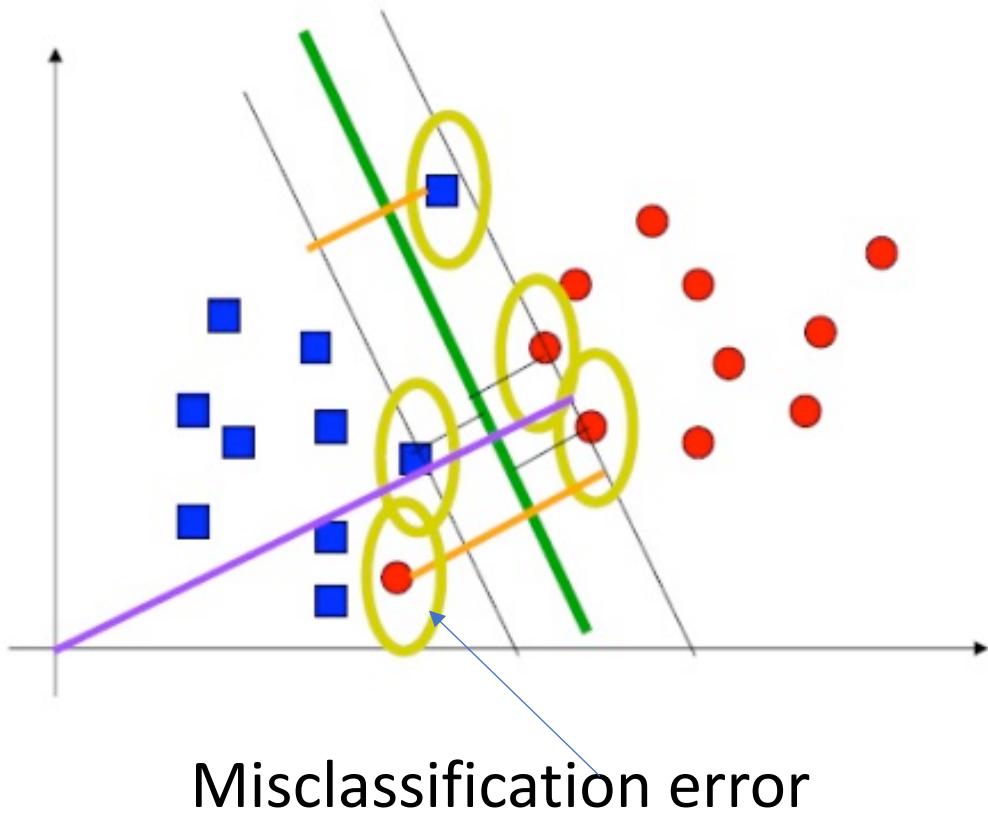
$$\vec{\omega} \cdot \vec{x} + b = \sum \alpha_i \cdot y_i \cdot \vec{x}_i \cdot \vec{x} + b$$

Where the samples x_i considered are only support vectors

Support Vector Machines

Nonlinear separable data

SVM nonseparable data Intuition



SVM nonseparable data

When data are non-linearly separable, we may get a separation between classes with a hyperplane only allowing that, after having defined the separating hyperplane, some patterns of the training set with positive label are classified as negative and viceversa. We must accept that some constraints are **violated**.

We introduce a **slack** variable ξ_i for each constraint, in order to allow an **error tolerance**:

$$y^{(i)}(\omega^T \cdot x^{(i)} + b) \geq 1 - \xi^{(i)}$$

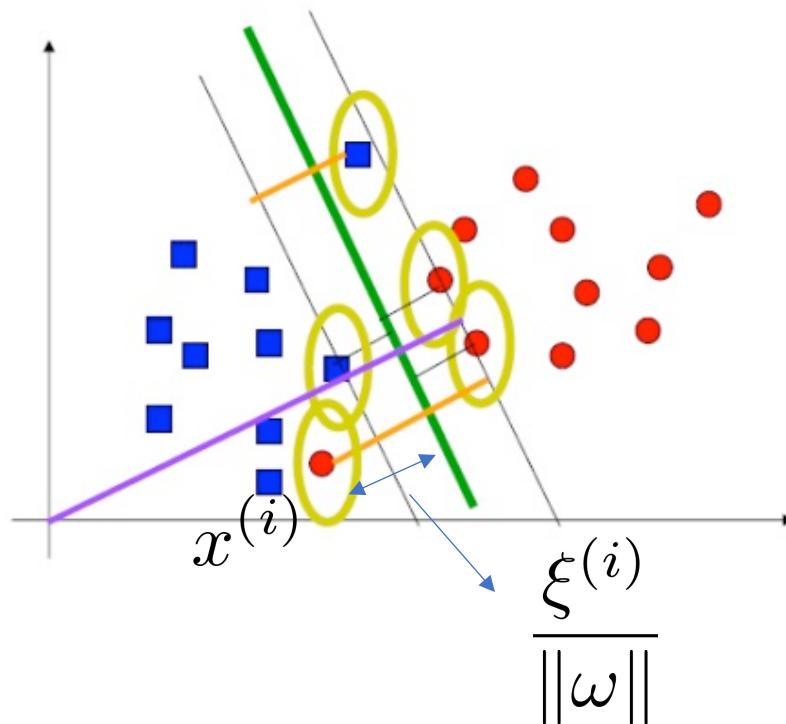
An additional term C is introduced in the cost function to **penalize misclassification** errors.

$$\frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^m \xi^{(i)}$$

SVM nonseparable data (cont.d)

ξ_i are cost variables proportional to how far the misclassified pattern is from the hyperplane.

$\xi_i > 1$ indicates a misclassification error



SVM nonseparable data (cont.d)

C (regularization parameter) lets to control the trade-off between hypothesis space complexity and the admissible number of errors.

A big value for C gives a stronger penalization to errors.

The optimization problem to solve becomes:

$$\text{Minimize: } \frac{1}{2} \|\omega\|^2 + C \sum_{i=1}^m \xi^{(i)}$$

$$\text{s.t. } y^{(i)}(\omega^T \cdot x^{(i)} + b) \geq 1 - \xi^{(i)}$$

$$\xi^{(i)} \geq 0$$

SVM nonseparable data (cont.d)

The dual problem becomes:

$$\max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j x_i^T x_j$$

s.t. $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^m \alpha_i y_i = 0$

Dual variables are now bounded with C .

The proposed solution could not be enough. It does not guarantee good performances since a hyperplane can only represent a dichotomy in the space of instances/patterns.

Cover's theorem

Cover's theorem on the separability of patterns

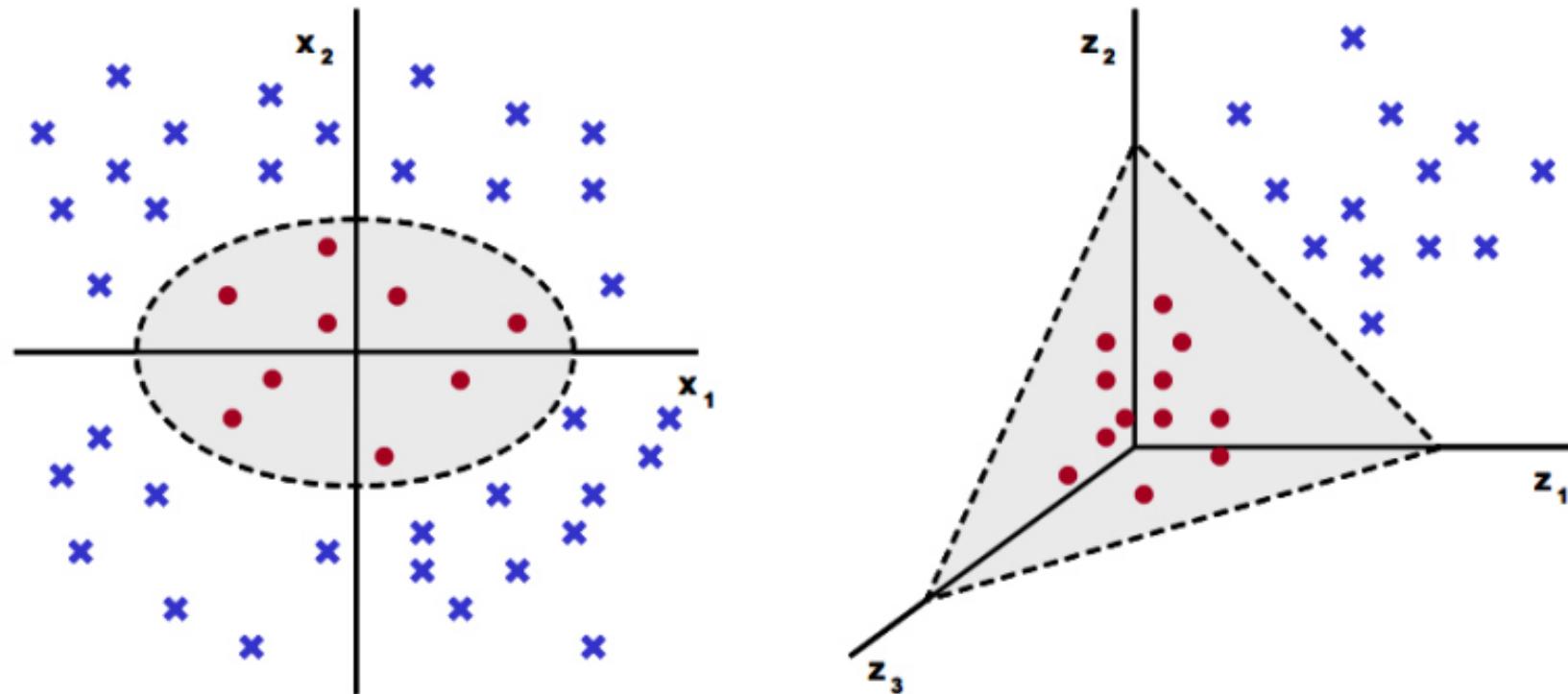
“A complex pattern-classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space”

1. Patterns (input space) are mapped into a space with (much) higher dimension (feature space) through kernel functions;
2. The optimal hyperplane is defined within this feature space.

Cover's theorem

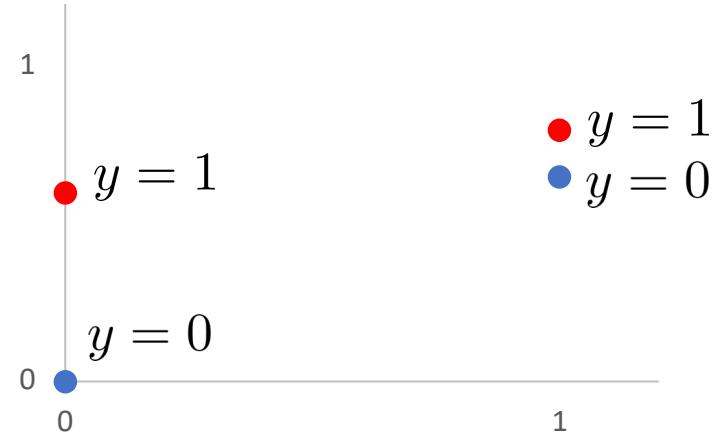
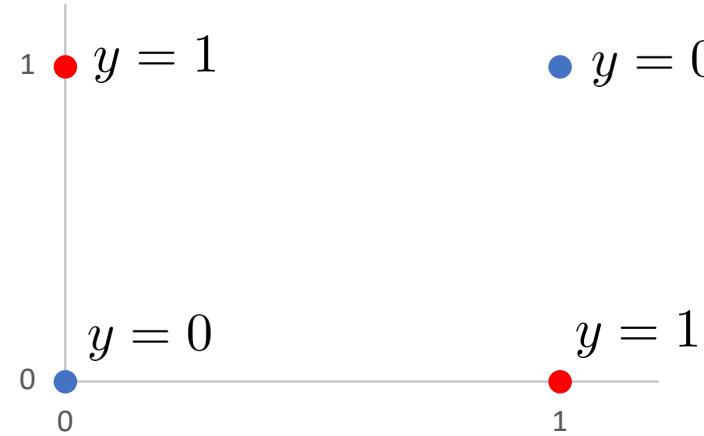
$$\varphi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



[Schölkopf, 2002 @; <http://kernel-machines.org/>]

The case of the non linearly separable space



In this case the samples are not linearly separable and we are not able to find a solution for our problem, but we can move to another space that is more convenient for our purposes:

$$\vec{x} \Rightarrow \Phi(\vec{x})$$

The case of the non linearly separable space

$$K(x_i, x_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$$

This means that the optimization can be done using pairs of samples, and the decision can be computed again using a sample and an unknown vector.

This implies that ALL WE NEED is a function K that corresponds to the dot product between the samples in the new space.

This function K is called **kernel function**.

We have no need of the transformation function Phi!!

~~$\Phi(\vec{x}_i)$~~

Kernel trick

- Instead of increasing the complexity of the classifier (still remains a hyperplane) we increase the **dimension** of features space.
- BUT a higher dimensional space provokes serious computational issues because the learning algorithm must work with high dimensional vectors. Actually, the **projection** in a higher dimensional space is only **implicit** thanks to appropriate *Kernel* functions (*Kernel trick*).

To solve the optimization problem, the product $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$ does not have to be explicitly computed in the feature space once we find a kernel function (specifically a positive definite kernel).

Kernel trick $K(x_i, x_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$

The kernel is a function that returns the (scalar) product of projections: it avoids you to explicitly compute the projection and make the product between the projected vectors. The explicit form of φ may be ignored.

Kernel trick (cont.d)

If a kernel function is defined, that is a function such that:

$$K(x_i, x_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j) = \sum_{k=1}^m \Phi(\vec{x}_i)_k \cdot \Phi(\vec{x}_j)_k$$

The optimization problem becomes:

$$\max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

$$\text{s.t. } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^m \alpha_i y_i = 0$$

Kernel trick (cont.d)

If a kernel function is defined, that is a function such that:

$$K(x_i, x_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j) = \sum_{k=1}^m \Phi(\vec{x}_i)_k \cdot \Phi(\vec{x}_j)_k$$

The optimization problem becomes:

$$\max_{\alpha} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

$$\text{s.t. } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^m \alpha_i y_i = 0$$

The projection $\Phi(\vec{x}_i)$ into the feature space must not be explicitly computed.

I should compute the scalar product $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$, but I don't have to, since I can **indirectly** obtain it with the kernel function

$$K(x_i, x_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$$

Kernel examples

Linear kernel

$$(\vec{u} \cdot \vec{v})$$

Polynomial kernel

$$(\vec{u} \cdot \vec{v} + 1)^d$$

Multi-Layer Perceptron tanh

$$\tanh(b(\vec{u} \cdot \vec{v}) - c)$$

Kernel examples (cont.d)

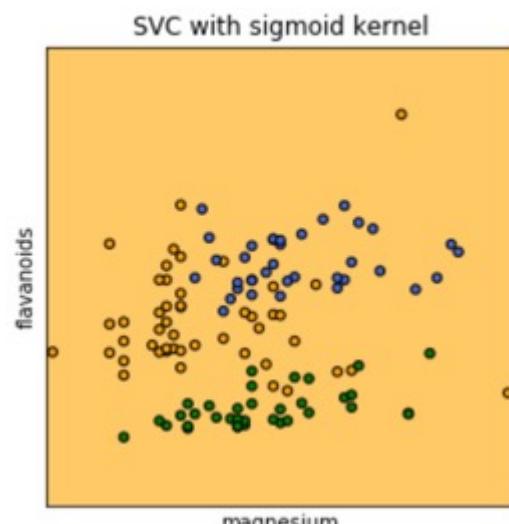
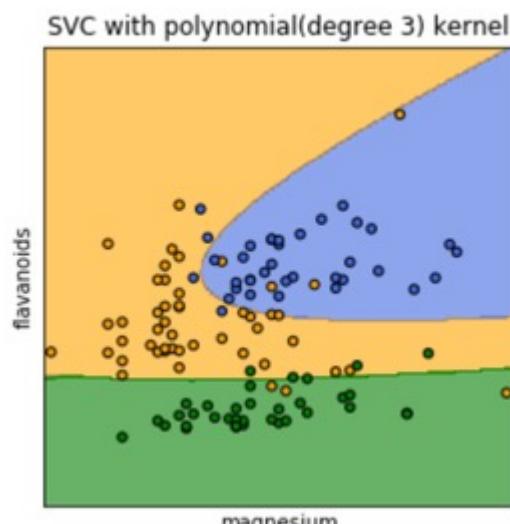
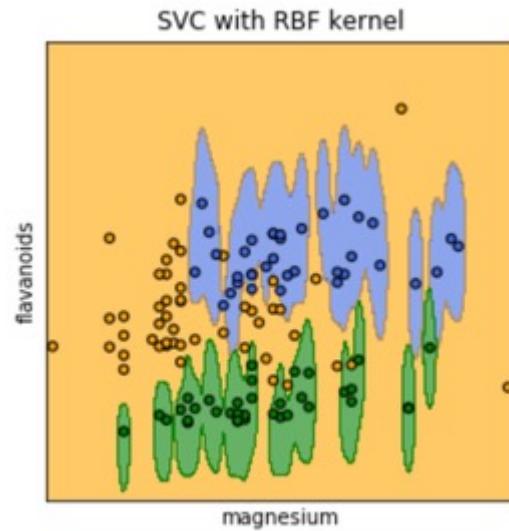
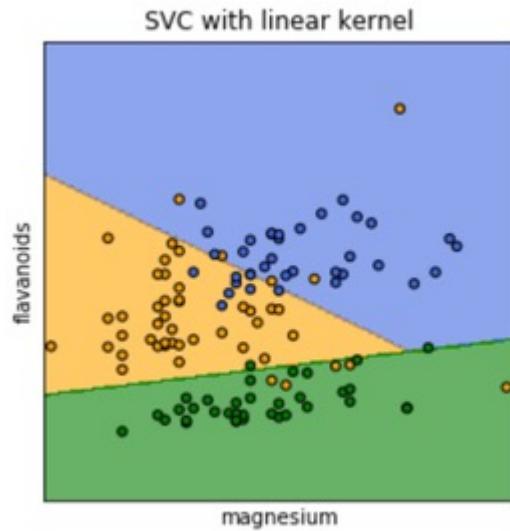
Radial basis function (RBF) kernel

$$e^{-\frac{\|x_i - x_j\|}{\sigma}}$$

Gaussian Radial basis function kernel

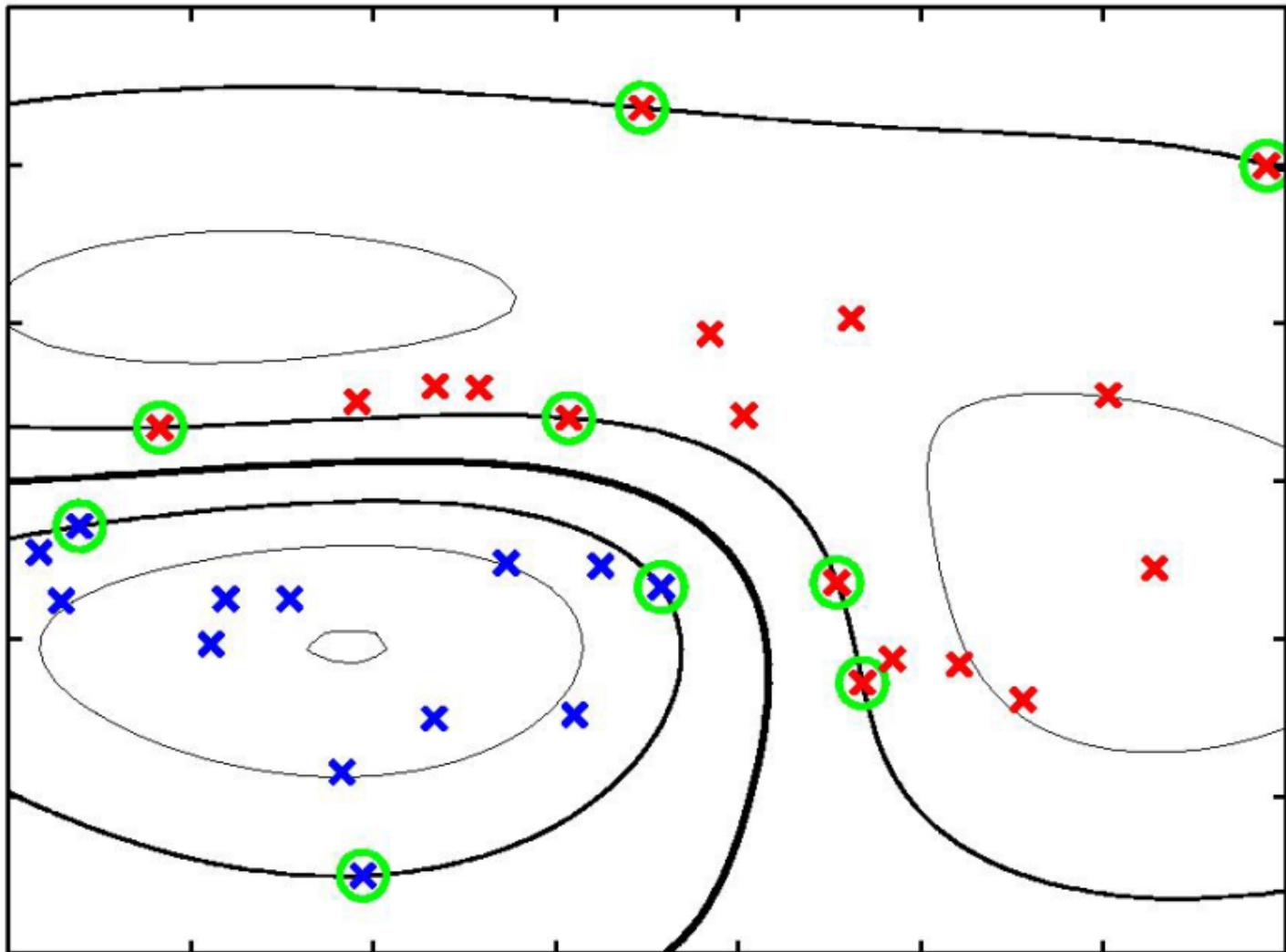
$$e^{-\frac{(x_i - x_j)^2}{2\sigma^2}}$$

Kernel examples (cont.d)



Kernel examples (cont.d)

Gaussian kernel



Parameters tuning

To use Support Vector Machines you have to define:

- the kernel function;
- potential parameters of the kernel function;
- The value for the regularization parameter C .

General rules for the set up do not exist, but you should make your choice on a validation set, usually through cross validation.

Advantages of SVMs

- There are **no local minima** (the optimization problem is quadratic -> $\exists!$ optimal solution)
- The optimal solution can be found in polynomial time.
- There are few parameters to set up (C , type of kernel and specific kernel parameters)
- Solution is stable (ex. there is no problem of randomly initializing of weights just as in Neural Networks)
- Solution is sparse: it just involves support vectors.

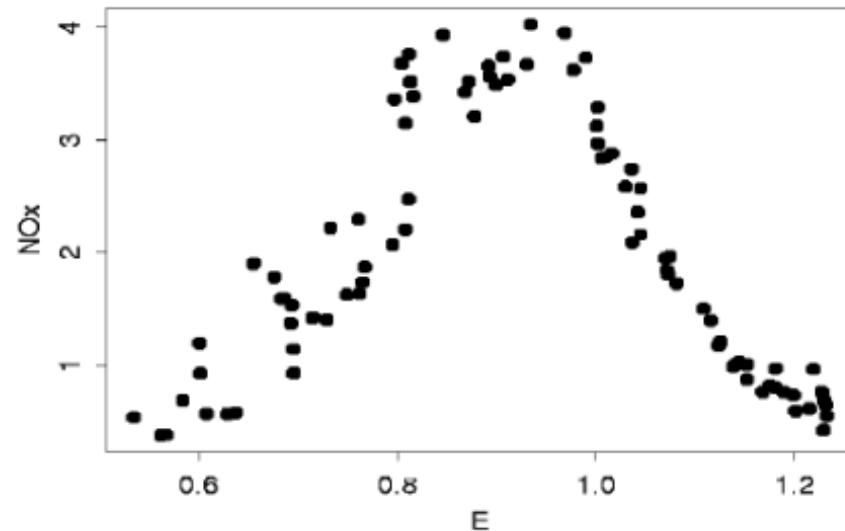
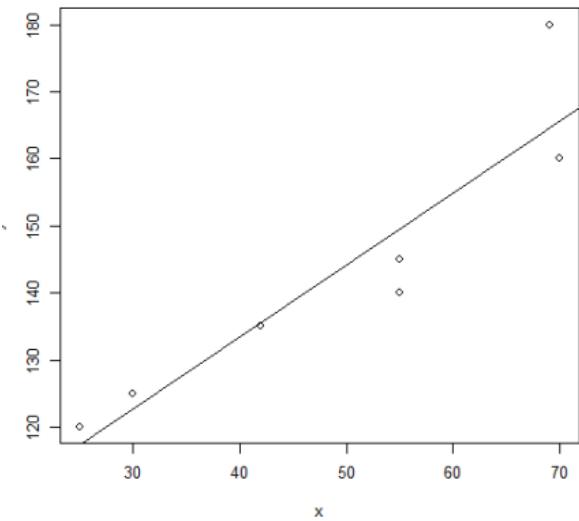
Generalized Linear Models

Basic functions

Generalized Linear Models

It is usual that our data cannot be approximated to a linear function.

What we need is to find a way to model nonlinear relations without increasing too much the complexity of the algorithm



Generalized Linear Models

The main and limiting characteristics of linear regression is the linearity of parameters

$$h(x) = \sum_{i=0}^n \theta_i \cdot x_i = \theta^T x$$

$h(x)$ keeps a linear relation w.r.t. the features space X . This linearity represents a limitation of the expressiveness of the model because the hypothesis is only able to approximate linear functions of the input. GLMs represent an extension to linear models that allow nonlinear transformations of the input

$$h(x) = \sum_{i=0}^n \theta_i \cdot \phi(x_i) = \theta^T \phi(x)$$

Phi are called basic functions

Example $x_1, x_2, x_3 \rightarrow x_1, x_2, x_3, x_1x_2, x_1x_3, x_1^2, x_2^2, x_3^2$

Generalized Linear Models

The main kinds of basic functions are:

Linear

$$\phi_i(x) = x_i$$

Polynomial

$$\phi_i(x) = x_i^p$$

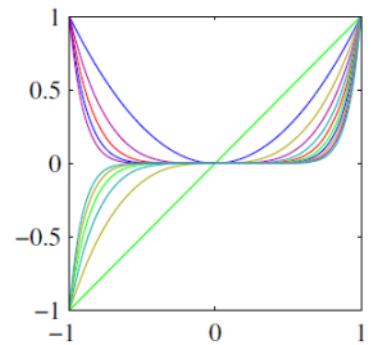
Gaussian

$$\phi_i(x) = e^{-\frac{x-\mu_i}{2\sigma^2}}$$

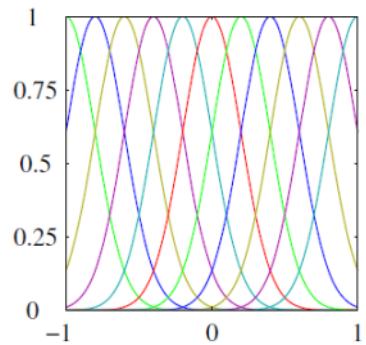
Sigmoidal

$$\phi_i(x) = \frac{1}{1 + e^{-\frac{x-\mu_i}{2\sigma^2}}}$$

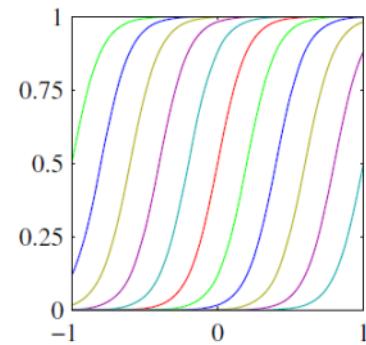
Polynomial



Gaussian



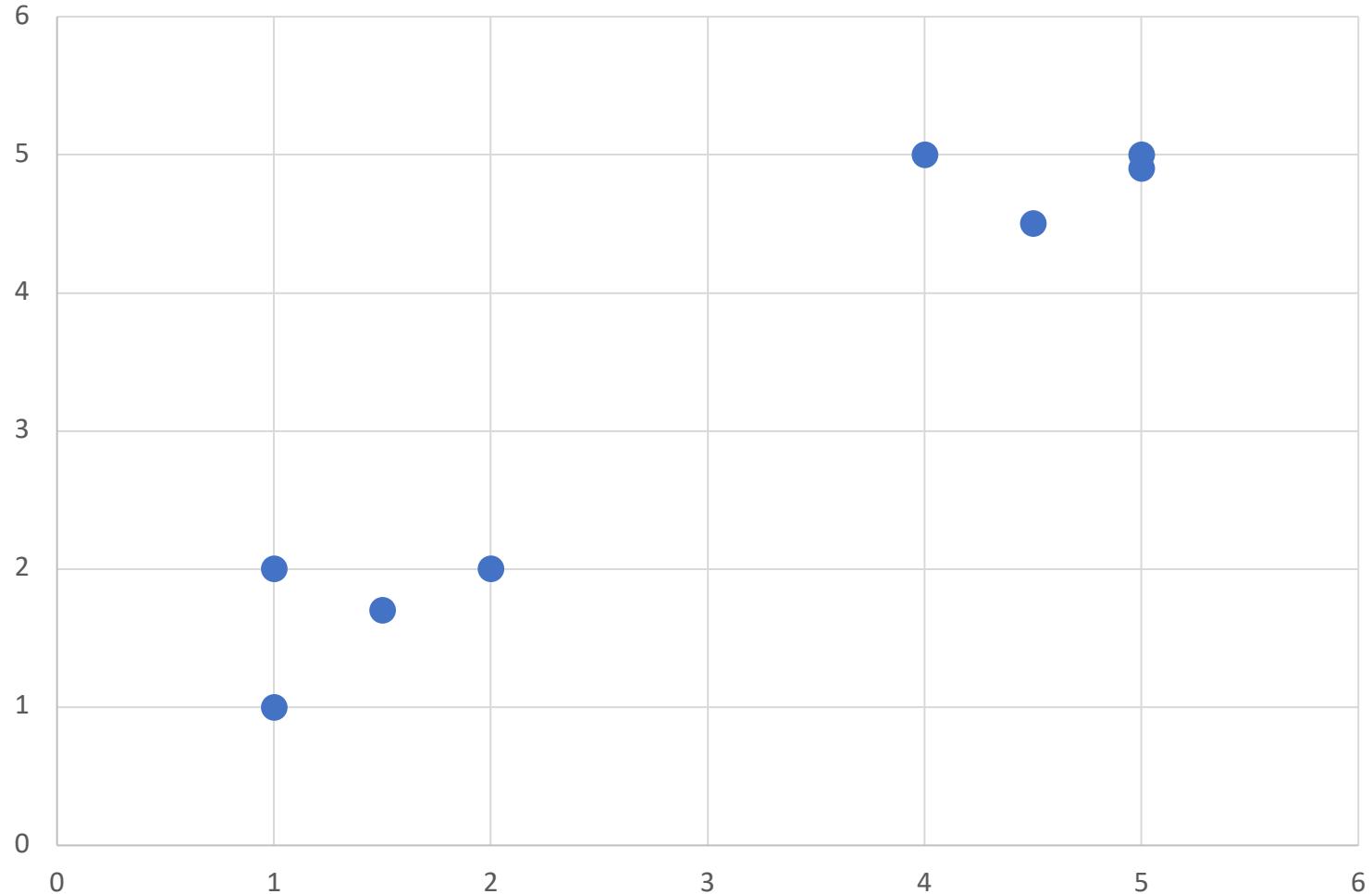
Sigmoidal



Unsupervised Learning

Introduction

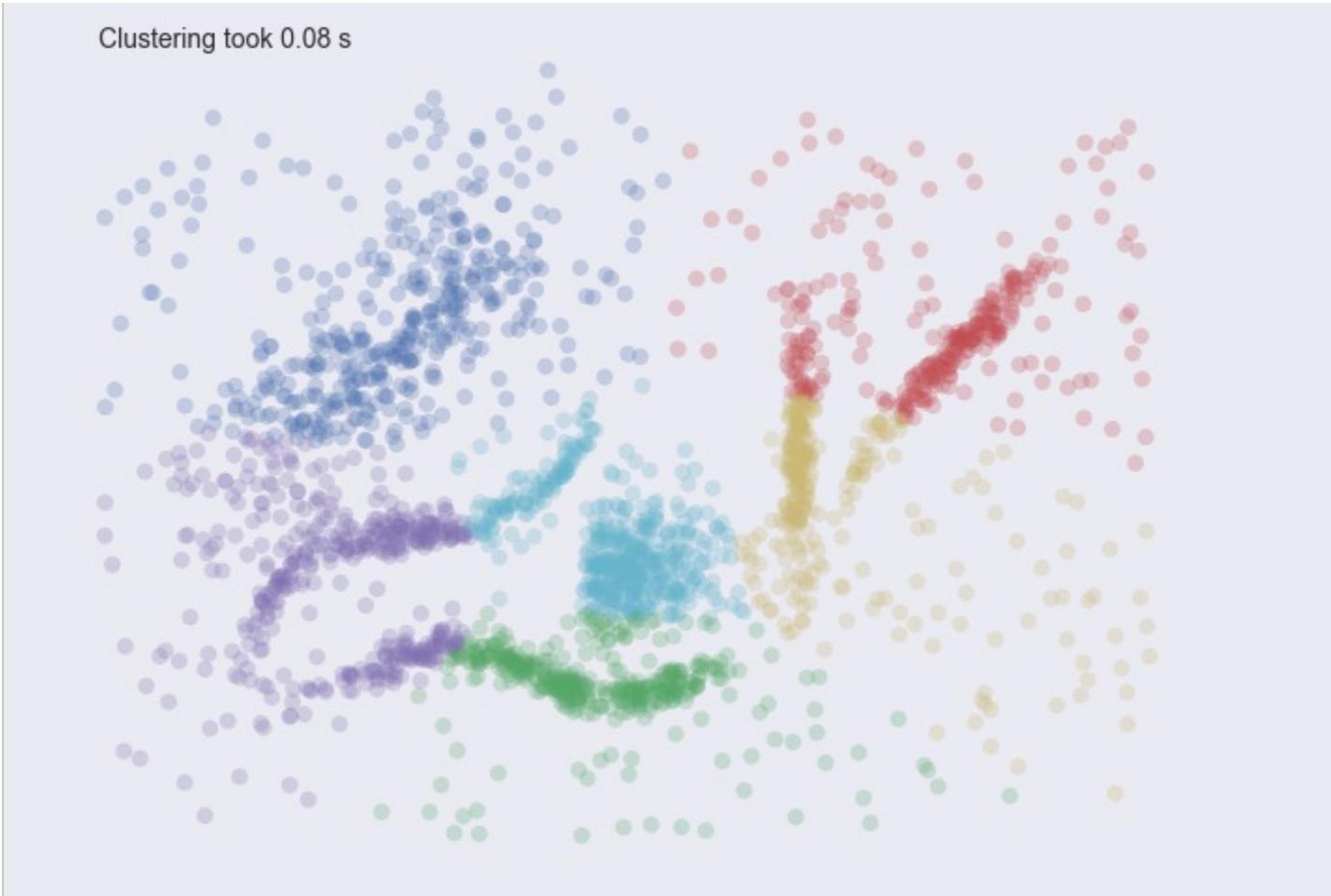
Unsupervised Learning



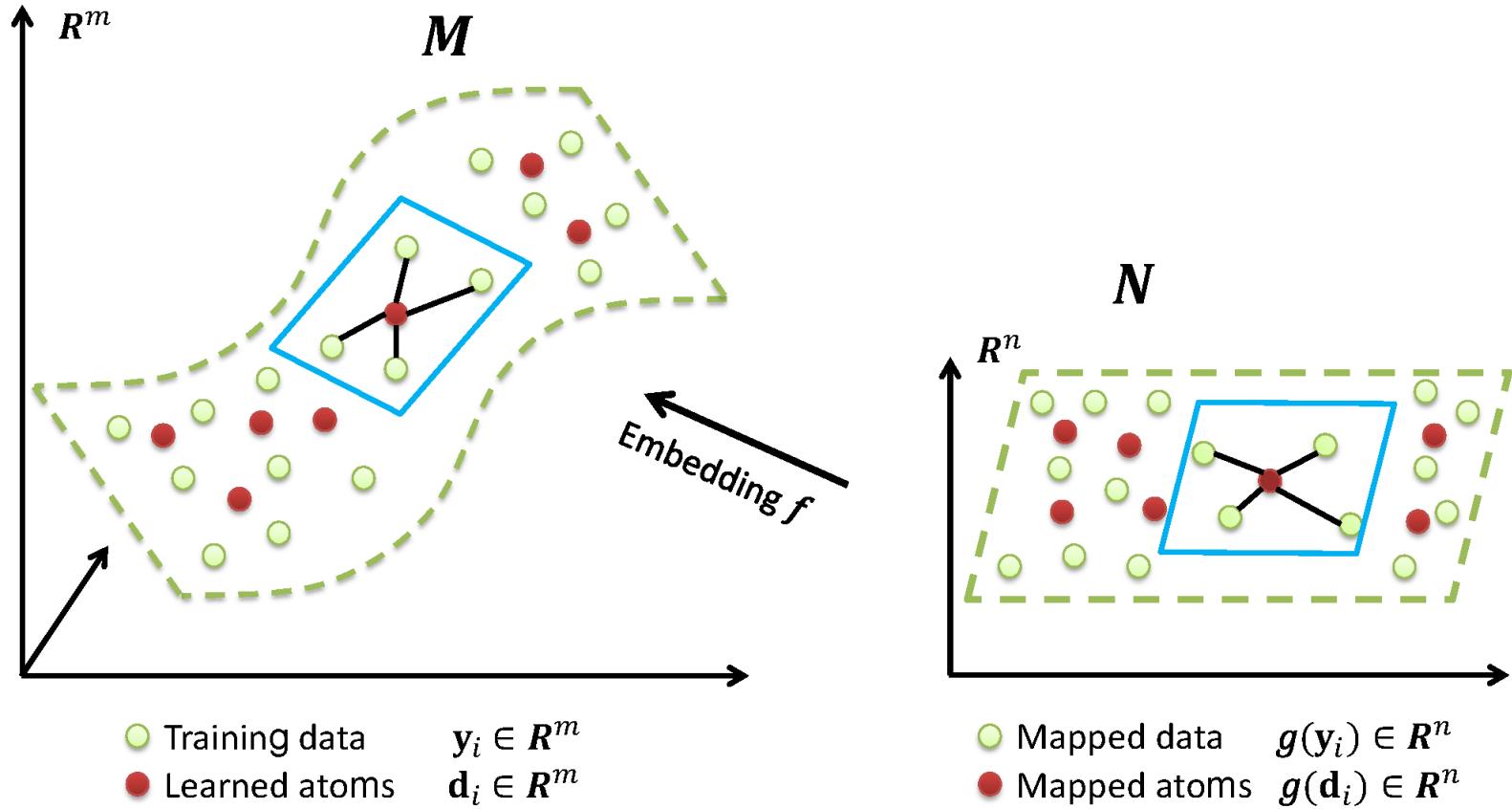
Unsupervised Learning



Unsupervised Learning



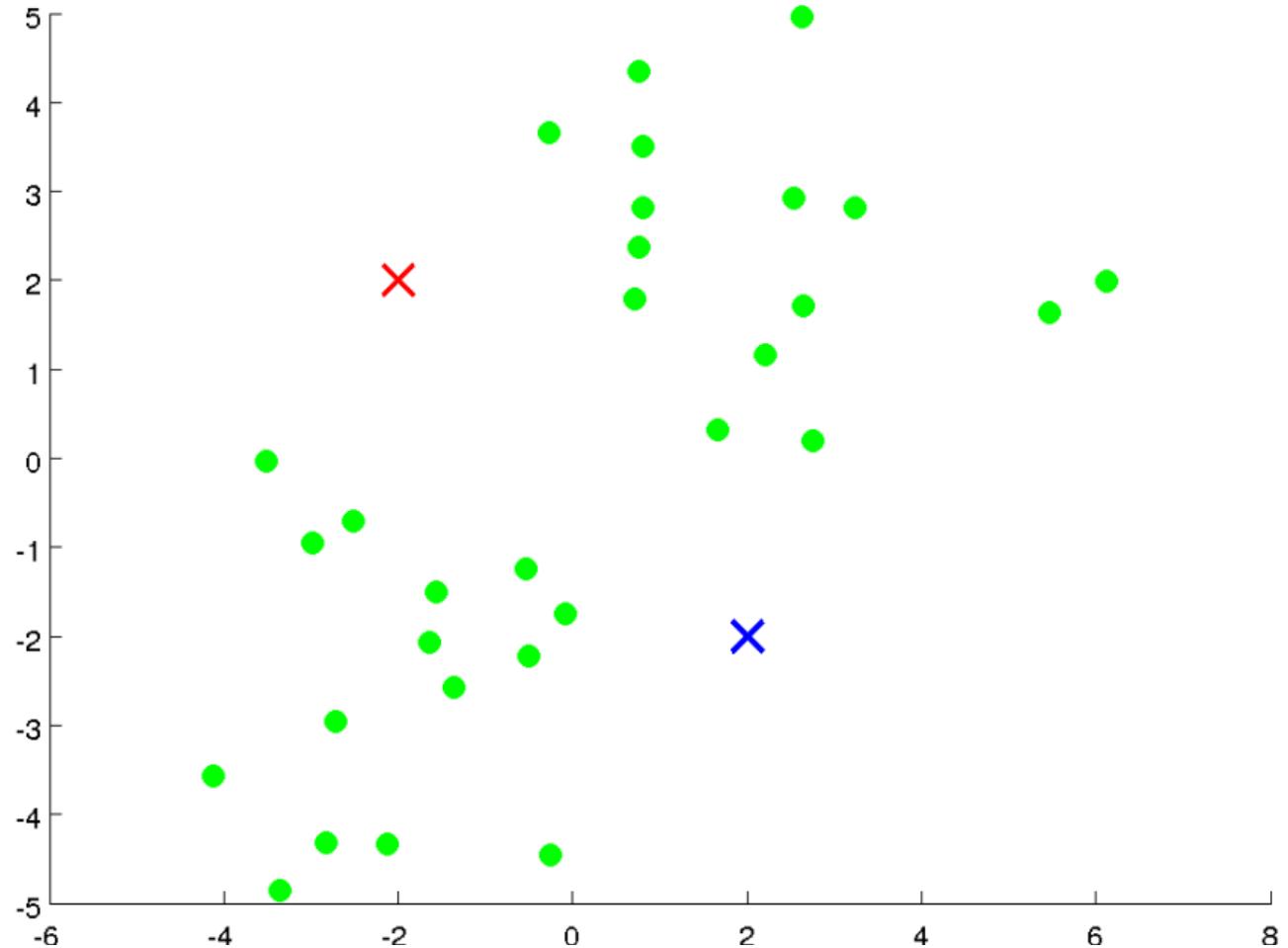
Unsupervised Learning



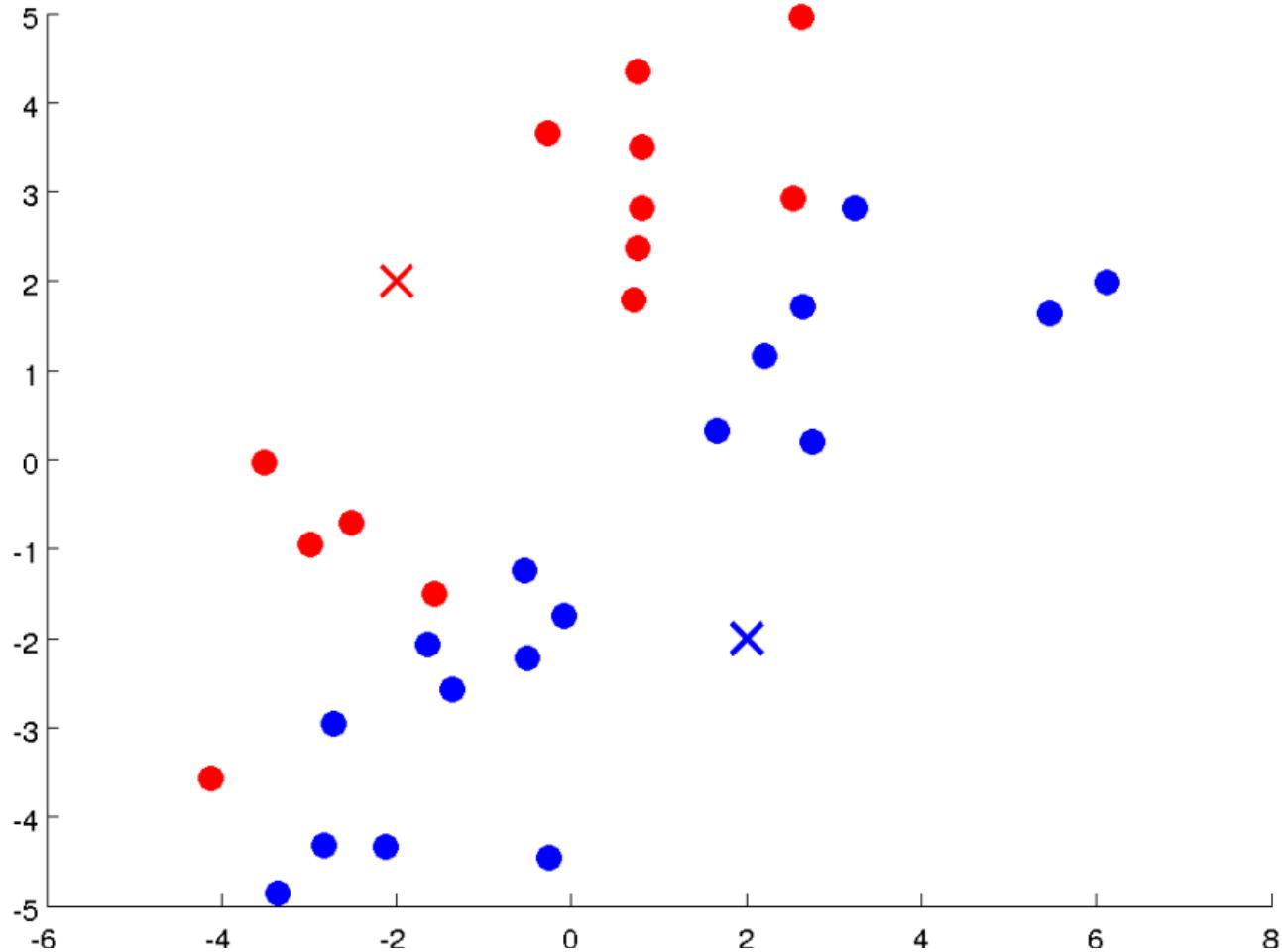
Clustering

K-means

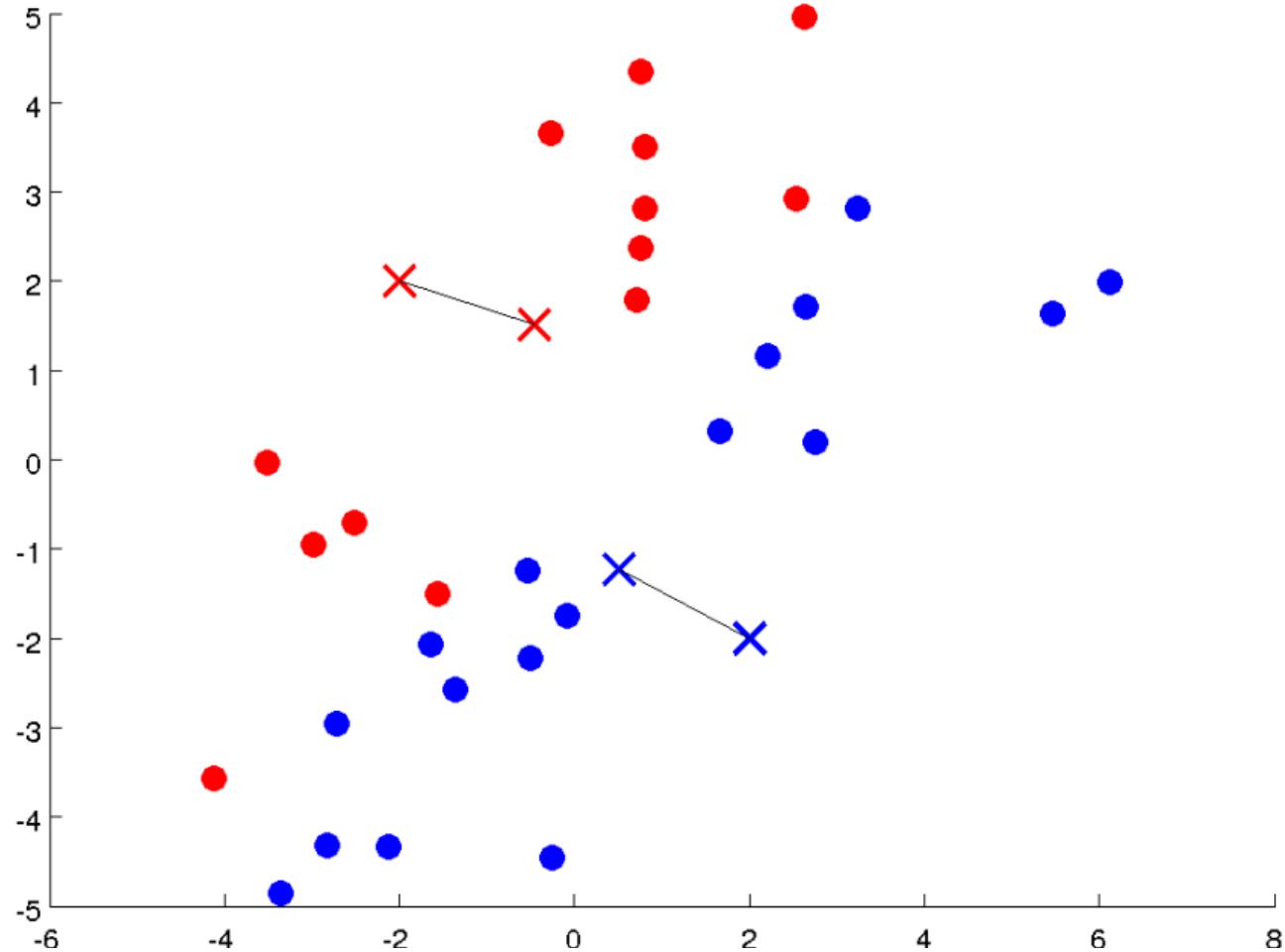
K-means



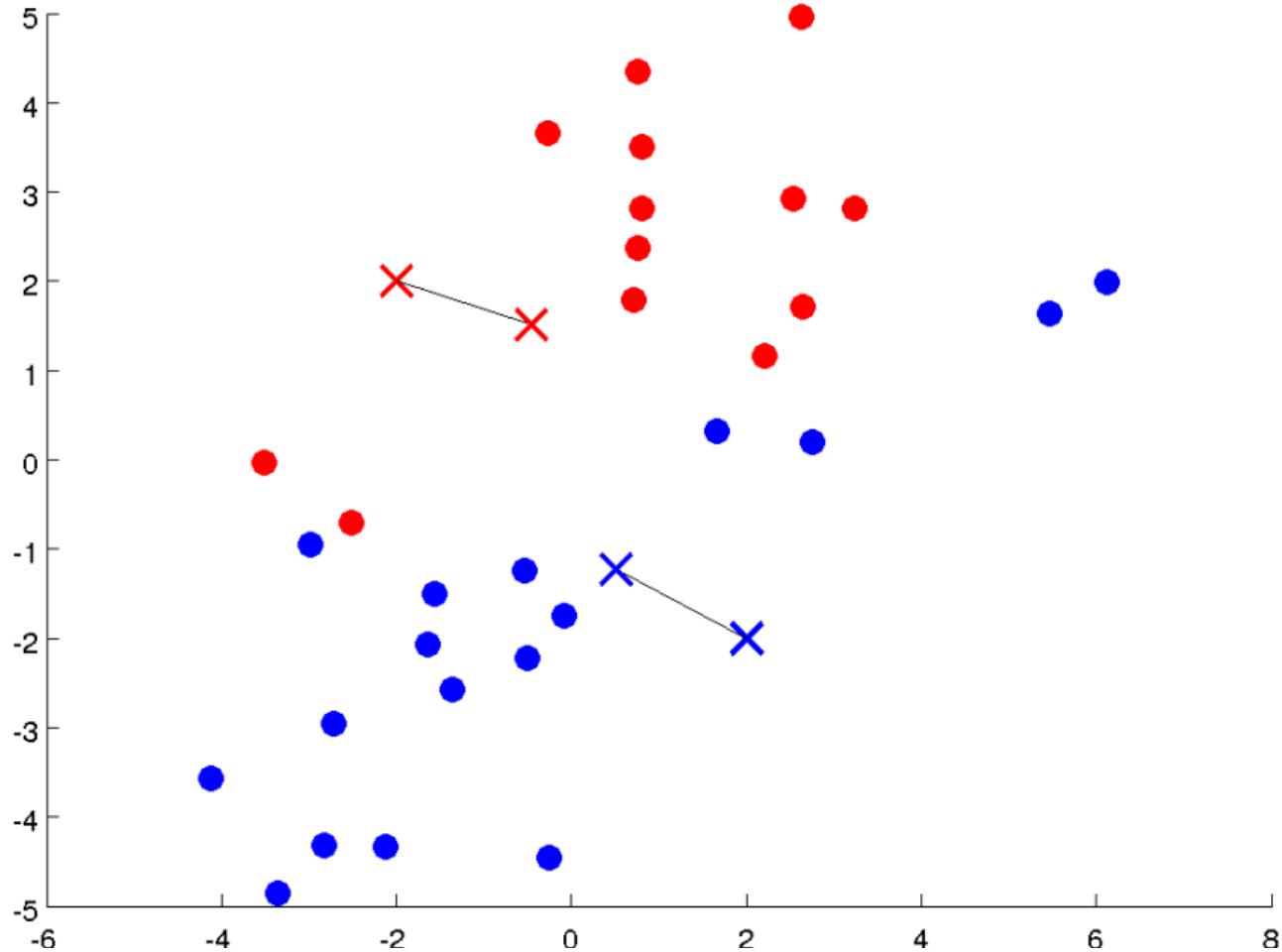
K-means



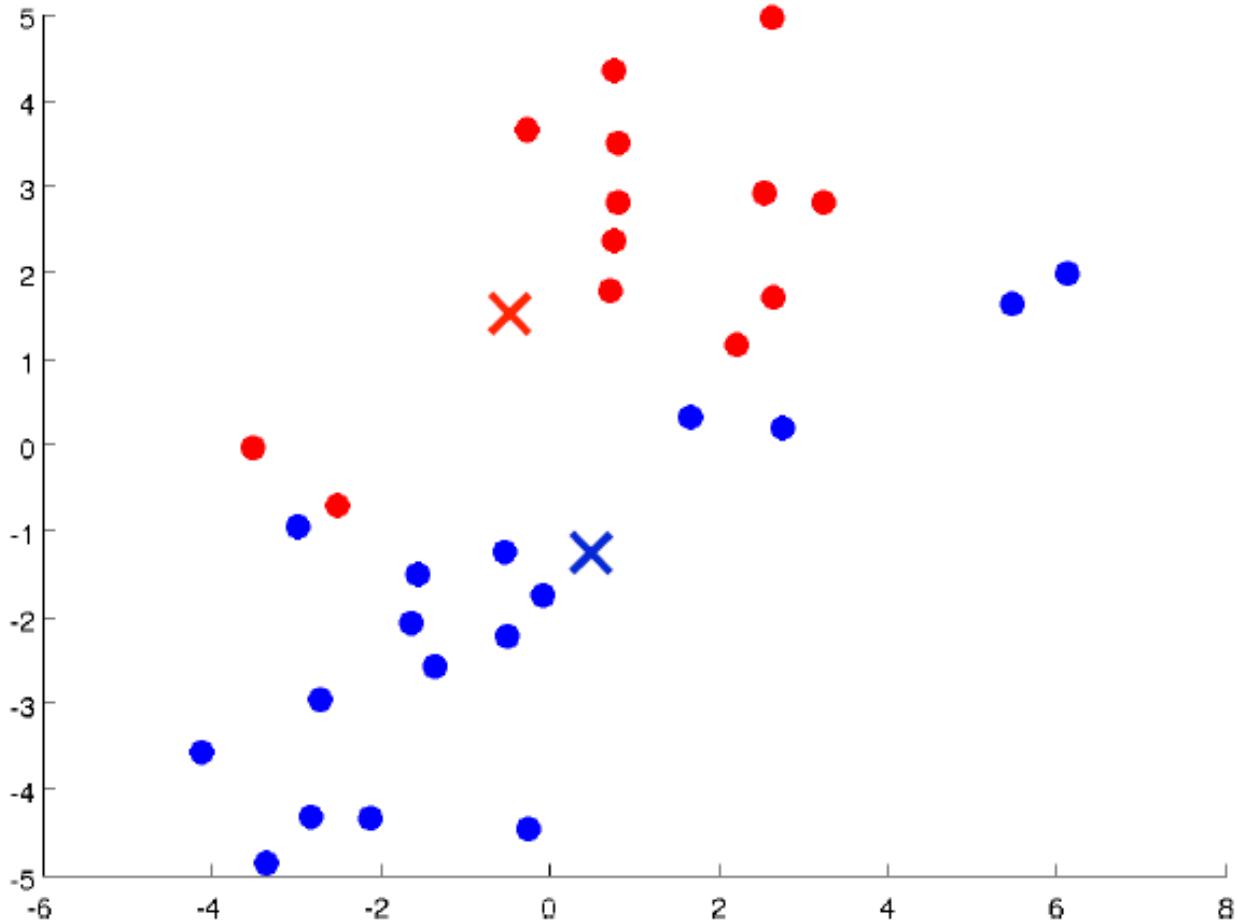
K-means



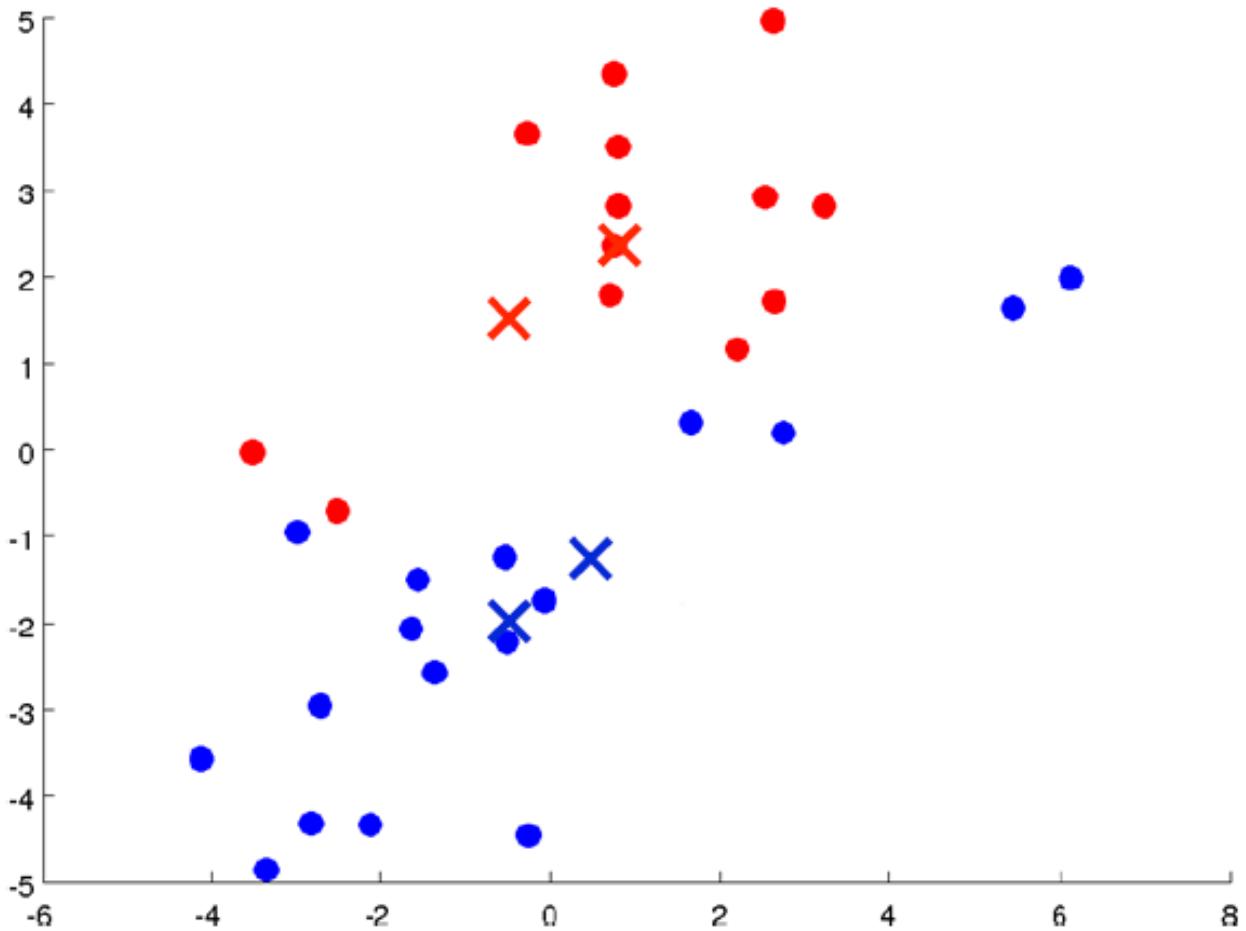
K-means



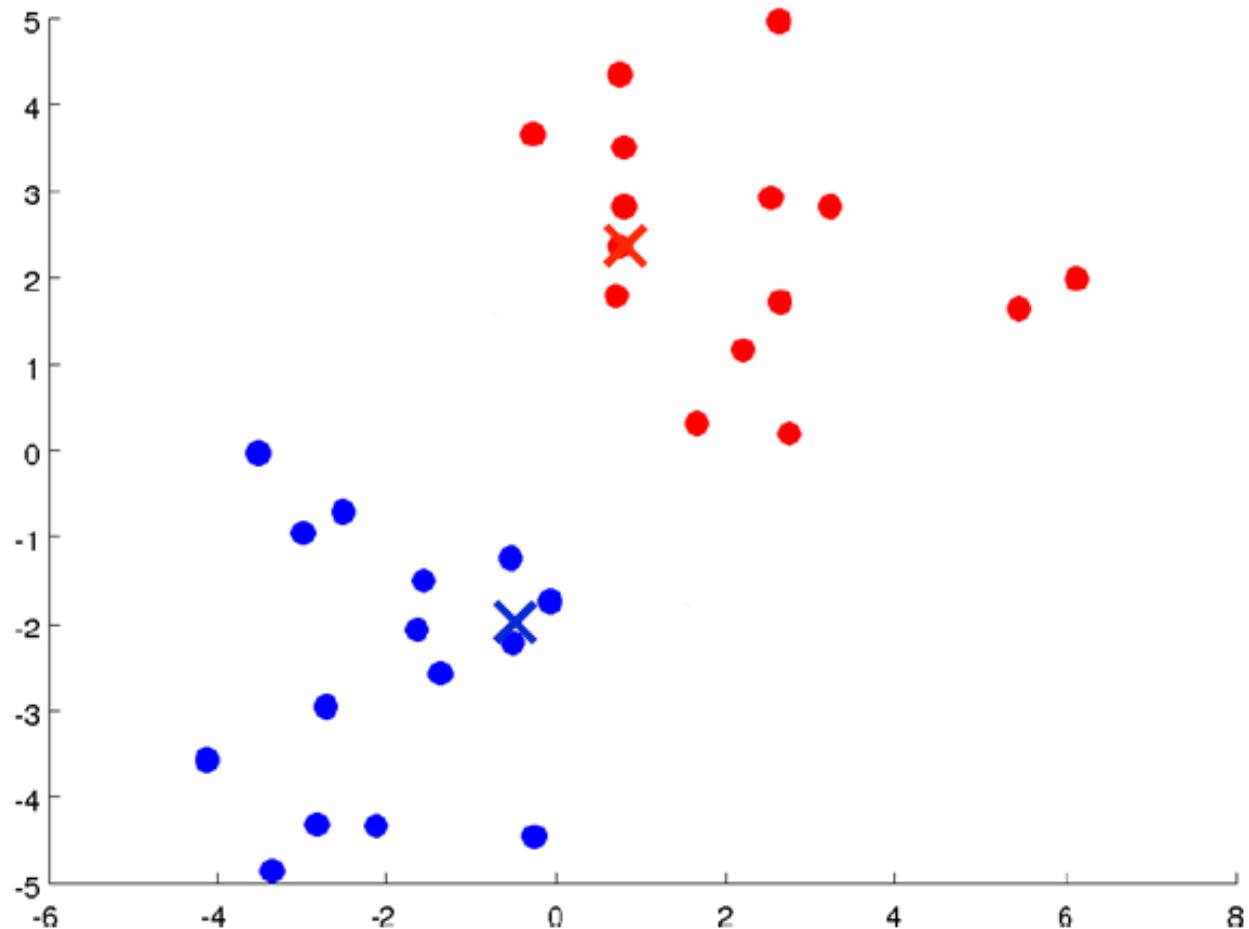
K-means



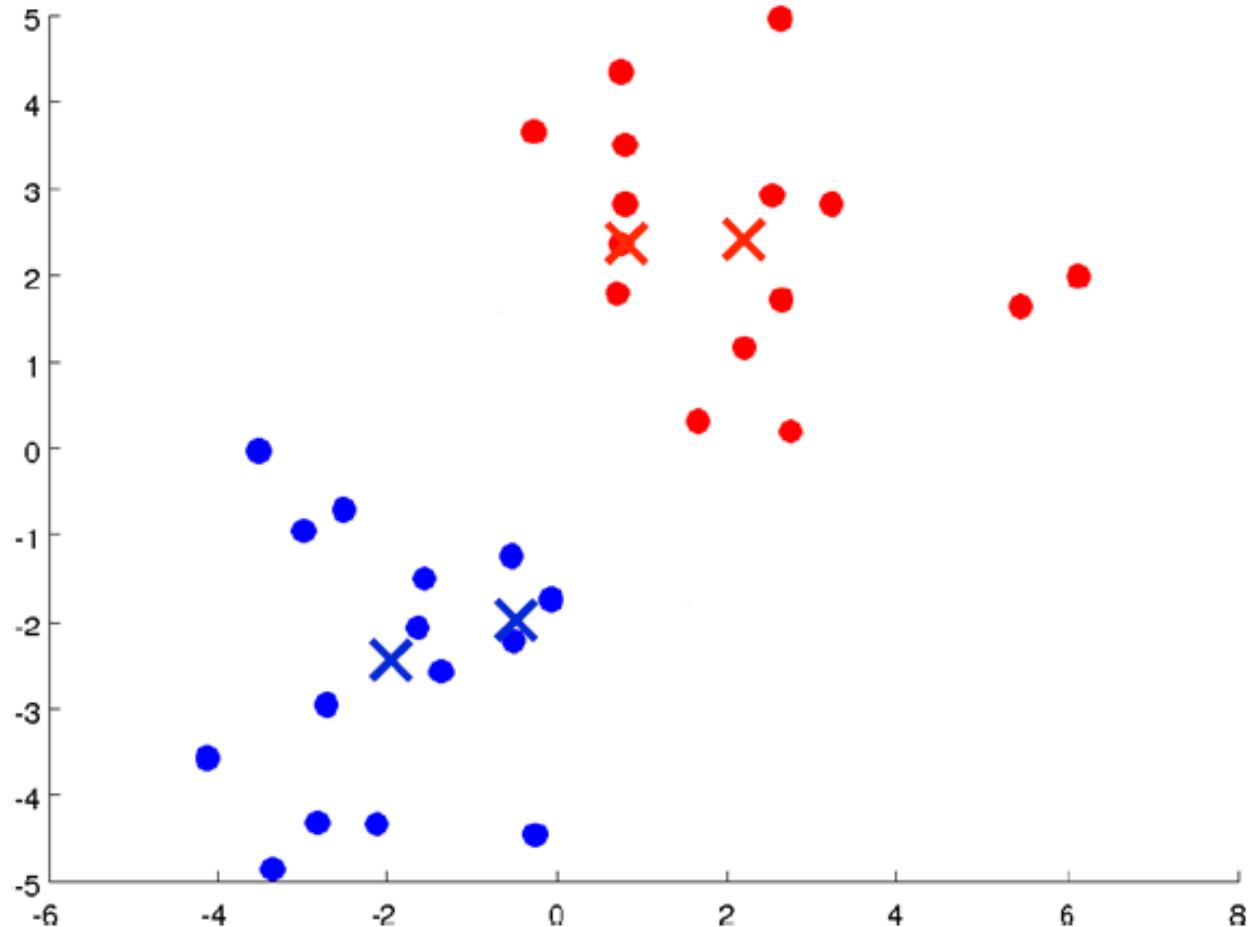
K-means



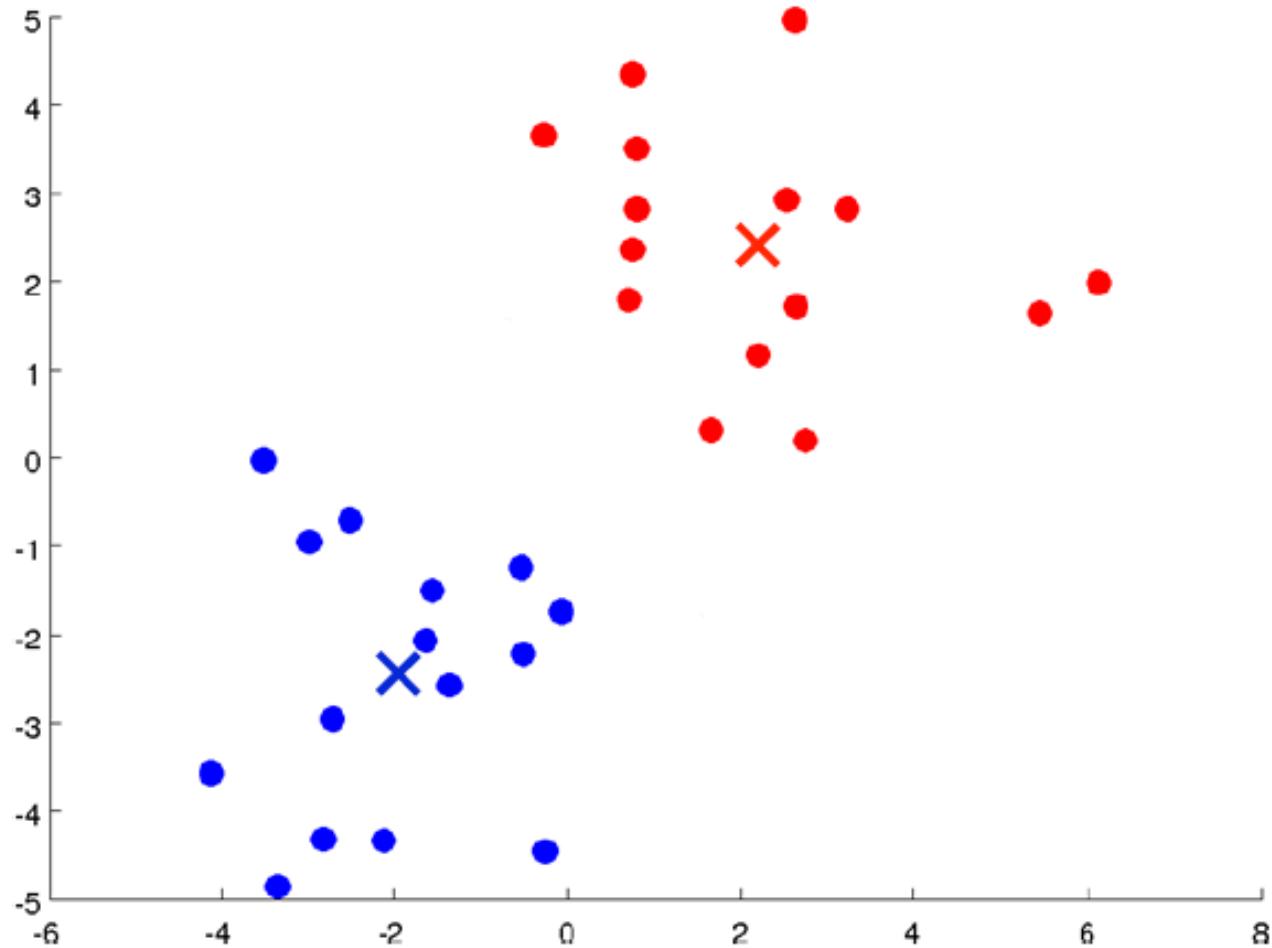
K-means



K-means



K-means



K-means algorithm

$$L = \sum_{k=1}^K \sum_{i=1}^m a_{ik} \|x^{(i)} - \mu_k\|^2$$

Minimize L with respect to a and μ following these two steps:

[Expectation] Choose optimal a for fixed μ by assigning $x^{(i)}$ to the nearest μ_k

$$a_{ik} = \begin{cases} 1 & \text{if } k = \arg \min_l \|x^{(i)} - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

[Maximization] Choose optimal μ for fixed a by updating μ_k to be the empirical mean of the points assigned to each cluster

$$\mu_k = \frac{1}{n_k} \sum_{i: x_i \in C_k} x^{(i)} \quad \text{where } n_k = \sum_{i=1}^m a_{ik} \text{ (number of data points in the k-th cluster } C_k)$$

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

 for i=1 to m

$c^{(i)} :=$ index of cluster centroid (from 1 to K) closest to $x^{(i)}$

 for k = 1 to K

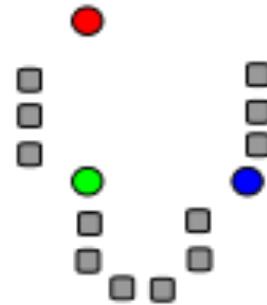
$\mu_k :=$ mean of points assigned to cluster k

}

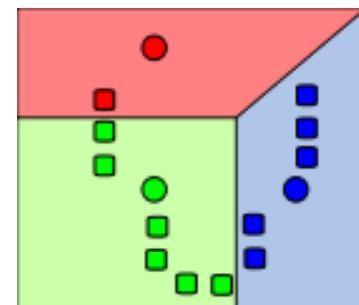
$$\mu_k = \frac{1}{|C_k|} \sum_{x^{(i)} \in C_k} x^{(i)} \quad x^{(i)} \in \mathbb{R}^n$$

K-means

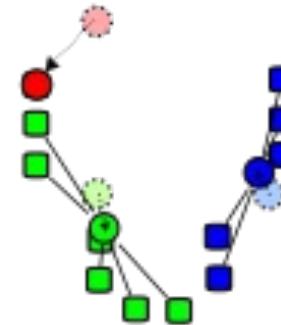
Choice randomly k centroids



Assignment to the clusters

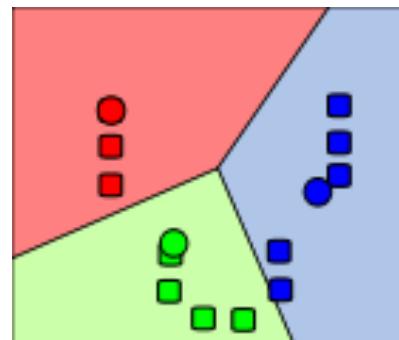


Recompute centroids



iterations

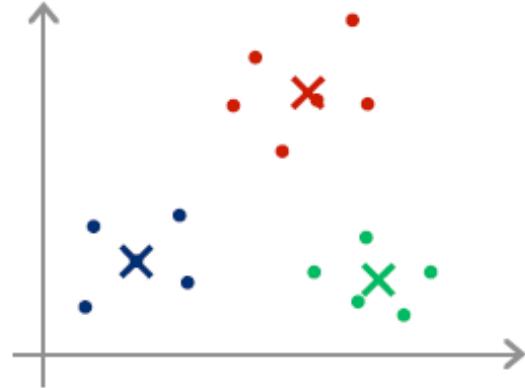
Clusters found after the convergence



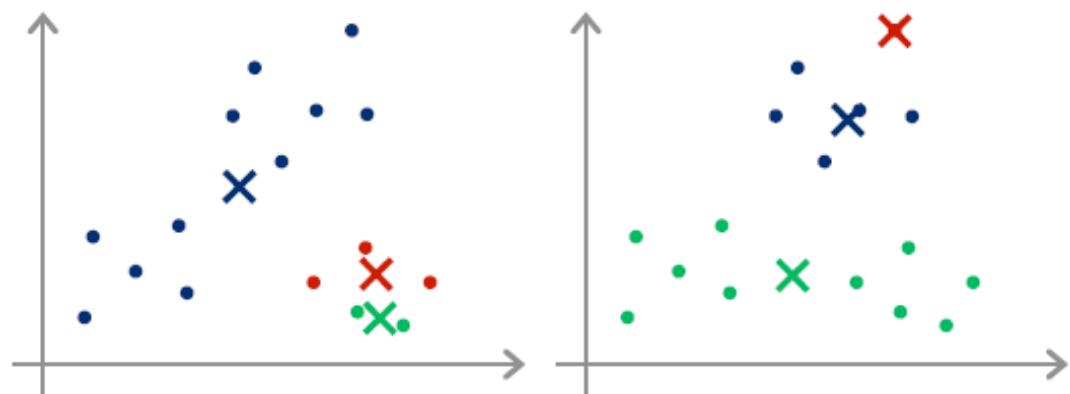
Random initialization

- We should have $k < m$
- Randomly pick K training examples
- Set the centroids equal to these examples

K-means – local minima



$$J(c^1, \dots, c^m, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^i - \mu_c^i\|^2$$



K-means – local minima

For i=1 to 100 {

 Randomly initialize K-means

 Run K-means. Get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k$

 Compute $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k)$

}

Choose clustering that gave the lowest cost $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

K-means – PROS/CONS

PROS

- Quite simple and computationally efficient.
- Always terminates

CONS:

- Finding the optimal clustering is not guaranteed
- Not applicable to categorical values (non-numeric)
- Requires to specify K
- Sensible to noise and outliers

Clustering

K-medoids

Partitioning Around Medoids (PAM)

1. Randomly initialize K cluster medoids μ^k
2. $M = \cup_{k=1}^K \mu^k$
3. Associate each $x^{(i)} \in X$ to the closest medoid k
4. $Cost =$ sum of the distances of samples from their medoids
5. $NewCost = 0$
6. **while**($NewCost \leq Cost$) {

 for $k = 1$ **to** K

 for $x^{(i)} \in X - M$

 swap $x^{(i)}$ and μ^k

 repeat steps 2) and 3)

 $NewCost =$ sum of the distances of samples from their medoids

 if ($NewCost > Cost$) **then** undo swap

 else $Cost = NewCost$

 }

}

K-Medoids – PROS/CONS

Pros

- The method is less sensitive to outliers than k-means

Cons

- Finding the optimal clustering is not guaranteed
- Not applicable to categorical values (non-numeric)
- Requires to specify K
- Yet sensible to noise and outliers
- Fails for non-linear data set
- Computationally more expensive than k-means

Clustering

Gaussian Mixture Models

Gaussian (Normal) distribution

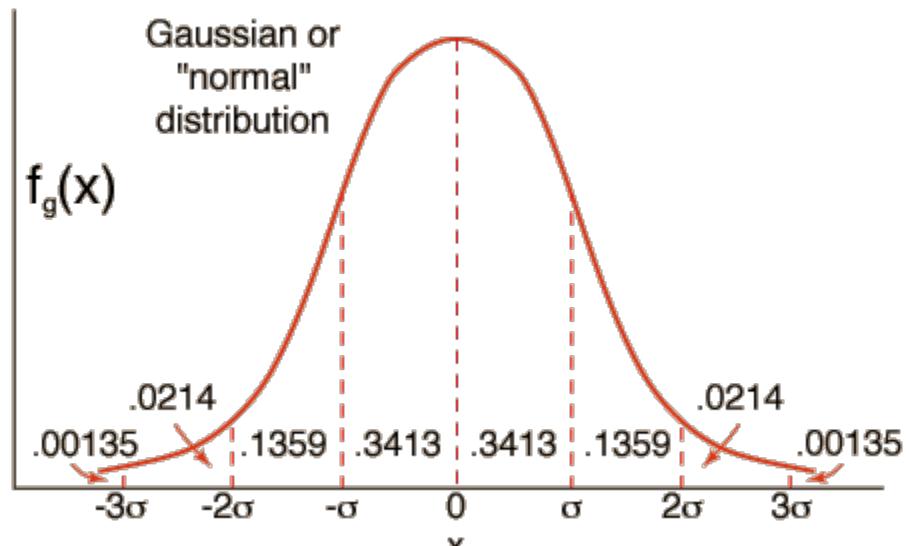
Say $x \in \mathbb{R}$.

x has a Gaussian distribution with mean μ and variance σ^2

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

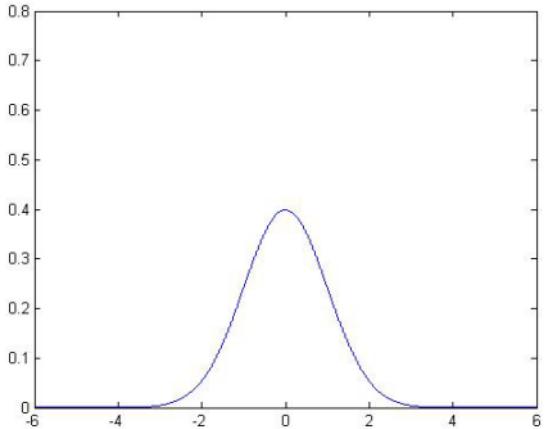
$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In the picture $\mu = 0$

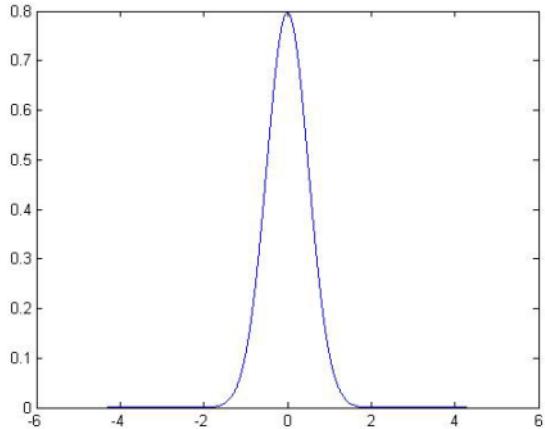


Gaussian distribution example

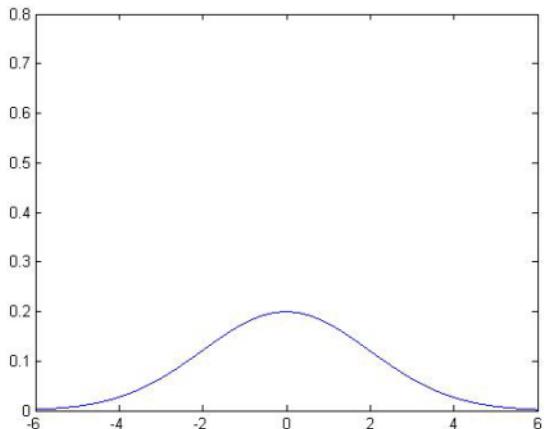
$$\mu = 0, \sigma = 1$$



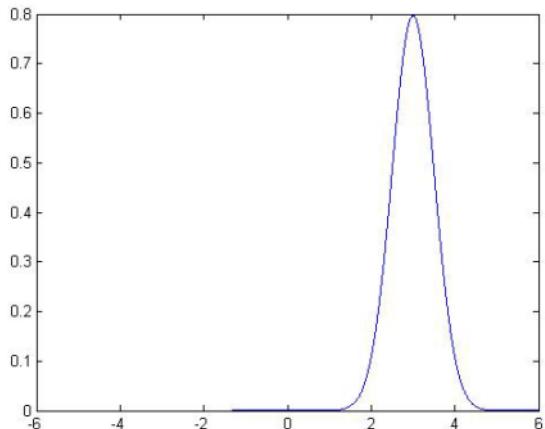
$$\mu = 0, \sigma = 0.5$$



$$\mu = 0, \sigma = 2$$

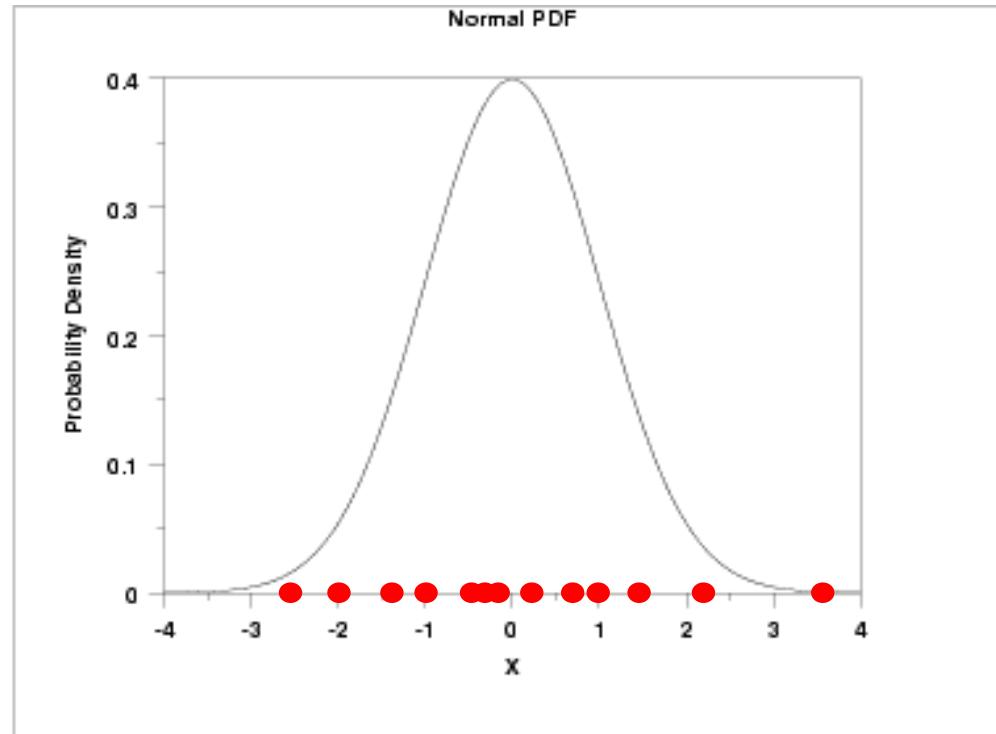


$$\mu = 3, \sigma = 0.5$$



Parameters estimation

Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\} | x^{(1)} \in \mathbb{R}$



$$x \sim \mathcal{N}(\mu, \sigma^2)$$

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$$

Multivariate Gaussian (Normal) distribution

With the previous model we have to model $p(x_1), p(x_2), \dots, p(x_n)$ separately.

$$x \in \mathbb{R}^n$$

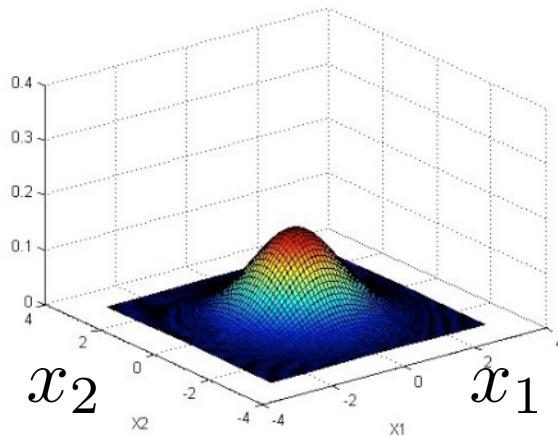
$$\mu \in \mathbb{R}^n$$

$$\Sigma \in \mathbb{R}^{n \times n} \text{ covariance matrix}$$

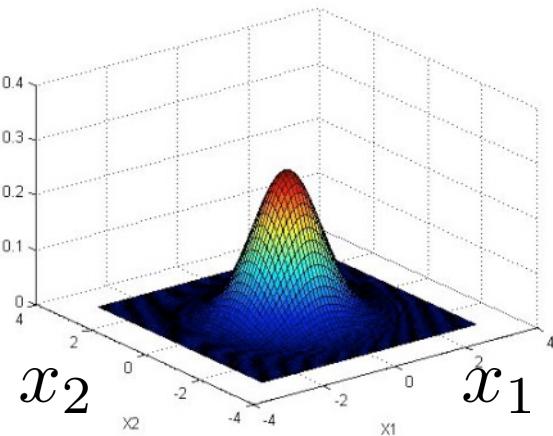
$$p(x; \mu, \Sigma) = \frac{e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))}}{\sqrt{(2\pi)^n |\Sigma|}}$$

Multivariate Gaussian examples

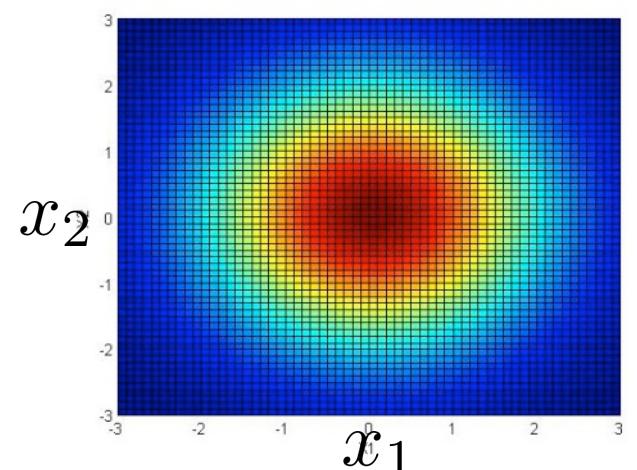
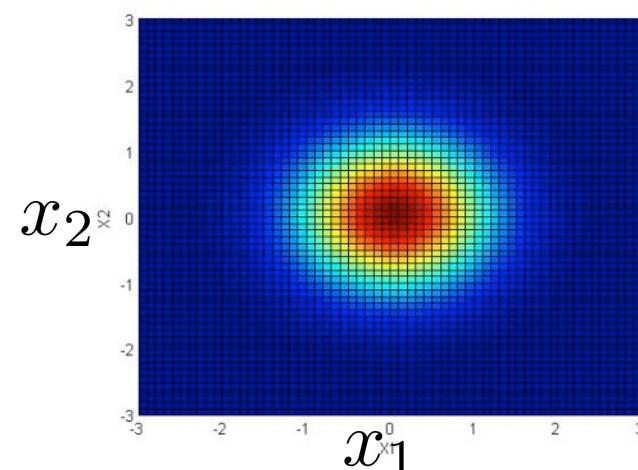
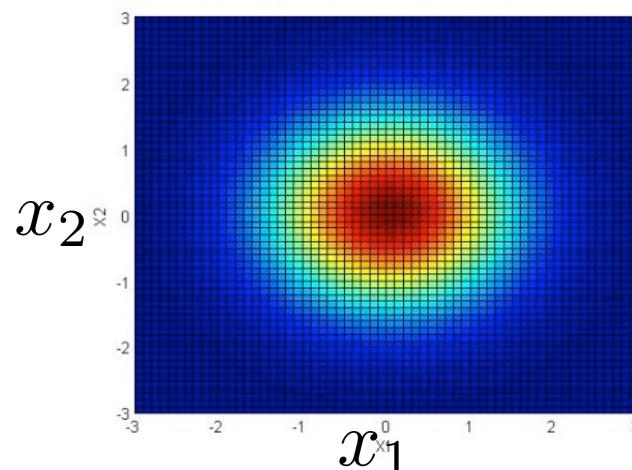
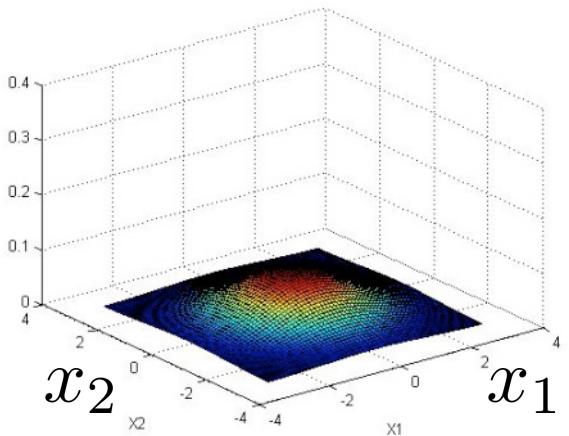
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

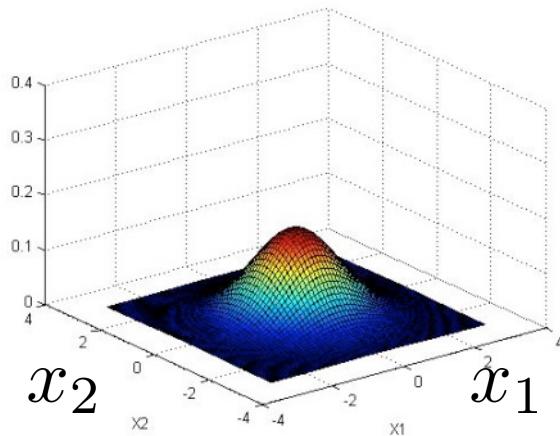


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

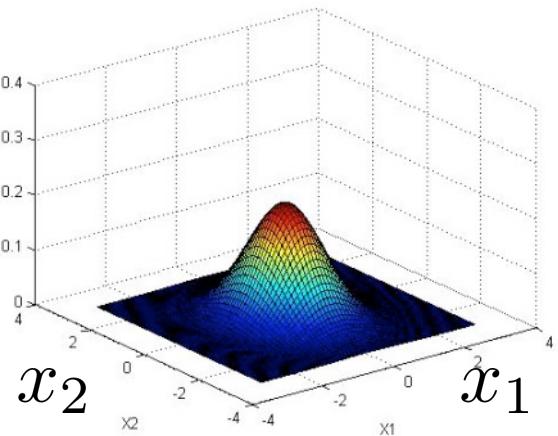


Multivariate Gaussian examples

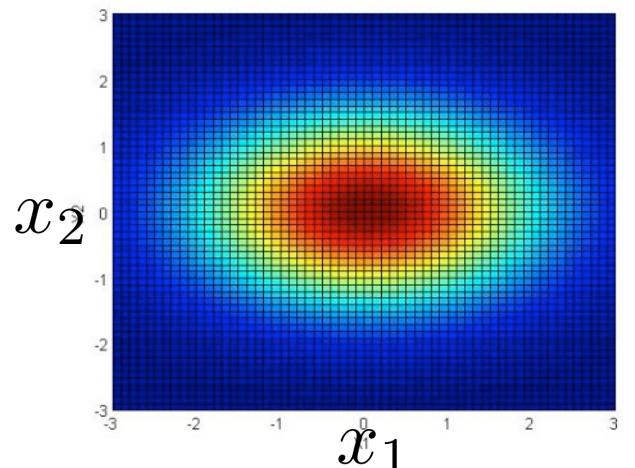
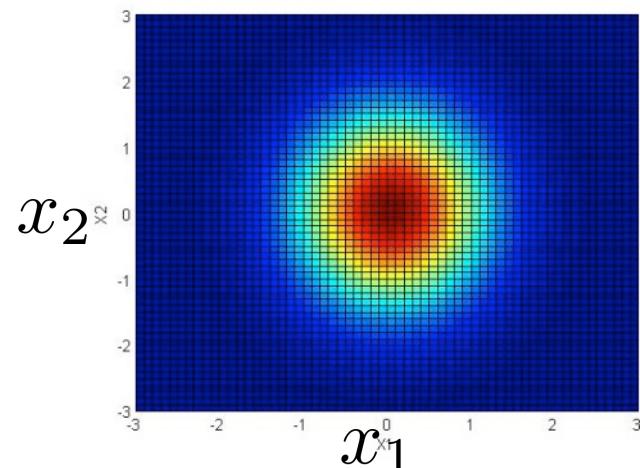
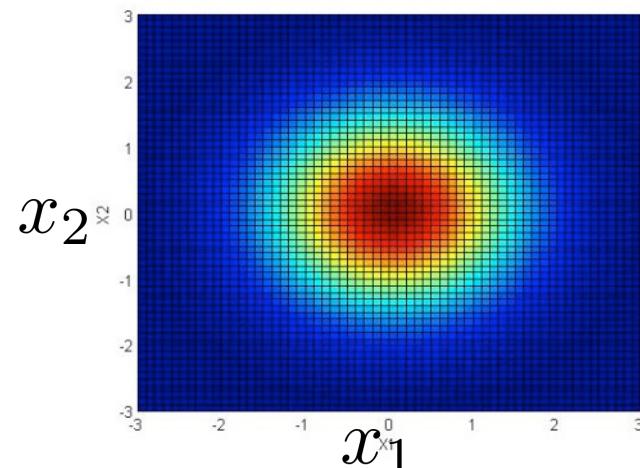
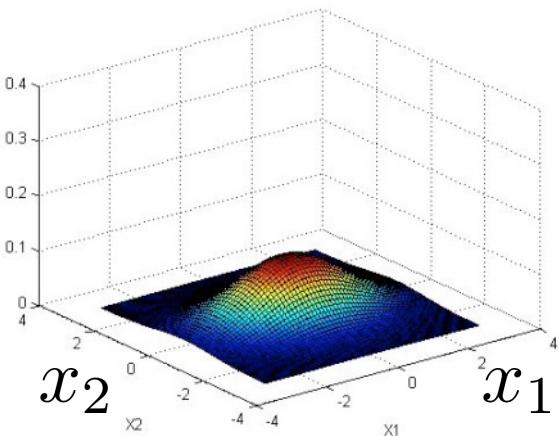
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$$

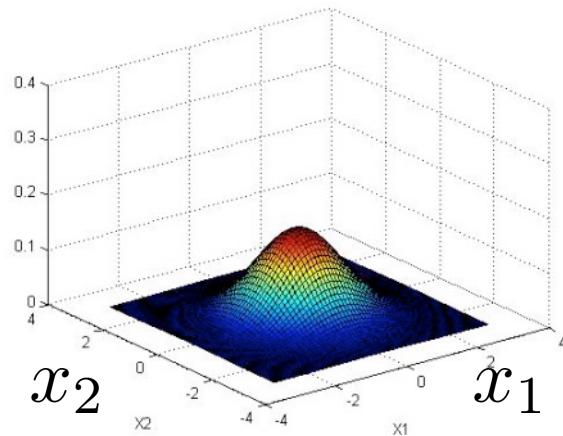


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

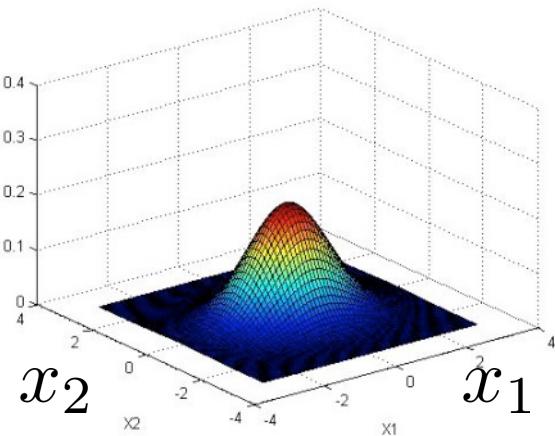


Multivariate Gaussian examples

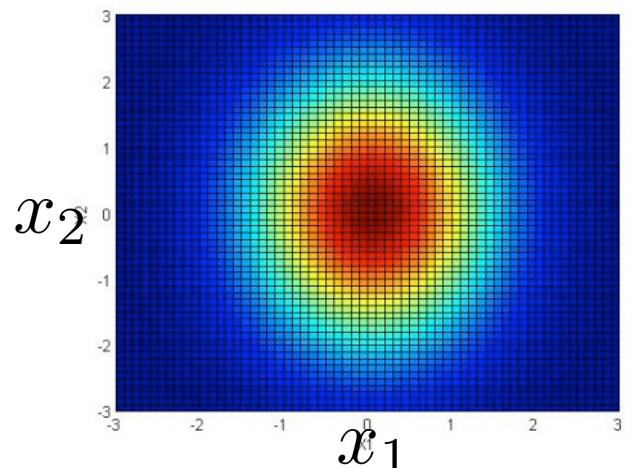
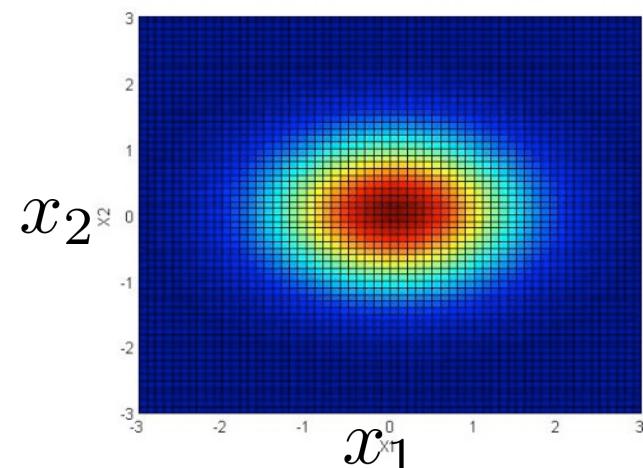
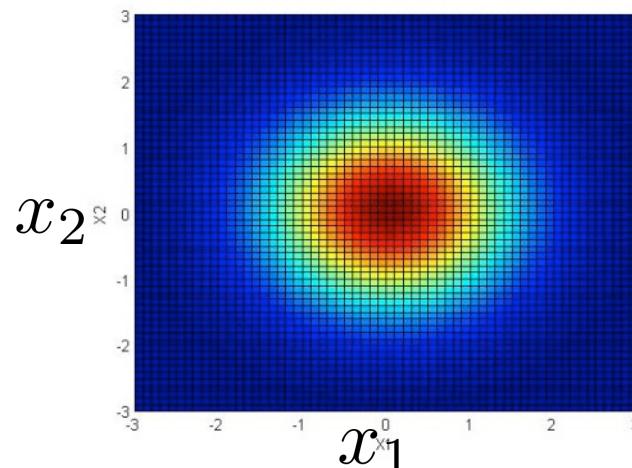
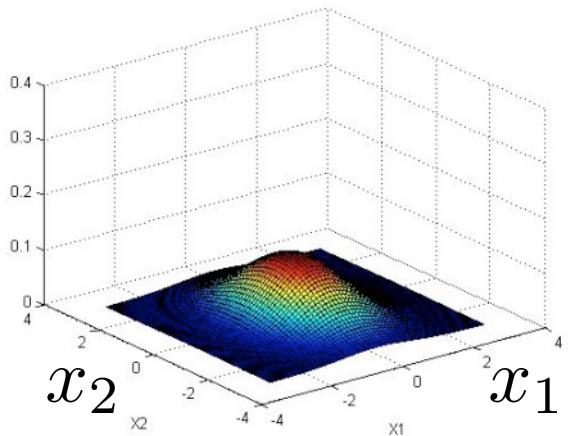
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$$

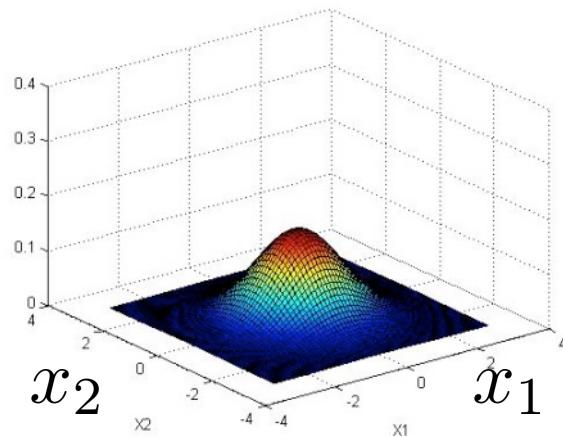


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

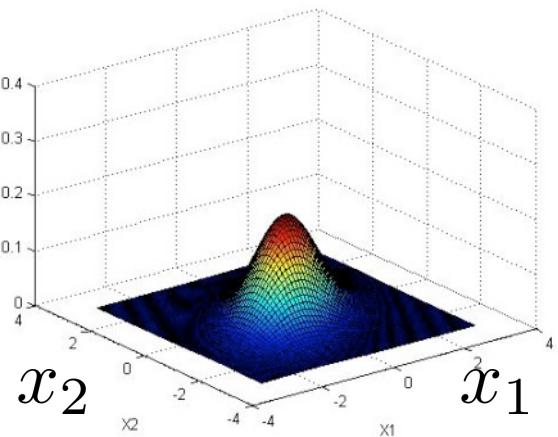


Multivariate Gaussian examples

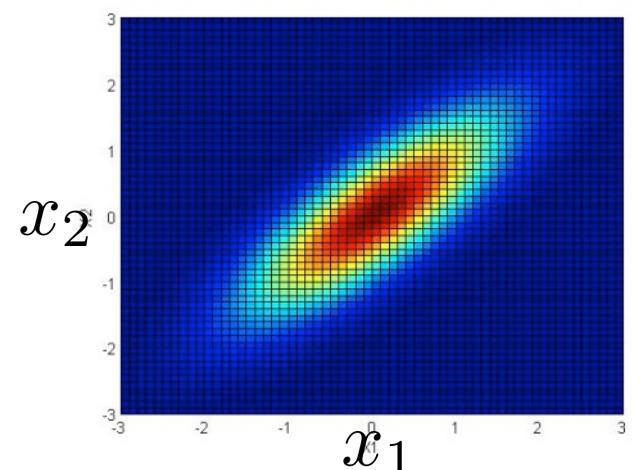
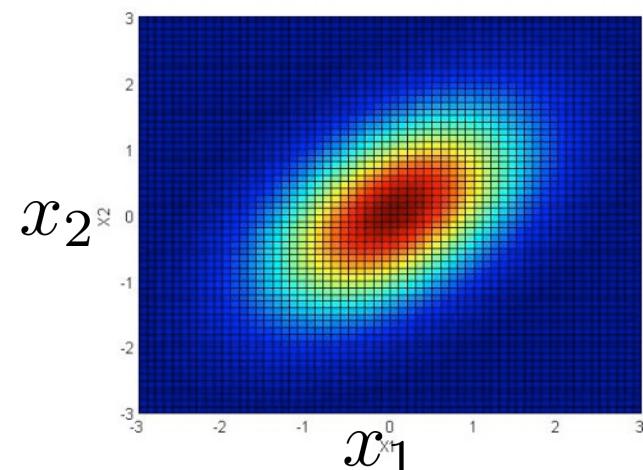
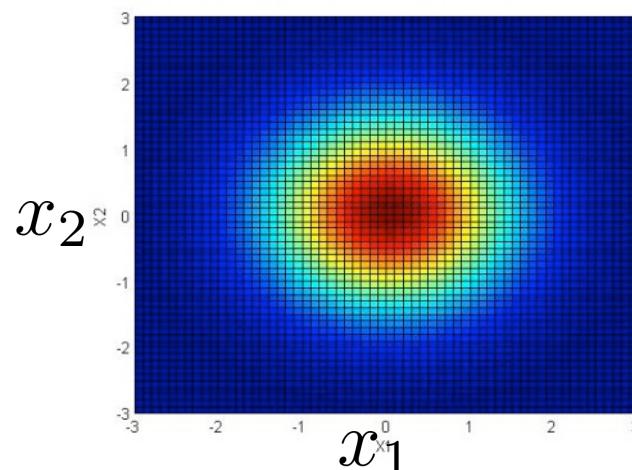
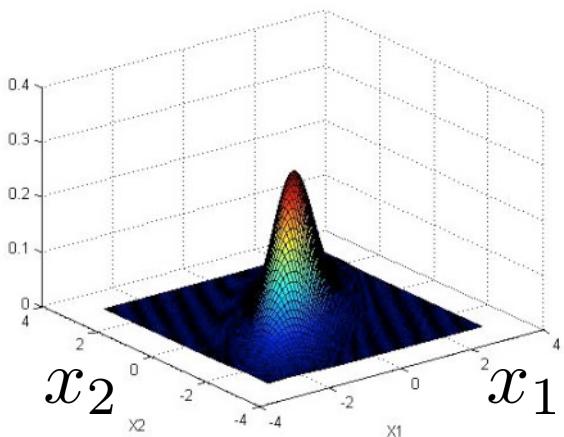
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

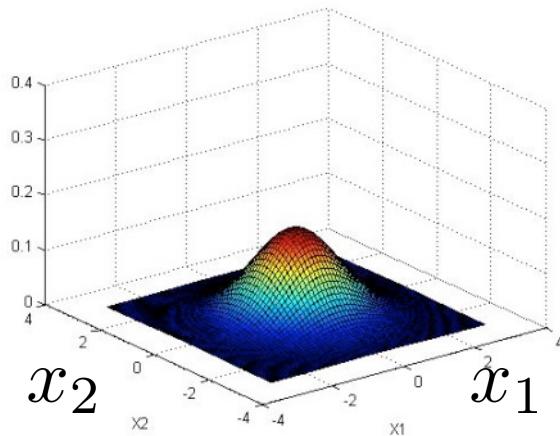


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

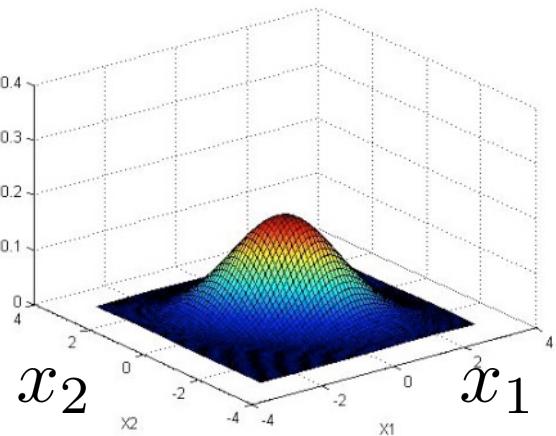


Multivariate Gaussian examples

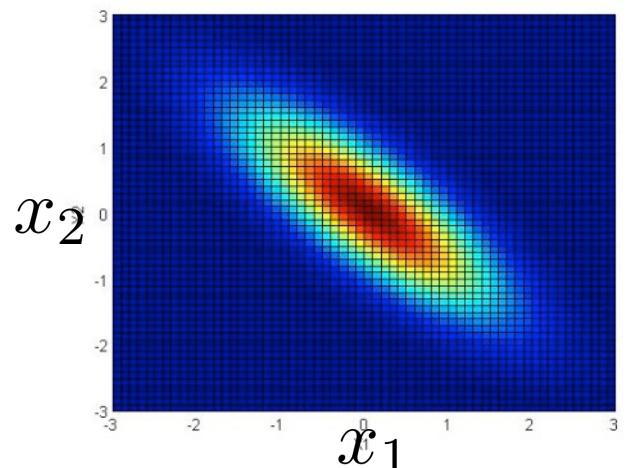
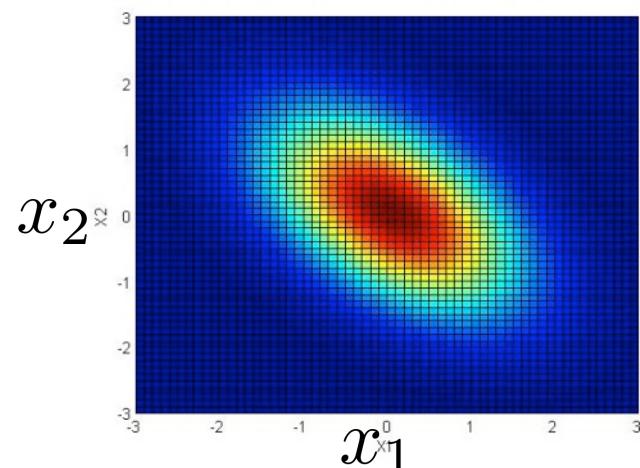
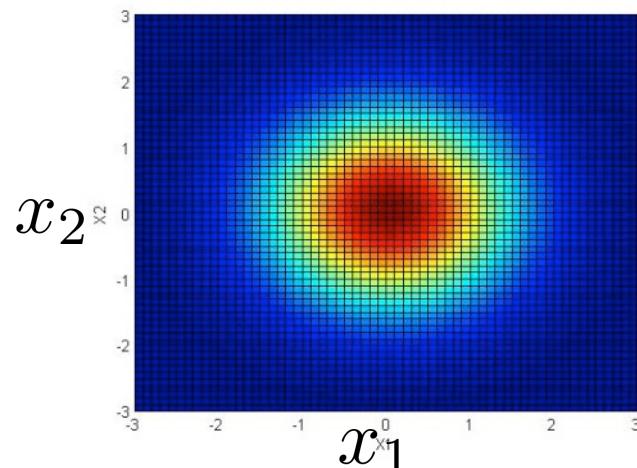
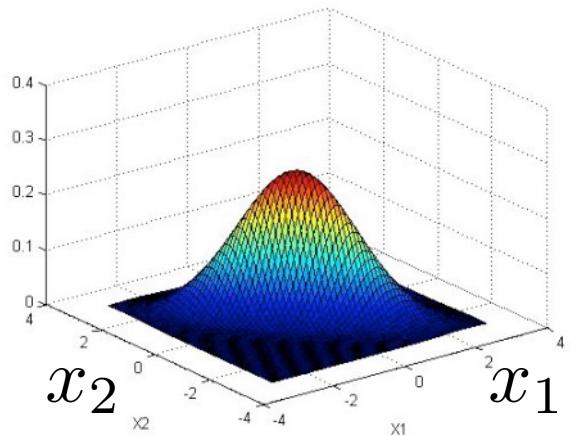
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

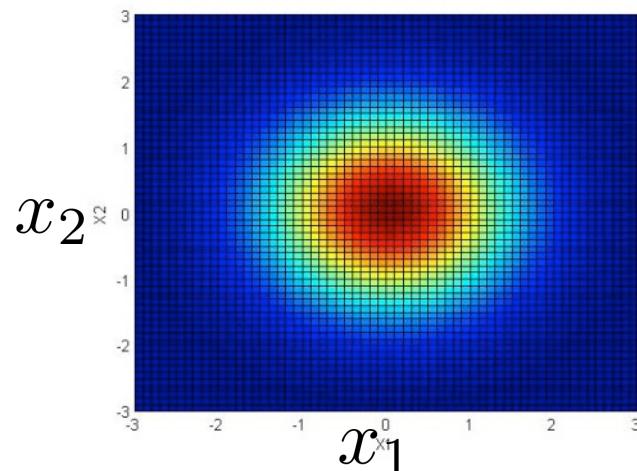
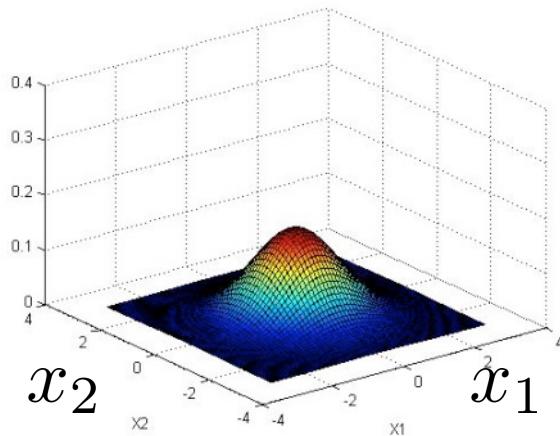


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$

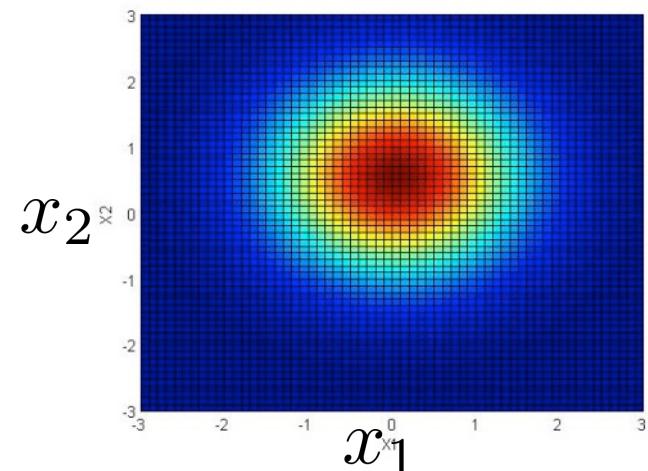
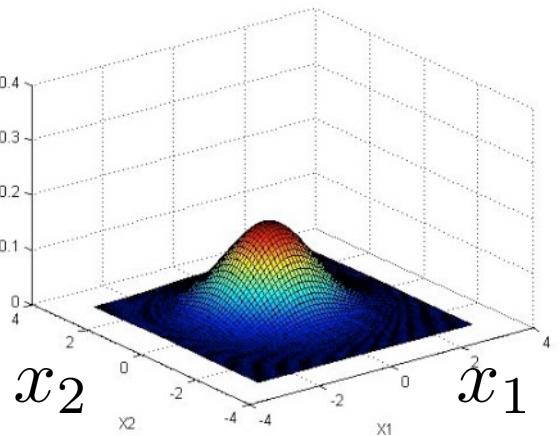


Multivariate Gaussian examples

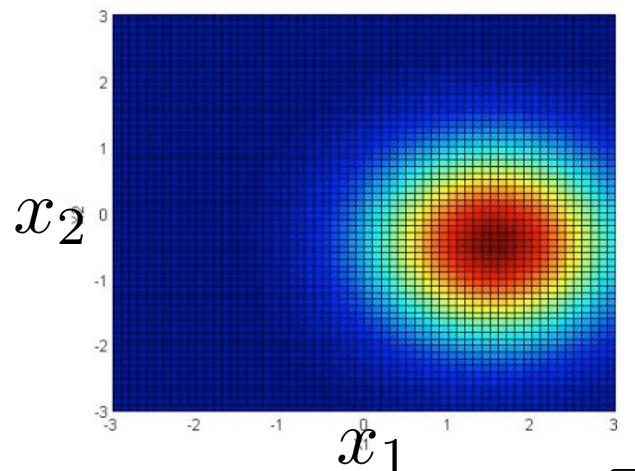
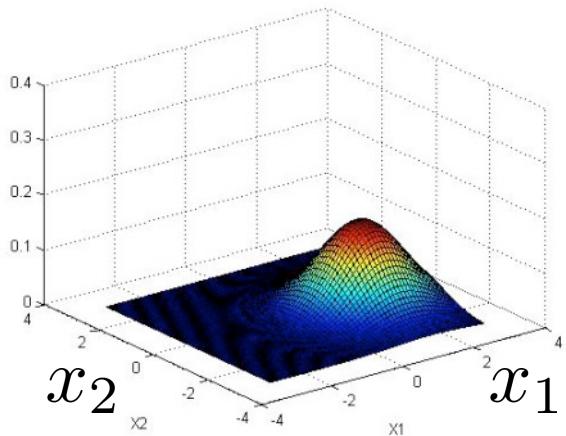
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



Univariate/Multivariate Gaussian

Univariate Gaussian

$$p(x_1; \mu_1, \sigma_1^2) \times \dots \times p(x_n; \mu_n, \sigma_n^2)$$

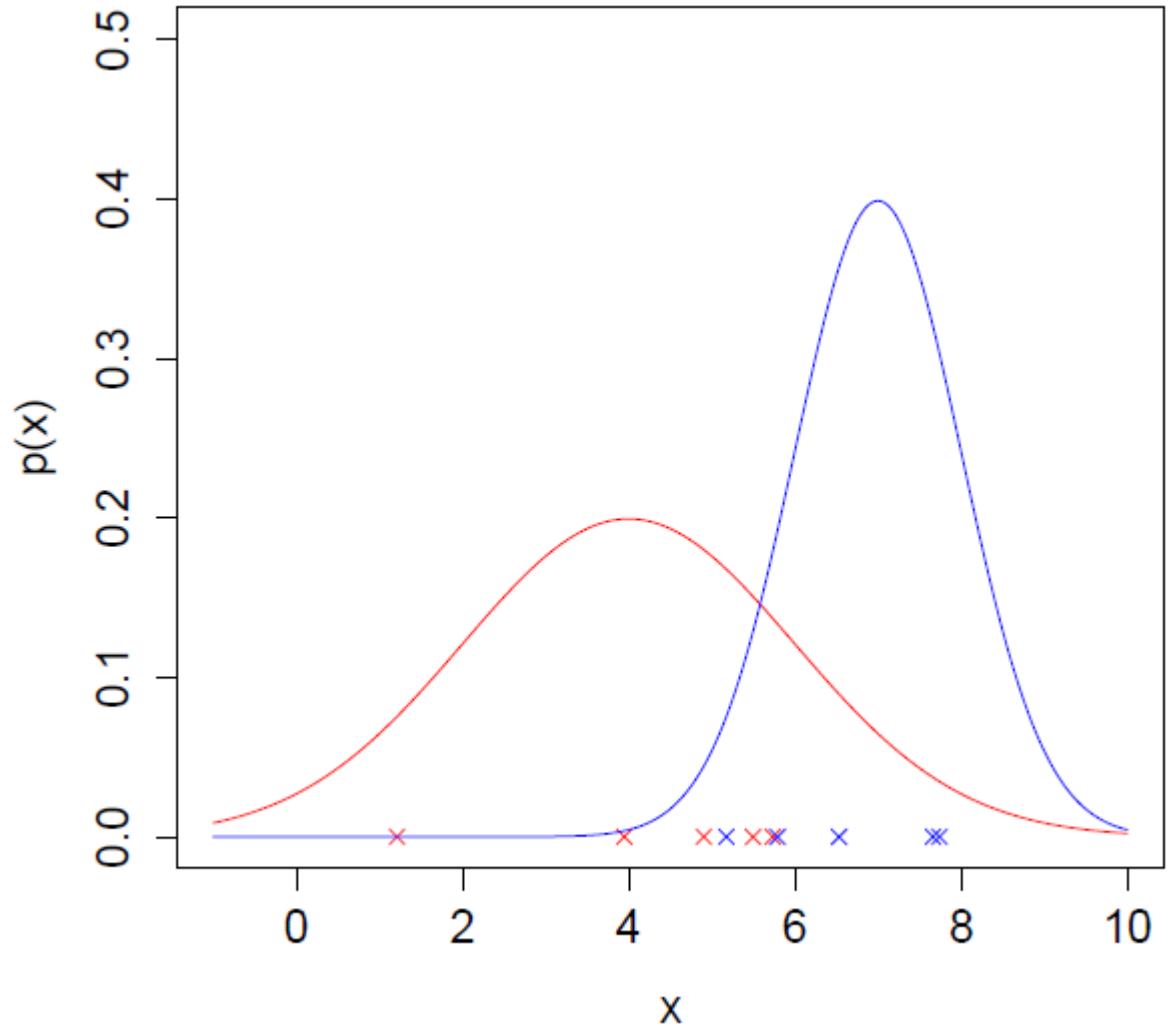
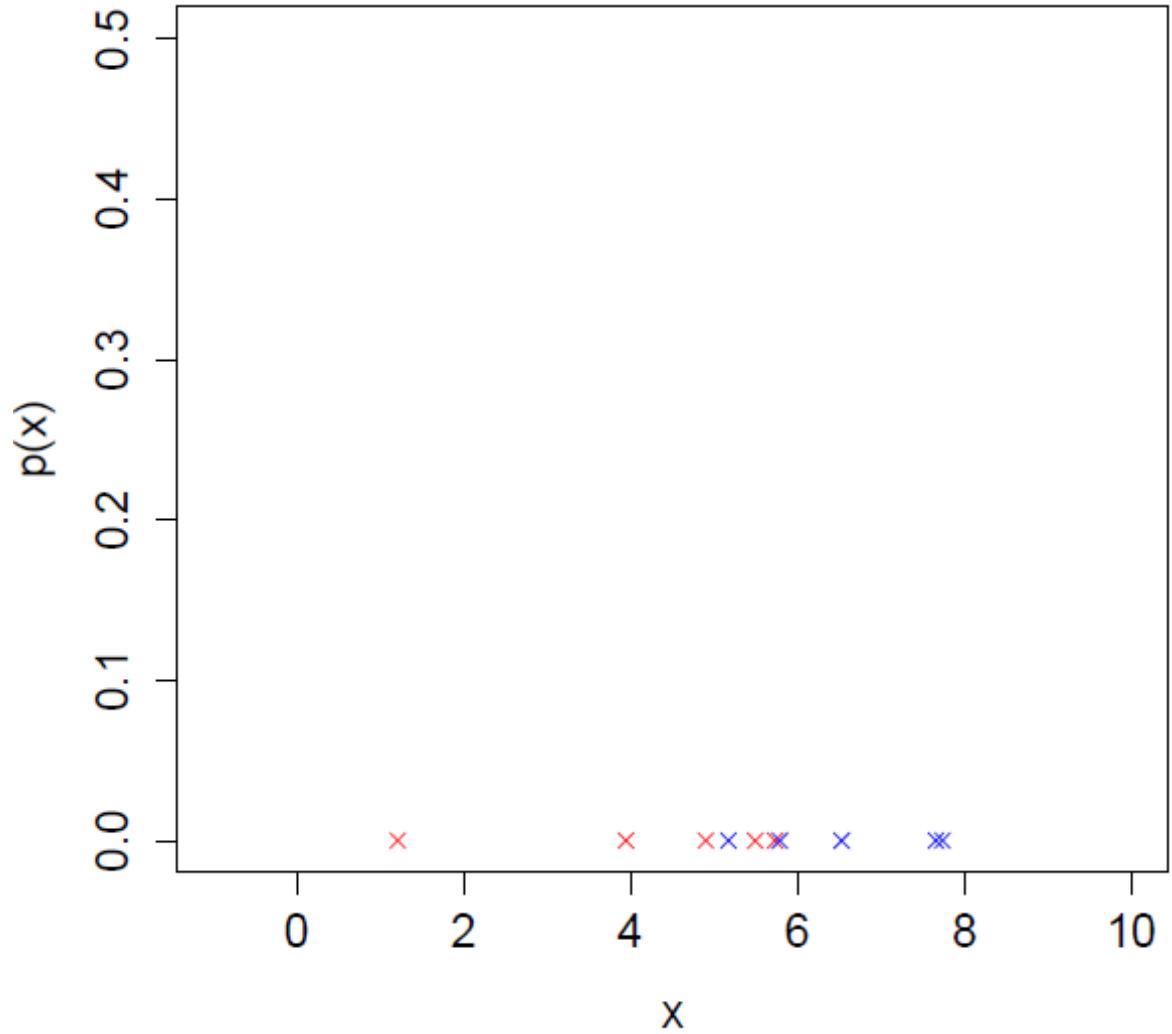
- You can manually create features with unusual combinations of values.
- Computationally cheaper
- Scales better for large n
- Works even if m is small

Multivariate Gaussian

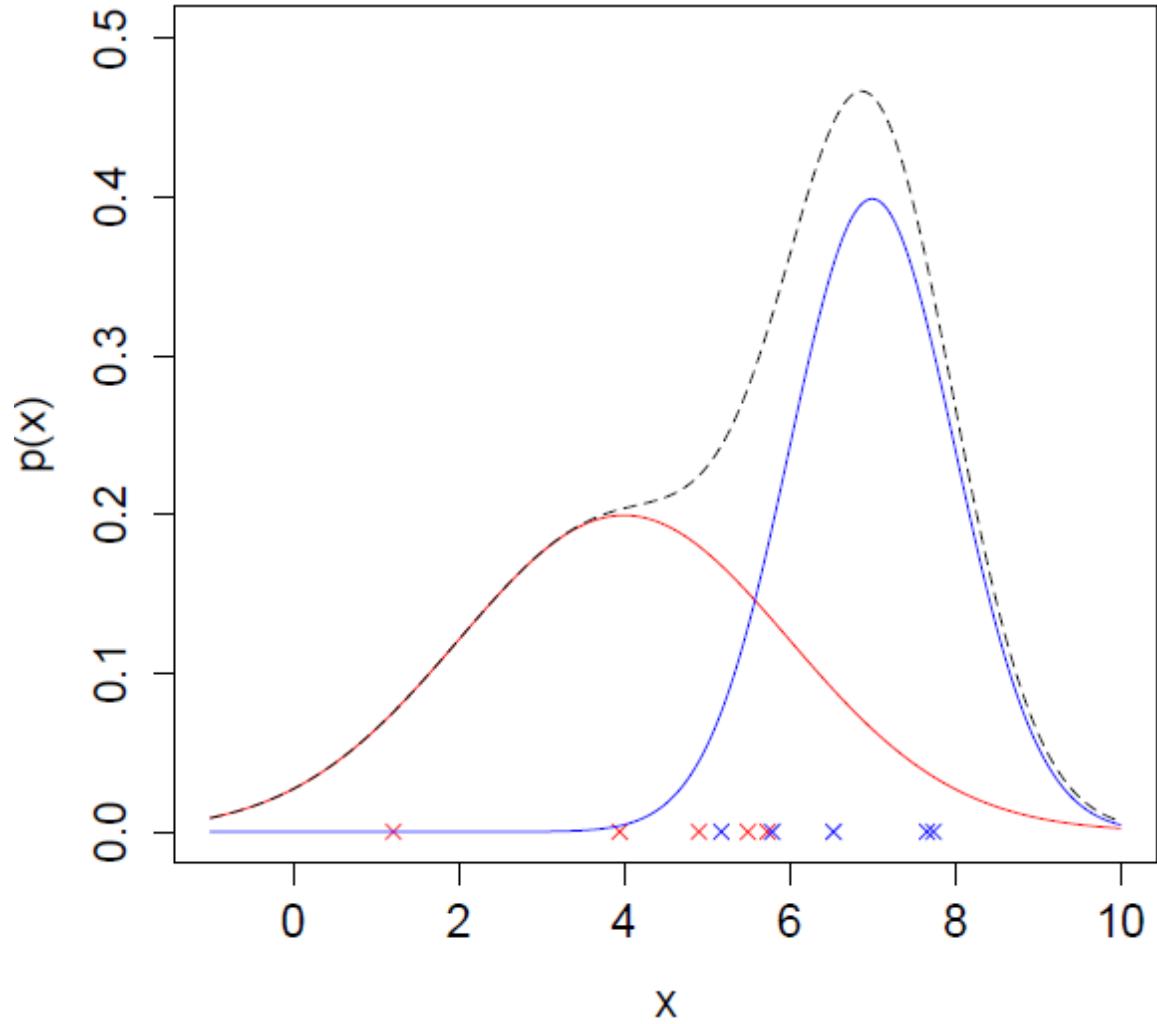
$$p(x; \mu, \Sigma) = \frac{e^{(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu))}}{\sqrt{(2\pi)^n |\Sigma|}}$$

- Automatically captures correlations between dimensions
- Computationally more expensive
- If $m \leq n$ then Σ is not invertible

Mixture Model



Mixture Model



Mixture Model

The probability of a point (sample) in the mixture is the sum of the probability (the marginal probability) of the same point in each k Gaussians.

$$p(\mathbf{x}) = \sum_{j=1}^k p(\mathbf{x} \wedge z = j).$$

The probability can be explicitly rewritten as

$$p(\mathbf{x}) = \sum_{j=1}^k p(\mathbf{x}|z = j)p(z = j).$$

Mixture Model

We can set:

- $p_j(x) = p(\mathbf{x}|z = j)$ the probability density of the j th gaussian in the point x
- $\pi_j = p(z = j)$ the probability of the j th gaussian among all gaussian, called mixing coefficient

We can rewrite:

$$p(\mathbf{x}) = \sum_{j=1}^k \pi_j p_j(\mathbf{x}).$$

Mixture Model

We can further make explicit the parameters of the gaussians (mean and variance), through a theta parameter.

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^k \pi_j p_j(\mathbf{x}|\boldsymbol{\theta}_j)$$

$\boldsymbol{\theta}_j$ represents the parameters of the jth gaussian

$\boldsymbol{\theta}$ represents the parameters of all gaussians

Mixture Model

A multivariate gaussian under the mixture perspective can be defined as:

$$p(x; \mu_j, \Sigma_j) = \frac{e^{(-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1} (x-\mu_j))}}{\sqrt{(2\pi)^n |\Sigma_j|}}$$

Where:

μ_j is the vector of the means of the jth gaussian

Σ_j is the covariance matrix of the jth gaussian

n is the number of the features

Mixture Model

What we want to know are the parameters of the Gaussians:

$$\boldsymbol{\mu}_j \text{ e } \boldsymbol{\Sigma}_j \quad j = 1, \dots, k.$$

Let us define C_j as the j th cluster, and $|C_j|$ the number of the points that belong to it. If we knew which point belongs to which cluster, we would compute the parameters as:

Univariate

$$\hat{\mu}_j = \frac{1}{|C_j|} \sum_{x^{(i)} \in C_j} x^{(i)}$$

$$\hat{\sigma}_j^2 = \frac{1}{|C_j| - 1} \sum_{x^{(i)} \in C_j} (x^{(i)} - \hat{\mu}_j)^2$$

Multivariate

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}^{(i)} \in C_j} \mathbf{x}^{(i)}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{1}{|C_j| - 1} \sum_{\mathbf{x}^{(i)} \in C_j} (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_j)(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_j)^T$$

Expectation-Maximization Algorithm (EM)

1. Randomly set the parameters (μ_j , Σ_j and π_j) for each of the K gaussians.
2. Repeat {
 1. **(E) Expectation step:** for each cluster-point pair the «responsibility» coefficient is computed. This coefficient gives an idea of how much the point is generated from the gaussian that is paired with it at the moment, w.r.t. the entire mixture.
 2. **(M) Maximization step:** the parameters (means, variances and mixing coefficient) are recomputed as a function of the responsibility coefficient to maximize the Likelihood.}
- until convergence

Expectation step

For each cluster-point pair, the «responsibility» coefficient is computed. This coefficient shows how much the point is generated from the Gaussian that is paired with it at the moment, w.r.t. the entire mixture.

$$r_{ij} := p(z = j | \mathbf{x}^{(i)}, \boldsymbol{\theta}^{t-1})$$

$\boldsymbol{\theta}_j^{(t-1)}$ are the parameters set in the previous iteration.

We can make the formula more explicit:

$$r_{ij} = \frac{\pi_j p_j(\mathbf{x}^{(i)} | \boldsymbol{\theta}_j^{(t-1)})}{\sum_{j=1}^k \pi_j p_j(\mathbf{x}^{(i)} | \boldsymbol{\theta}_j^{(t-1)})}$$

Maximization step

The parameters (means, variances and mixing coefficient) are recomputed as a function of the responsibility coefficient to maximize the likelihood.

$$\pi_j = \frac{1}{m} \sum_{i=1}^m r_{ij}$$

$$\mu_j = \frac{\sum_i r_{ij} \mathbf{x}^{(i)}}{\sum_i r_{ij}}$$

$$\Sigma_j = \frac{\sum_i r_{ij} (\mathbf{x}^{(i)} - \mu_j)(\mathbf{x}^{(i)} - \mu_j)^T}{\sum_i r_{ij}}$$

Gaussian Mixture Models – PROS/CONS

Pros

- It is a flexible algorithm that is able to approximate multivariate gaussian to fit data maximizing the likelihood
- It is not affected by bias approaching zero means
- Less sensible to outliers

Cons

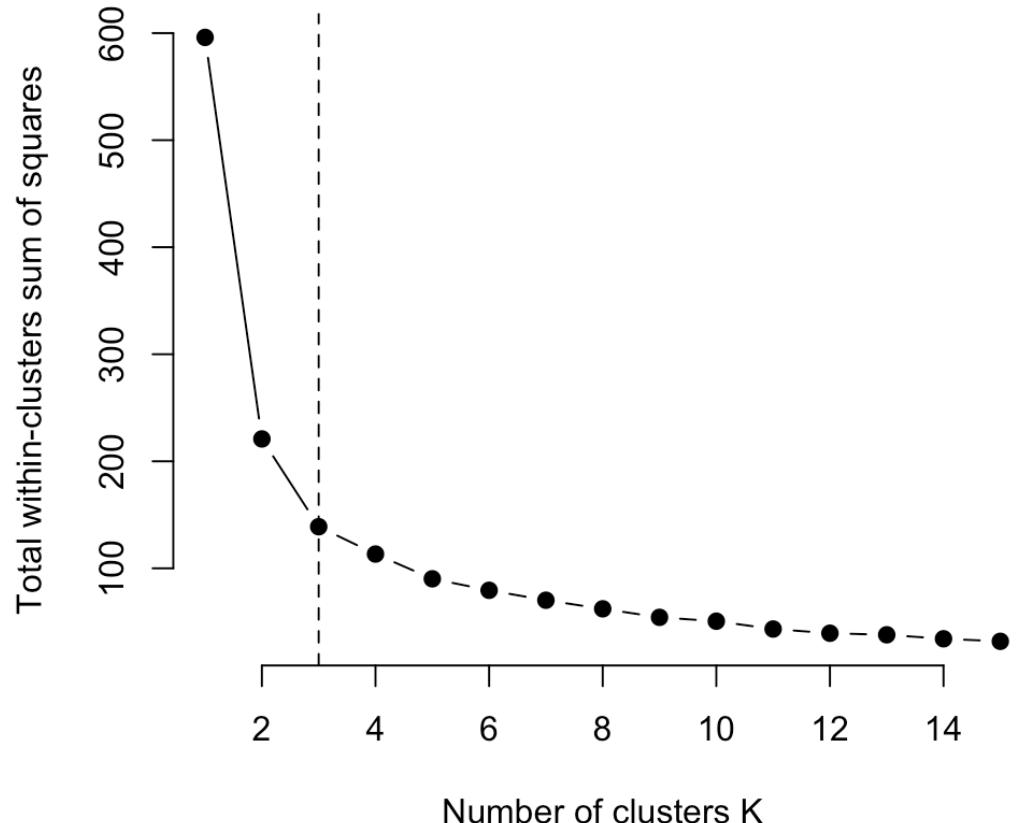
- The algorithm can diverge if there are only few points w.r.t. the overall mixture
- All the flexibility of the model is always used, even if it is not needed

Clustering

Choosing the value of K

Choosing the value of K

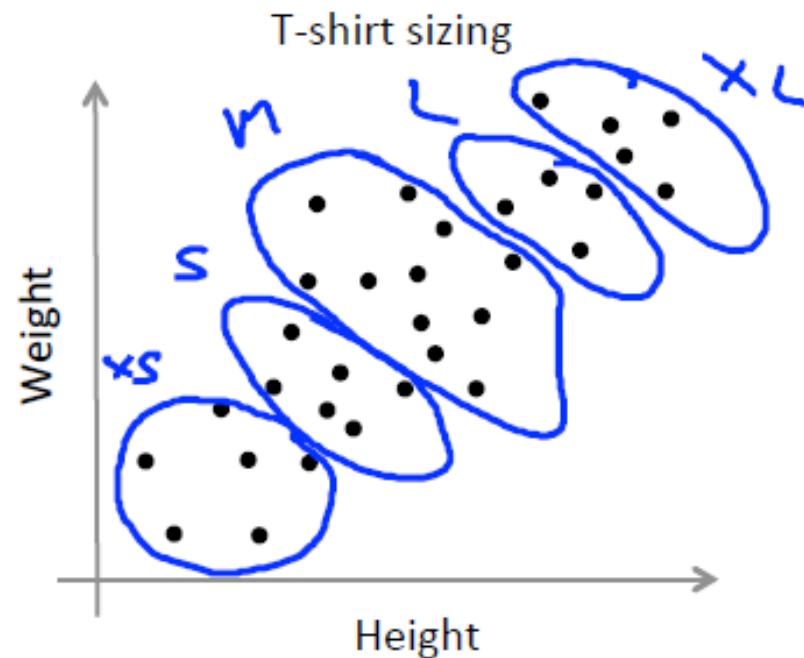
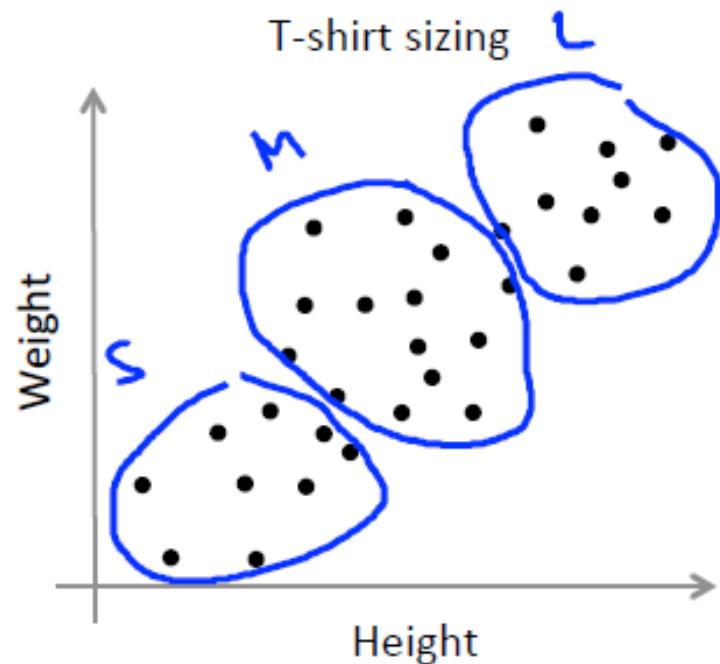
Elbow method:



Choosing the value of K

Sometimes our ML system must fit some external constraints, in those cases the number of clusters depends on how well the clusters performs on an external task.

E.g.



Kullback–Leibler divergence

It can be used to measures how the «true» probability distribution $P(i)$ diverges from the expected probability distribution, i.e.,the distribution of the model, $Q(i)$.

It can assume values from 0 (when the two distributions behave in the same way) to 1 (when their behaviour is completely different)

$$D_{KL}(P\|Q) = - \sum_i P(i) \log \frac{Q(i)}{P(i)} = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Information criterion approach

AIC: Akaike information criterion

BIC: Bayesian information criterion

DIC: Deviance information criterion

BF: Bayes factors

MDL: Minimum Description Length

FIA: Fisher Information Aproximation

Recap Likelihood $\hat{L} = L(\hat{\theta}) = \max_{\theta} L(\theta; X; y; M) = p(y|X; \hat{\theta}, M)$

Akaike information criterion (AIC)

AIC derived as asymptotic approximation of Kullback-Liebler information distance between the model of interest and the truth:

$$AIC = 2k - 2 \ln(\hat{L})$$

↑
number of parameters

It can also be used AICc, in which a correction related to the number of samples is injected:

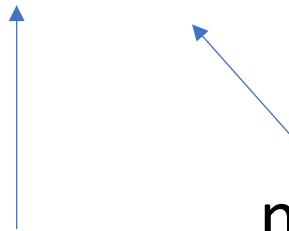
$$AICc = AIC + \frac{2k \cdot (k + 1)}{m - k - 1}$$

The model that minimizes AIC is selected

Bayesian information criterion

BIC is the asymptotic approximation under the assumption that the model can result in an exponential distribution.

$$BIC = \ln(m) \cdot k - 2 \ln(\hat{L})$$



number of samples
number of parameters

The model that minimizes BIC is selected

Deviance information criterion

It represents an asymptotic approximation of the deviance under the assumption that the posterior distribution is a multivariate normal

$$D(\theta) = -2 \log (p(y|X; \theta, M)) \quad \text{deviance}$$

$$D(\bar{\theta}) = \mathbb{E}[D(\theta)]$$

$$\tilde{\theta} = \mathbb{E}[\theta|y] \quad \text{posterior mean deviance}$$

$$p_D = \mathbb{E}_{\theta|y}[-2 \log (p(y|X; \theta, M)) + 2 \log (p(y|X; \tilde{\theta}, M))]$$

$$DIC = D(\bar{\theta}) + p_D$$

The model that minimizes DIC is selected

Information criterions comparison

- AIC lacks the finite samples correction in its base form
- AIC is asymptotically equivalent to k-folds cross-validation on linear regression models with a least squares optimization
- BIC is able to detect the true model (if present in the comparison)
- BIC is only valid if n is much greater than k
- BIC penalizes complex models more than AIC
- DIC tends to suggest overfitted models

Silhouette Coefficient

Silhouette Coefficient combines the ideas of cohesion and separation for single samples, clusters, and clusterings

For an individual sample, $x^{(i)}$

- coh = average distance of $x^{(i)}$ to the samples in the same cluster
- sep = min (average distance of $x^{(i)}$ to samples in another cluster)
- silhouette coefficient of $x^{(i)}$:

$$s_i = 1 - \frac{coh}{sep} \text{ if } coh < sep$$

The closer to 1 the better.

Can calculate the Average Silhouette width for a cluster or a clustering

Clustering

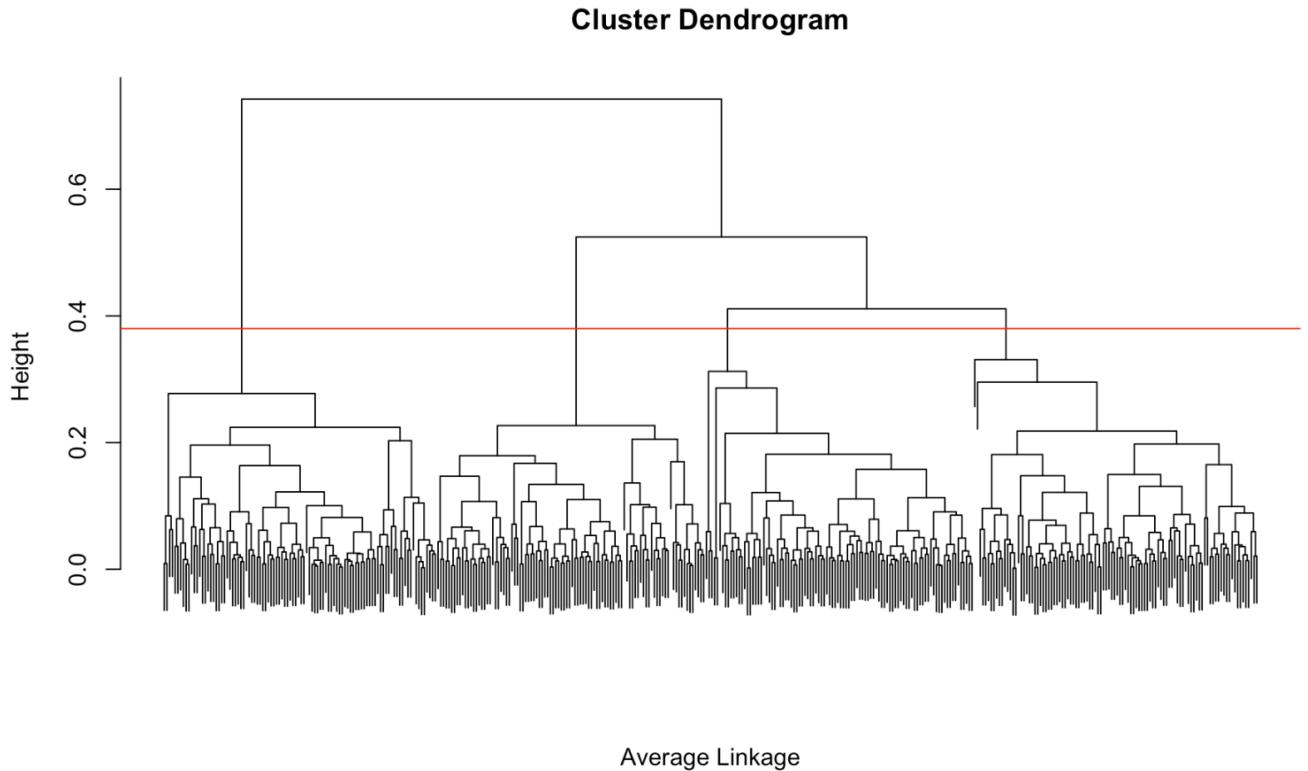
Hierarchical Clustering

Hierarchical Clustering

Hierarchical cluster involves building a hierarchy of (potentially overlapping) clusters.

- Agglomerative (“bottom-up”)
- Divisive (“top-down”)

Once the hierarchy is built,
cluster extraction is done
by selecting some height to ‘cut’



Hierarchical Clustering

Deciding how to *merge* or *split* clusters depends on 1) the **similarity measure/metric** and 2) the **linkage criterion**

1. Euclidean Distance, Mahalanobis distance, etc.
2. Could be:
 - **Distance based:** Max/min/mean distance (complete/single/ average linkage)
 - **Probability based:** the probability that the candidate clusters come from the same distribution
 - The product of the **in-and-out degree** of a node (graph degree linkage)
 - **Instance-based constraints** (see Wagstaff 2002)

Hierarchical Clustering: Single Linkage

Perhaps the most well-known hierarchical clustering algorithm is called the **Single-Linkage algorithm**

The idea:

1. Start with every single point in its own cluster (n clusters)
2. Merge the two closest points to form a new cluster
3. Repeat: each time *merging the two closest pair of points*

The configuration:

- The **similarity metric** is **Euclidean distance**
- The **linkage criterion** is the [pairwise] minimum distance

Hierarchical Clustering: Wards Criterion

Rather than use distance metrics as measures of similarity, why not merge based on a more statistically intuitive notion

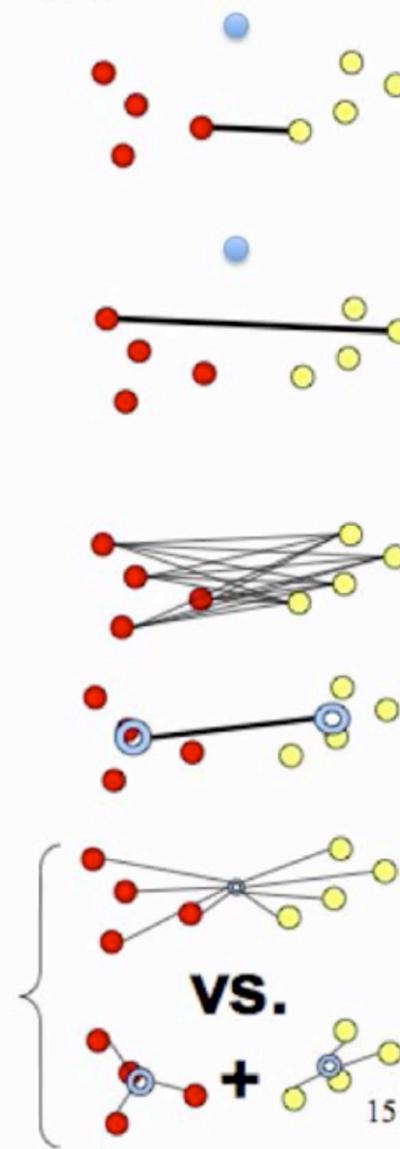
Perhaps its fair to assume observations come from an approximately elliptically-shaped distribution

Ward proposed agglomeration where observations are merged in such a way that minimizes the total squared error (maximizes r^2 at each merge)

“...Interpreted as the proportion of variation explained by a particular clustering of the observations.”

Cluster distance measures

- Single link: $D(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$
 - distance between closest elements in clusters
 - produces long chains a→b→c→...→z
- Complete link: $D(c_1, c_2) = \max_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$
 - distance between farthest elements in clusters
 - forces “spherical” clusters with consistent “diameter”
- Average link: $D(c_1, c_2) = \frac{1}{|c_1|} \frac{1}{|c_2|} \sum_{x_1 \in c_1} \sum_{x_2 \in c_2} D(x_1, x_2)$
 - average of all pairwise distances
 - less affected by outliers
- Centroids: $D(c_1, c_2) = D\left(\left(\frac{1}{|c_1|} \sum_{x \in c_1} \vec{x}\right), \left(\frac{1}{|c_2|} \sum_{x \in c_2} \vec{x}\right)\right)$
 - distance between centroids (means) of two clusters
- Ward's method: $TD_{c_1 \cup c_2} = \sum_{x \in c_1 \cup c_2} D(x, \mu_{c_1 \cup c_2})^2$
 - consider joining two clusters, how does it change the total distance (TD) from centroids?



Clustering

DBSCAN

DBSCAN

DBSCAN stands for **Density-Based Spatial Clustering of Applications with Noise**

DBSCAN is a **density-based agglomerative** algorithm

Density-based Clustering locates regions of high density that are separated from one another by regions of low density

In this case **Density** is defined as the number of points within a specified radius (**Eps**)

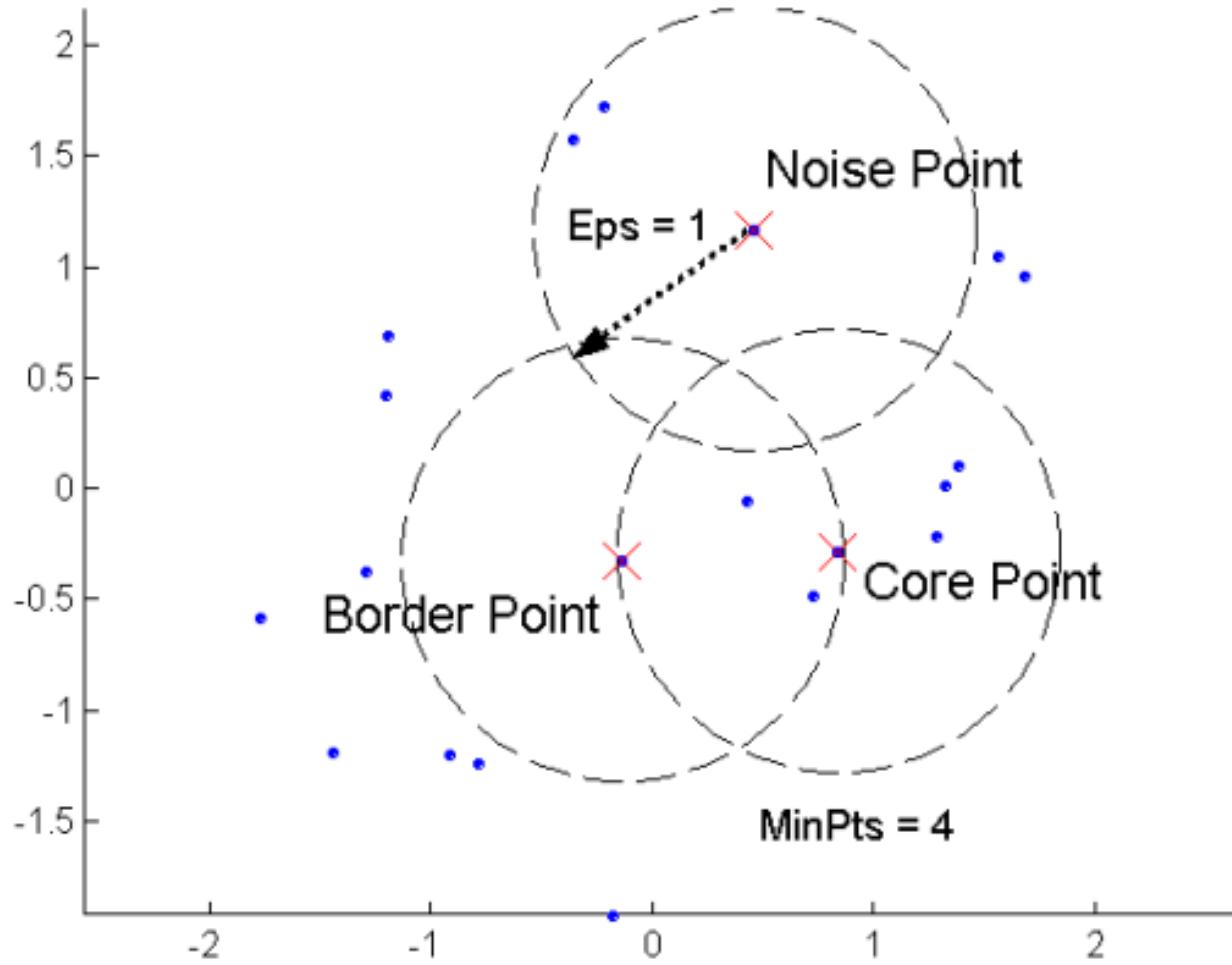
DBSCAN concepts

- A point is a **core point** if it has more than a specified number of points (MinPts) within Eps
 - These are points that are at the interior of a cluster
- A **border point** has fewer than MinPts within Eps, but is in the neighborhood of a core point
- A **noise point** is any point that is not a core point or a border point
- **Eps-Neighborhood** are the points within a radius of Eps from a point

DBSCAN concepts

- Any two core points are close enough, within a distance Eps of one another, are put in the same cluster
- Any border point that is close enough to a core point is put in the same cluster as the core point
- Noise points are discarded

DBSCAN



DBSCAN

We have to set two different parameters:

Epsilon := physical distance or radius

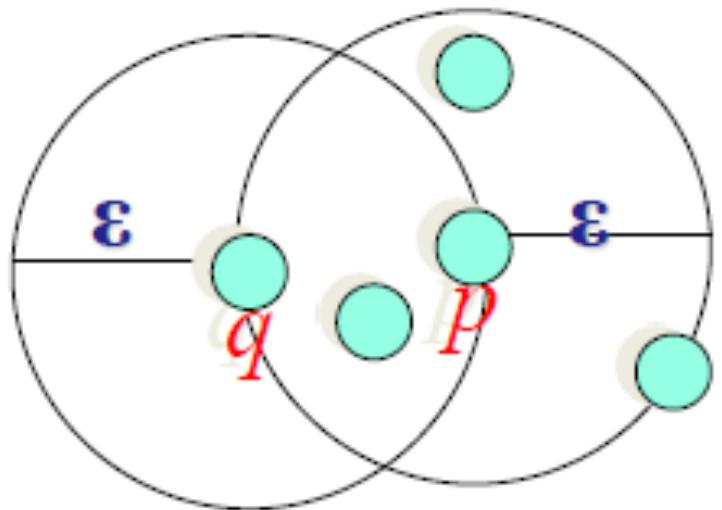
minPts := desired minimum cluster size

- Epsilon can be chosen using a k-distance graph
- minPts can be chosen with the empirical rule:
 $\text{minPts} \geq (\text{number of Dimensions}) + 1$

DBSCAN Reachability

Directly density-reachable

An object q is directly density-reachable from object p if q is within the ϵ -Neighborhood of p and p is a core object.

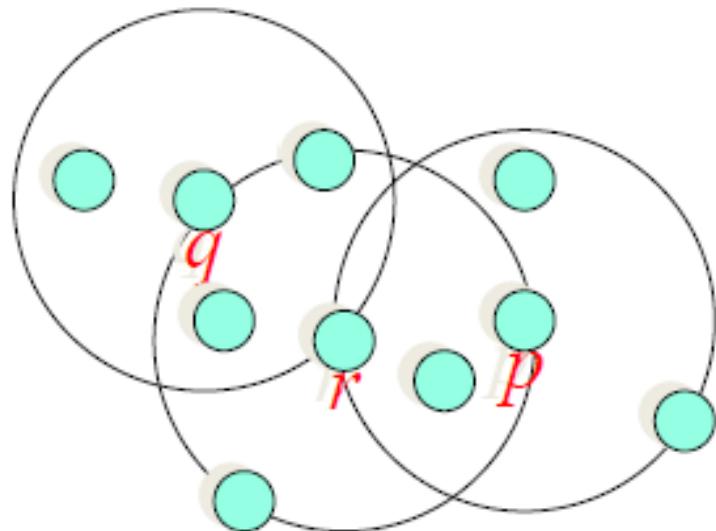


q is directly density-reachable from p
p is not directly density-reachable from q

DBSCAN Connectivity

Density-connectivity

Object p is density-connected to object q w.r.t ϵ and $MinPts$ if there is an object r such that both p and q are density-reachable from r w.r.t ϵ and $MinPts$



p and q are density-connected to each other by r
Density-connectivity is symmetric

DBSCAN Algorithm

CorePoints =[]

NoisePoints = []

For $x^{(i)}$ in X:

 Retrieve all points density-reachable from $x^{(i)}$

 if $x^{(i)}$ is a core point, a cluster is formed.

 CorePoints.push($x^{(i)}$)

For $x^{(i)}$ in CorePoints :

 Retrieve all the connected components.

 Merge the clusters

For $x^{(i)}$ in X/CorePoints :

 if distance from the nearest cluster $< \epsilon$:

 Assign $x^{(i)}$ to that cluster

 else

 NoisePoints.push($x^{(i)}$)

DBSCAN Pros

- DBSCAN does not require one to specify the number of clusters in the data *a priori*, as opposed to k-means.
- DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced.
- DBSCAN has a notion of noise, and is robust to outliers.
- DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database.
- DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.
- The parameters minPts and ϵ can be set by a domain expert, if the data is well understood.

DBSCAN Cons

- DBSCAN is not entirely deterministic: border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data is processed.
- The quality of DBSCAN depends on the distance measure. The most common distance metric used is Euclidean distance. Especially for high-dimensional data, this metric can be rendered almost useless due to the so-called “Curse of dimensionality”
- DBSCAN cannot cluster data sets well with large differences in densities, since the minPts- ε combination cannot then be chosen appropriately for all clusters.
- If the data and scale are not well understood, choosing a meaningful distance threshold ε can be difficult.

DBSCAN Complexity

- **Time Complexity:** $O(n^2)$
 - for each point it has to be determined if it is a core point.
 - can be reduced to $O(n * \log(n))$ in lower dimensional spaces by using efficient data structures (n is the number of objects to be clustered);
- **Space Complexity:** $O(n)$.

Clustering

HDBSCAN

Hierarchical DBSCAN

- Shown that DBSCAN corresponds *exactly* with cutting the Robust Single Linkage tree at height Eps
- ‘Flat’ extraction method proposed based on stability-based cluster validation methods

Hierarchical DBSCAN

1. Transform the space according to the density/sparsity.
2. Build the minimum spanning tree of the distance weighted graph.
3. Construct a cluster hierarchy of connected components.
4. Condense the cluster hierarchy based on minimum cluster size.
5. Extract the stable clusters from the condensed tree.

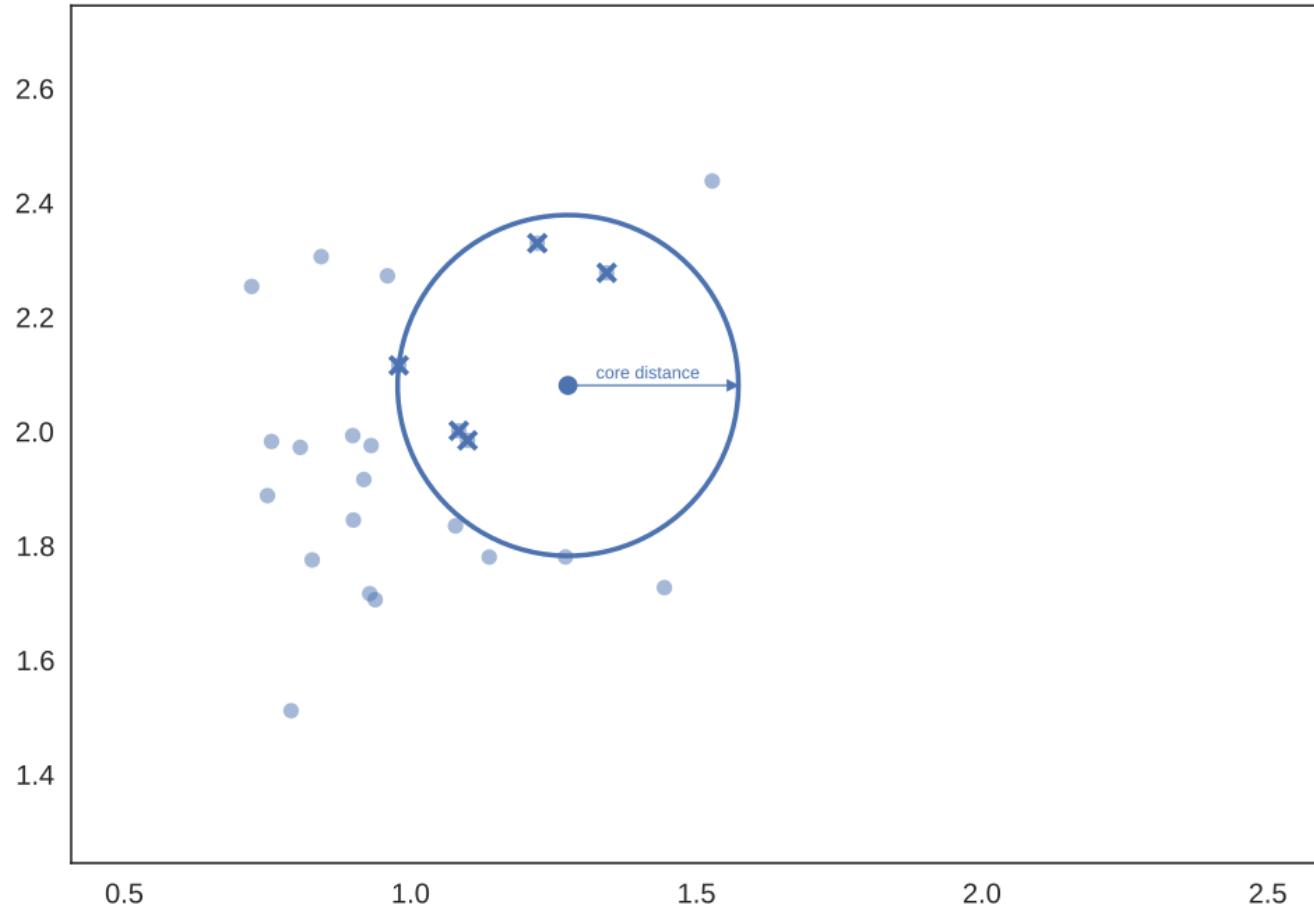
Mutual reachability distance

- First we define the core distance, $\text{core}_k(x)$ as the distance from x to the $k - th$ nearest neighbor
- **NOTE:** Points in denser regions would have smaller core distances while points in sparser regions would have larger core distances.
- Core distance is what makes these methods “density-based”.
- $d(a, b)$ is the distance between two different samples

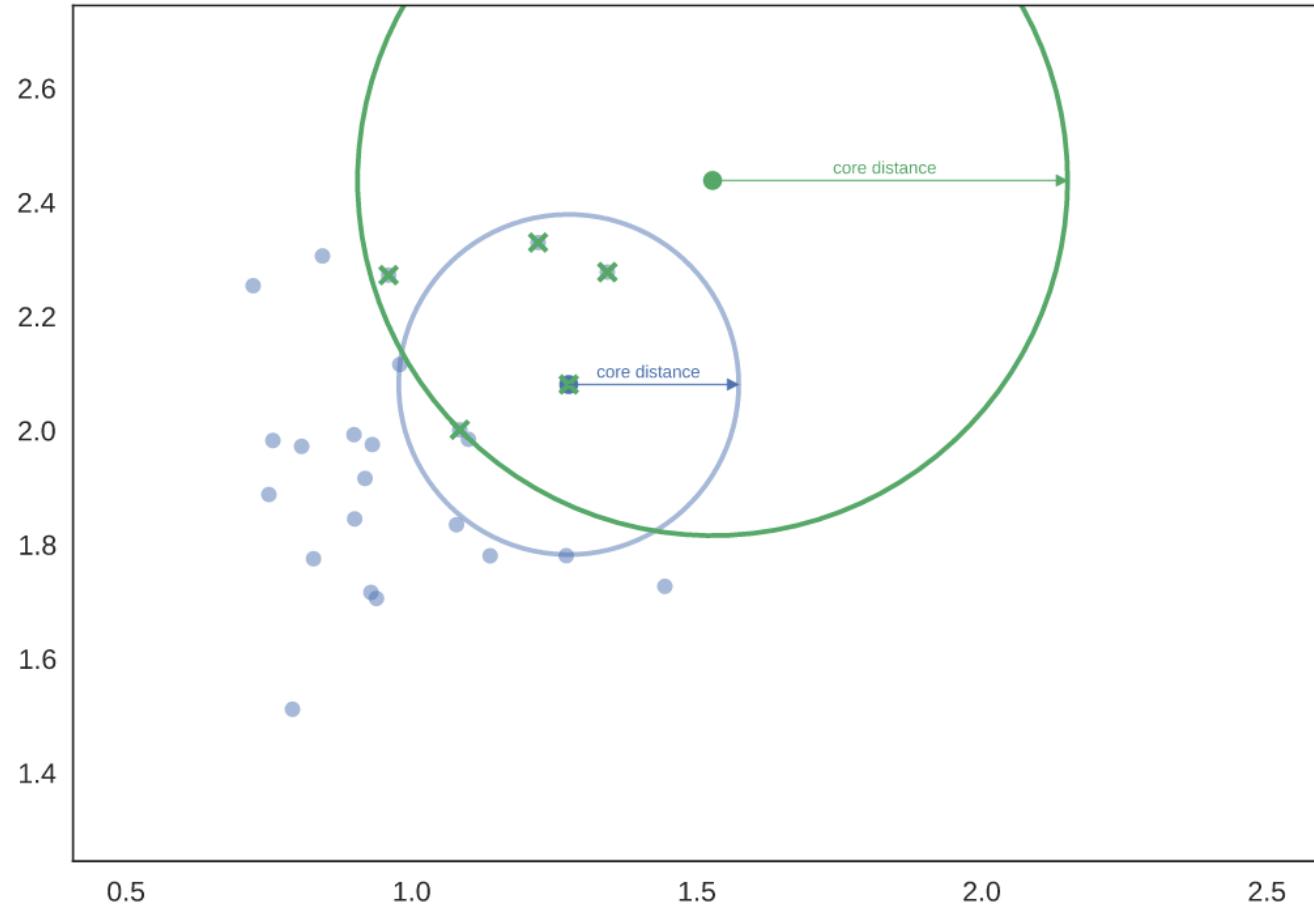
The mutual reachability distance is finally defined as:

$$d_{mreach-k}(a, b) = \max\{\text{core}_k(a), \text{core}_k(b), d(a, b)\}$$

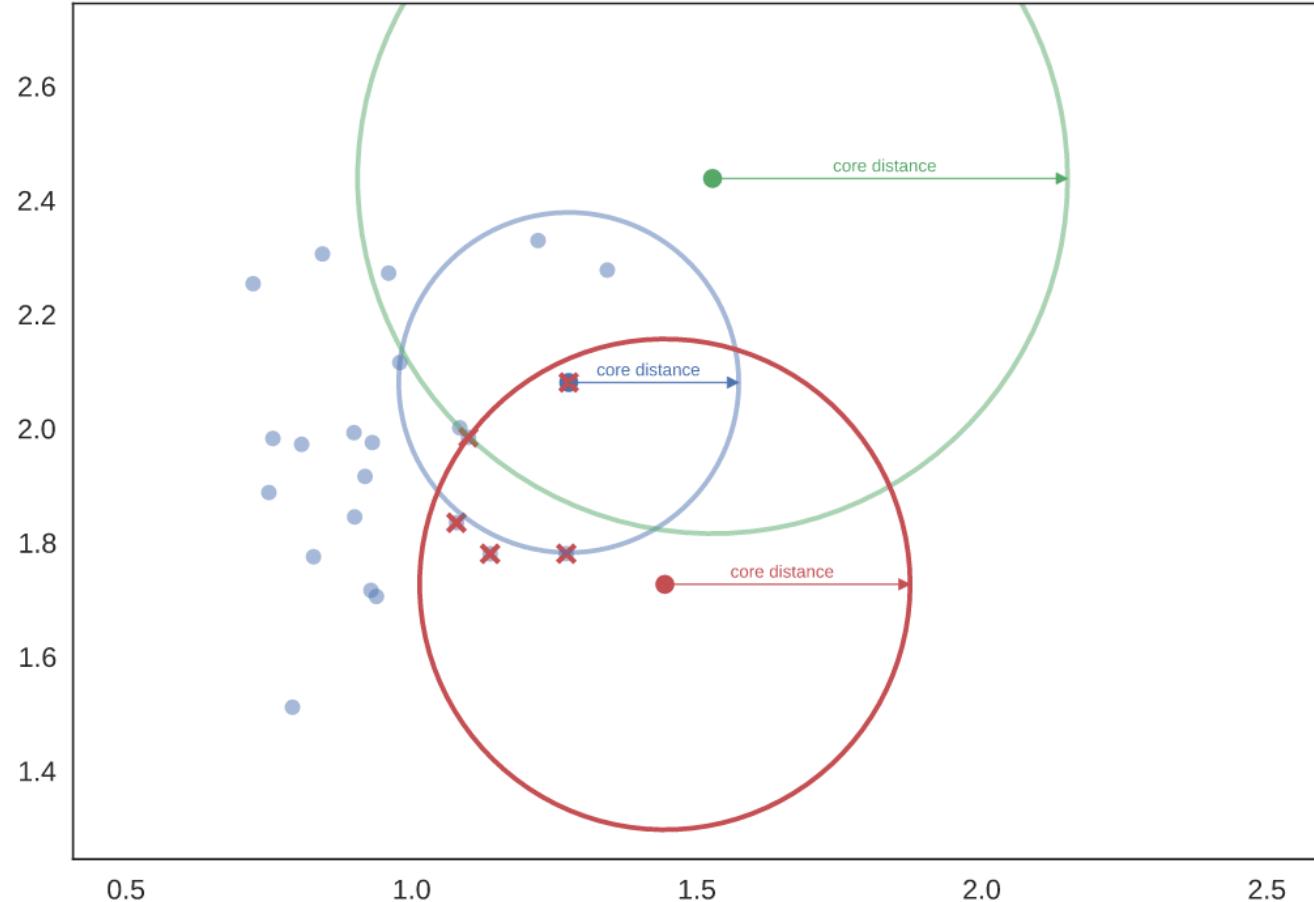
Mutual reachability distance



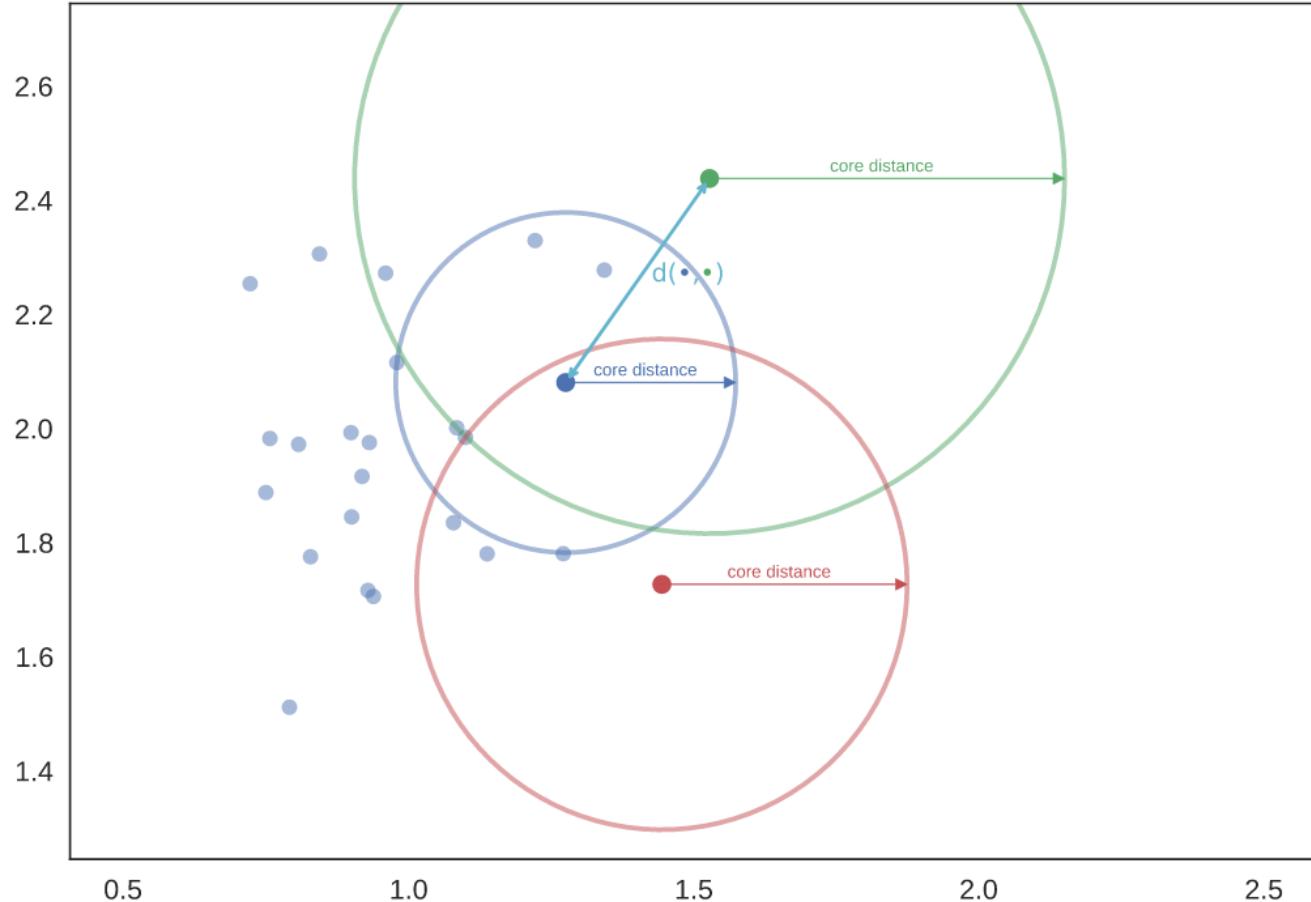
Mutual reachability distance



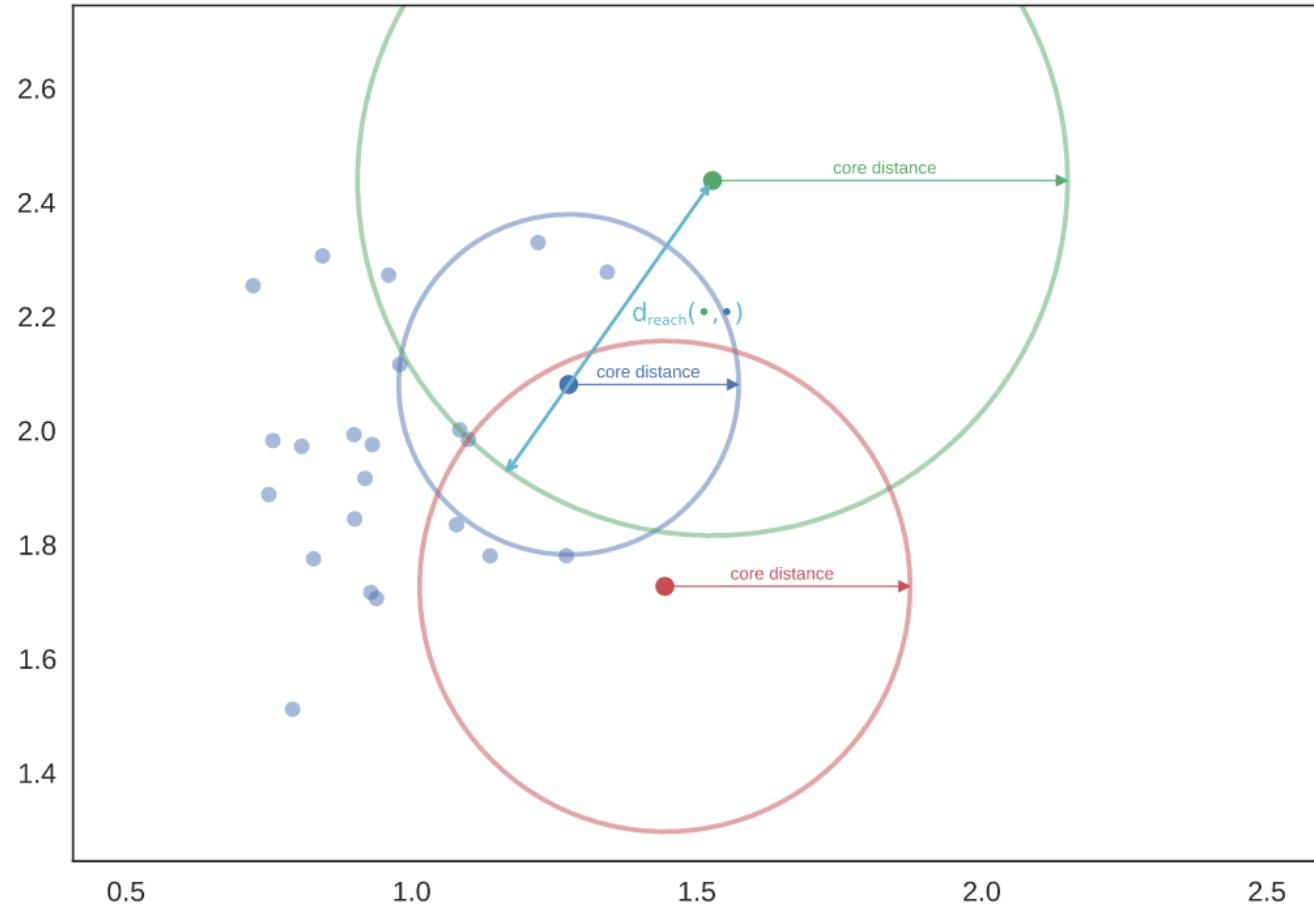
Mutual reachability distance



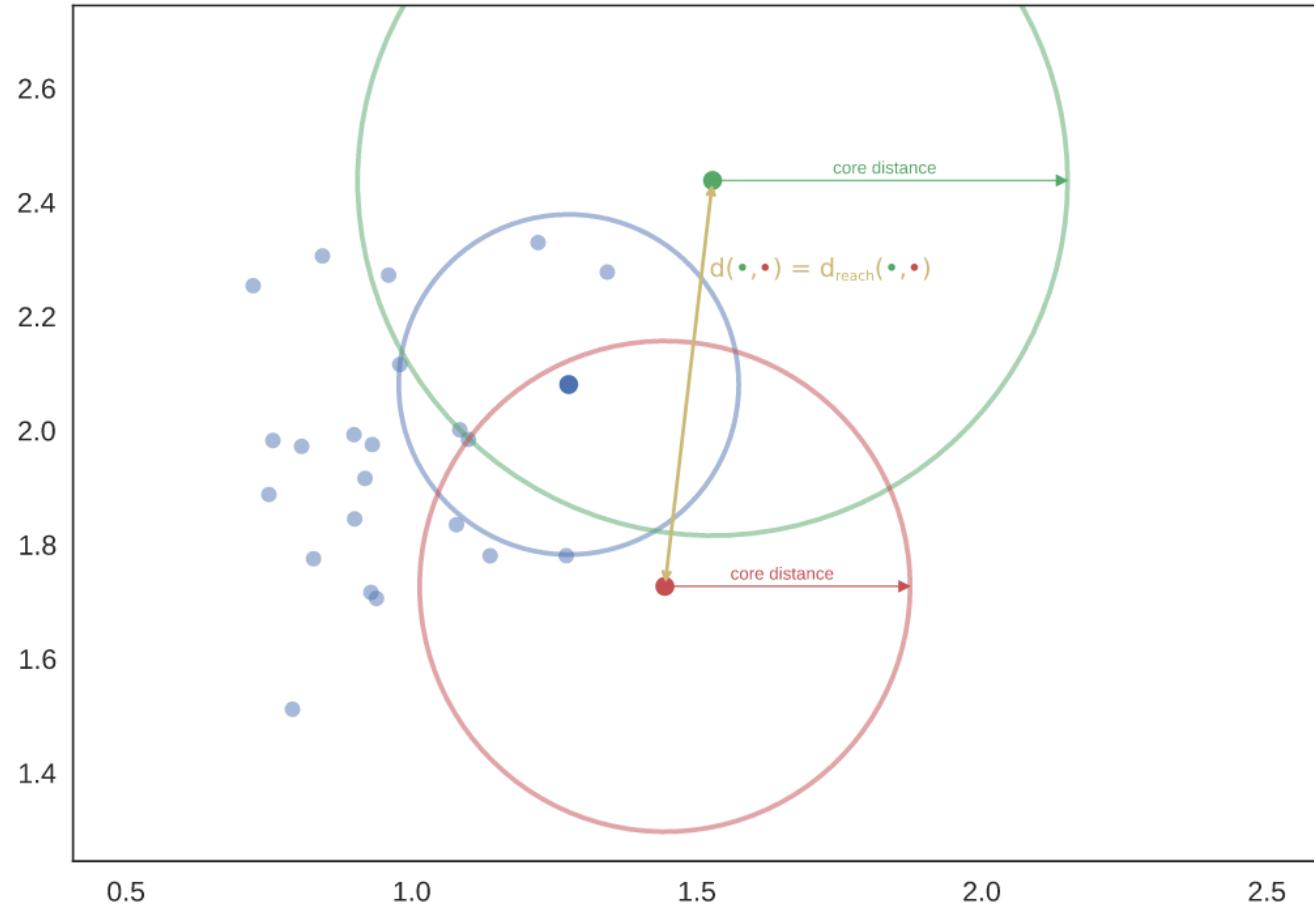
Mutual reachability distance



Mutual reachability distance



Mutual reachability distance



Build the minimum spanning tree

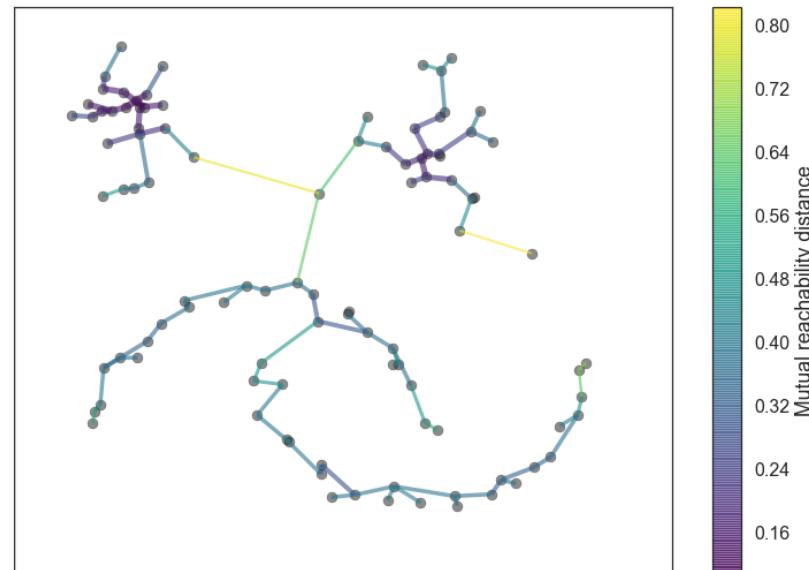
Very common algorithms to find the minimum spanning tree are:

Prim's algorithm

Kruskal's algorithm

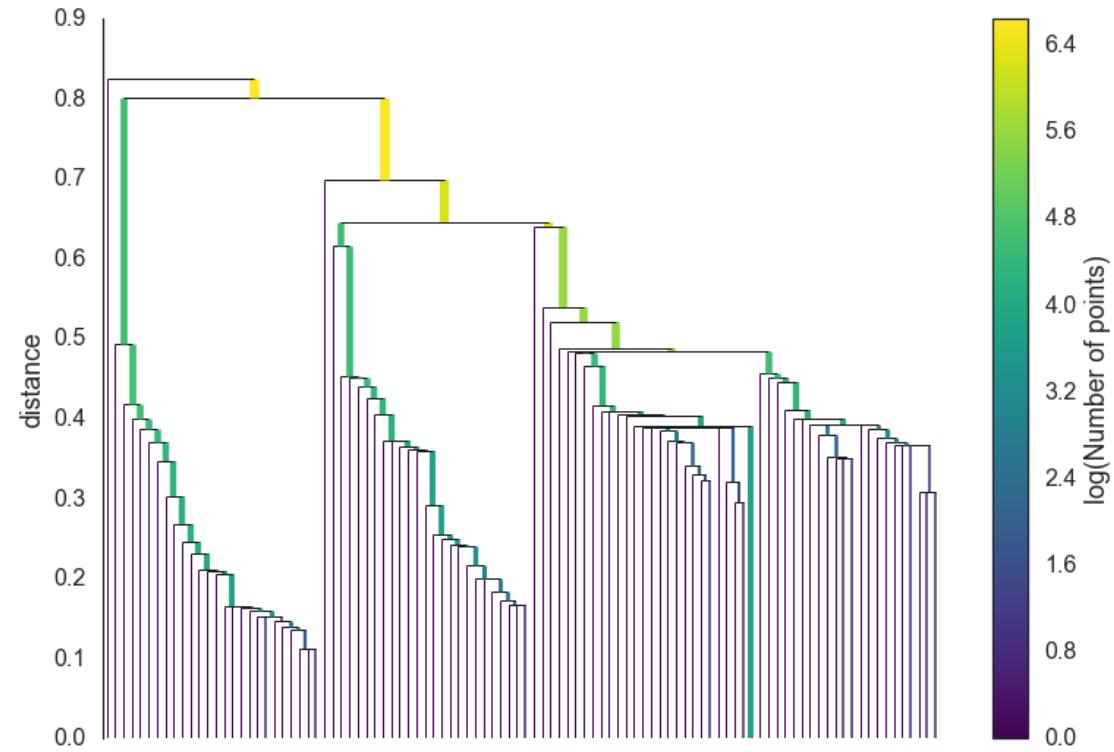
and Borůvka's algorithm

The procedure builds the tree one edge at a time, adding the lowest weighted edge that connects the tree to a vertex not yet in the tree.



Cluster hierarchy

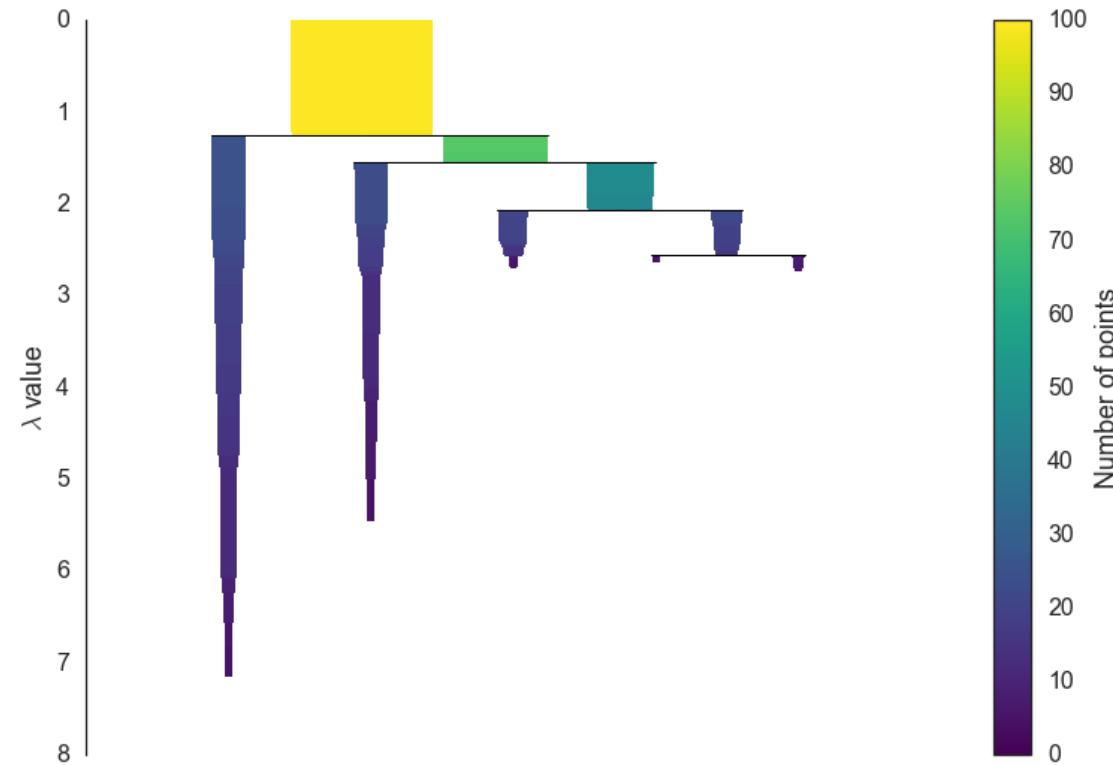
In an agglomerative perspective we can build a hierarchy of clusters:



But we still need a linkage stop condition

Condense the cluster tree

To condense the tree, we set a `minClusterSize`, a threshold above which a set of points is considered a cluster.



Flat the clusters

Now we will briefly focus on the stability-based cluster validation method

We define λ as a new quantity called persistence, equal to the inverse of the distance

We can explore the tree Top-Down: at a certain point the tree is splitted and new clusters are created, that point of the tree is the λ_{birth} of the clusters. When the cluster is splitted again, or we reach the end, that point is called λ_{death} .

When a sample falls out of the cluster (because of the distance or because the cluster is splitted in smaller clusters) that point is called λ_p

At the end we can compute the stability as

$$\lambda = \frac{1}{distance}$$

$$\lambda_{birth}, \lambda_{death}, \lambda_p$$

$$stability = \sum_{p \in cluster} (\lambda_p - \lambda_{birth})$$

Flat the clusters

Suppose we have three clusters, one parent (C_1) and two childs (C_2, C_3).

Suppose all leaf nodes are clusters. Now we explore Bottom-Up.

If $\text{stability}(C_2) + \text{stability}(C_3) > \text{stability}(C_1)$

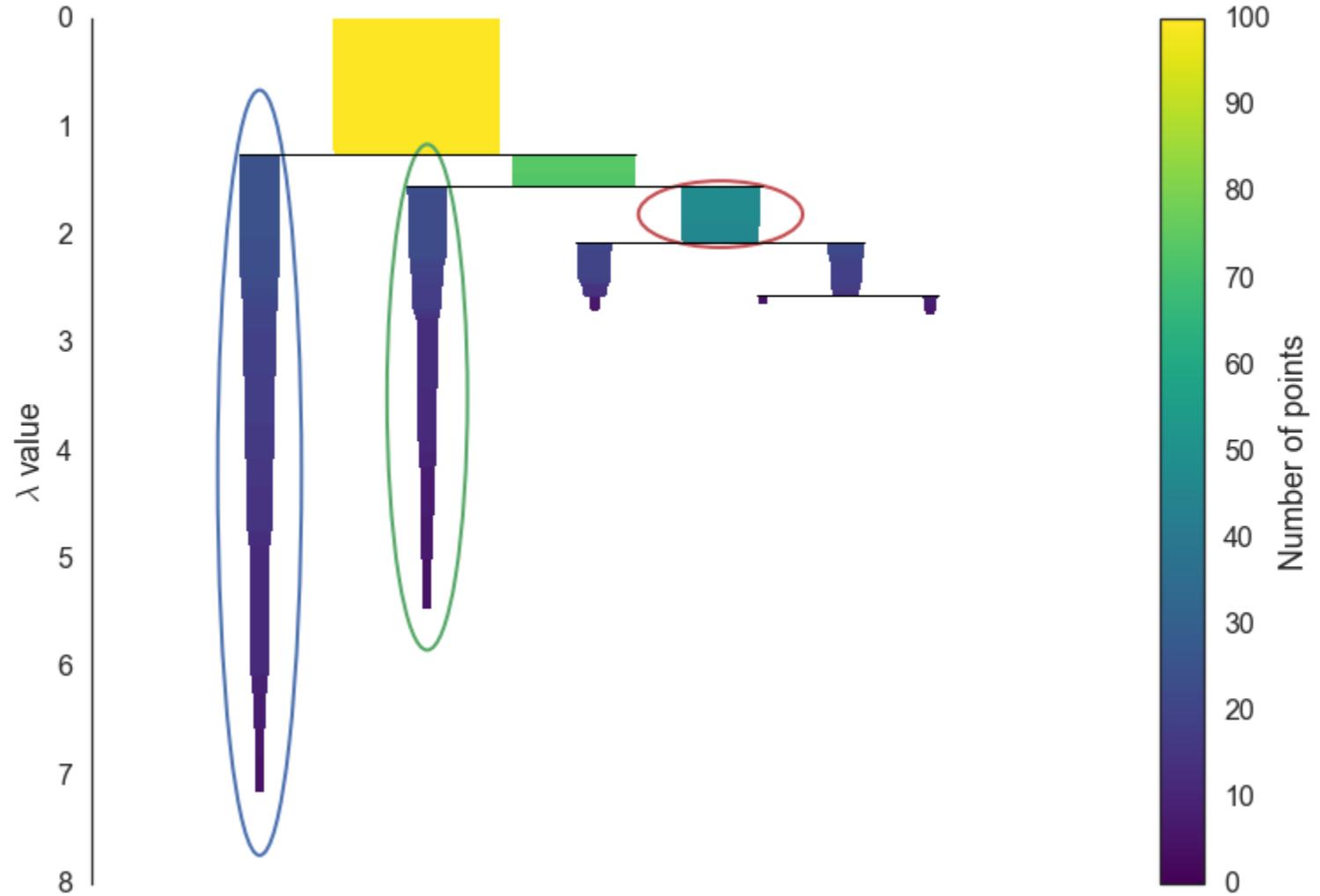
the cluster C_1 is not stable

we set $\text{stability}(C_1) = \text{stability}(C_2) + \text{stability}(C_3)$

Else if $\text{stability}(C_1) \geq \text{stability}(C_2) + \text{stability}(C_3)$

C_1 is a stable cluster, unset all descendants as clusters

Flat the clusters



HDBSCAN – PROS/CONS

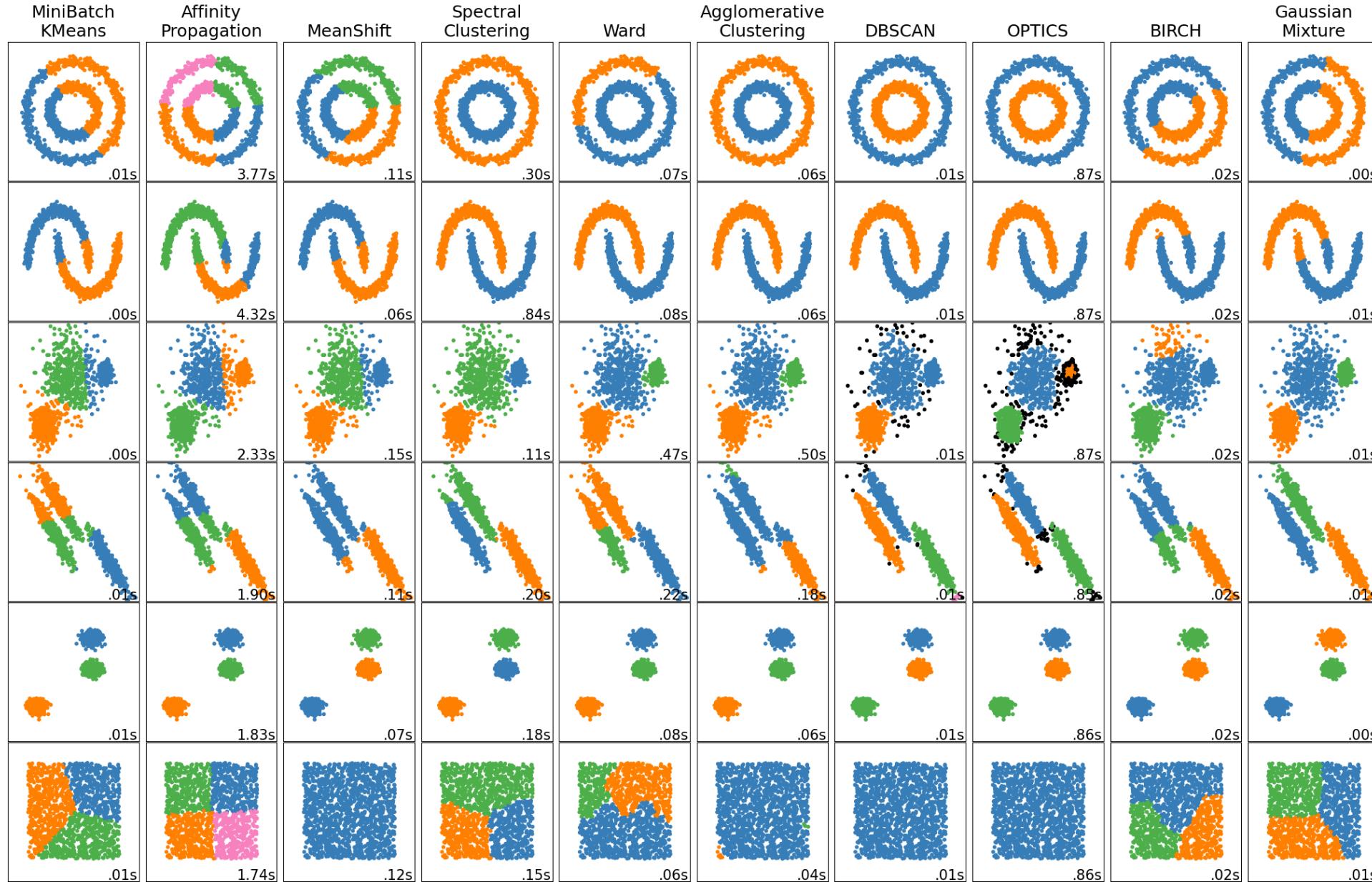
Pros

- Can handle varying densities
- The algorithm is stable over runs and parameter choices
- Robust to outliers

Cons

- Some important parameters must be set by hand

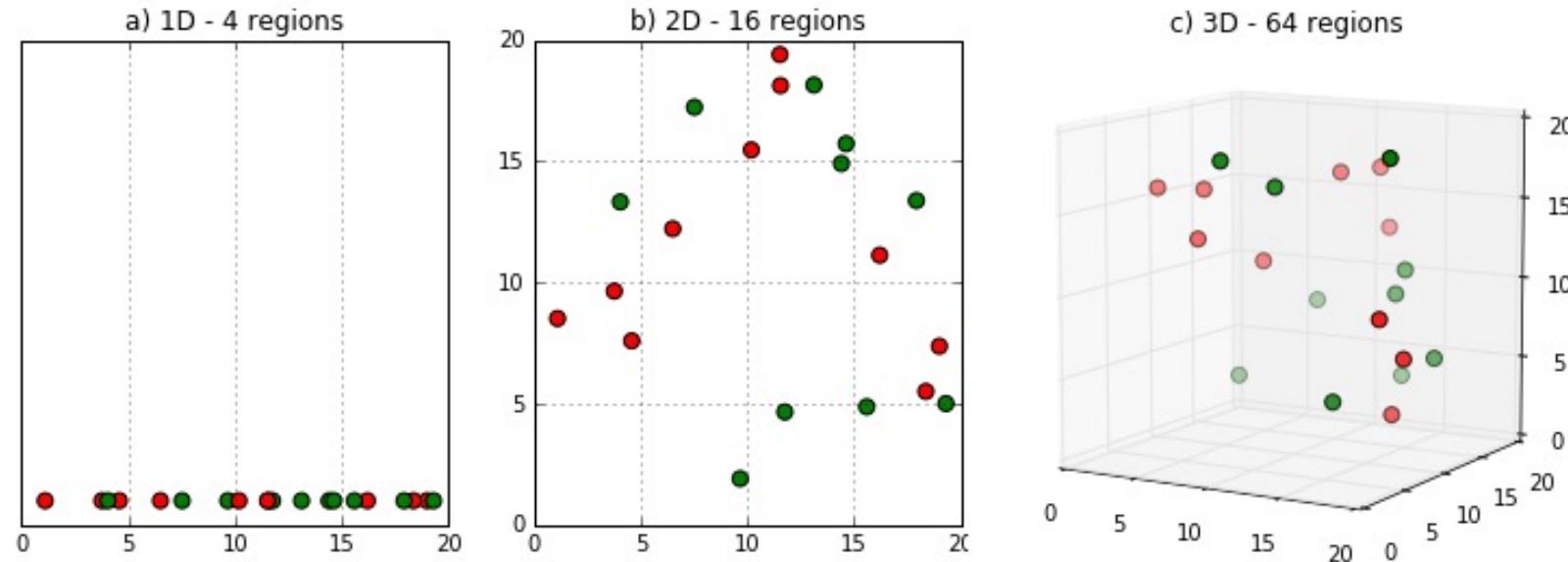
A comparison (implementation in scikit-learn)



ML principles

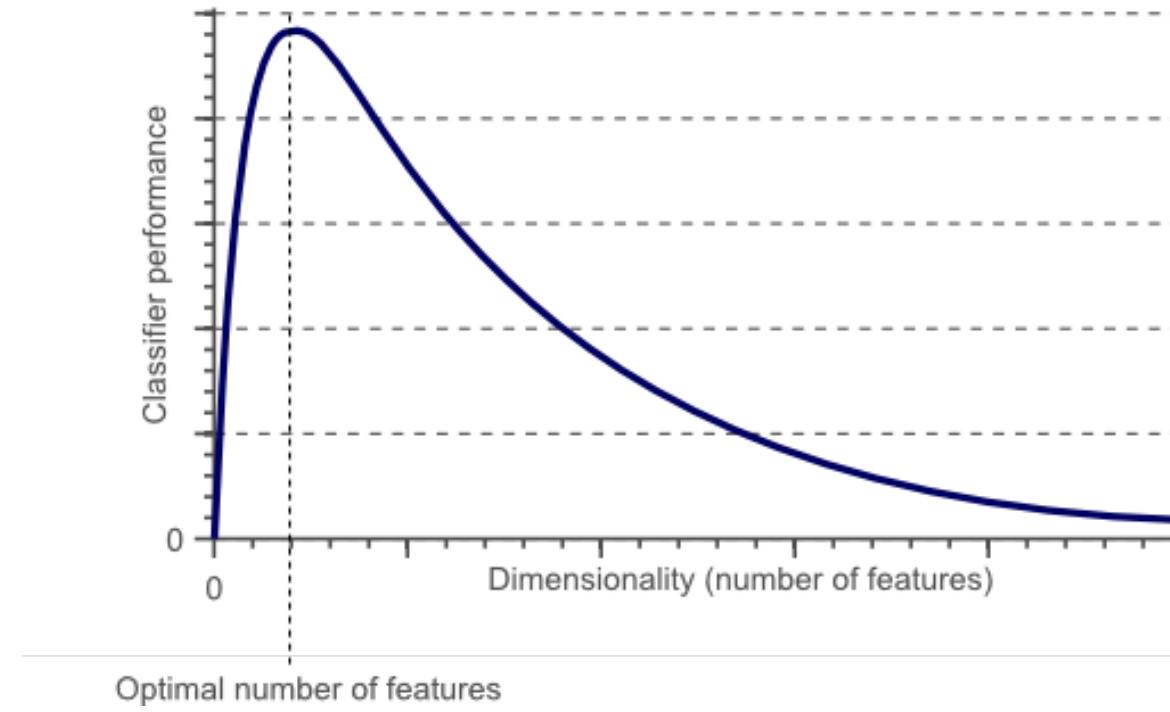
The Curse of Dimensionality

The Curse of Dimensionality



- A single feature is not sufficient to build a perfect classifier (1d).
- Adding a second feature still does not result in a linearly separable classification problem (2d).
- Imagine that adding a third feature results in a linearly separable classification problem in our example (3d).
- Strategy: Can we build a perfect classifier by defining a very big number of features? The answer is no!

The Curse of Dimensionality



As the dimensionality increases, the classifier's performance increases until the optimal number of features is reached. Further increasing the dimensionality without increasing the number of training samples results in a decrease in classifier performance.

Why couldn't we simply increase the number of features?

In the example we built a perfect classifier BUT note how **the density of the training samples decreases exponentially** when we increase the dimensionality of the problem.

- 1d: we almost covered the complete 1d feature space; sample density $\sim 30/4$.
- 2d: the 2D feature space has an area of $4 \times 4 = 16$ unit squares; sample density $\sim 30/16$.
- 3d: a feature space volume of $4 \times 4 \times 4 = 64$ unit cubes; sample density $\sim 30/30$.

The Curse of Dimensionality

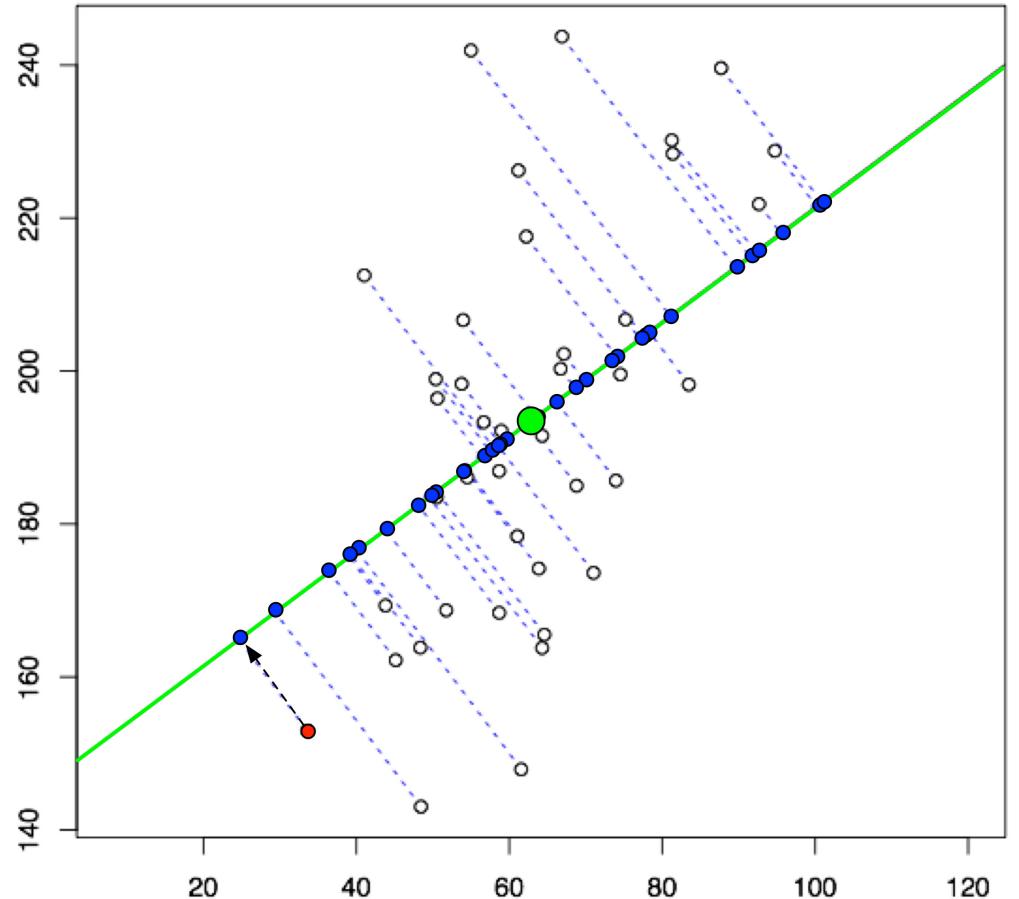
- If the amount of available training data is fixed, then **overfitting** occurs if we keep adding dimensions.
- The “curse” of dimensionality introduces the **sparseness** of the training data. The more features we use, the sparser the data becomes, and consequently accurate estimation of the classifier’s parameters (i.e., its decision boundaries) becomes more difficult.
- To maintain the same coverage and avoid overfitting, the amount of training data would need to grow **exponentially** fast, but this condition rarely occurs.

Unsupervised Learning

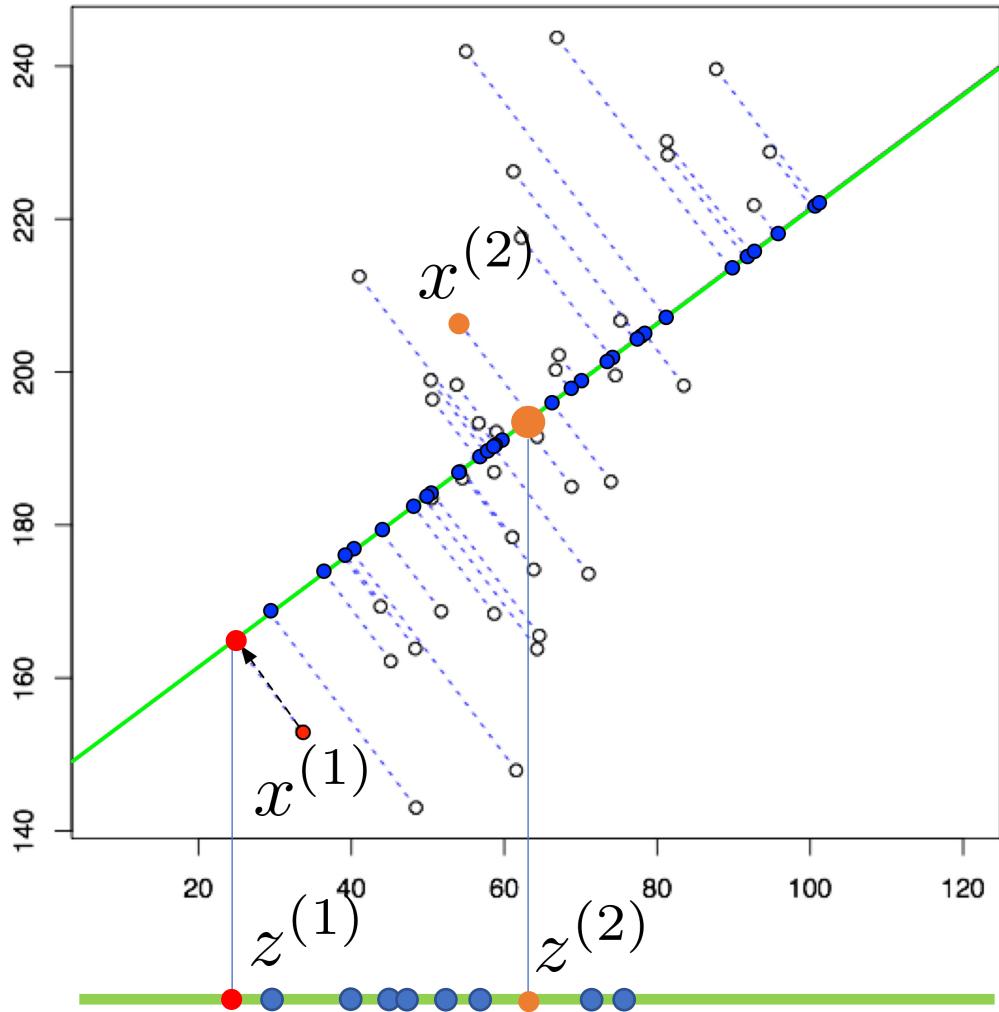
Dimensionality Reduction

Motivation I: Data Compression

Reduce data from 2D to 1D



Motivation I: Data Compression



Reduce data from 2D to 1D

$$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$$

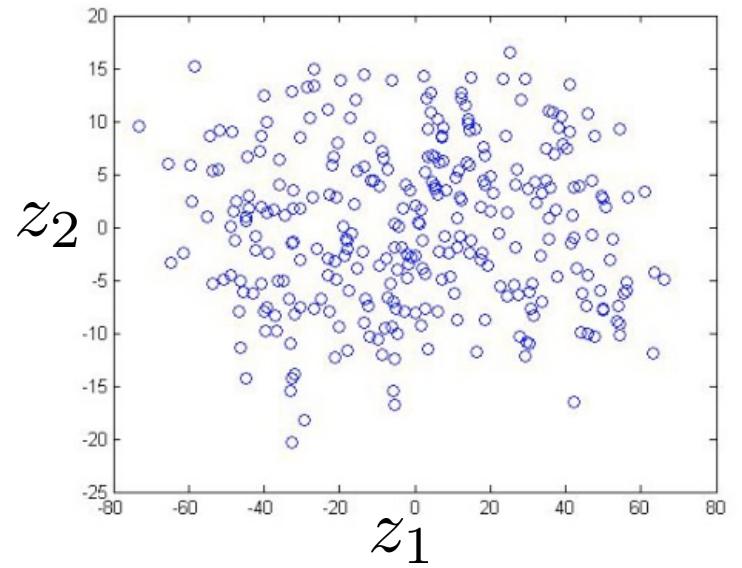
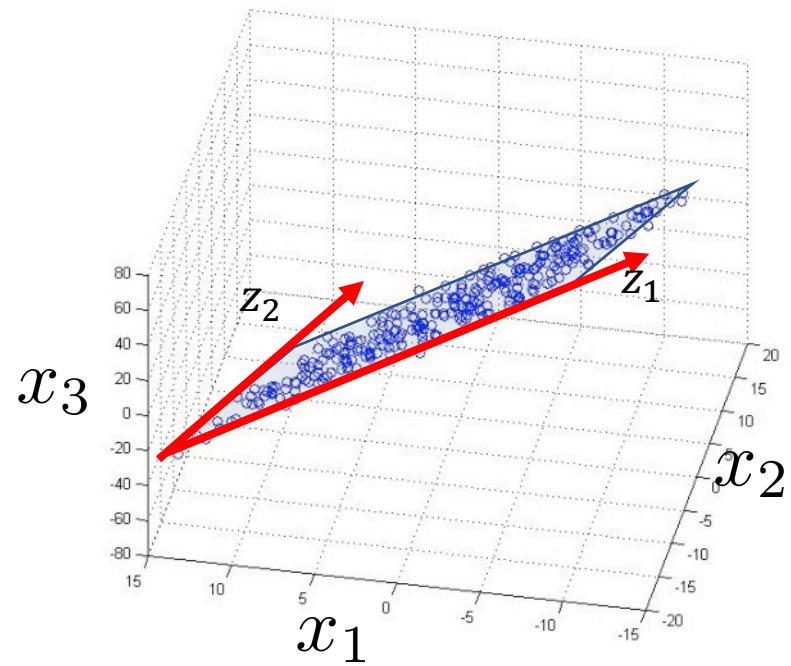
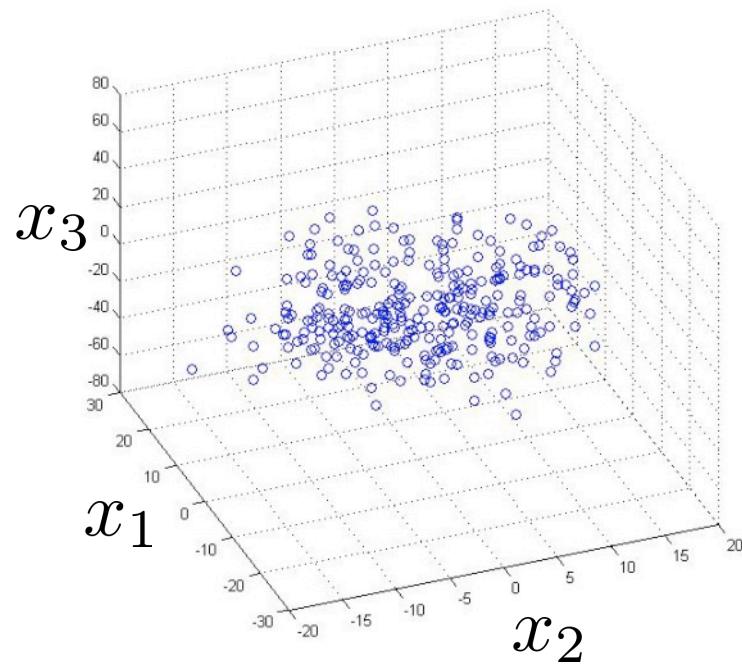
$$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$$

...

$$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$$

Motivation I: Data Compression

Reduce data from 3D to 2D



$$x^{(i)} \in \mathbb{R}^3 \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \mathbf{z}^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

Motivation II: Data Visualization

		x_1	x_2	x_3	x_4	x_5	
Country	Region	Population	Area	Pop. Density	Coastline
Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0	0,00
Albania	EASTERN EUROPE	3581655	28748	124,6	1,26
Algeria	NORTHERN AFRICA	32930091	2381740	13,8	0,04
American Samoa	OCEANIA	57794	199	290,4	58,29
Andorra	WESTERN EUROPE	71201	468	152,1	0,00
Angola	SUB-SAHARAN AFRICA	12127071	1246700	9,7	0,13
Anguilla	LATIN AMER. & CARIB	13477	102	132,1	59,80
Antigua & Barbuda	LATIN AMER. & CARIB	69108	443	156,0	34,54
Argentina	LATIN AMER. & CARIB	39921833	2766890	14,4	0,18
...

$$x \in \mathbb{R}^{50}$$

$$x^{(i)} \in \mathbb{R}^{50}$$

Motivation II: Data Visualization

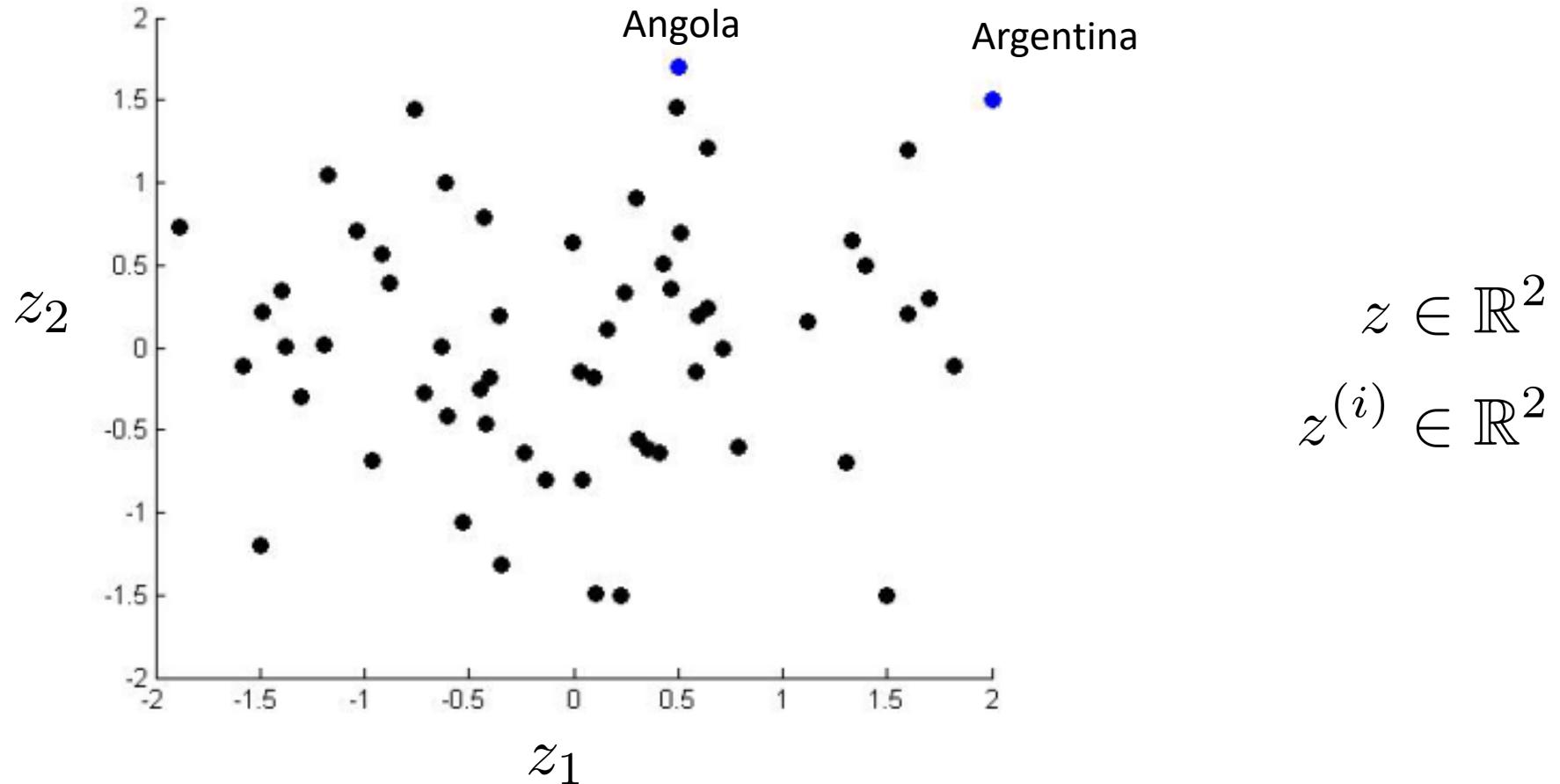
	z_1	z_2
Country	z_1	z_2
Afghanistan	1.6	1.2
Albania	1.7	0.3
Algeria	1.6	0.2
American Samoa	1.4	0.5
Andorra	0.5	1.7
Angola	2	1.5
Anguilla	1.4	1.5
Antigua & Barbuda	0.5	0.5
Argentina	2	1.7
...

$$z \in \mathbb{R}^2$$

$$z^{(i)} \in \mathbb{R}^2$$

We reduced data from 50D to 2D

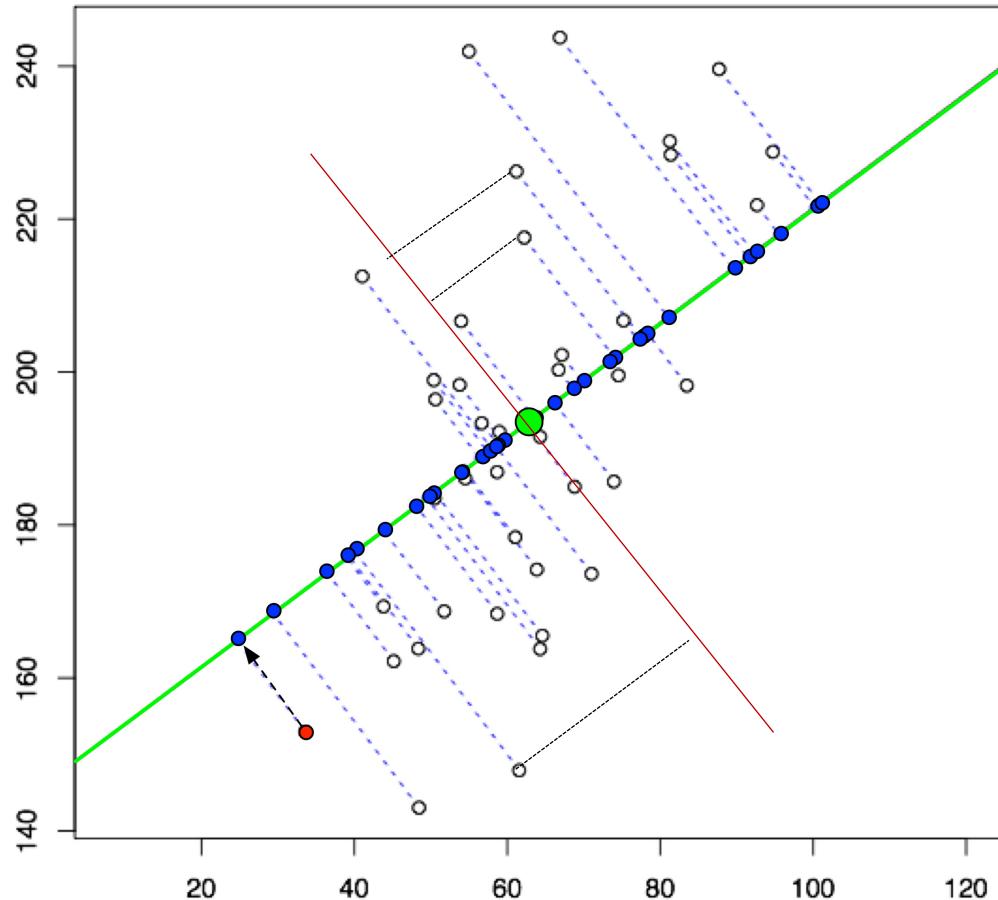
Motivation II: Data Visualization



Dimensionality Reduction

Principal Component Analysis

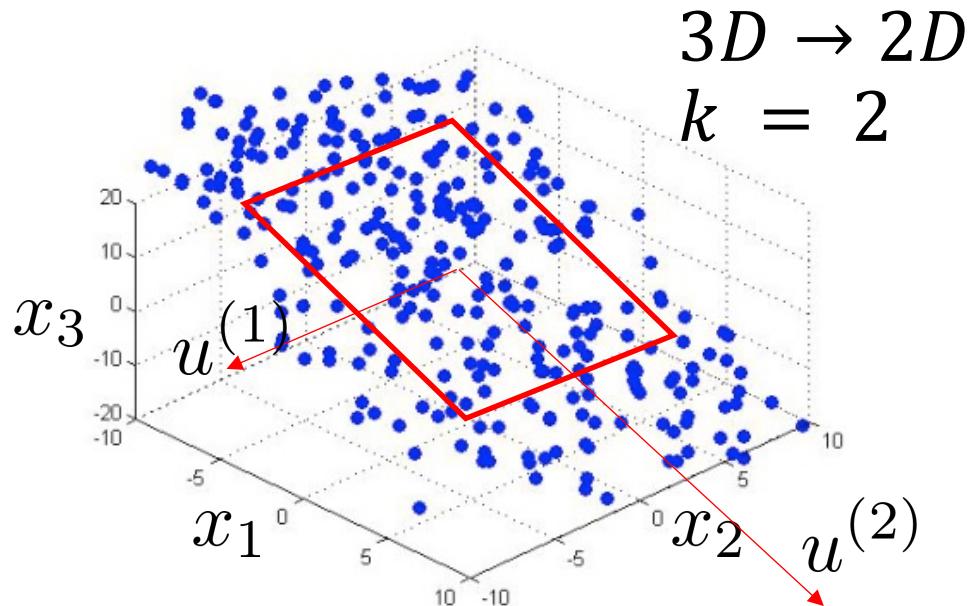
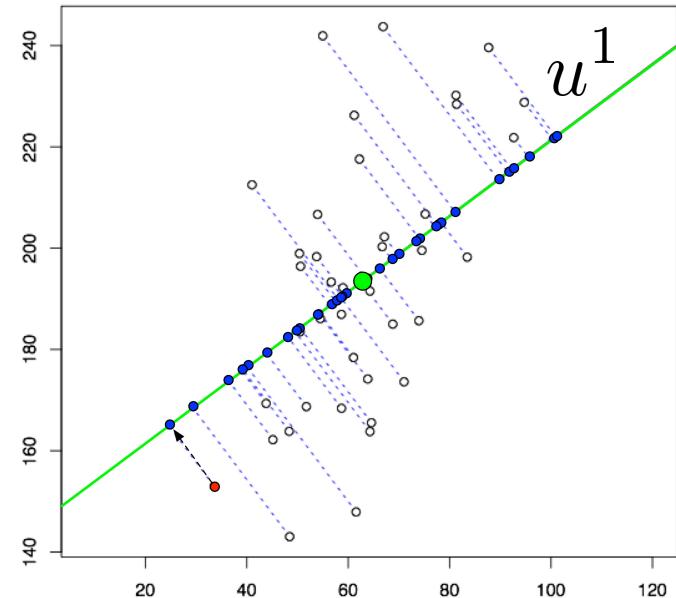
Problem formulation



$$z \in \mathbb{R}^2$$

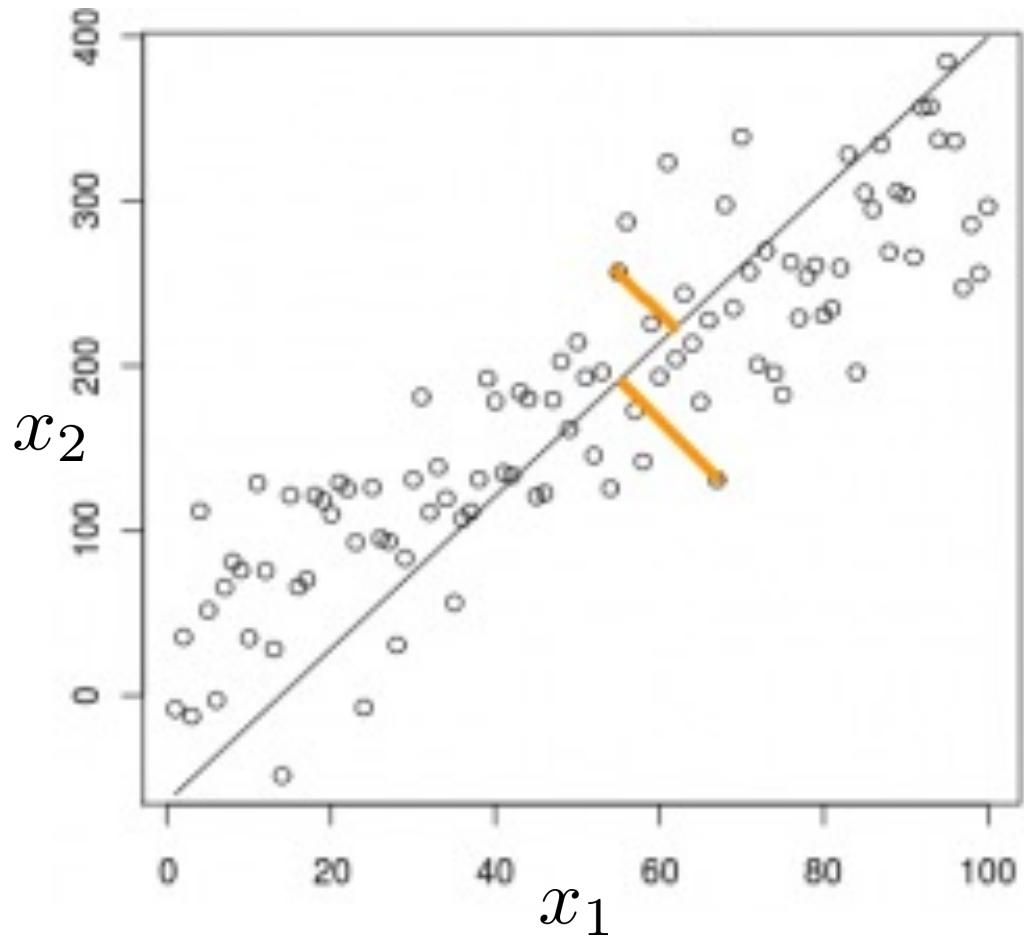
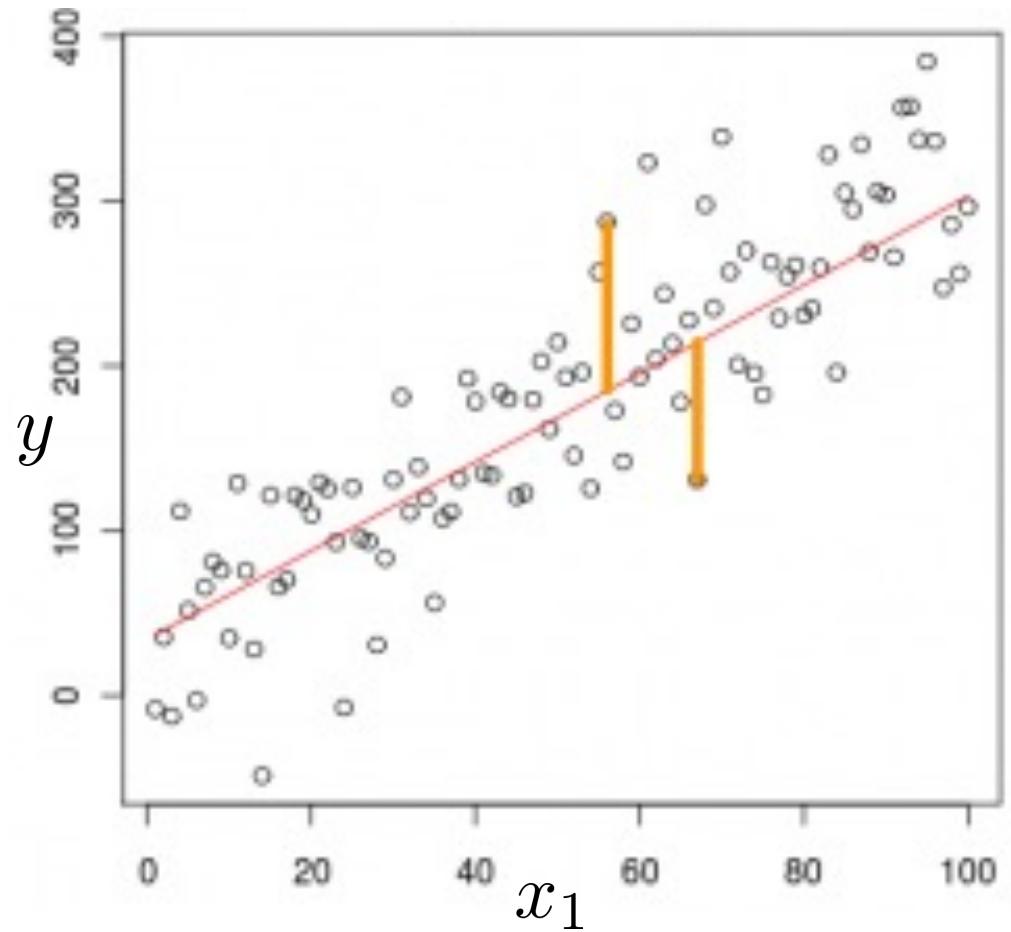
$$z^{(i)} \in \mathbb{R}^2$$

Problem formulation

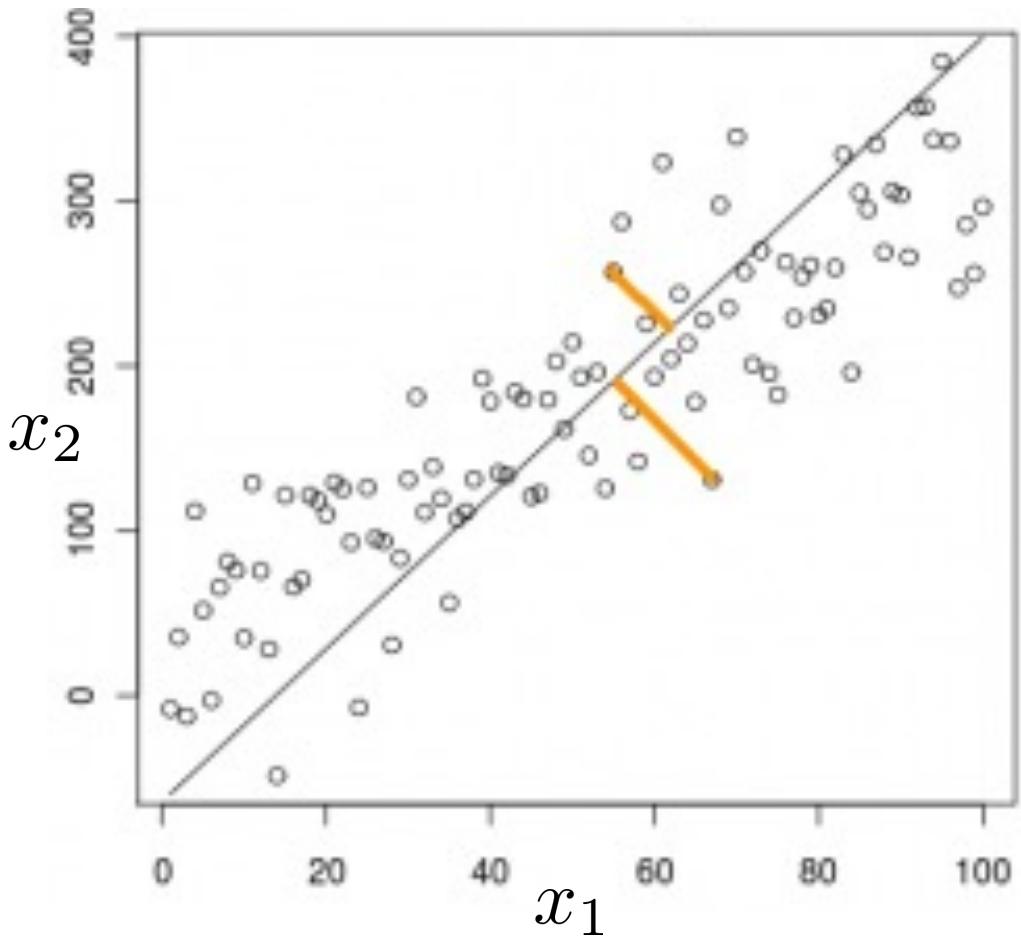
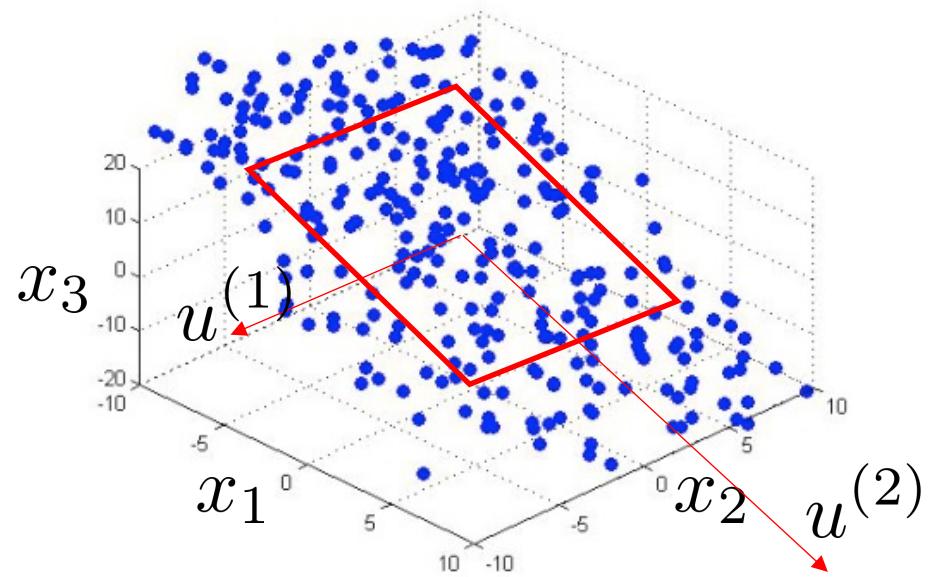


- Reduce from 2D to 1D: Find a direction (a vector u in \mathbb{R}^n) onto which to project to minimize the projection error.
- Reduce from nD to kD: Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project in order to minimize the projection error.

PCA is NOT linear regression



PCA – PROS/CONS



PCA Algorithm summary

1. Preprocess Data
2. Compute covariance matrices
3. Compute svd
4. Transform data in the new space

Data preprocessing

Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

Feature scaling/Mean normalization

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

$$x_j^{(i)} \leftarrow x_j^{(i)} - \mu_j \quad (\text{zero mean})$$

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{stdev_j} \quad (\text{same scale among features})$$

Compute covariance matrix

Now we can compute the covariance **matrix** between each pair of features. Each component is defined as:

$$\begin{aligned}\Sigma_{ij} &= Cov(x_i, x_j) \\ &= \mathbb{E}[(x_i - \mu_i)(x_j - \mu_j)] = \frac{1}{m} \cdot \sum_{l=1}^m (x_i^{(l)} - \mu_i) \cdot (x_j^{(l)} - \mu_j) \\ &= \mathbb{E}[(x_i)(x_j)]\end{aligned}$$

This operation can be easily vectorized:

$$\Sigma = \frac{1}{m} \sum_{l=1}^m \underbrace{x^{(l)}}_{n \times 1} \cdot \underbrace{x^{(l)T}}_{1 \times n}_{n \times n}$$

Singular Value Decomposition

In order to compute the eigenvectors of Σ we can compute the Singular Value Decomposition (SVD):

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = \text{svd}(\Sigma)$$

SVD is a factorization that takes as input a generic matrix A in $C_{m \times n}$ such that

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^*$$

Where:

- \mathbf{U} is the left eigenvectors matrix, a left unitary matrix (the product with its conjugate transpose is the identity matrix $\mathbf{U} \cdot \mathbf{U}^* = \mathbf{I}$, for real matrices $\mathbf{U}^{-1} = \mathbf{U}^T$ is valid) of shape $m \times m$
- \mathbf{S} is a diagonal matrix of shape $m \times n$ (the first n components are not null and they are the eigenvalues of A , ordered in decreasing order)
- \mathbf{V}^* is the conjugate transpose of a right unitary matrix of shape $n \times n$

But we are computing the SVD of a covariance matrix (a symmetric matrix), hence $\mathbf{U} = \mathbf{V}$

$$\Sigma = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{U}^T$$

Singular Value Decomposition

From $[U, S, V] = svd(\Sigma)$ we get:

$$\mathbf{U} = \left[\begin{array}{cccc} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{array} \right] \in \mathbb{R}^{n \times n}$$

$\underbrace{\phantom{u^{(1)} \quad u^{(2)} \quad \dots \quad u^{(n)}}}_k$

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

$$\mathbf{z}^{(i)} = \left[\begin{array}{cccc} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{array} \right]^T \in \mathbb{R}^{n \times k}$$

$\underbrace{\phantom{u^{(1)} \quad u^{(2)} \quad \dots \quad u^{(k)}}}_{n \times k}$

Ureduce

$$\mathbf{x}^{(i)} = \left[\begin{array}{ccc} \phantom{(u^{(1)})^T} & (u^{(1)})^T & \phantom{(u^{(1)})^T} \\ \vdots & \vdots & \vdots \\ \phantom{(u^{(k)})^T} & (u^{(k)})^T & \phantom{(u^{(k)})^T} \end{array} \right]_{n \times 1}$$

$\underbrace{\phantom{(u^{(1)})^T \quad (u^{(2)})^T \quad \dots \quad (u^{(k)})^T}}_{k \times n}$

$\underbrace{\phantom{(u^{(1)})^T \quad (u^{(2)})^T \quad \dots \quad (u^{(k)})^T}}_{k \times 1}$

PCA Algorithm summary

1. Preprocess Data
2. Compute Sigma

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(1)})_{n \times 1} (x^{(1)})_{1 \times n}^T$$

3. Compute svd

$$[U, S, V] = svd(\Sigma)$$

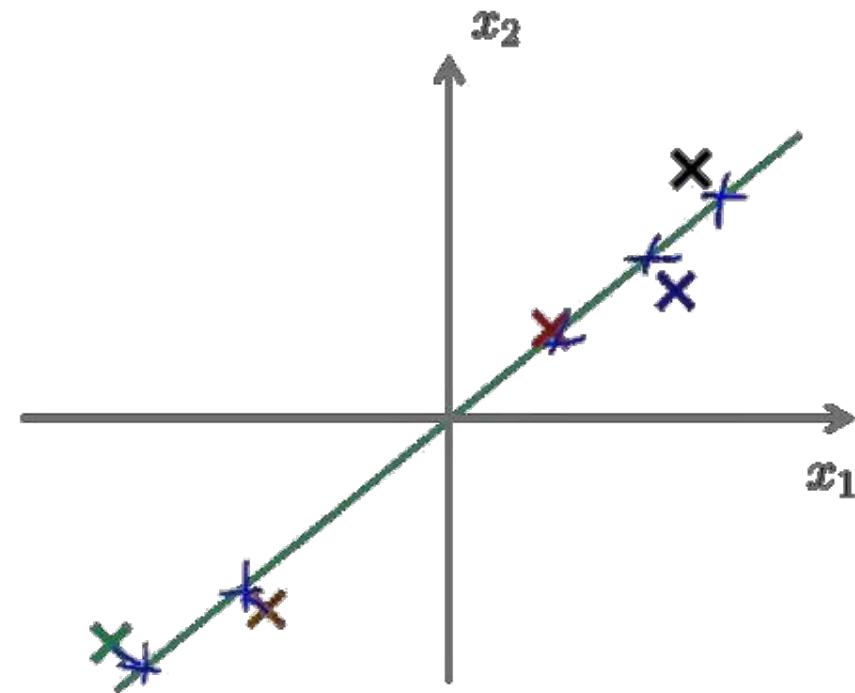
4. Choose the number of dimension and extract the eigenvectors

Ureduce = U(:,1:k)

5. Map data in the new space

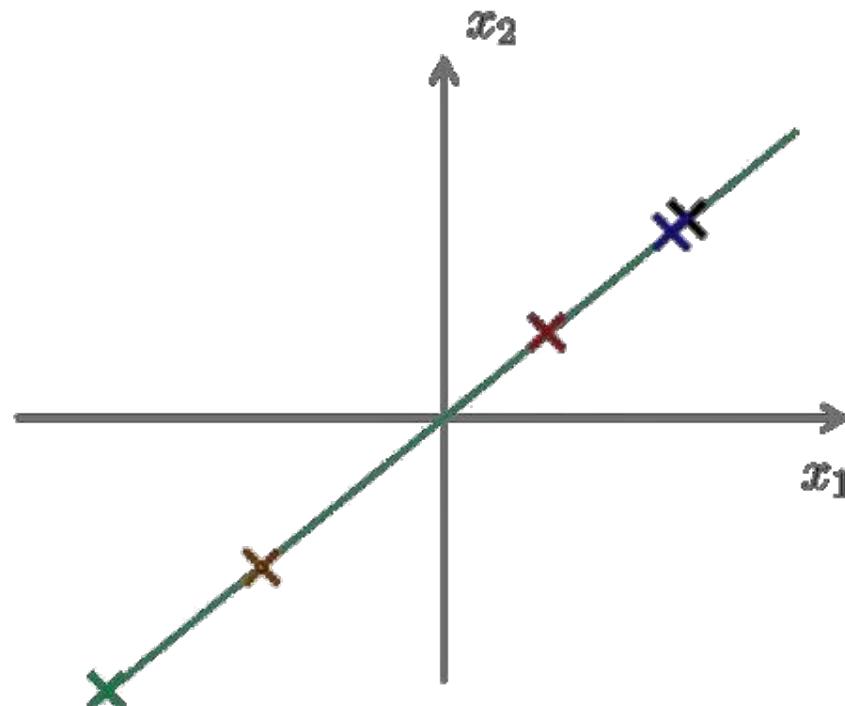
$z = Ureduce^T * x$

Reconstruction from compressed representation



$$z = Ureduce^T x$$





$$z \in \mathbb{R} \rightarrow x \in \mathbb{R}^2$$

$$\underbrace{Xapprox^{(i)}}_{n \times 1} = \underbrace{Ureduce}_{n \times k} \cdot \underbrace{z^{(i)}}_{k \times 1}$$

$n \times 1$

Choosing the number of Principal Components

Average squared projection error: $\sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

It is usual to choose k to be the smallest value so that

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01 \quad (1\%)$$

99% of variance is retained

Choosing the number of Principal Components

Algorithm:

Try PCA with $k=1,2,3\dots$

Compute:

$U_{\text{reduce}}, z(1), z(2), \dots, z(m)$

$X_{\text{approx}}(1), \dots, X_{\text{approx}}(m)$

Check if

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{\text{approx}}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01$$

$[U, S, V] = \text{svd}(Sigma)$

$$S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & \ddots & \\ & & & S_{nn} \end{bmatrix}$$

for a given k

$$1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \leq 0.01 \Rightarrow \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.99$$

PCA – PROS/CONS

Pros

- Can handle varying densities
- The algorithm is stable over runs and parameter choices
- Robust to outliers

Cons

- Some important parameters must be set by hand

Dimensionality Reduction

Independent Component Analysis

Problem definition

Decomposing a mixed signal into independent sources

Example

Given: Mixed Signal

Goal: retrieve the original signals

Source1 News

Source2 Song

Common example

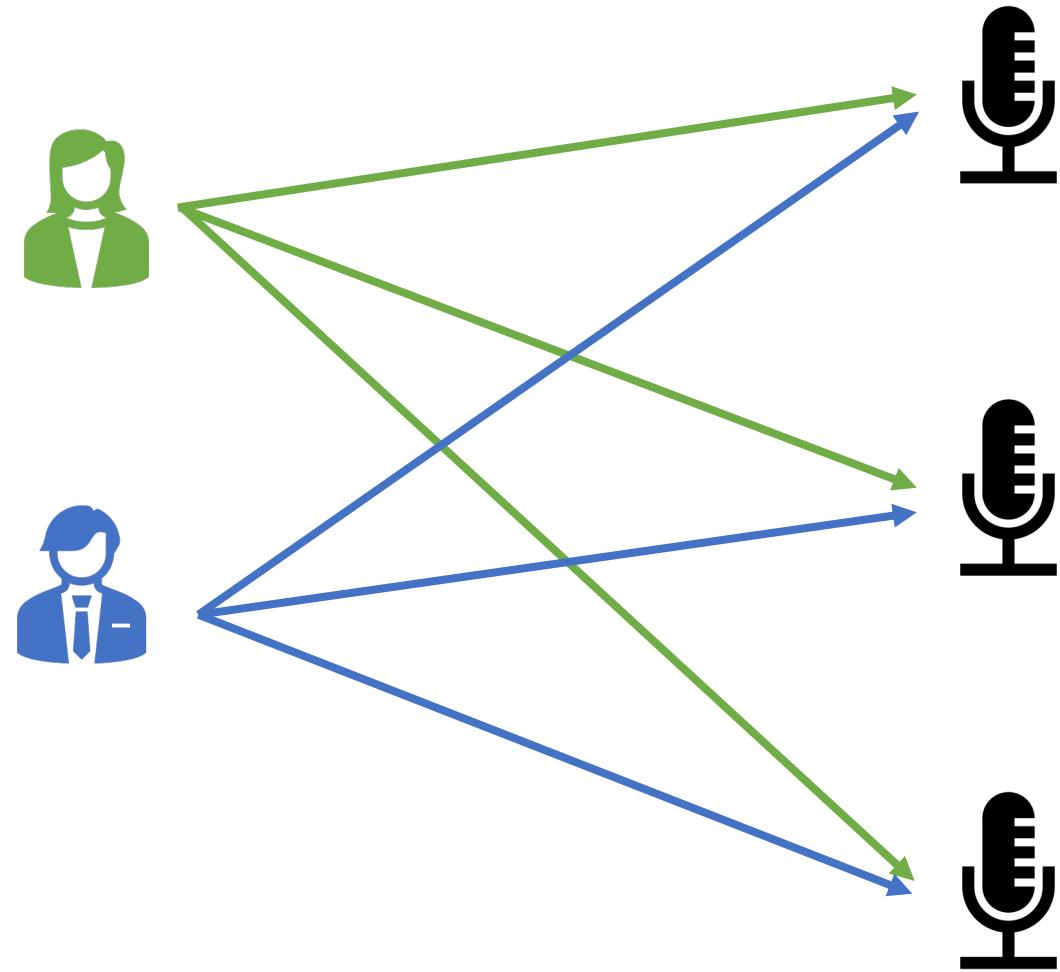
Cocktail party or Blind Source Separation (**BSS**) problem

Definitions: Statistical independence

In probability theory, saying that two events are **independent** means that the occurrence of one event makes it neither more nor less probable that the other occurs.

Examples of such independent random variables are the value of a dice thrown and of a coin tossed

Cocktail party problem



Problem

The microphones give us three recorded time signals.

We denote them with:

$$\mathbf{x} = (x_1(t), x_2(t), x_3(t)).$$

x_1 , x_2 , and x_3 are the amplitudes and t is the time index.

We denote the independent signals by

$$\mathbf{s} = (s_1(t), s_2(t));$$

A - mixing matrix (3x2)

$$x_1(t) = a_{11}s_1 + a_{12}s_2$$

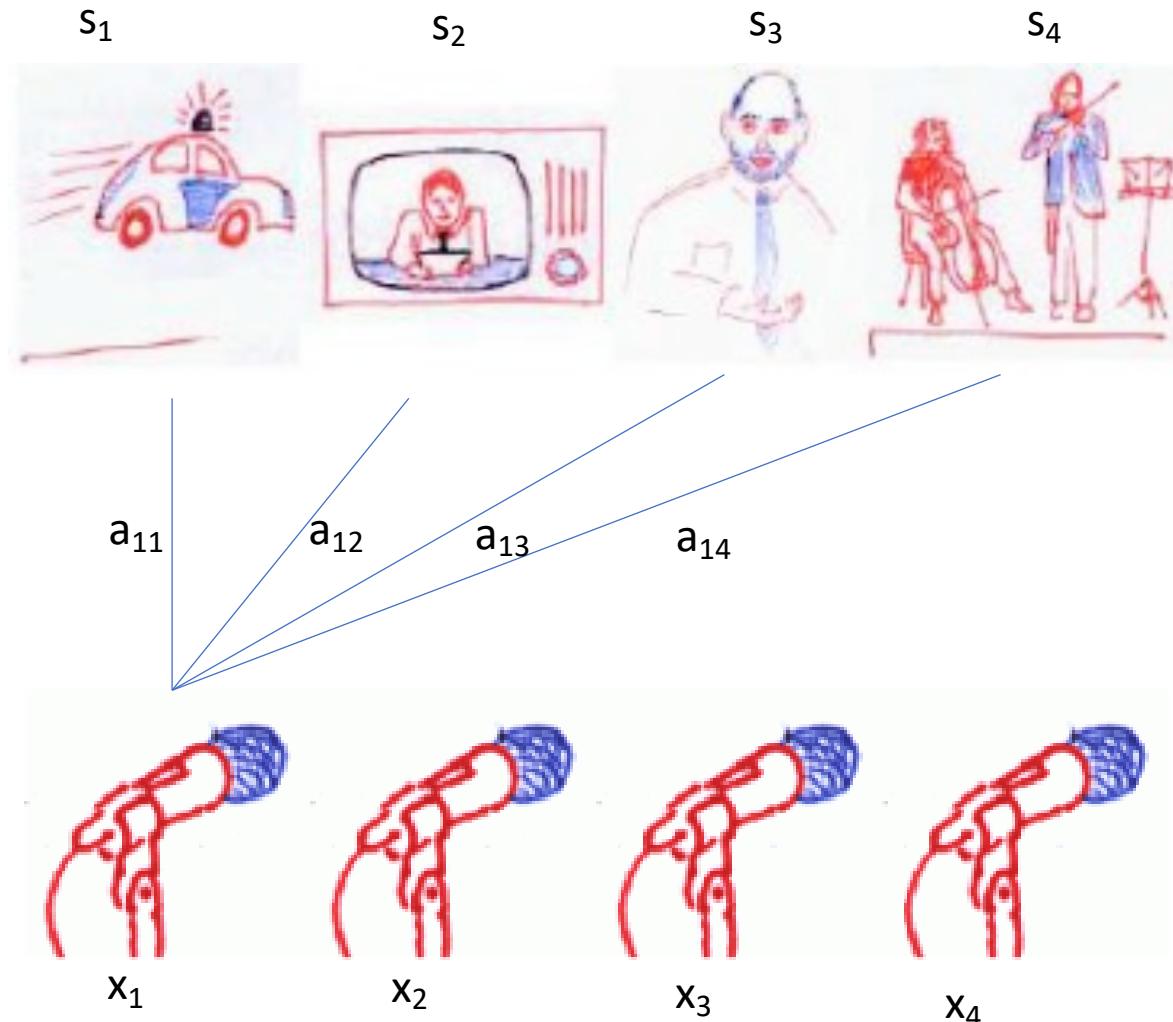
$$x_2(t) = a_{21}s_1 + a_{22}s_2$$

$$x_3(t) = a_{31}s_1 + a_{32}s_2$$

a_{ij} are some parameters that depend on the distances of the microphones from the speakers.

It would be very good if we could estimate the two original speech signals $s_i(t)$ using only the recorded signals $x_j(t)$. We need to estimate the a_{ij} , but it is enough to assume that $s_1(t)$ and $s_2(t)$, at each time instant t , are statistically independent. The main task is to transform the data $\mathbf{s}(x)=\mathbf{Ax}$ to independent components, measured by function: $F(s_1, s_2)$

ICA model



$$x_i(t) = a_{i1} * s_1(t) + a_{i2} * s_2(t) + a_{i3} * s_3(t) + a_{i4} * s_4(t)$$

$$i=1, \dots, 4.$$

In vector-matrix notation,
and dropping index t, this is

$$\mathbf{x} = \mathbf{A} * \mathbf{s}$$

Restrictions

In order to make ICA working some assumptions have to be made:

1. The independent components are assumed statistically *independent*
2. The independent components must have *non-Gaussian* distributions
3. For simplicity, we assume that the unknown mixing matrix is *squared*.

The number of independent components is equal to the number of observed mixtures.

Building bricks (recap)

Observed event: $x = As$ A:= mixing matrix

Observed streams $\{x^{(i)}; i = 1, \dots, m\}$

Original sources: $s^{(i)} \in \mathbb{R}^k$

$$s_j^{(i)}$$

Unmixing Matrix: $W = A^{-1}$

$$s^{(i)} = Wx^{(i)}$$

Unmixing element: w_i^T

$$s_j^{(i)} = w_i^T x^{(i)}$$

$$\mathbf{W} = \begin{bmatrix} \quad & (w^{(1)})^T & \quad \\ & \vdots & \\ \quad & (w^{(k)})^T & \quad \end{bmatrix}$$

ICA Ambiguities

Ambiguous permutation:

Let \mathbf{P} be a $n \times n$ permutation matrix: $\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\mathbf{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

If z is a vector, then $\mathbf{P} \cdot z$ is another vector that contains a permutation of z coordinates. Given only the $x(i)$'s, there is no way to distinguish between \mathbf{W} and $\mathbf{P} \cdot \mathbf{W}$ ($s = \mathbf{W} \cdot x$).

$$\mathbf{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

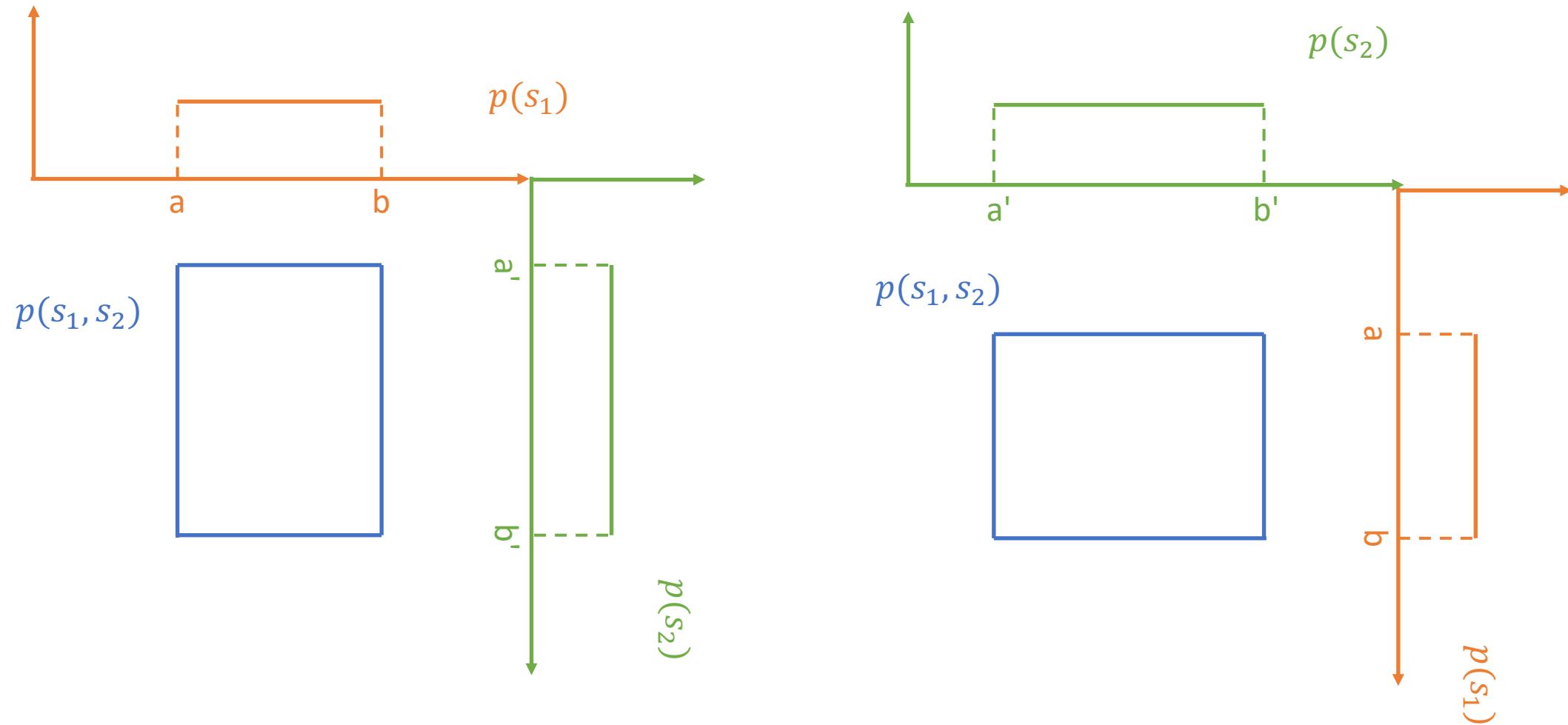
$$\mathbf{W} \cdot x = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 19 \\ 24 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$$

$$P \cdot W \cdot x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 & 6 \\ 3 & 5 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 24 \\ 19 \end{bmatrix}$$

$$x = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

ICA Ambiguities - Permutations



ICA Ambiguities

Ambiguous scale:

We said that:

$$x^{(i)} = A \cdot s^{(i)}$$

There is no way to recover the correct scaling of the w_i 's. For instance, if A was replaced with $2 \cdot A$, and every $s(i)$ were replaced with $0.5 \cdot s(i)$, then we obtain

$$x^{(i)} = 2A \cdot (0.5)s^{(i)}$$

The observed x is still the same.

Gaussian Ambiguity

Why sources have to be non-Gaussian?

Let us suppose our sample shows a Gaussian distribution:

$$s \sim \mathcal{N}(0, I)$$

What we observe is $x = A \cdot s$

The distribution of x will also be Gaussian with zero mean and variance:

$$\mathbb{E}[xx^T] = \mathbb{E}[A s s^T A^T] = AA^T$$

Gaussian Ambiguity

Now let R be an orthogonal matrix such that:

$$RR^T = R^T R = I$$

$$A' = AR$$

If A' was used as mixing matrix we would observe $x' = A' \cdot s$.

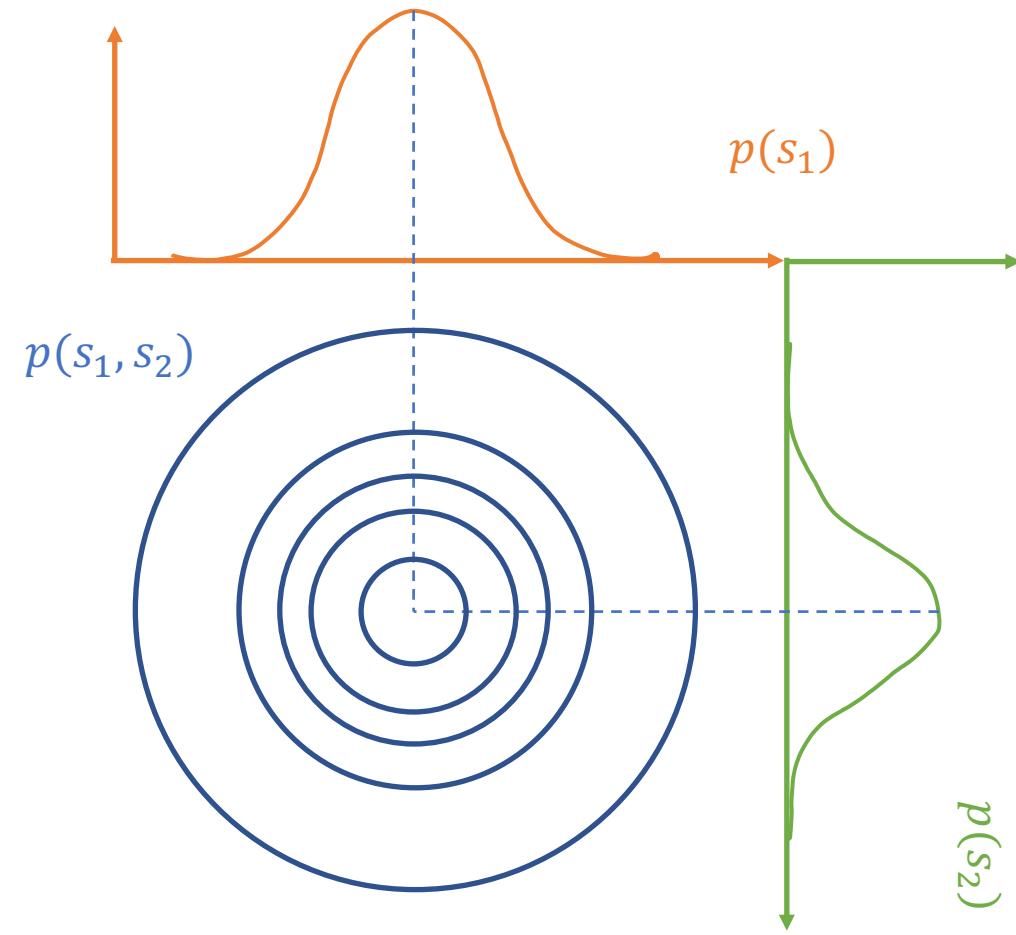
Also x' will show a Gaussian distribution with zero mean and variance:

$$\mathbb{E}[x'(x')^T] = \mathbb{E}[A'ss^T(A')^T] = \mathbb{E}[ARss^T(AR)^T] = ARR^TA^T = AA^T$$

Also in this case we will observe data in $\mathcal{N}(0, AA^T)$

If sources are Gaussian there is no way to know if they were mixed with A or A' .

ICA Ambiguities – Gaussian Distributions



Central Limit Theorem (practical remind)

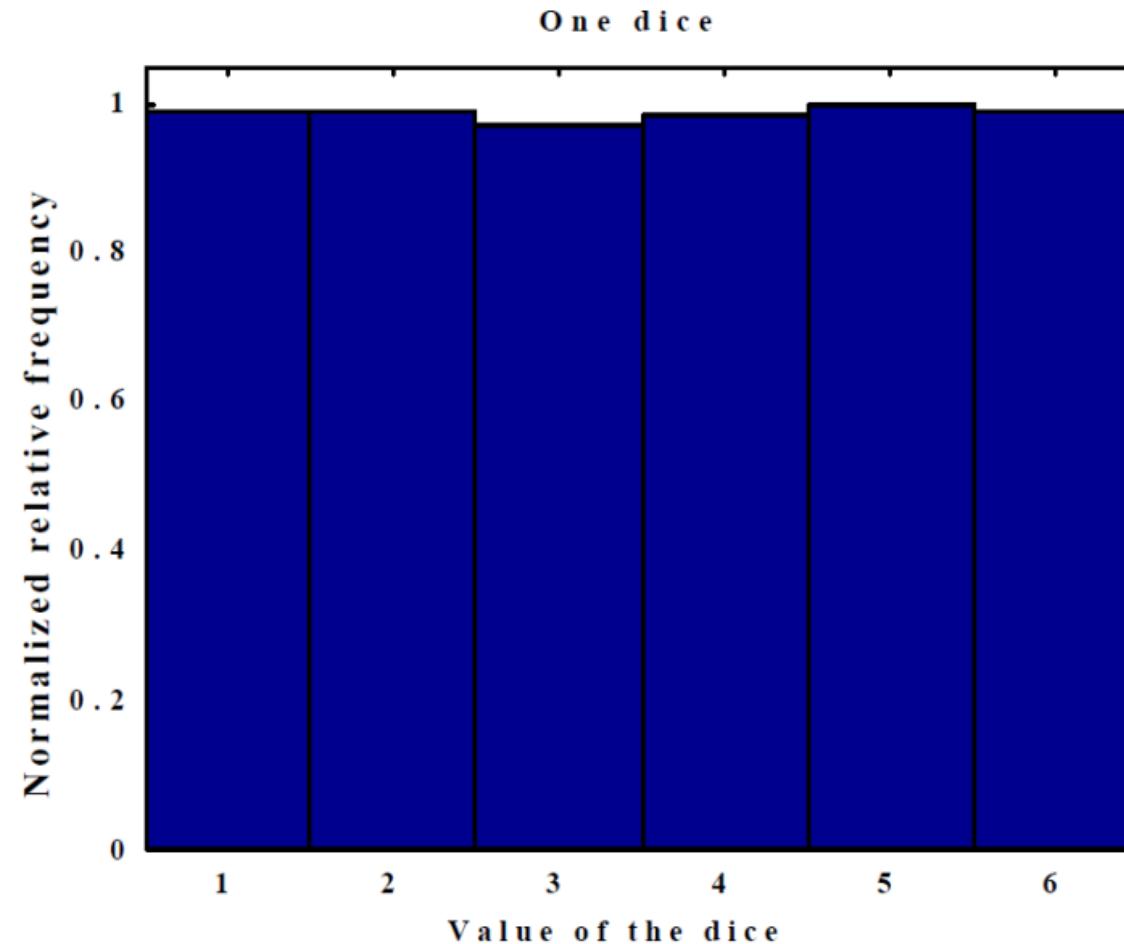
Central limit theorem states that linear combination of given random variables is more Gaussian as compared to the original variables themselves.

Example:

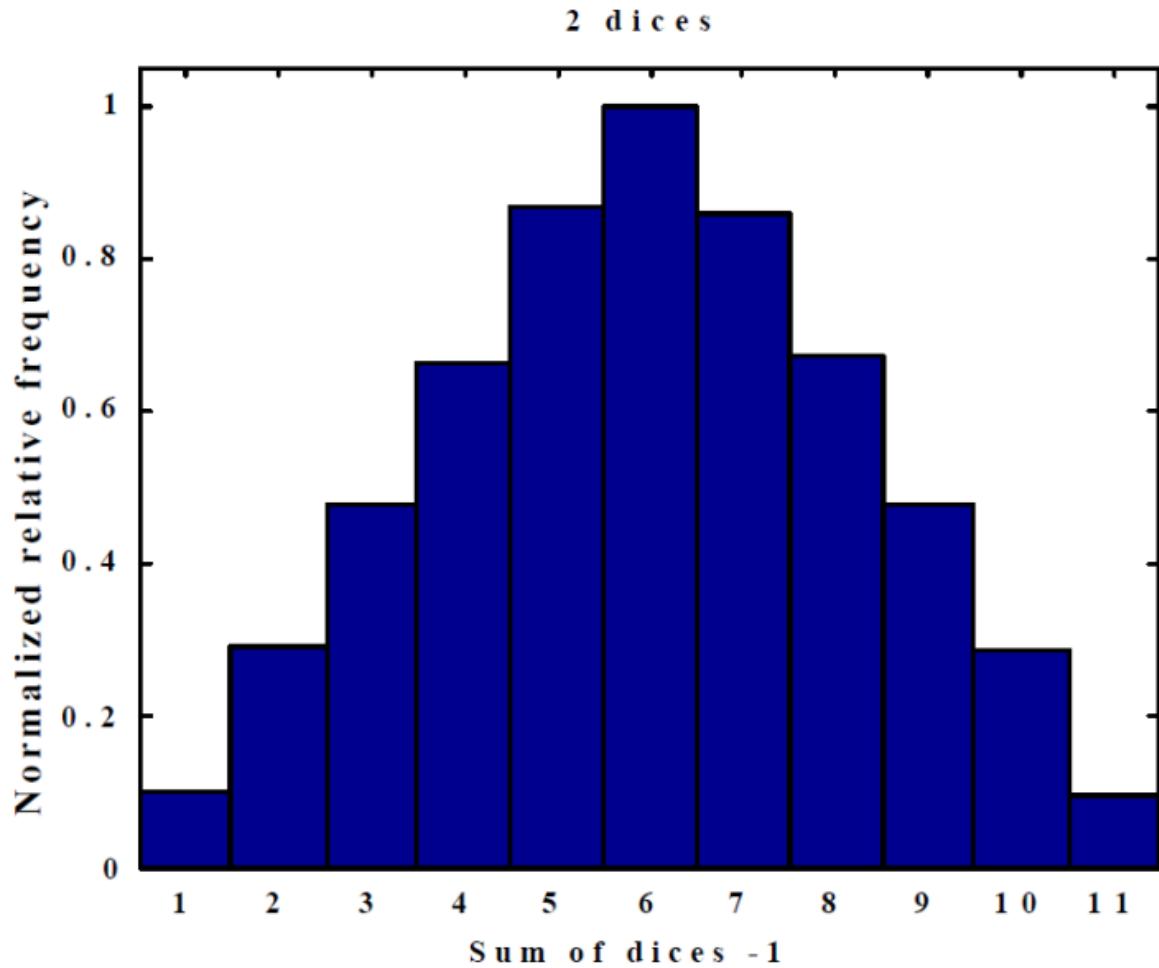
Repeated throwing of nonbiased dices, with the result given by the sum of the dice faces.

- For one dice the distribution is uniform;
- For two dices the sum is piecewise linearly distributed;
- For a set of k dices, the distribution is given by a k -order polynomial distribution;
- With the increase of k , the distribution of the sum tends towards a more Gaussian one.

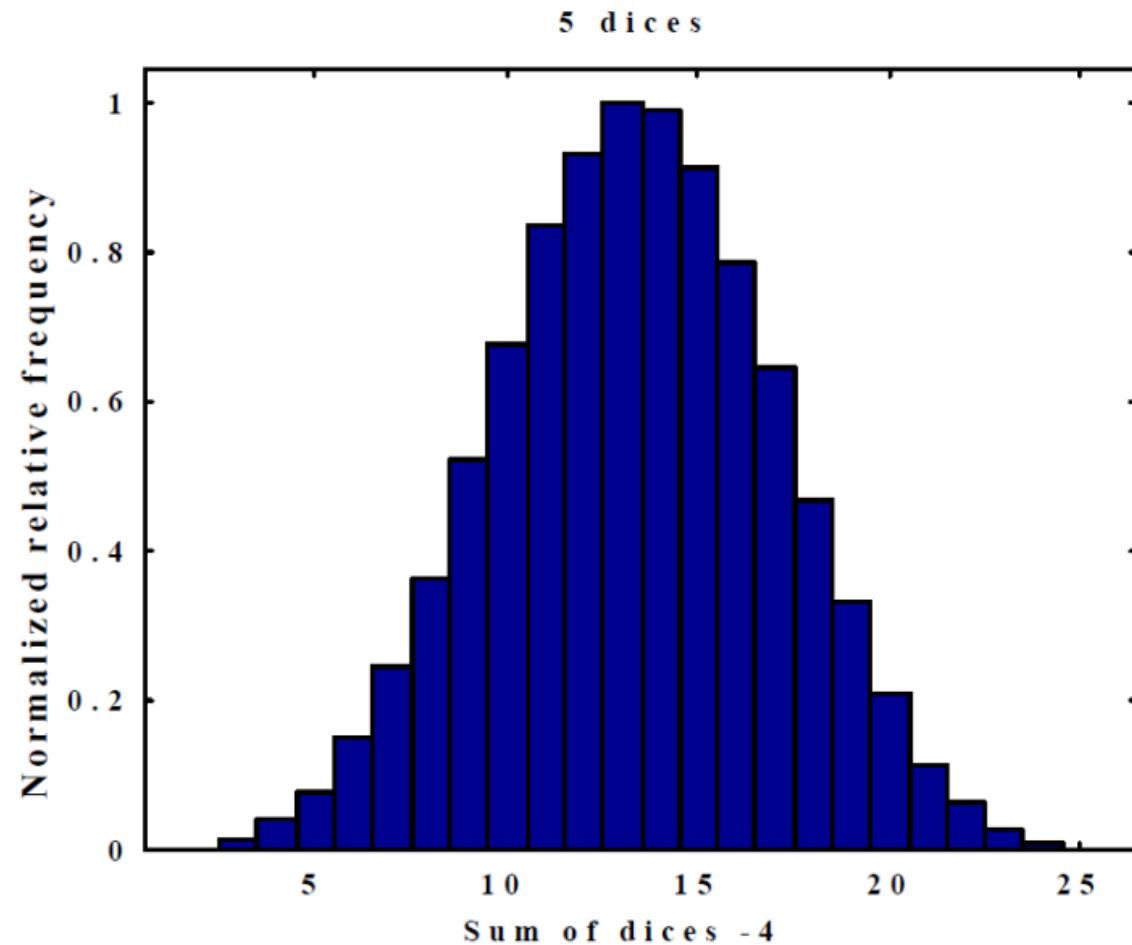
One dice sum distribution



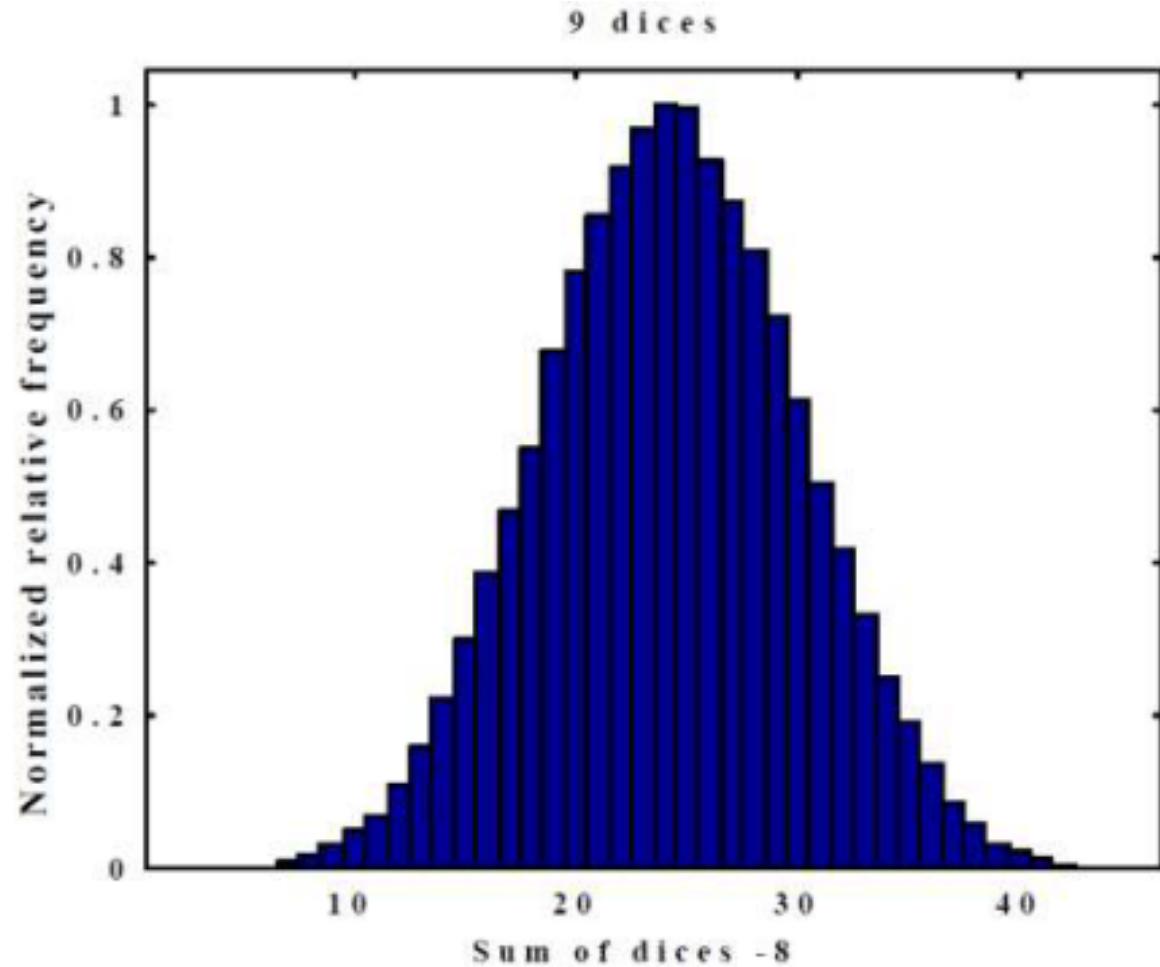
Two dices sum distribution



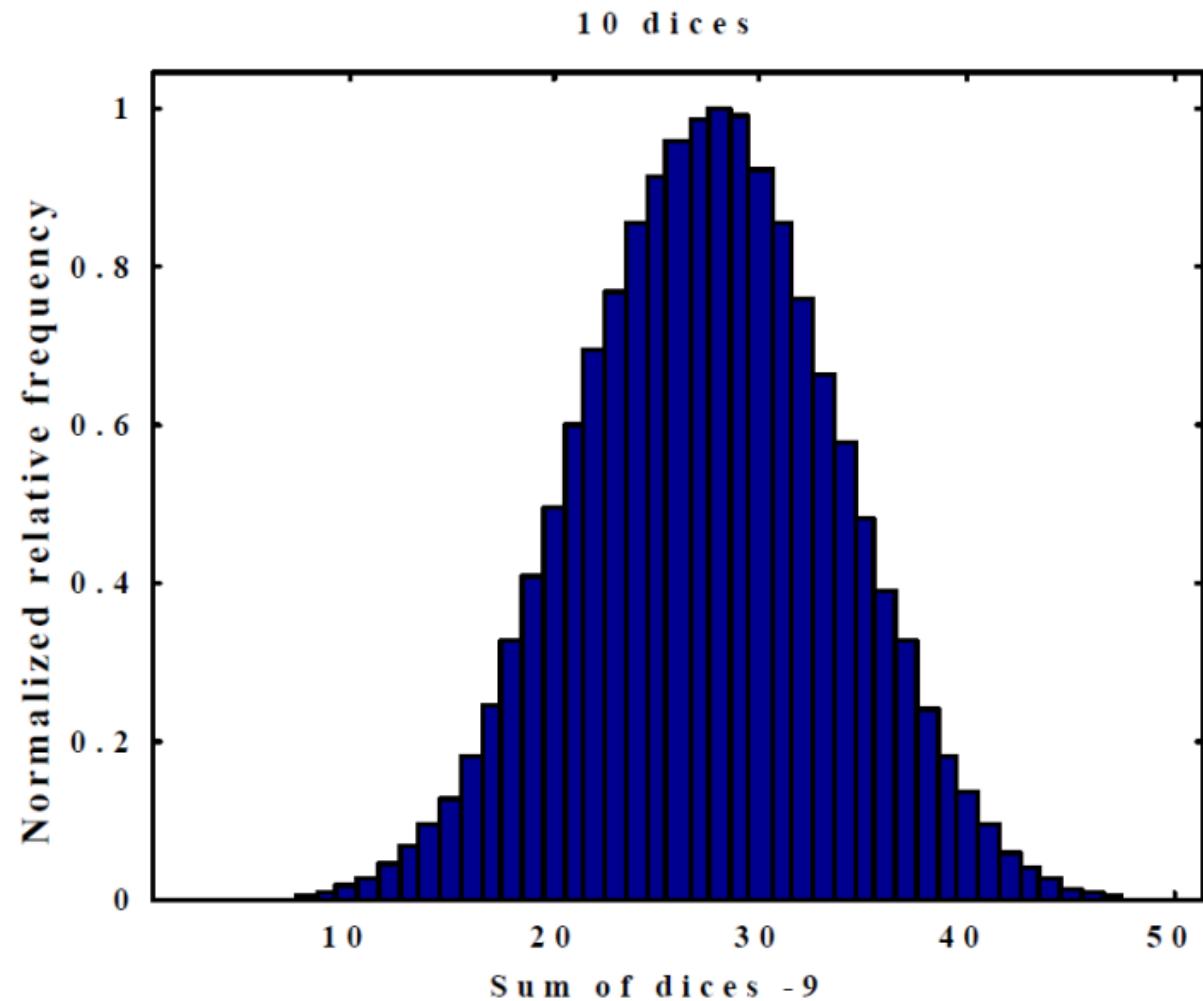
Three dices sum distribution



Nine dices sum distribution



Ten dices sum distribution



Why does ICA require signals to be non-Gaussian ?

Central limit theorem states that linear combination of given random variables is more Gaussian as compared to the original variables themselves.

So, if we have two independent Gaussian sources after mixing them, they become more Gaussian as central limit theorem state, which make it impossible to recover the original sources

Densities and linear transformations

Assuming that s could be substituted to $W \cdot x$ in density definition leads to **the following erroneous relation:** $p_x(x) = p_s(Wx)$

Let us see through an example the correct procedure to derive densities:

$$x = As \text{ with } s \in \mathbb{R}, x \in \mathbb{R}, A \in \mathbb{R}$$

$$s = Wx$$

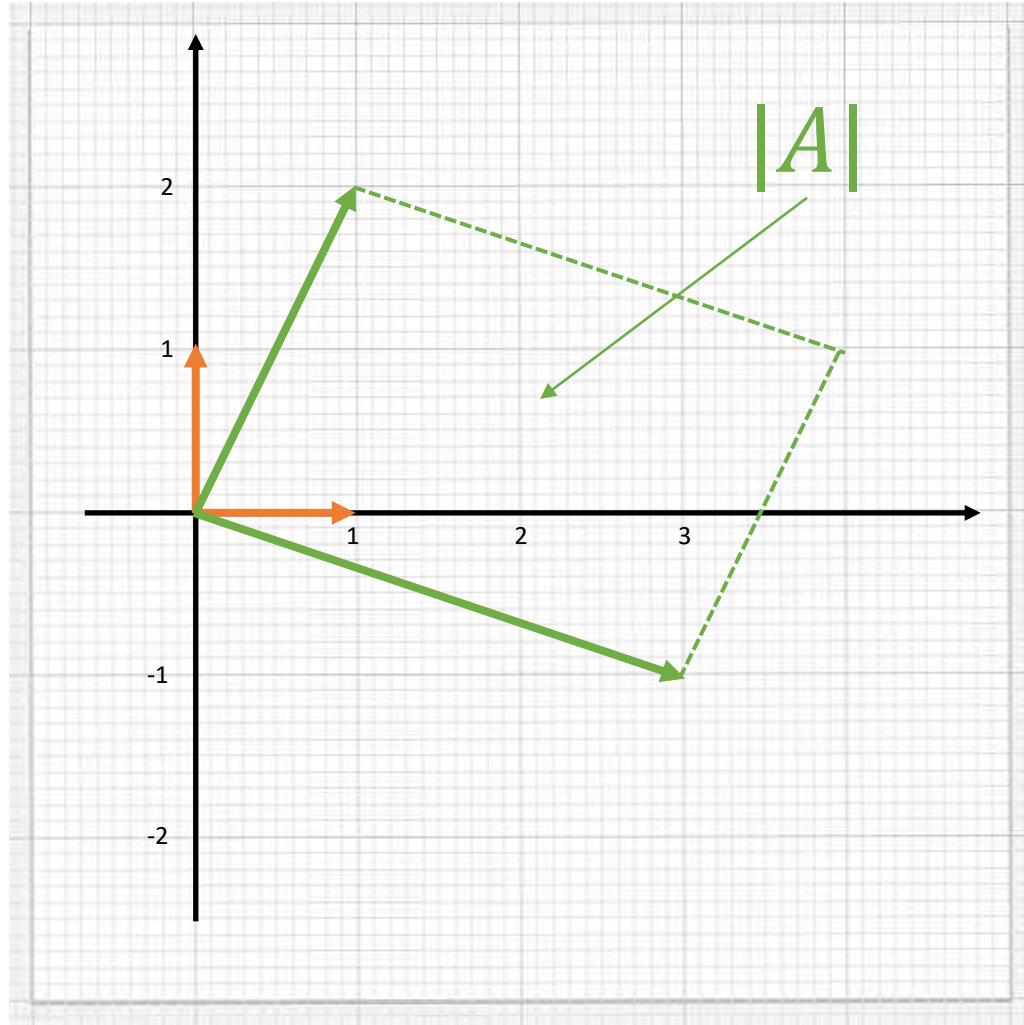
$$s \sim Uniform[0, 1] \Rightarrow p_s(s) = 1\{0 \leq s \leq 1\}$$

$$\text{If } A = 2; x = 2s \Rightarrow x \sim Uniform[0, 2]$$

$$p_x(x) = (0.5)1\{0 \leq x \leq 2\}$$

$$p_x(x) = p_s(Wx)|W|$$

$$p_x(x) = p_s(W \cdot x) \cdot |W|$$



$$A = \begin{bmatrix} 1 & 3 \\ 2 & -2 \end{bmatrix} \quad s_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad s_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$x_1 = A \cdot s_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad x_2 = A \cdot s_2 = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

$$p_x(x) = \frac{p_s(W \cdot x)}{|A|} = p_s(W \cdot x) \cdot |W|$$

ICA Algorithm

We can define the joint distribution as product of the marginal distributions:

$$p(s) = \prod_{i=1}^n p_s(s_i)$$

Now we can take advantage of the former derivation:

$$p(x) = \prod_{i=1}^n p_s(w_i^T x) \cdot |W|$$

ICA Algorithm

Now we will maximize $p(x)$. What does it mean?

We want to find the value for the unknown parameter that maximize the probability of obtaining the observed values.

But this is the definition of likelihood, hence we can compute the log likelihood of $p(x)$:

$$l(W) = \sum_{i=1}^m \left(\sum_{j=1}^n \log p_s(w_j^T x^{(i)}) + \log|W| \right)$$

Usually a high-Kurtosis model for p_s is assumed

$$p_s(s) = (1 - \tanh(s)^2) = (1 - \tanh(w_j^T x^{(i)}))^2$$

Sigmoid-based $p_s(s)$

- $p_s(s)$ is such that $P(s) = \int_{s_0}^{s_1} p_s(s)ds \in [0, \dots, 1]$
- We know that $\text{sigmoid}(s) \in [0, \dots, 1]$
- Idea: $P(s) = \text{sigmoid}(s) = g(s)$ then $p_s(s) = g'(s)$

$$l(W) = \sum_{i=1}^m \left(\sum_{j=1}^n \log g'(w_j^T x^{(i)}) + \log|W| \right)$$

ICA Algorithm

Now we know from linear algebra that the gradient of a determinant (in case W is invertible) is equal to:

$$\nabla_W = |W|(W^{-1})^T$$

Now we can derive a stochastic gradient learning rule:

$$W := W + \alpha \begin{pmatrix} \left[\begin{array}{c} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{array} \right] x^{(i)} + (W^T)^{-1} \end{pmatrix}$$

Dimensionality Reduction

Kernel PCA

PCA in a nutshell

PCA finds principal axes by working on the covariance matrix:

$$Cov = \frac{1}{m} \sum_{i=1}^m x_i x_i^T$$

Cov is positive definite, and can be diagonalized with non-negative eigenvalues.

$$\lambda v = Cov v$$

PCA in terms of dot products

$$Cov \ v = \frac{1}{m} \sum_{i=1}^m x_i x_i^T v = \lambda v$$

From the former we can compute v :

$$\begin{aligned} v &= \frac{1}{m\lambda} \sum_{i=1}^m x_i x_i^T v \\ &= \frac{1}{m\lambda} \sum_{i=1}^m (x_i \cdot v) x_i \end{aligned}$$

All the solutions v with $\lambda \neq 0$ can be expressed as $v = \sum_{i=1}^m \alpha_i x_i$

Kernel PCA

If we map our data in another space $x \in \mathbb{R}^n \rightarrow \Phi(x) \in \mathbb{R}^d$ we can write the covariance matrix as:

$$Cov = \frac{1}{m} \sum_{i=1}^m \Phi(x_i) \Phi(x_i)^T$$

When we consider its eigenvectors we have:

$$Cov v = \frac{1}{m} \sum_{i=1}^m \Phi(x_i) \Phi(x_i)^T v = \lambda v$$

$$v = \frac{1}{\lambda m} \sum_{i=1}^m \Phi(x_i) \Phi(x_i)^T v = \frac{1}{\lambda m} \sum_{i=1}^m (\Phi(x_i) \cdot v) \Phi(x_i)^T$$

Kernel PCA

All the solutions v_l with $\lambda \neq 0$ can be expressed as:

$$v_l = \sum_{i=1}^m \alpha_{li} \cdot \Phi(x_i)$$

This means that *finding the eigenvectors is equivalent to finding the alpha coefficients.*

If we substitute the result in the second last equation of the previous slide:

$$Cov \cdot v_l = \frac{1}{m} \cdot \sum_{i=1}^m \Phi(x_i) \cdot \Phi(x_i)^T \cdot \left(\sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_j) \right) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_j)$$

Kernel PCA

$$Cov \cdot v_l = \frac{1}{m} \cdot \sum_{i=1}^m \Phi(x_i) \cdot \sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_i)^T \cdot \Phi(x_j) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_j)$$

Kernel function: $k(x_i, x_j) = \Phi(x_i)^T \cdot \Phi(x_j) \in \mathbb{R}$

$$Cov \cdot v_l = \frac{1}{m} \cdot \sum_{i=1}^m \Phi(x_i) \cdot \left(\sum_{j=1}^m \alpha_{lj} \cdot k(x_i, x_j) \right) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_j)$$

Kernel PCA

$$Cov \cdot v_l = \frac{1}{m} \cdot \sum_{i=1}^m \Phi(x_i) \cdot \left(\sum_{j=1}^m \alpha_{lj} \cdot k(x_i, x_j) \right) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_j)$$

We can multiply by a data point $\Phi(x_k)^T$ that let us express the equation as composed by the same kernel value

$$Cov \cdot v_l = \frac{1}{m} \cdot \sum_{i=1}^m \Phi(x_k)^T \cdot \Phi(x_i) \cdot \left(\sum_{j=1}^m \alpha_{lj} \cdot k(x_i, x_j) \right) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot \Phi(x_k)^T \cdot \Phi(x_j)$$

$$Cov \cdot v_l = \frac{1}{m} \cdot \sum_{i=1}^m k(x_k, x_i) \cdot \sum_{j=1}^m \alpha_{lj} \cdot k(x_i, x_j) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot k(x_k, x_j)$$

Kernel PCA

$$\frac{1}{m} \cdot \sum_{i=1}^m k(x_k, x_i) \cdot \sum_{j=1}^m \alpha_{lj} \cdot k(x_i, x_j) = \lambda_l \cdot \sum_{j=1}^m \alpha_{lj} \cdot k(x_k, x_j)$$

Can be rewritten in matrix notation as

$$K^2 \cdot a_l = m \cdot \lambda_l \cdot K \cdot a_l$$

with $K \in \mathbb{R}^{d \times d}$. Since K is invertible we have

$$K \cdot a_l = m \cdot \lambda_l \cdot a_l$$

Kernel PCA

Now the equation can be normalized

$$K\alpha = \lambda\alpha$$

And we can obtain the same formulation as before defining:

- The kernel matrix K $m \times m$ in which the ij -th element is $K(x_i, x_j)$
- The vector α $m \times 1$ whose j -th element is α_j

Now we can compute the principal components in the new space by:

$$\Phi(\mathbf{x})^T \cdot \mathbf{v}_l = \sum_{i=1}^m a_{li} \cdot K(\mathbf{x}, \mathbf{x}^{(i)})$$

Kernel PCA recap and how-to

- $C^{n \times n} = \frac{1}{m} \sum_{i=1}^m \Phi(x^{(i)}) \cdot \Phi(x^{(i)})^T$ **Covariance Matrix in the projected space**

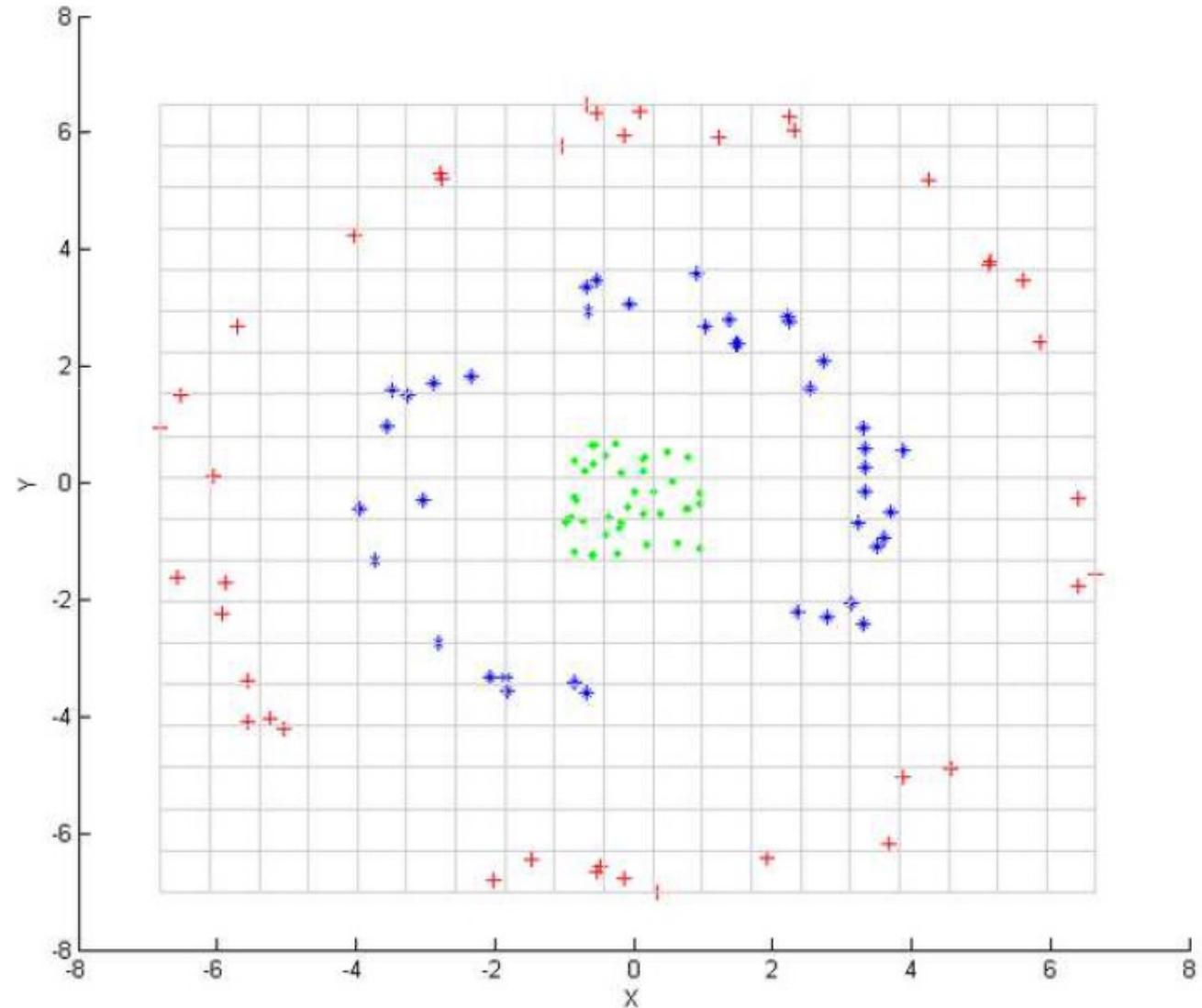
- $C \cdot v_l = \lambda_l \cdot v_l$ **Eigenvectors and eigenvalues of C**

↑ This is what we want ↑
 ↓ This is what we do ↓

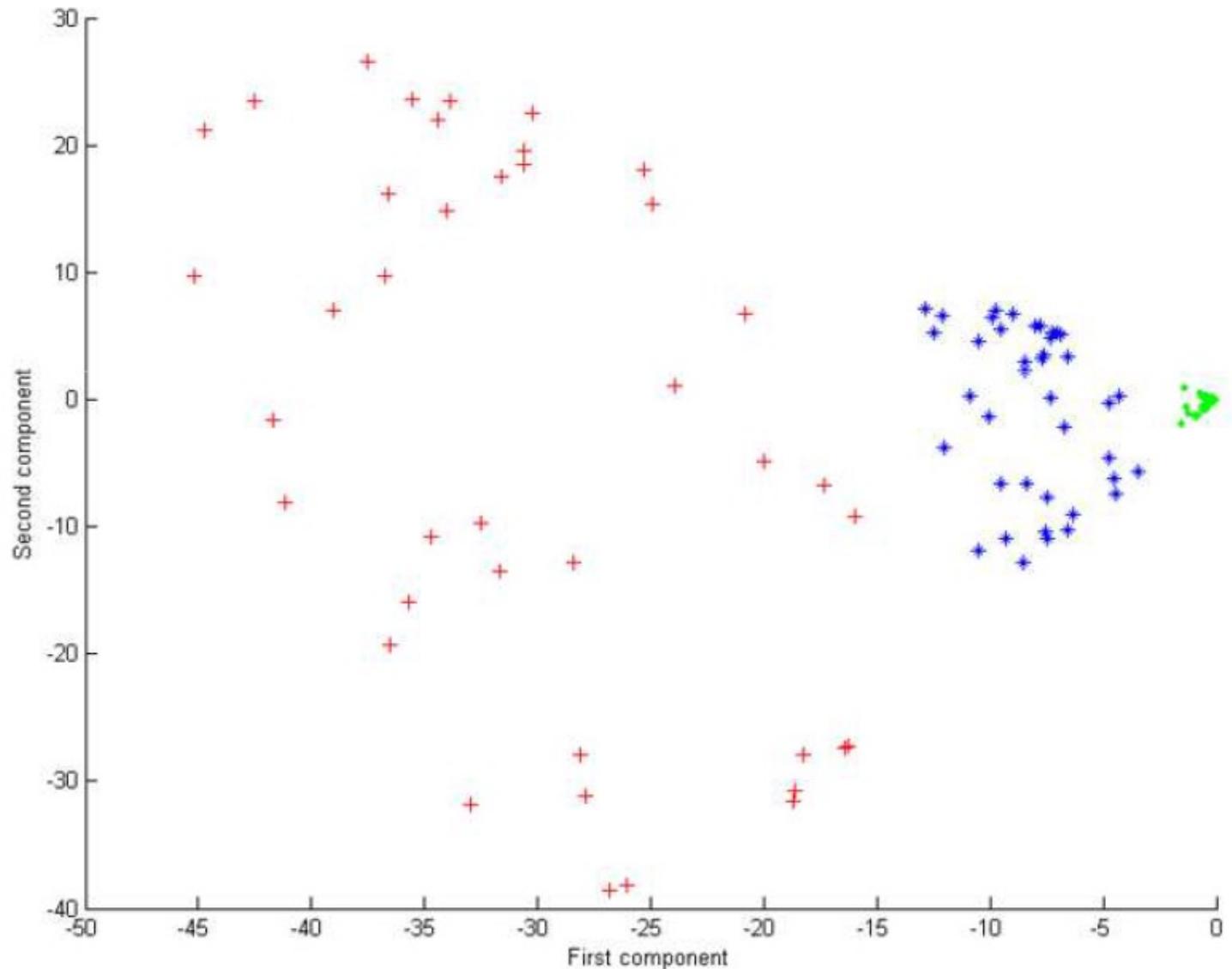
- $K^{m \times m} \cdot a_l = \lambda_l \cdot a_l$ **Eigenvectors and eigenvalues of K**

- $\Phi(x)^T \cdot v_l = \sum_{i=1}^m a_{li} \cdot K(x, x^{(i)})$ **Projection of $\Phi(x)$ on the l-th principal dimension of the Covariance Matrix**

Kernel PCA – Example 1



Kernel PCA – Example 2



Kernel PCA – Denoising images

Original data



Data corrupted with Gaussian noise



Result after linear PCA



Result after kernel PCA, Gaussian kernel



Dimensionality Reduction

Embeddings

Vector Space Models

Represent an item (e.g., word) as a vector of numbers.

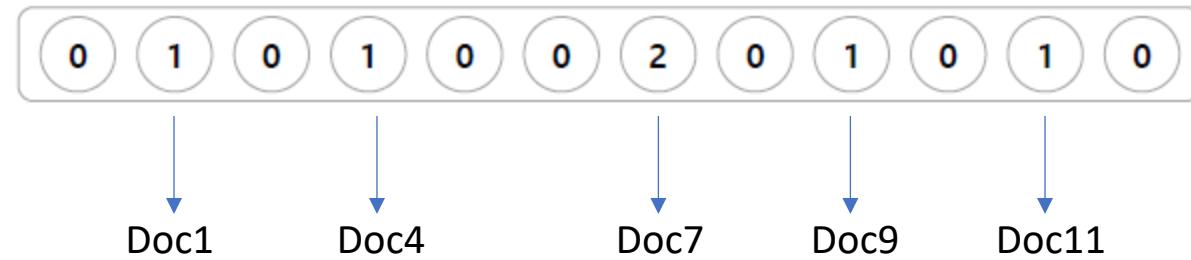
banana



Vector Space Models

Represent an item (e.g., word) as a vector of numbers.

banana

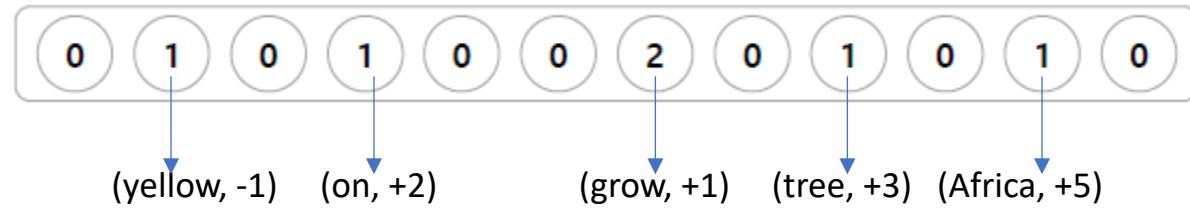


The vector can correspond to documents in which the word occurs.

Vector Space Models

Represent an item (e.g., word) as a vector of numbers.

banana



The vector can correspond to neighboring word context with respect to the following two documents.

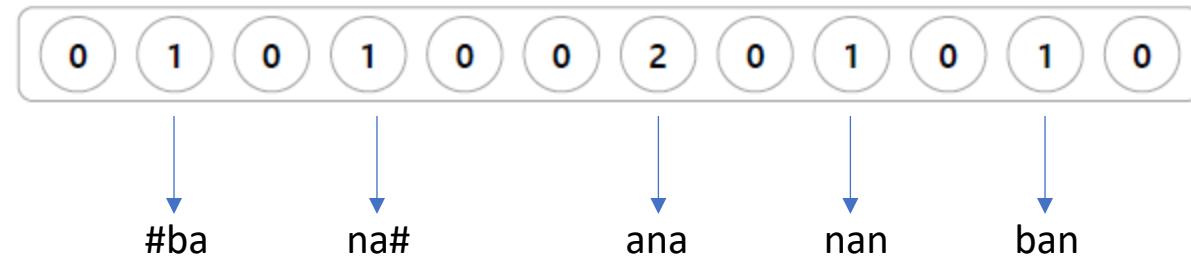
“Bananas grow best in soil that is rich and fertile”

“yellow bananas grow on trees in Africa”

Vector Space Models

Represent an item (e.g., word) as a vector of numbers.

banana



The vector can correspond to character trigrams in the word.

Notions of Relatedness

Comparing two vectors (e.g., using cosine similarity) estimates how similar the two words are. However, the notion of relatedness depends on what vector representation you have chosen for the words.

seattle similar to **denver**?

Because they are both cities.

seattle similar to **seahawks**?

Because “Seattle Seahawks”.

Important note: In previous slides I showed raw counts. They should either be normalized (e.g., using pointwise-mutual information) or (matrix) factorized.

Let's consider the following example...

We have four documents,

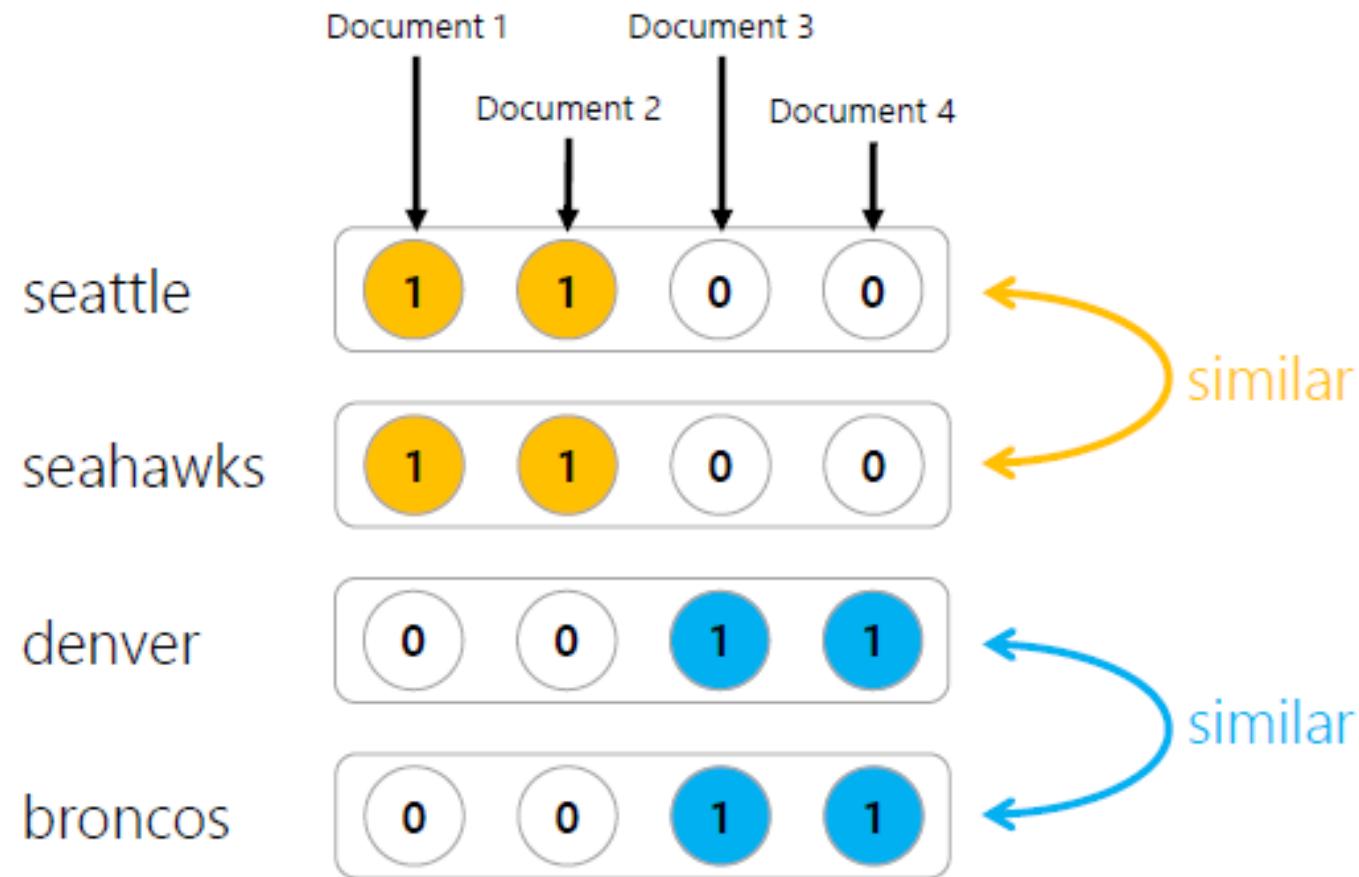
Document 1 : “seattle seahawks jerseys”

Document 2 : “seattle seahawks highlights”

Document 3 : “denver broncos jerseys”

Document 4 : “denver broncos highlights”

If we use document occurrence vectors...



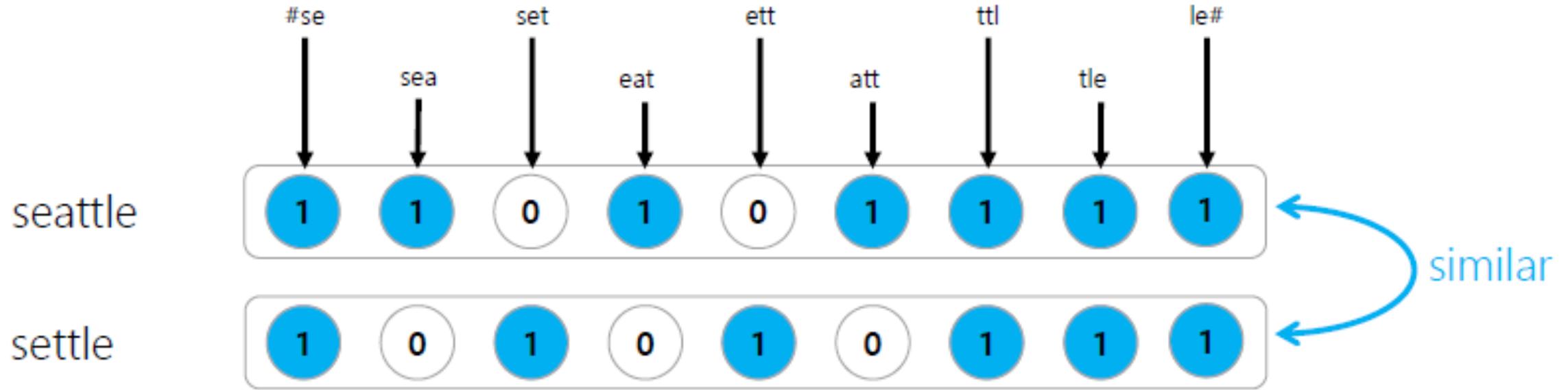
We refer to this notion of relatedness as **Topical** similarity.

If we use word context vectors...



We refer to this notion of relatedness as **Typical** (by-type) similarity

If we use character trigram vectors...



This notion of relatedness is similar to string edit-distance.

Word Analogy Task

man is to *woman* as *king* is to _____ ?

good is to *best* as *smart* is to _____ ?

china is to *beijing* as *russia* is to _____ ?

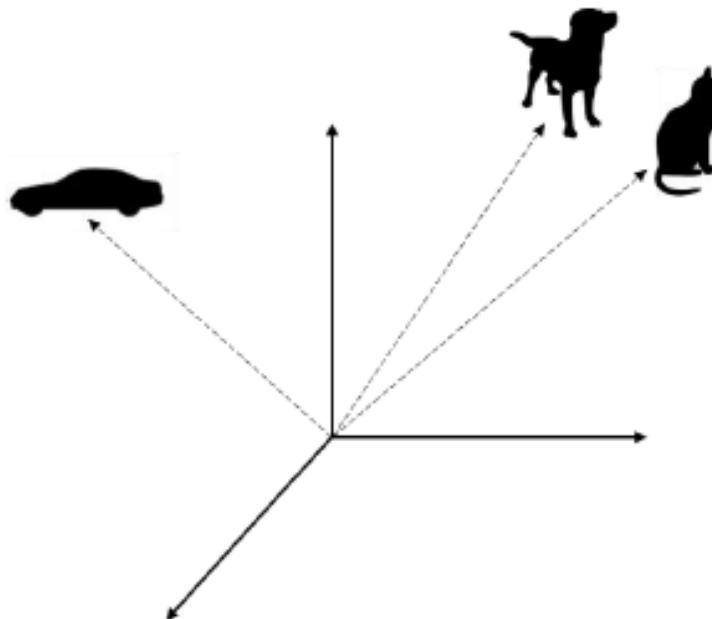
It turns out that the word-context based vector model we just learnt is good for such analogy tasks

$$[\text{king}] - [\text{man}] + [\text{woman}] \approx [\text{queen}]$$

Embeddings

The vectors we have been discussing so far are very high-dimensional (thousands, or even millions) and sparse. But there are techniques to learn lower-dimensional dense vectors for words using the same intuitions.

These dense vectors are called embeddings.



Learning Dense Embeddings

Matrix Factorization

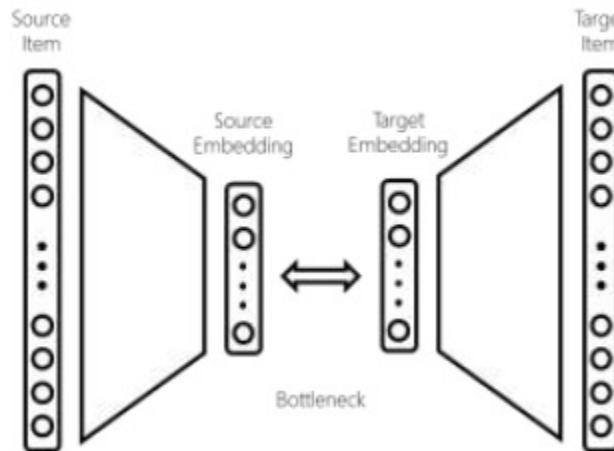
Factorize word-context matrix.

	Context ₁	Context ₁	Context _k
Word ₁				
Word ₂				
⋮				
Word _n				

E.g., LDA (Word-Document),
GloVe (Word-NeighboringWord)

Neural Networks

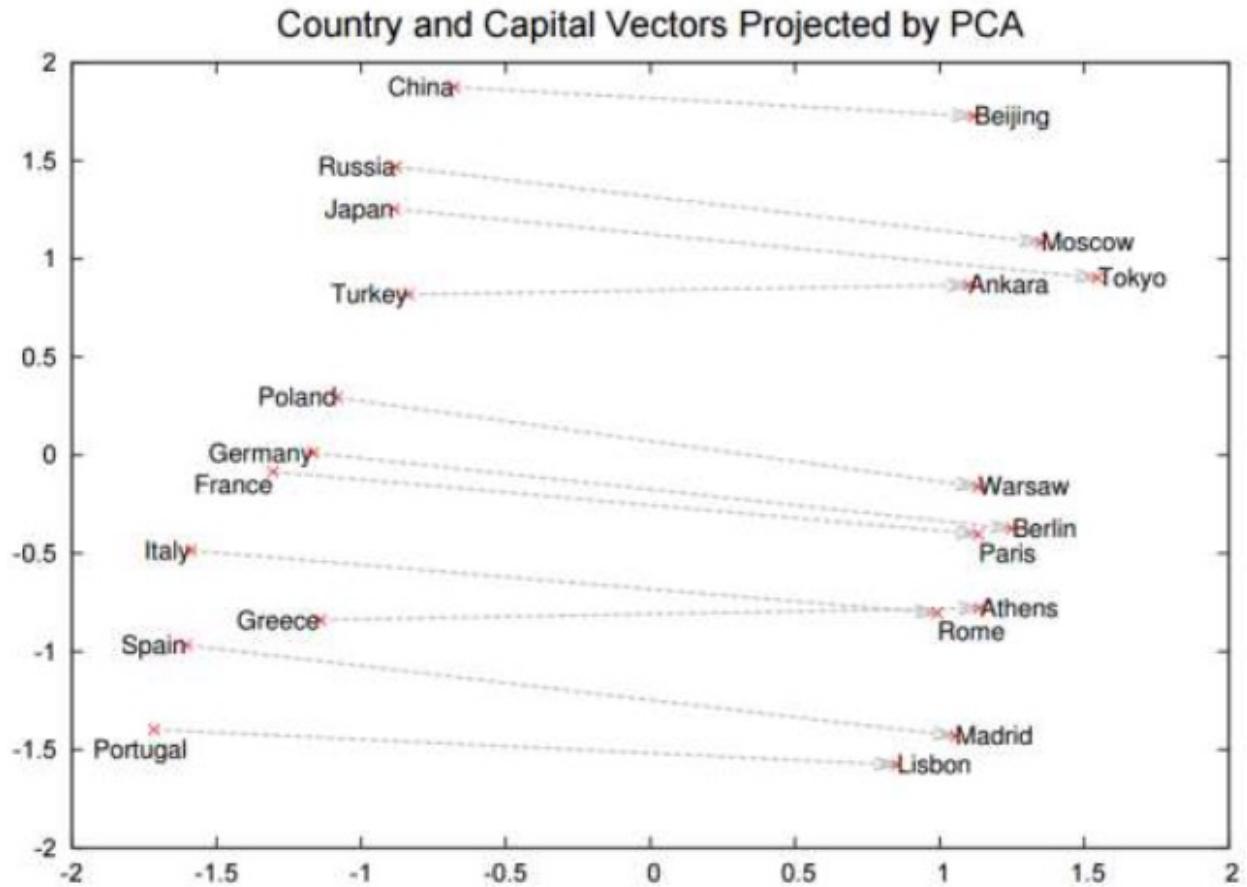
A neural network with a bottleneck,
word and context as input and output
respectively.



E.g., Word2vec (Word-NeighboringWord)

How do word analogies work?

Visually, the vector
 $\{\text{china} \rightarrow \text{beijing}\}$
turns out to be almost parallel
to the vector
 $\{\text{russia} \rightarrow \text{moscow}\}$.



Word embeddings for Document Ranking

Traditional IR uses Term matching,
→ # of times the doc says *Albuquerque*

We can use word embeddings to
compare all-pairs of query-document
terms,
→ # of terms in the doc that relate to
Albuquerque

Albuquerque is the most populous city in the U.S. state of New Mexico. The high-altitude city serves as the county seat of Bernalillo County, and it is situated in the central part of the state, straddling the Rio Grande. The city population is 557,169 as of the July 1, 2014, population estimate from the United States Census Bureau, and ranks as the 32nd-largest city in the U.S. The Metropolitan Statistical Area (or MSA) has a population of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.

Passage about Albuquerque

Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in *Albuquerque*, New Mexico in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.

Passage not about Albuquerque

Dimensionality Reduction

Embeddings – Word2vec

Word2vec

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words/Context of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

The context represents the meaning of a word

Word2vec

- The context represents the meaning of a word

«*Rome is the capital of the country Italy*»

«*London is the capital of the country UK*»

Word2vec

- The context represents the meaning of a word

«*Rome* ~~is the~~ capital ~~of the~~ country *Italy*»

«*London* ~~is the~~ capital ~~of the~~ country *UK*»

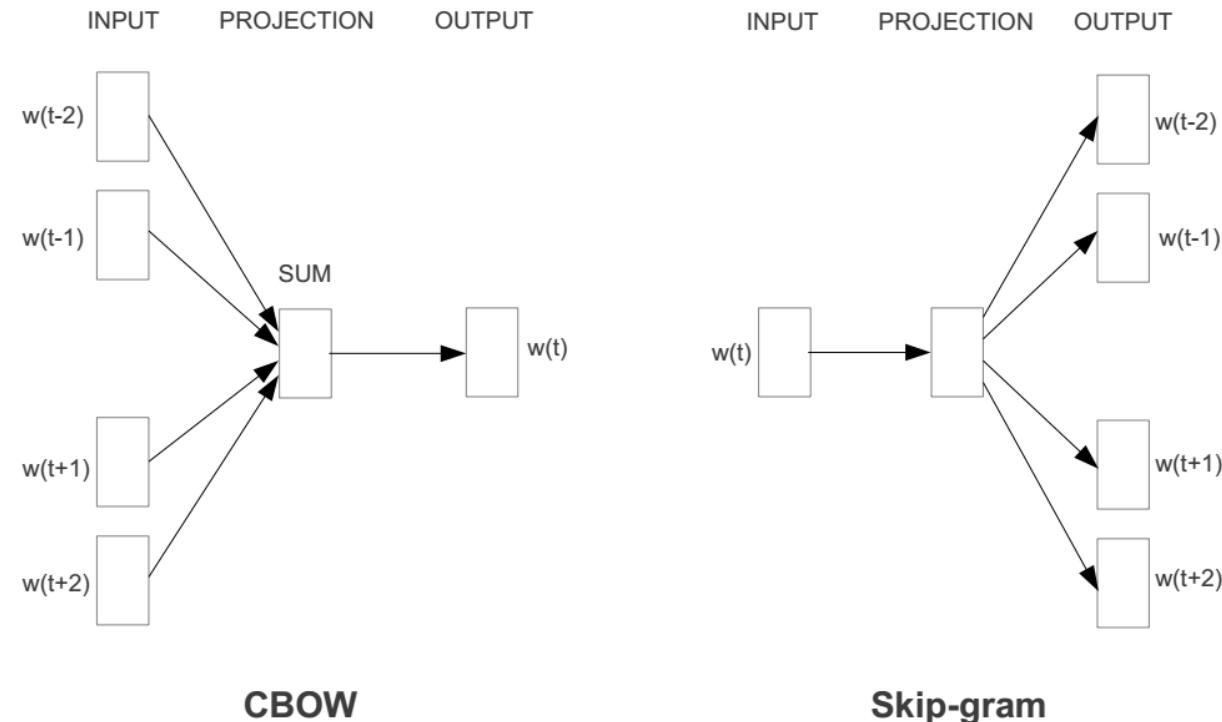
Rome is similar to *London* as they share the same context *capital*

Italy is similar to *UK* as they share the same context *country*

Represent the meaning of word

2 basic (quasi) Neural Network models:

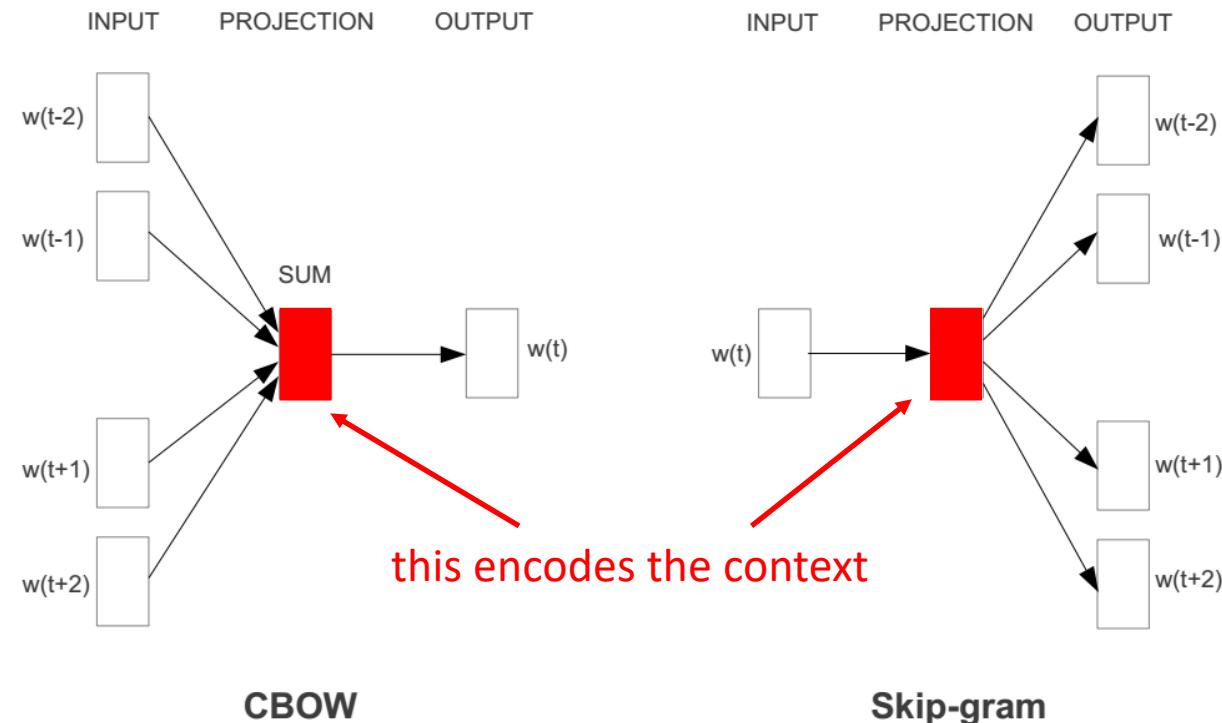
- Continuous Bag of Word (CBOW): uses a window of words to predict the middle word
- Skip-gram (SG): uses a word to predict the surrounding ones in window.



Represent the meaning of word

2 basic (quasi) Neural Network models:

- Continuous Bag of Word (CBOW): uses a window of words to predict the middle word
- Skip-gram (SG): uses a word to predict the surrounding ones in window.



Word2vec – Continuous Bag of Word

“The cat sat on the floor”

“I always sat on my favorite chair”

“Just sat on the sofa”

Input	Output
the	cat
cat	sat
sat	on
the	floor
I	always
always	sat
on	my
my	favorite
...	...

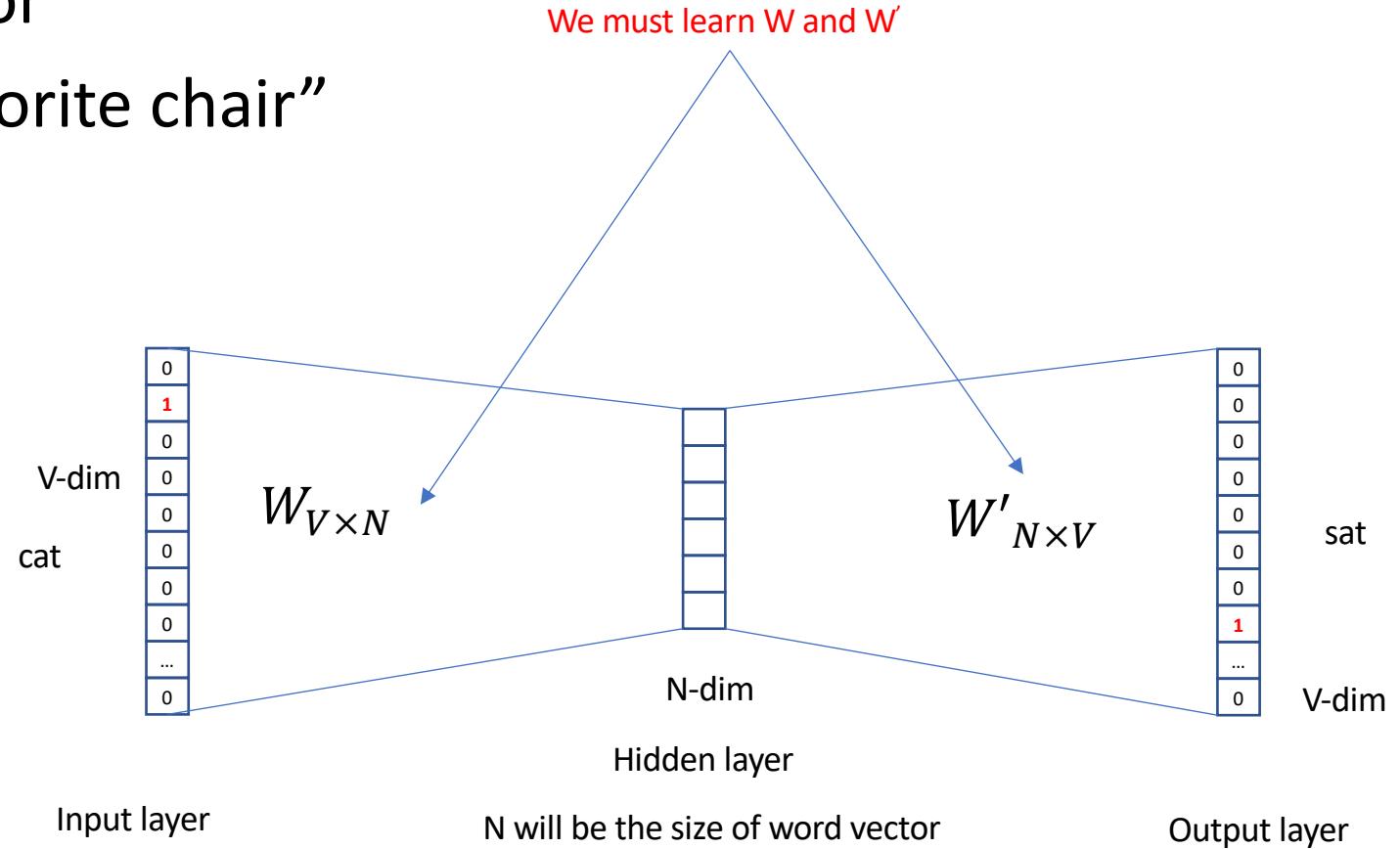
the	cat	sat	on	floor	I	always	my	favorite	...
1	0	0	0	0	0	0	0	0	...
0	1	0	0	0	0	0	0	0	...
0	0	1	0	0	0	0	0	0	...
0	0	0	1	0	0	0	0	0	...
0	0	0	0	1	0	0	0	0	...
0	0	0	0	0	1	0	0	0	...
0	0	0	0	0	0	1	0	0	...
0	0	0	0	0	0	0	0	1	...

Word2vec – Continuous Bag of Word

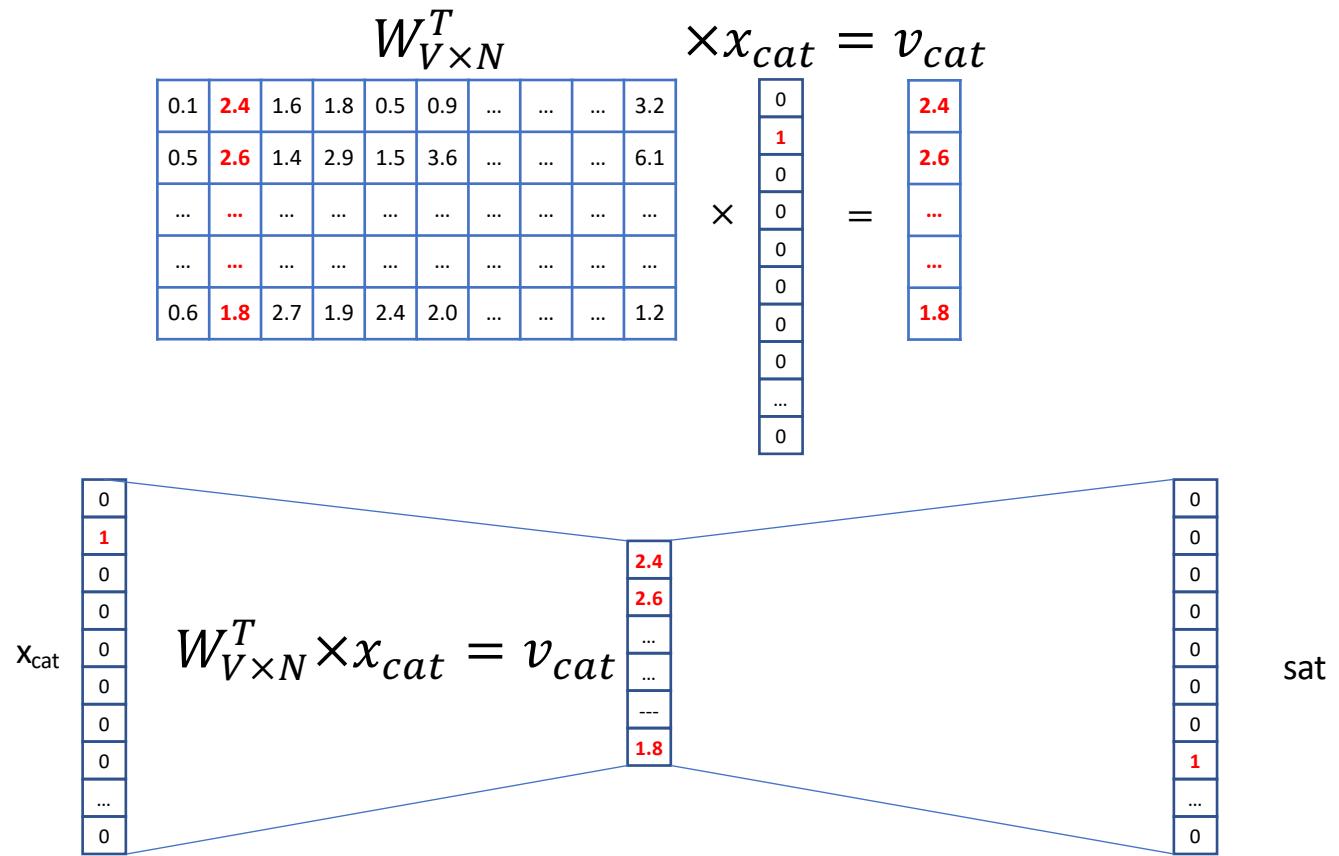
“The cat sat on the floor”

“I always sat on my favorite chair”

“Just sat on the sofa”



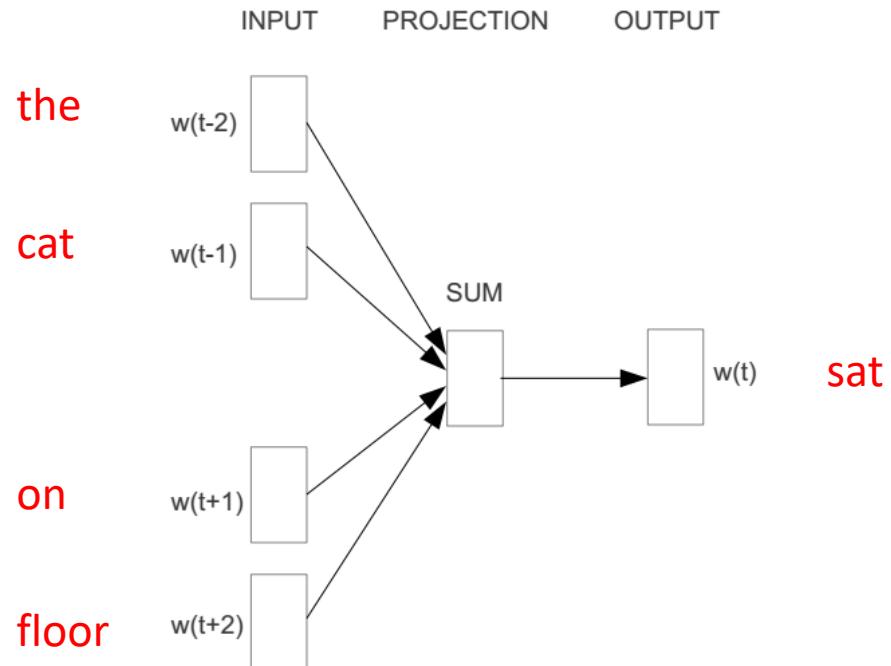
Word2vec – Continuous Bag of Word



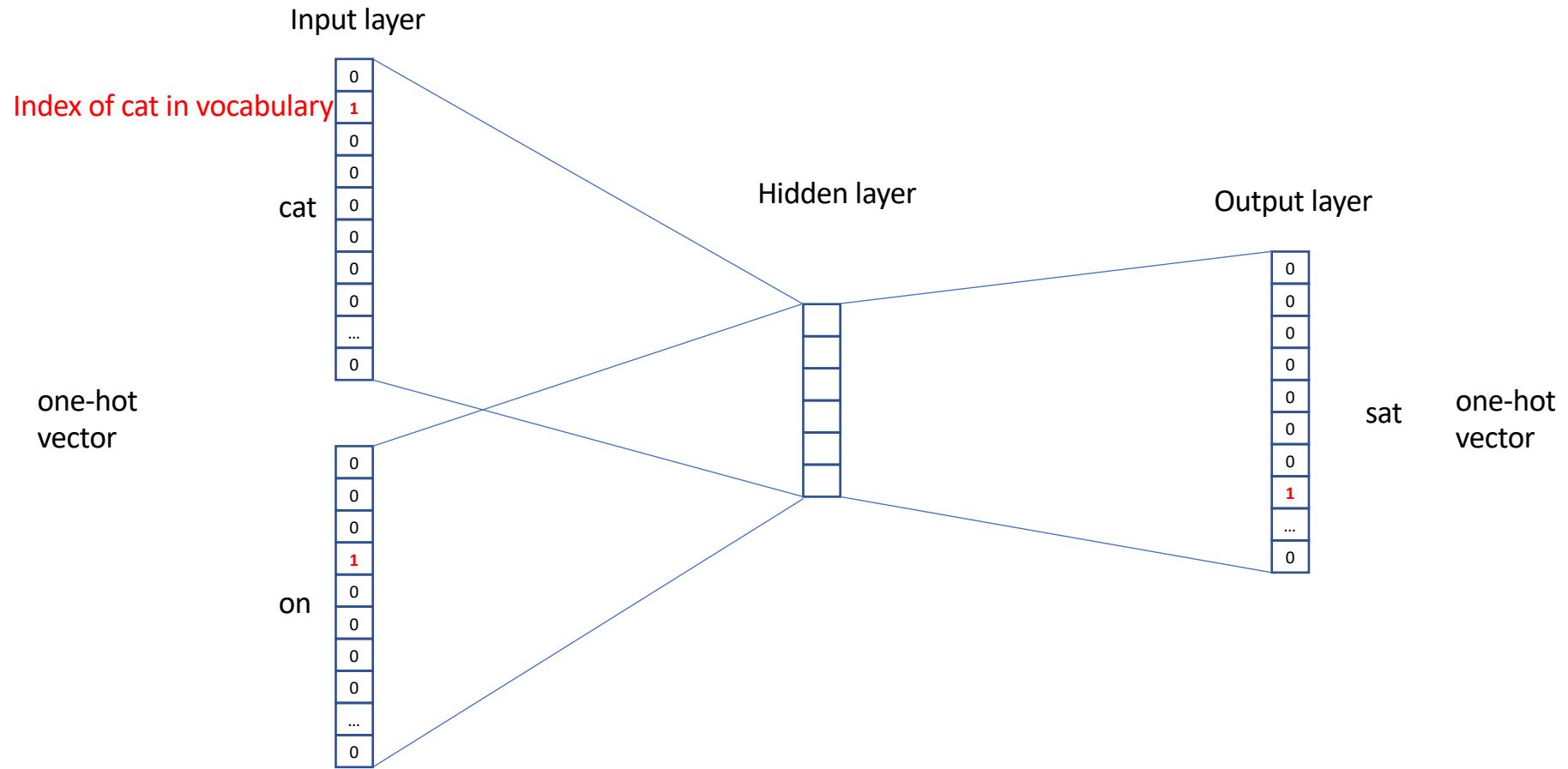
Word2vec – Continuous Bag of Word

E.g. “The cat sat on floor”

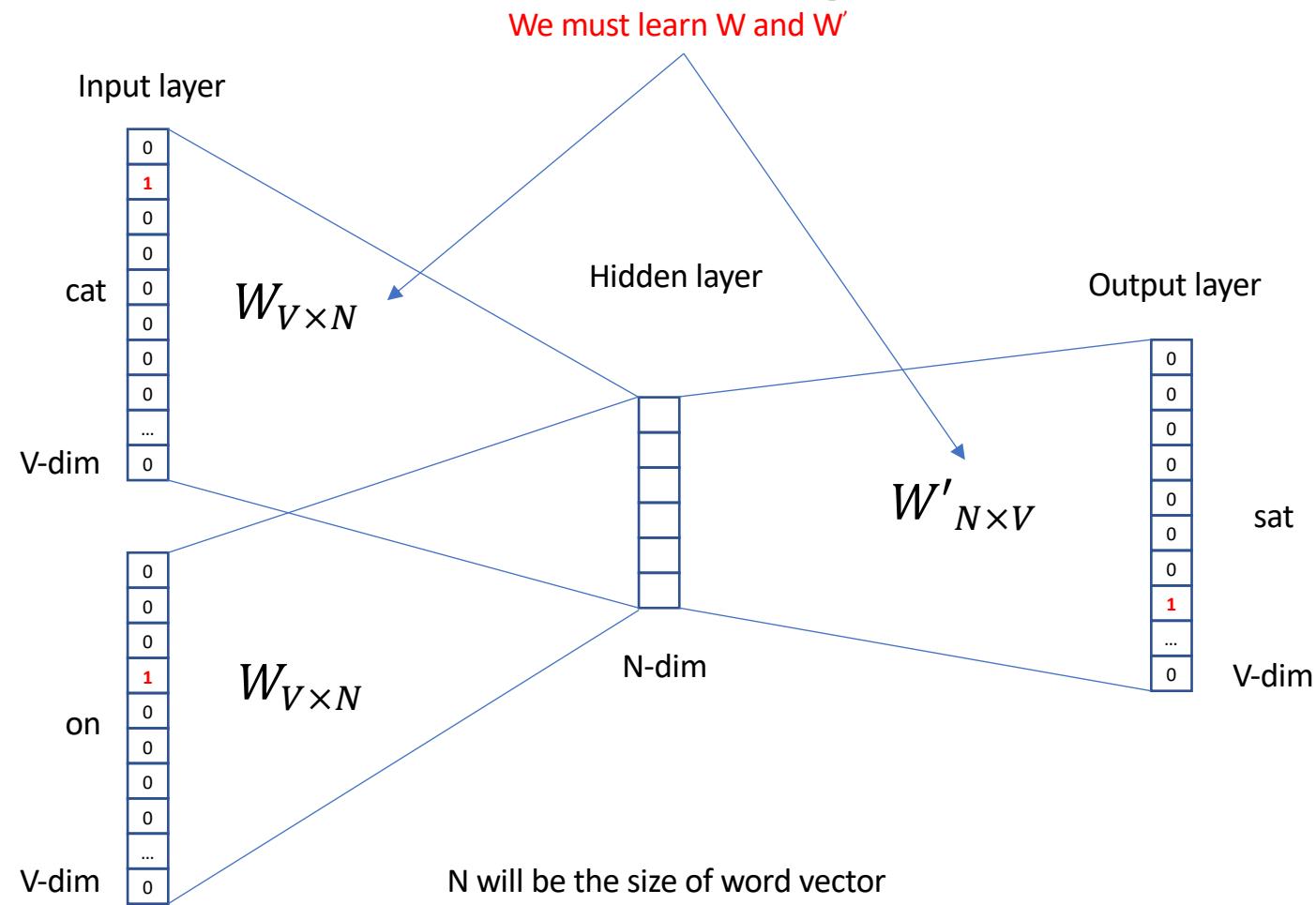
- Window size = 2



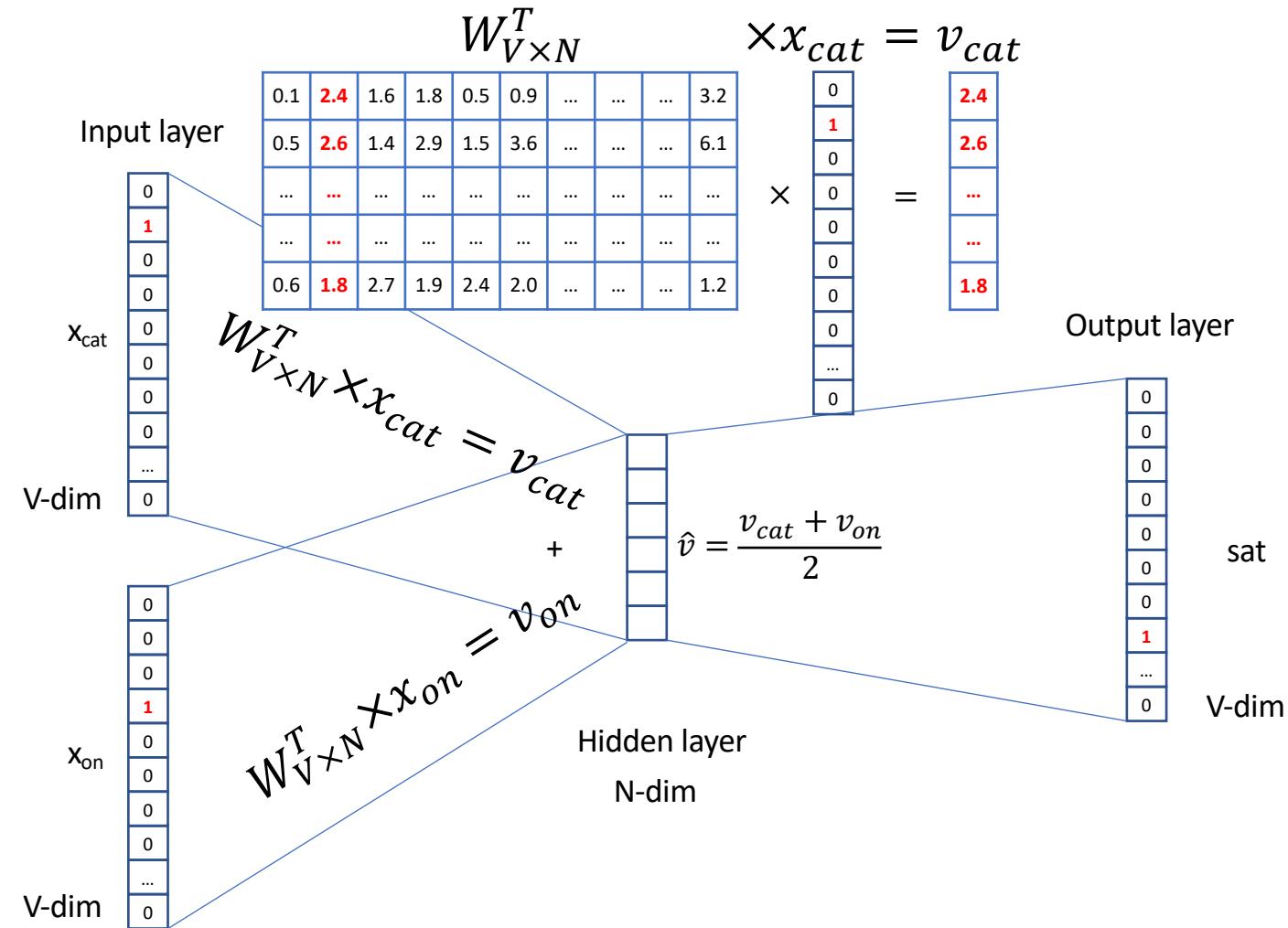
Word2vec – Continuous Bag of Word



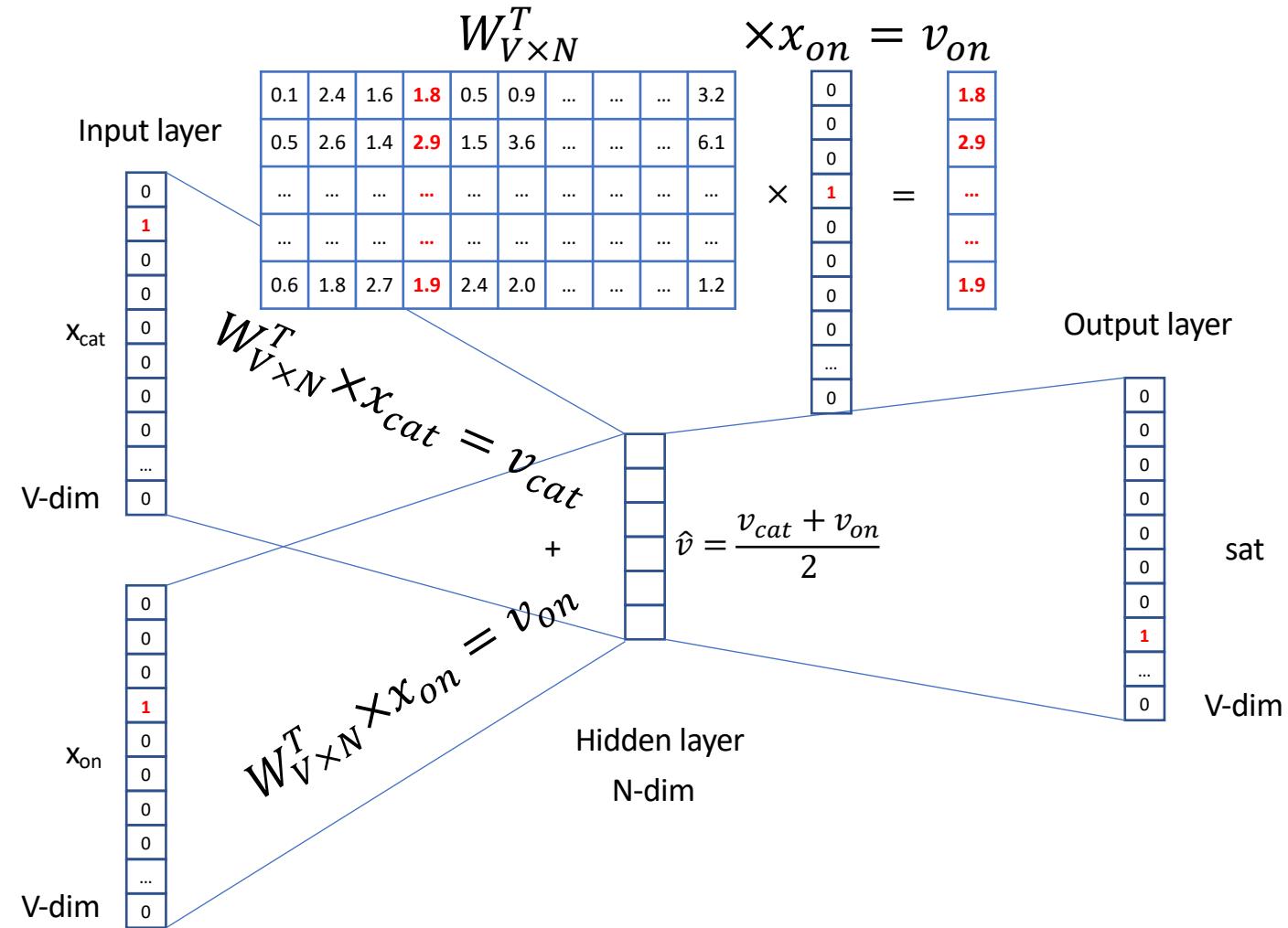
Word2vec – Continuous Bag of Word



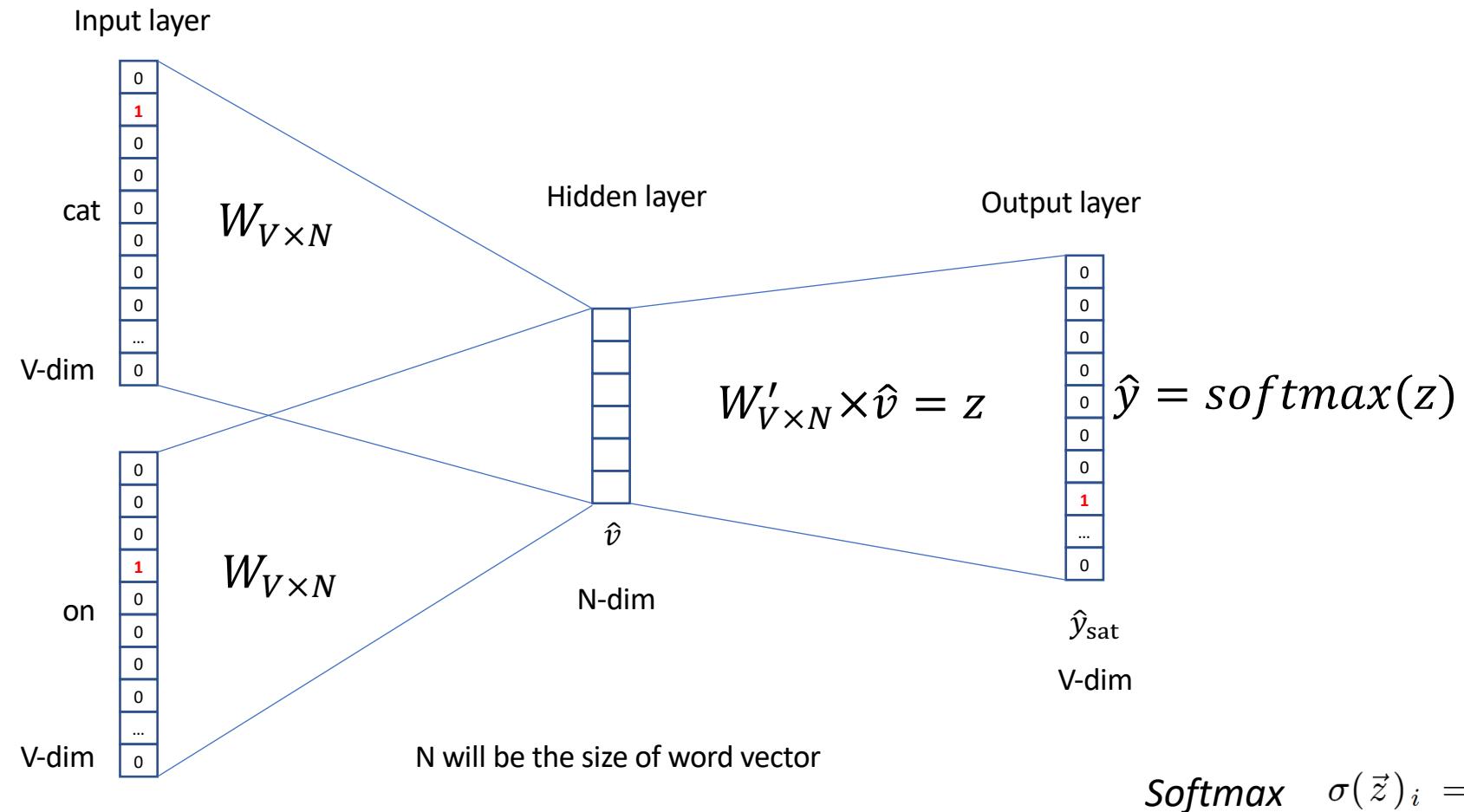
Word2vec – Continuous Bag of Word



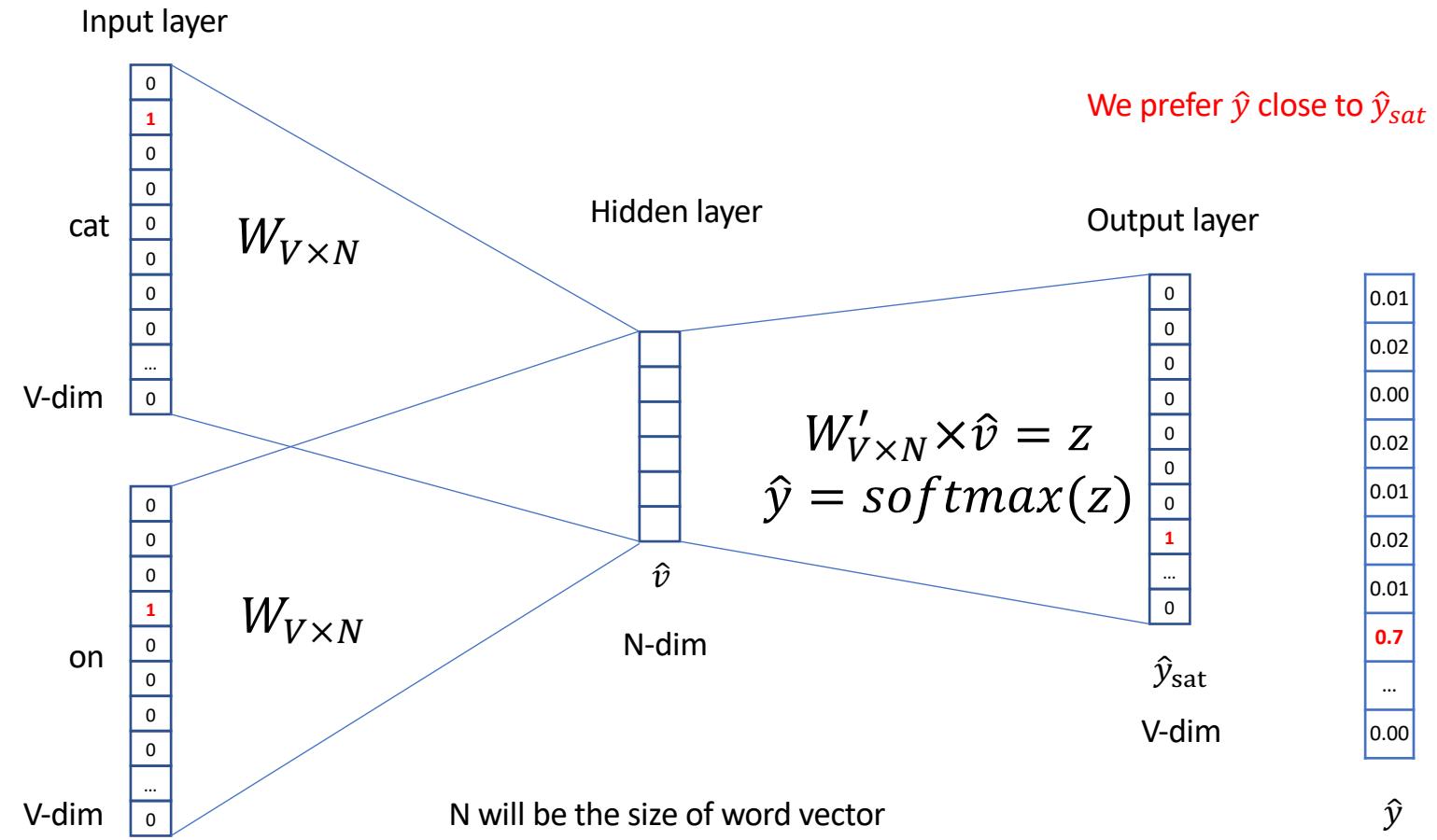
Word2vec – Continuous Bag of Word



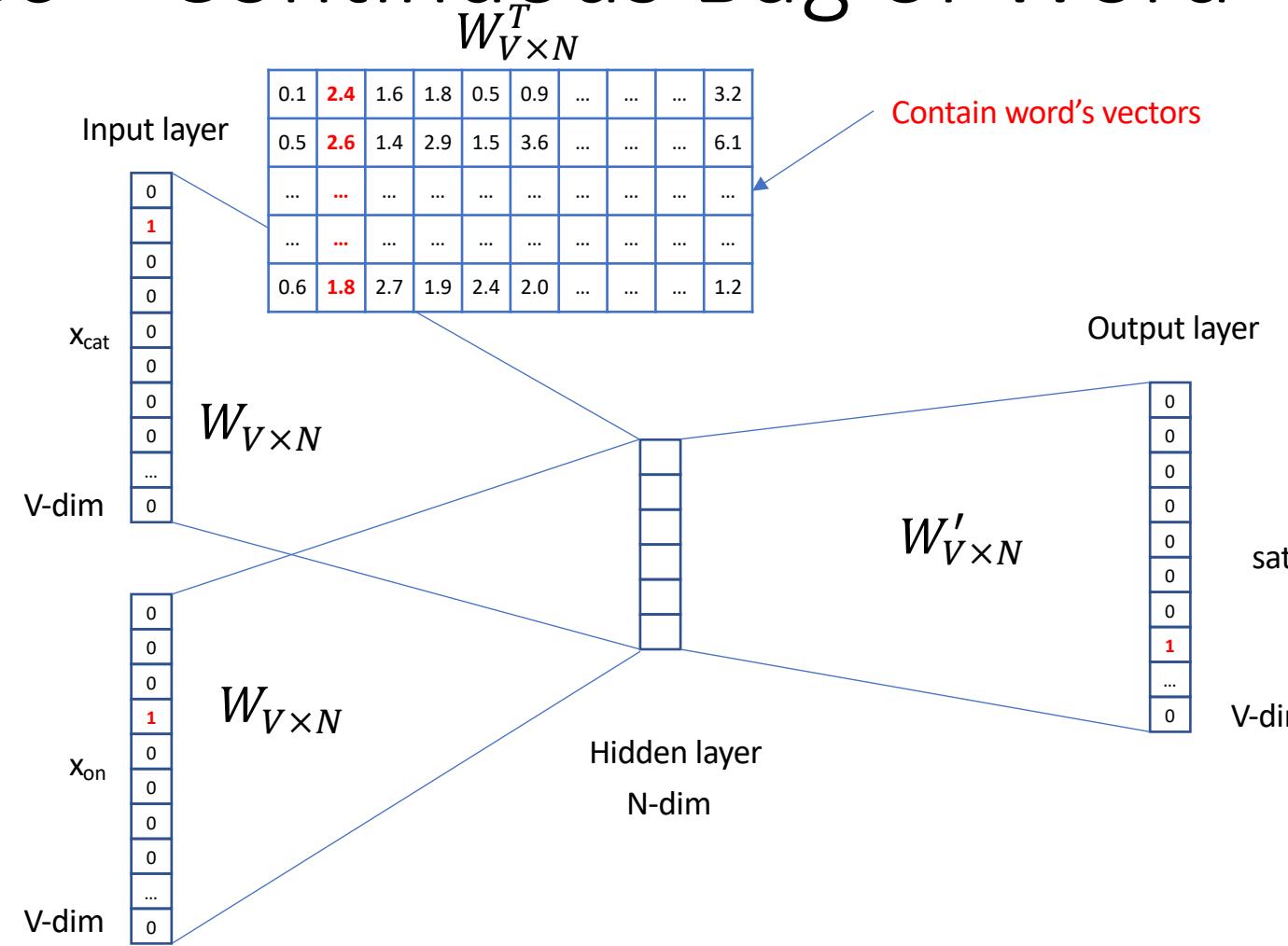
Word2vec – Continuous Bag of Word



Word2vec – Continuous Bag of Word



Word2vec – Continuous Bag of Word



Dimensionality Reduction

Autoencoders

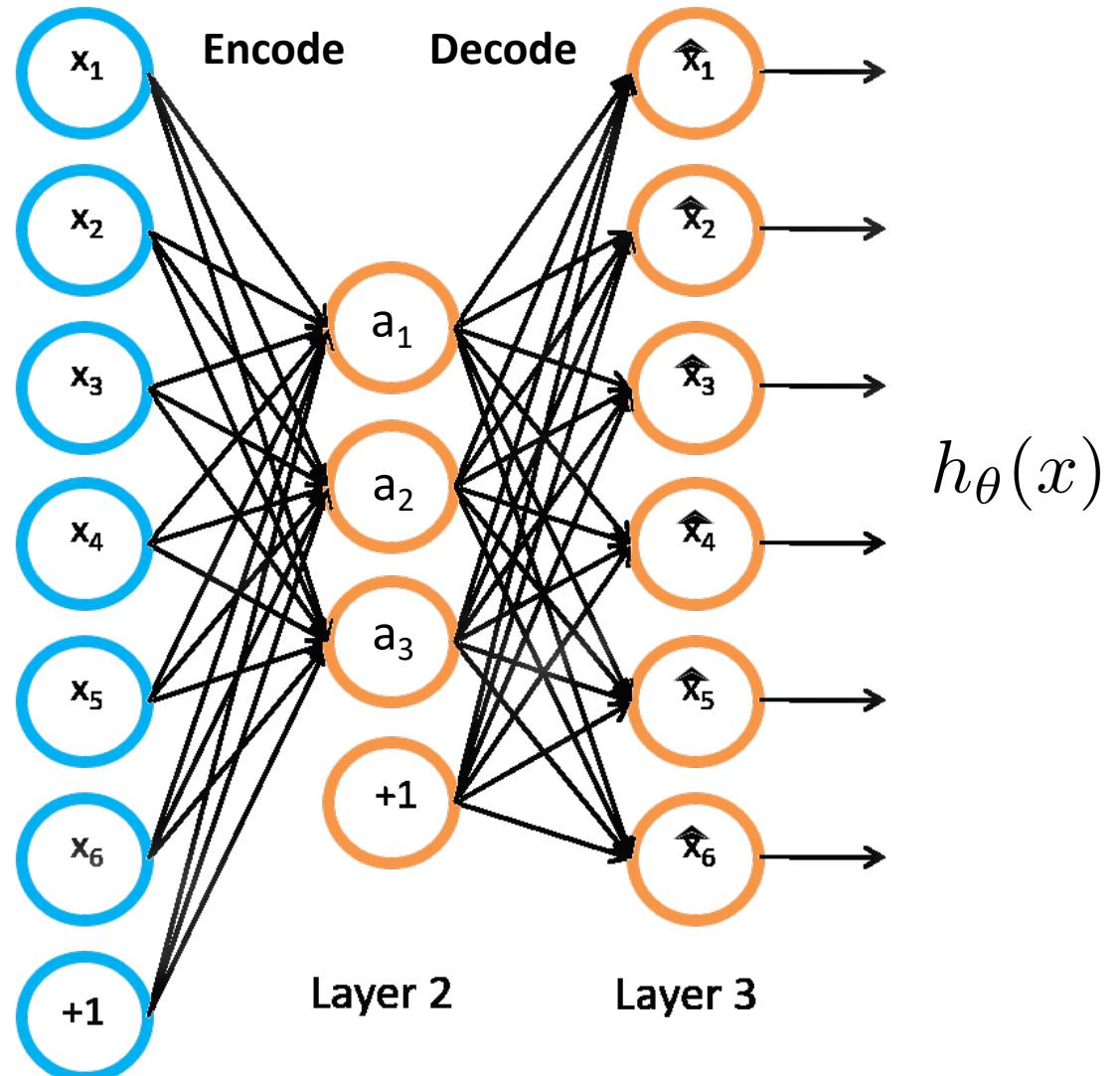
Autoencoder

Network is trained to output the input (learn identity function).

$$h_{\theta}(x) \approx x$$

Trivial solution unless:

- Constrain number of units in Layer 2 (learn a compressed representation) [Autoencoder]
- Constrain Layer 2 to be sparse [Sparse Autoencoder]



Autoencoder as Dimensionality Reduction algorithm

An autoencoder takes an input $x \in [0, 1]^d$ and first maps it (with an encoder) to a hidden representation $y \in [0, 1]^{d'}$ through a deterministic mapping

$$y = s(Wx + b)$$

where s is a non-linear activation function (such as sigmoid).

y is mapped back (with a decoder) into a reconstruction z of the same shape as x ,

$$z = s(W'y + b')$$

z is seen as a prediction of x .

Training a sparse Autoencoder

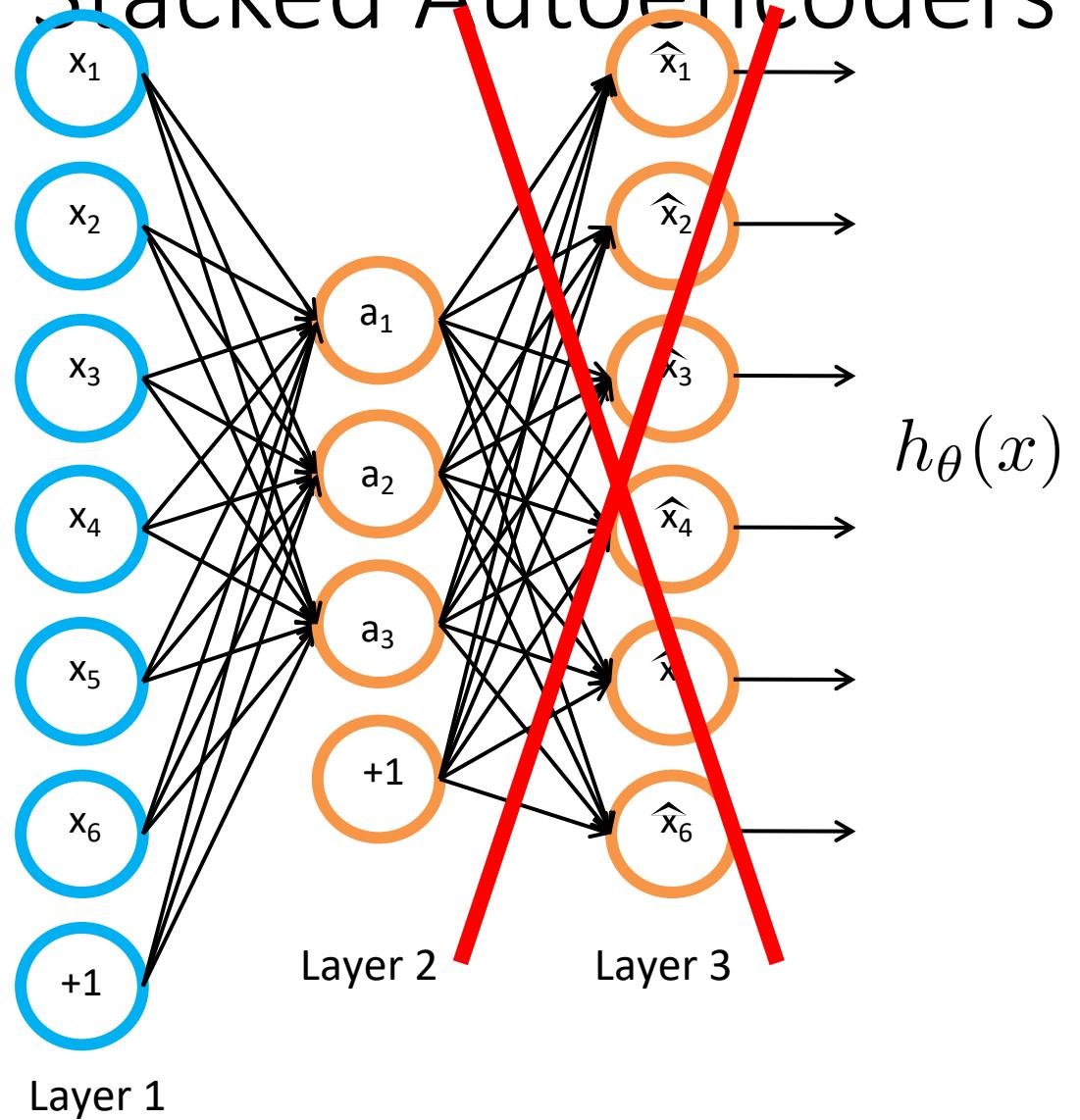
What we want to train a sparse hidden representation is penalizing the values assigned to the hidden neurons:

$$\min_{\theta} \|h_{\theta}(x) - x\|^2 + \lambda \sum_i |a_i|$$

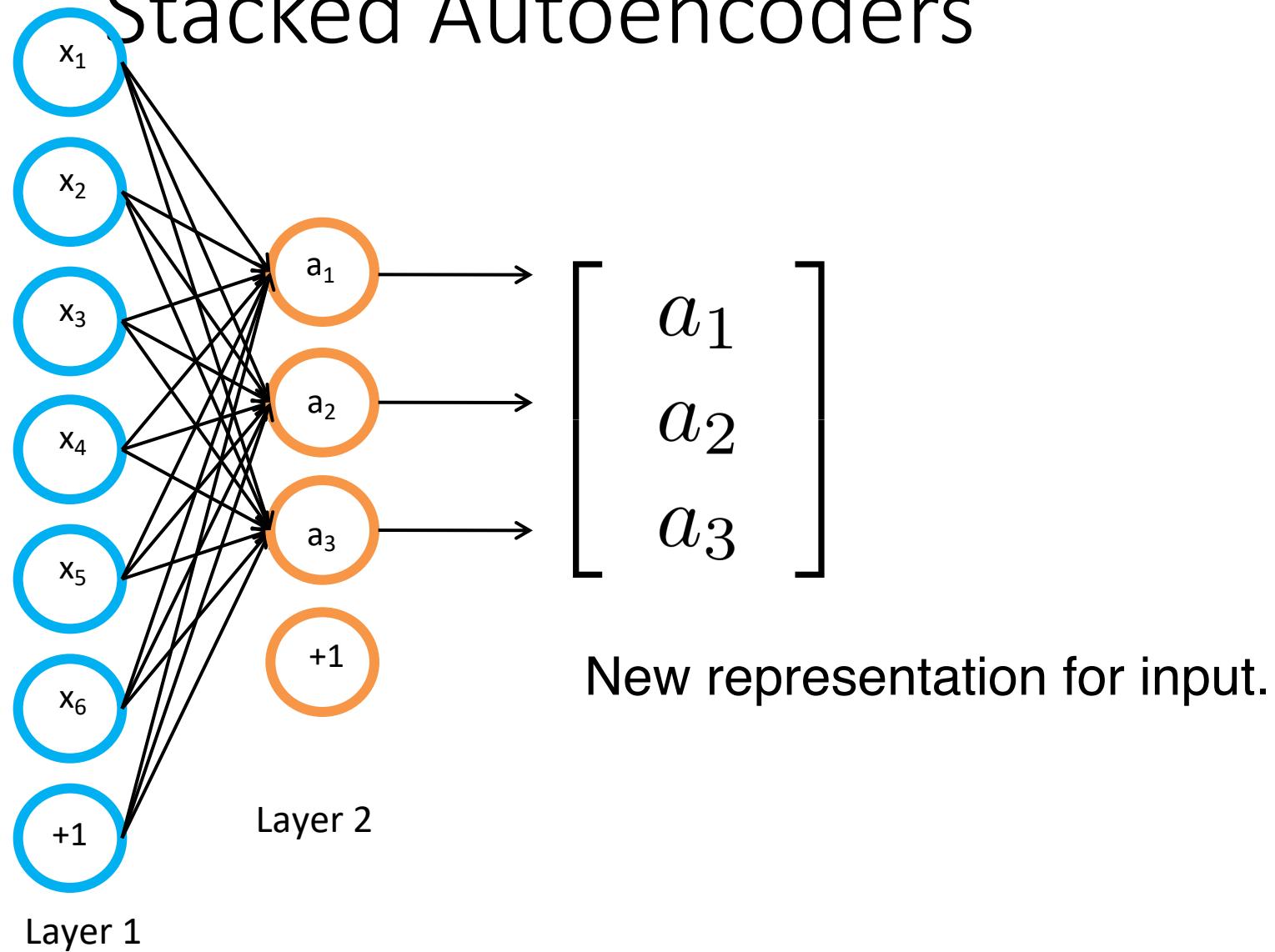
Reconstruction
error term

L1 sparsity term

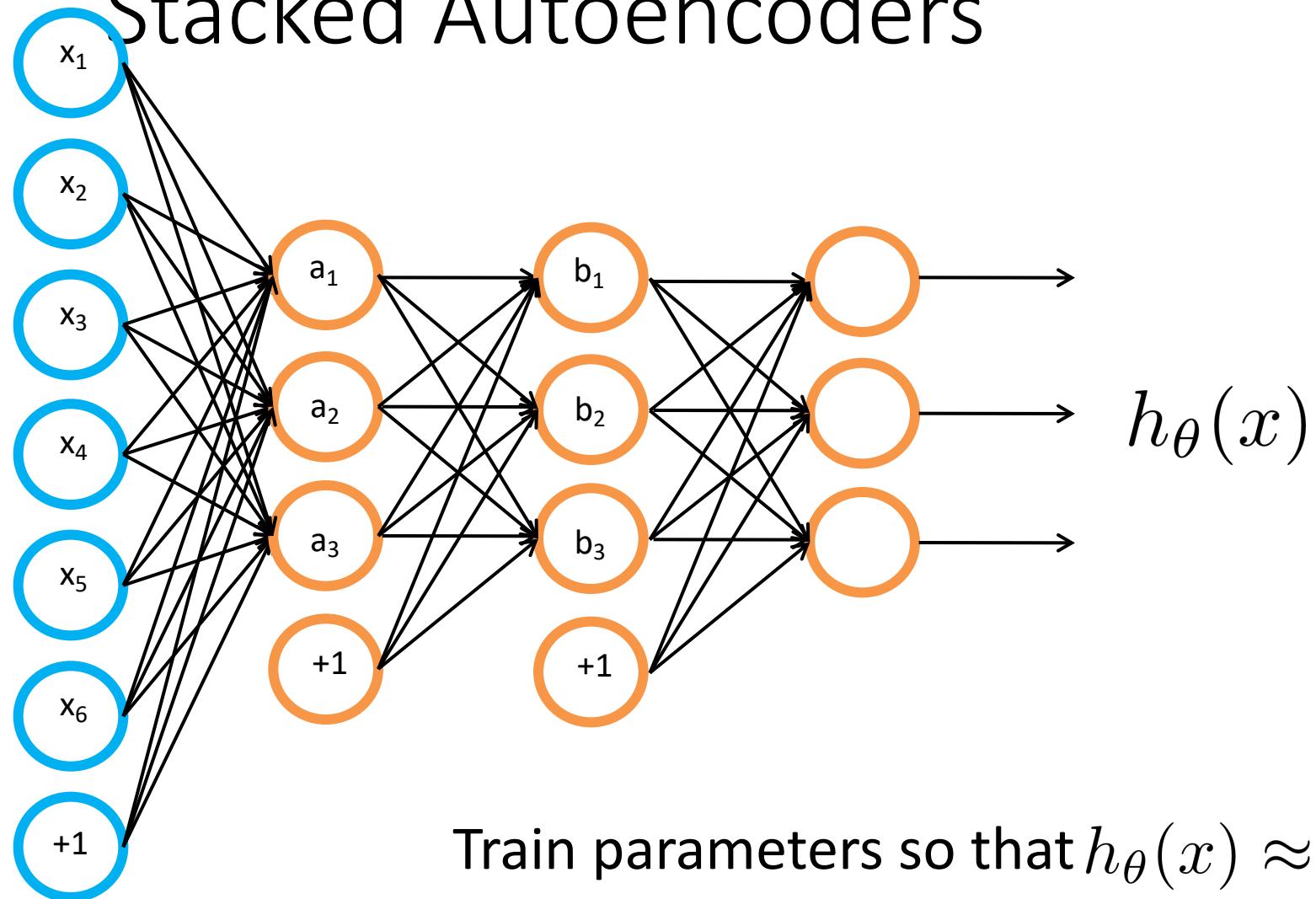
Stacked Autoencoders



Stacked Autoencoders



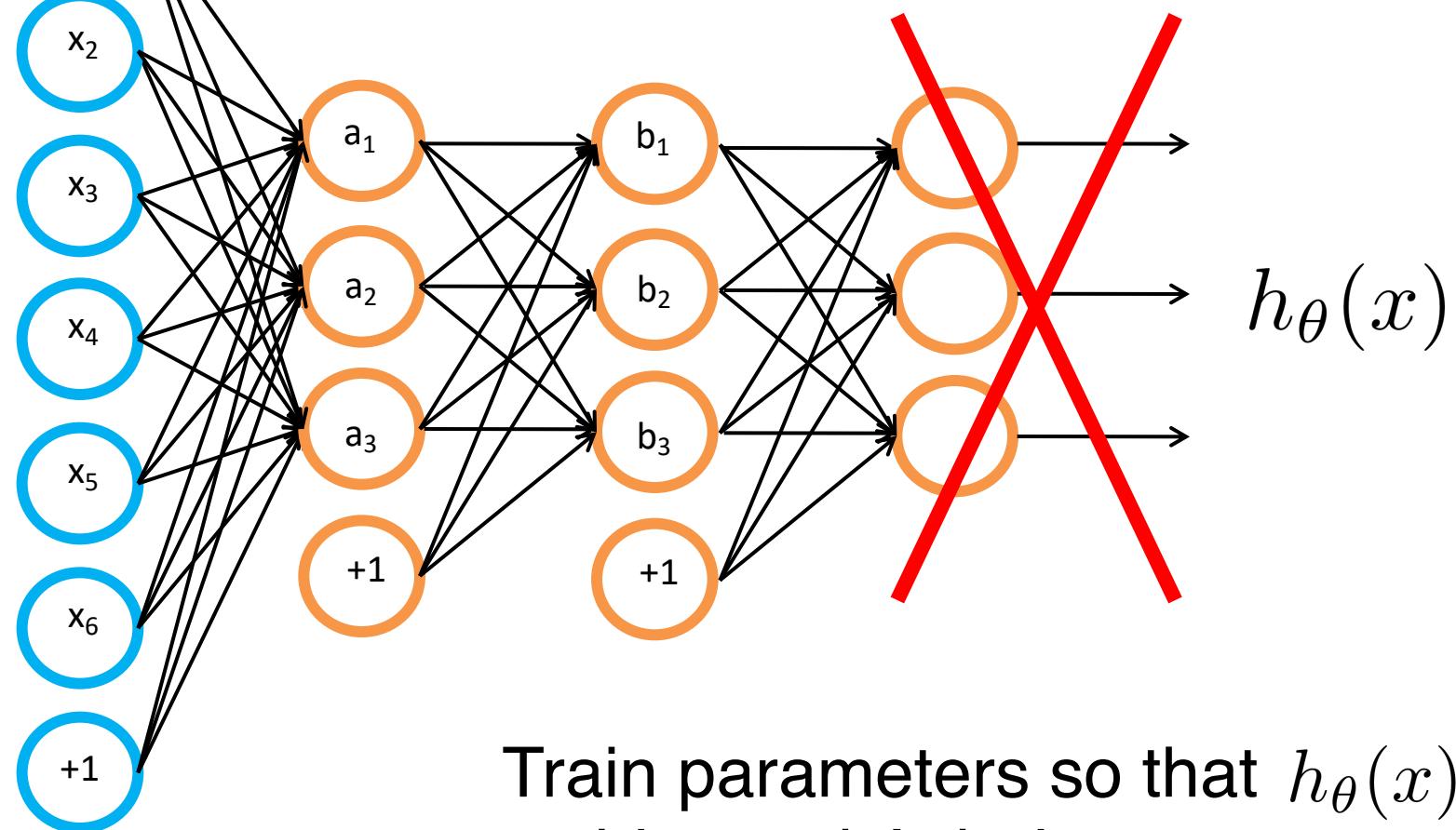
Stacked Autoencoders



$$h_{\theta}(x)$$

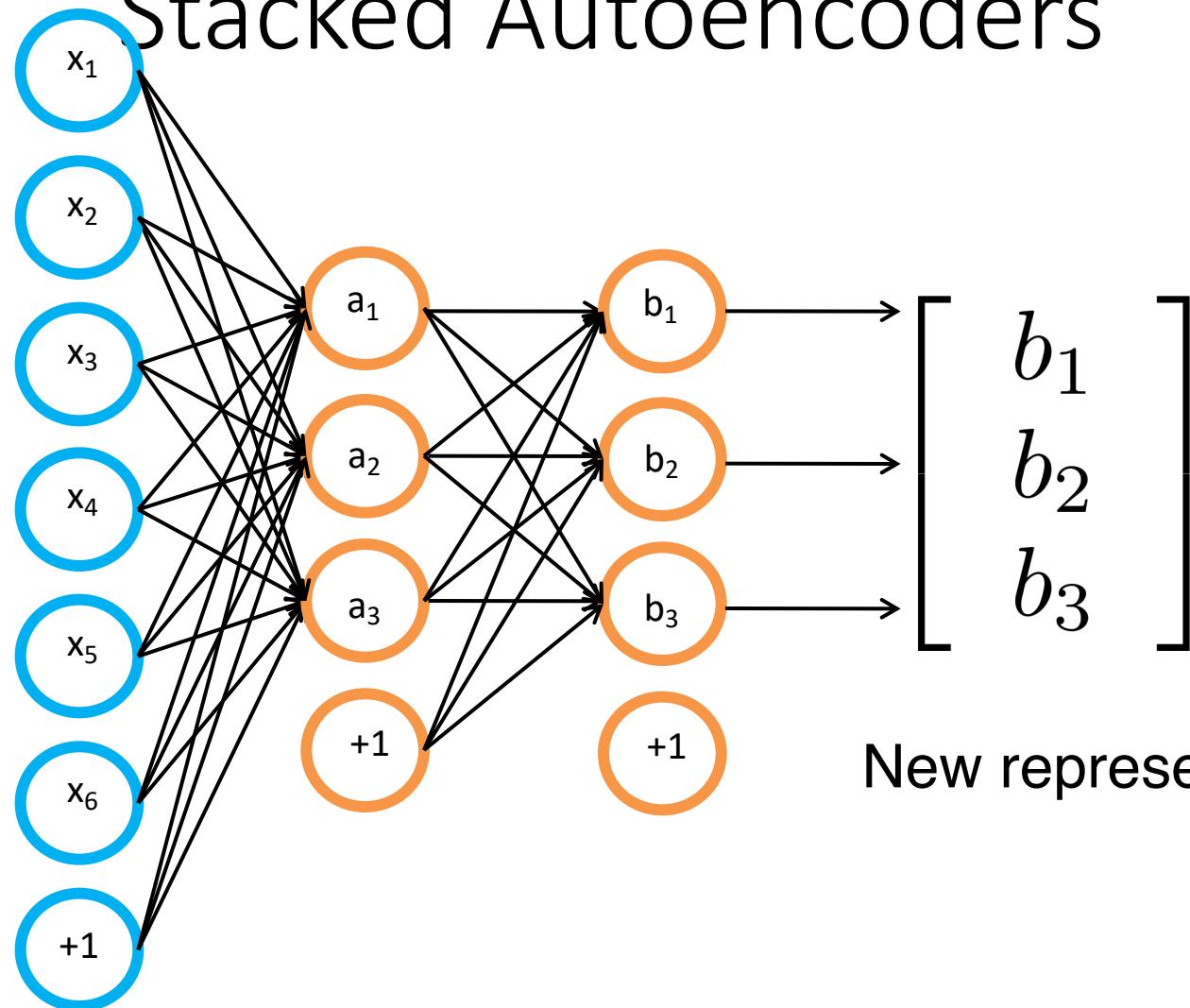
Train parameters so that $h_{\theta}(x) \approx a_i$,
subject to b_i 's being sparse.

Stacked Autoencoders



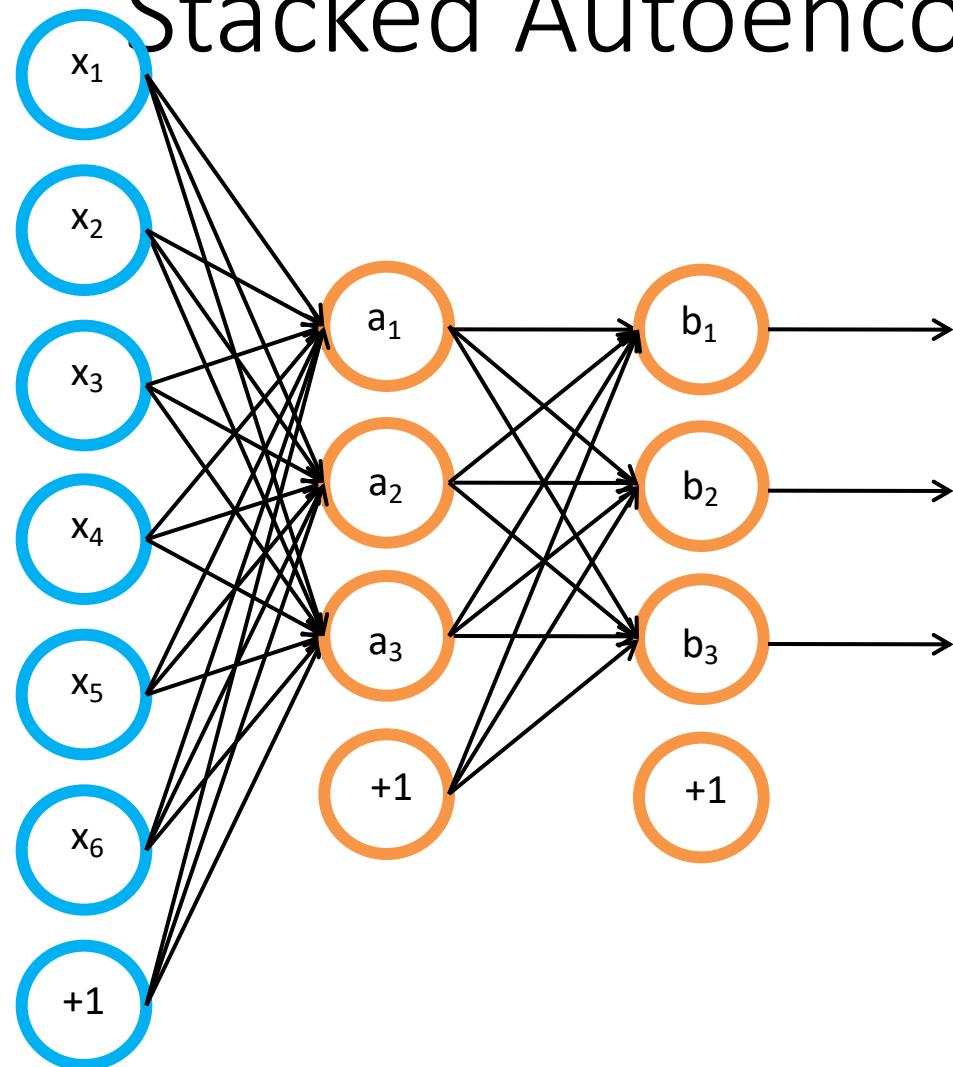
Train parameters so that $h_{\theta}(x) \approx a_i$,
subject to b_i 's being sparse.

Stacked Autoencoders

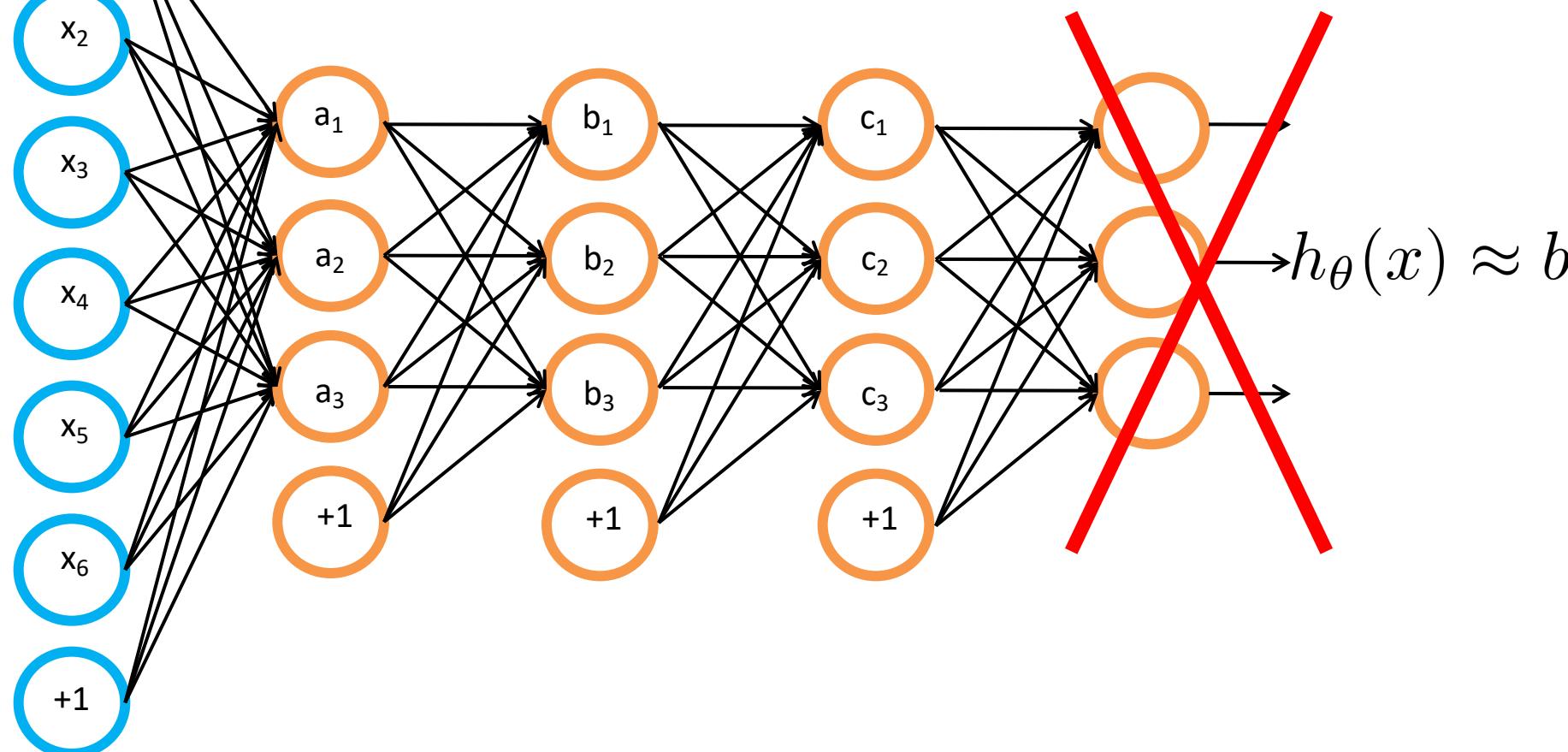


New representation for input.

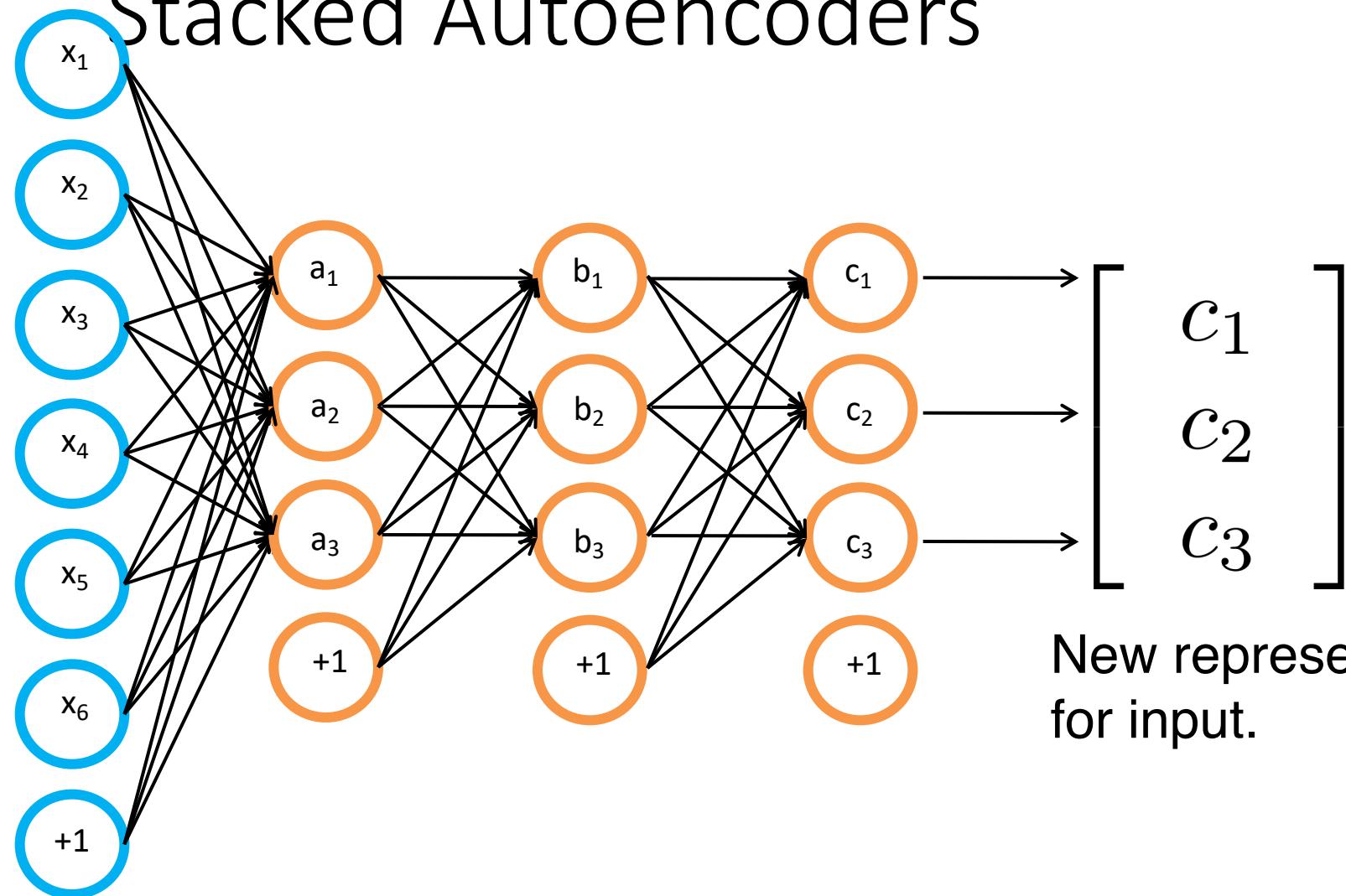
Stacked Autoencoders



Stacked Autoencoders



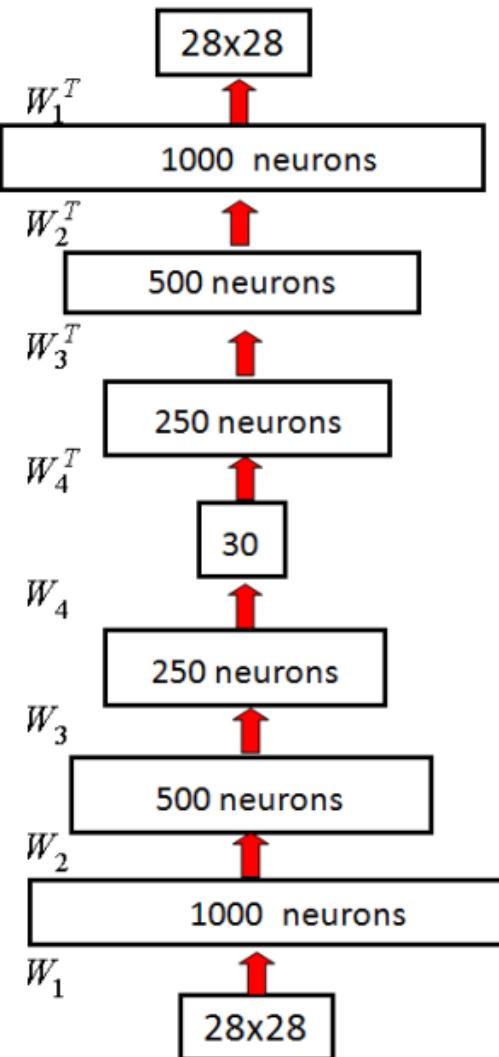
Stacked Autoencoders



New representation
for input.

Use $[c_1, c_2, c_3]$ as representation to feed the learning algorithm.
Foundations of Machine Learning

Stack multiple encoders (and their corresponding decoders)



Compressing images example



real
data

30-D
deep auto

30-D logistic
PCA

30-D
PCA

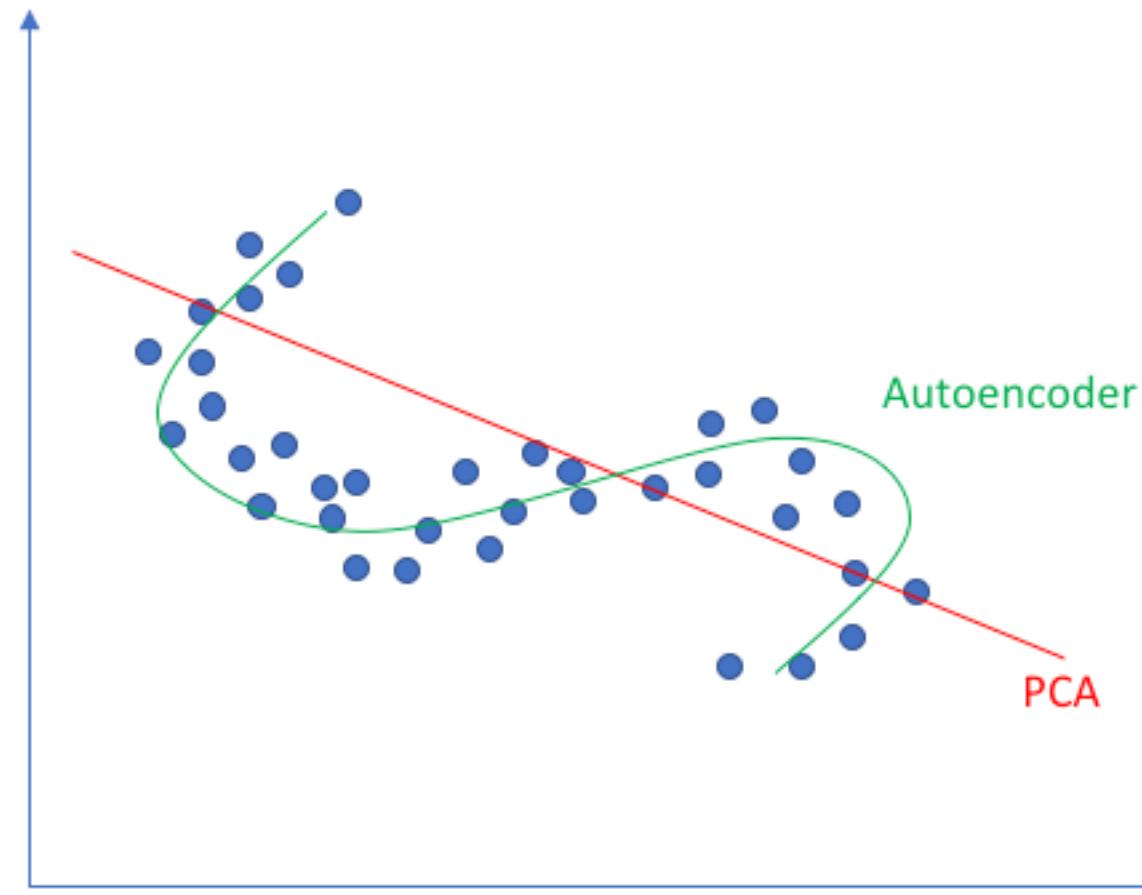


real
data

30-D
deep auto

30-D PCA

Linear vs nonlinear dimensionality reduction



ML principles

Occam's (Ockham's) razor

Occam's razor

- The simplest answer is usually the correct answer
- Simplicity is the ultimate sophistication
- Of two equivalent theories or explanations, all other things being equal, the simpler one is to be preferred
- We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances
- The simplest explanation is usually the best

ML principles

No free lunch (NFL) theorem

No free lunch (NFL) theorem

- If an algorithm performs well on a certain class of problems, then it necessarily performs bad on the set of all remaining problems

Theorem 1: For any pair of algorithms a_1 and a_2

$$\sum_f P(d_m^y | f, m, a_1) = \sum_f P(d_m^y | f, m, a_2).$$

- For any pair of algorithms a_1 and a_2 , the average of the performance over all the optimization problem f is always the same

Implications of NFL theorem

- If an algorithms performs particularly well for a class of problems, then it must do worse on average over the remaining problems
- If an algorithms performs better than random search on some class of problems, then it must performs worse than random search on the remaining problems
- No algorithm can give good results on all the problems.
You should not generalize good results obtained on some problems to other problems.