

# EE2211

<b>Lec1: What is machine learning?</b>	<b>8</b>
AI, Machine Learning, and Deep Learning.....	10
When do we need machine learning?.....	11
Application of Machine Learning.....	12
Types of Machine Learning.....	14
Supervised Learning.....	14
Regression.....	15
Classification.....	17
Unsupervised Learning.....	19
Clustering.....	20
Reinforcement Learning.....	21
Example.....	23
Walking Through A Toy Example: Token Classification.....	23
Step 1: Feature Extraction.....	24
Step 2: Nearest Neighbour Classifier.....	25
Inductive vs. Deductive Reasoning.....	26
Summary and Practise Questions.....	27
<b>Lec2: Data Engineering</b> .....	<b>28</b>
Types of Data.....	28
Levels/Scales of Measurement.....	28
Nominal Data.....	30
Ordinal Data.....	30
Interval Data.....	31
Ratio Data.....	32
Example.....	32
Based on Numerical/Categorical.....	34
Other Aspects.....	34
Data Wrangling and Cleaning (Data Preprocessing).....	35
Formatting Data.....	36
Binary Coding.....	36
Normalisation.....	37
Data Cleaning.....	38
Data Integrity and Visualisation.....	40
Data Integrity.....	40
Data Visualization.....	42
Why Visualization is Necessary.....	45
Summary and Practice Question.....	45
<b>Lec3: Introduction to Linear Algebra, Probability and Statistics</b> .....	<b>46</b>
Introduction to Linear Algebra.....	46
Notations, Vectors, Matrices.....	46
Linear dependence and independence.....	50
Systems of Linear Equations.....	51
Causality.....	52

Causality is a Statistical Relationship.....	55
Correlation vs Causality.....	55
Correlation does not imply causation!.....	56
Simpson's paradox.....	57
Example.....	57
Random Variable.....	58
Axioms of Probability.....	59
Discrete Random Variable.....	60
Continuous Random Variable.....	62
Example 1.....	64
Example 2.....	65
Example 3.....	66
Bayes' Rule.....	66
Example.....	67
Summary and Practice Question.....	67
<b>Lec4: Systems of linear equations.....</b>	<b>68</b>
Vector and Matrix Operation.....	68
Dot Product = Inner Product.....	68
Inverse of Matrix.....	69
Determinant and Cofactor to find Inverse.....	70
Systems of Linear Equations.....	71
Types of System.....	72
Square or even-determined system.....	74
Over-determined system.....	76
Under-determined system.....	79
Least Norm Solution.....	82
Set.....	84
Function.....	85
The inner product function.....	86
Linearity and Superposition.....	87
Affine Function.....	88
Class Quiz.....	88
Summary.....	89
<b>Lec5: Least Squares and Linear Regression.....</b>	<b>90</b>
Functions: Maximum and Minimum.....	90
Derivative and Gradient.....	91
Differentiation of a scalar function w.r.t. a vector.....	93
$\nabla = (\partial \partial x, \partial \partial y, \partial \partial z)$ .....	93
$\nabla f = (\partial f \partial x, \partial f \partial y, \partial f \partial z)$ .....	93
Differentiation of a vector function w.r.t. a vector.....	94
Properties.....	94
Linear Regression.....	95
Learning (Training).....	99
Least Squares Regression.....	99
Example 1: one dimension input, 1 feature.....	101
Example 2: 3 dimension input, 3 features.....	103

Learning of Vectored Function (Multiple Outputs).....	104
Example 3:.....	106
Example 4.....	109
Class Quiz.....	111
Summary.....	112
<b>Lec6: Ridge regression &amp; Polynomial regression.....</b>	<b>113</b>
Linear Regression for Classification.....	113
Binary Classification.....	113
Example.....	114
Multi-Categorical/Class Classification.....	116
Ridge Regression.....	120
Derivation.....	121
Primal and Dual Form.....	121
Polynomial Regression.....	126
Example.....	127
Polynomial Expansion.....	127
number of terms in polynomial = $k+dCd= k+dCk$ .....	128
Example.....	130
Polynomial Regression for Classification.....	134
In Class Quiz.....	135
<b>Lec7: Over-fitting, bias/variance trade-off.....</b>	<b>136</b>
Overfitting, Underfitting & Model Complexity.....	136
Overfitting.....	136
Underfitting.....	137
Good Fit.....	138
Overfitting / Underfitting Schematic.....	138
Feature Selection.....	139
Selecting Features With Pearson's r.....	140
Regularisation.....	141
Regularisation Example.....	142
Bias-Variance Trade-off.....	143
Bias-Variance Decomposition Theorem.....	144
Example.....	145
Test 1.....	146
Test 2.....	146
Test 3.....	147
Test 4.....	147
Test 5.....	148
All together.....	148
Average.....	149
Conclusion.....	149
In Class Quiz.....	149
<b>Lec8: Optimization, Gradient descent.....</b>	<b>150</b>
Loss Function.....	150
Review.....	150
Loss Function & Learning Model.....	151

Gradient Descent.....	153
Motivation for Gradient Descent.....	153
Gradient Descent Algorithm.....	154
Example 1: Find $x$ to minimise $g(x) = x^2$ .....	155
Example 2: What if the learning rate, $\eta$ , is too small?.....	157
Example 3: What if the learning rate, $\eta$ , is too big?.....	158
Python Demo.....	161
Different Learning Models.....	162
Different Loss Functions.....	164
Square Error Loss for Regression.....	164
Binary Loss for Classification.....	164
In Class Quiz.....	165
Summary.....	166
<b>Lec 9: Decision Trees &amp; Random Forest.....</b>	<b>166</b>
Decision Tree Classification.....	167
Example.....	167
Basic Terminologies.....	169
Building a Classification Decision Tree.....	170
Node Impurity.....	170
Node Impurity Measures.....	171
Gini Impurity.....	172
Entropy.....	173
Misclassification rate.....	174
Classification Tree Learning.....	175
Advantages / Disadvantages.....	176
Regression Trees.....	177
Regression Tree Learning.....	178
Example 1.....	179
Example 2.....	183
Python for Decision Tree Regression.....	186
Reducing Tree Instability.....	189
Random Forest.....	190
Random forest vs Tree.....	191
Summary and In Class Quiz.....	192
<b>Lec 10: Performance Issues.....</b>	<b>193</b>
Dataset Partition: Training/Validation/Testing.....	195
Example.....	196
Python Demo.....	198
k-fold Cross Validation.....	198
Step 1.....	198
Step 2.....	199
Step 3.....	199
Step 4.....	201
Closing Note.....	202
Evaluation Metrics.....	203
Evaluating Regression.....	203

Evaluating Classification.....	204
Recall, Precision and Accuracy.....	205
Cost Matrix.....	206
Handling unbalanced data.....	208
TPR, FNR, TNR, FPR.....	209
Confusion Matrix Depends on Threshold for Classification.....	209
Confusion Matrix for Multicategory Classification.....	211
Other Issues.....	211
Practice Question.....	212
<b>Lec 11: Unsupervised Learning.....</b>	<b>213</b>
K-means Clustering.....	215
Basic/Naïve K-means Clustering.....	216
Step 1: Initialisation.....	219
Step 2: Assignment.....	220
Step 3: Centroid Update.....	221
Step 4: Convergence.....	222
Optimization Objective Function (within-cluster variance).....	223
Initialization.....	226
K-means Clustering Example (1D).....	226
Ways to Choose Initialising Clusters.....	227
Hard vs Soft Clustering.....	228
Fuzzy Clustering.....	228
Visualisation of Fuzzy C-means Iterations.....	230
Naïve K-means versus Fuzzy C-means.....	233
Practice Question.....	234
<b>Lec 12: Neural Networks.....</b>	<b>235</b>
Introduction to Neural Networks.....	235
Perceptron.....	235
Activation Functions.....	236
Sigmoid.....	236
ReLU.....	237
Multi-layer Perceptron (Neural Network).....	238
Training and Testing of Neural Networks.....	241
Training: Forward and Backward.....	241
Testing: Forward.....	243
Python Demo.....	244
Convolutional Neural Networks (CNN).....	244
Practice Question.....	249
<b>Python and Conda Stuff.....</b>	<b>249</b>
<b>MATLAB.....</b>	<b>250</b>
<b>Tutorial 1.....</b>	<b>251</b>
<b>Tutorial 2.....</b>	<b>251</b>
Q1.....	251
Q2.....	252
Q3.....	256
Q5.....	259

<b>Tutorial 3.....</b>	<b>259</b>
Q5.....	259
Q10-Q11.....	260
<b>Tutorial 4.....</b>	<b>260</b>
Q1.....	260
Q2.....	261
Q6.....	262
<b>Tutorial 5.....</b>	<b>264</b>
Q1.....	264
Q2.....	265
Q3.....	266
Q4.....	267
Without removing duplicates VS removing the duplicates.....	267
Q5.....	269
Q6.....	270
Q8.....	272
Q9.....	272
<b>Tutorial 6.....</b>	<b>272</b>
Q1.....	272
Q2.....	273
Q3.....	274
number of terms in polynomial = $k+dCd= k+dCk= 6C3=20$ .....	274
Q4.....	275
Q5.....	276
Q6.....	277
Q7.....	278
Q8.....	278
<b>Tutorial 7.....</b>	<b>279</b>
Q1.....	279
Q2.....	280
<b>Tutorial 8.....</b>	<b>285</b>
Q1.....	285
Q2.....	286
Q3.....	288
Q4.....	290
Q5.....	291
<b>Tutorial 9.....</b>	<b>292</b>
Q1.....	292
Q2.....	294
Q3.....	295
Q4.....	296
<b>Tutorial 10.....</b>	<b>297</b>
<b>Tutorial 11.....</b>	<b>299</b>
Q3.....	300
Q4.....	300
<b>PY.....</b>	<b>301</b>

final PY.....	306
<b>Functions.....</b>	<b>313</b>
A2 Dump.....	314
A3 Dump.....	315
Admin.....	320

## Lec1: What is machine learning?

Learning is any process by which a system improves performance from experience.

- Herbert Simon

A computer program is said to learn

- from **experience E**
- with respect to some class of **tasks T**
- and **performance measure P**,

if its performance at tasks in T, as measured by P, improves with experience E.

- Tom Mitchell

# Machine Learning (Supervised Learning)



Data      Output



Cat

⋮



Dog

When applied

$f(\cdot)$



) → Cat !

New image

Machine Learning: field of study that gives computers the ability to learn without being explicitly programmed

- Arthur Samuel

- Computer with machine learning algorithm
- Output also called label
- Data output pair
- the function is said to be learned



Cat

⋮



Dog

$f(\text{cat}) = \text{'cat'}$

$f(\text{dog}) = \text{'dog'}$

- - $f(\text{data}) = \text{output}$

- Can predict after learnt

# ARTIFICIAL INTELLIGENCE

Any technique which enables computers to mimic human behavior



## MACHINE LEARNING

AI techniques that give computers the ability to learn without being explicitly programmed to do so



## DEEP LEARNING

A subset of ML which make the computation of multi-layer neural networks feasible



### Example of AI but not ML: Deductive Reasoning

NUS is in Singapore, Singapore is in Asia  $\rightarrow$  NUS is in Asia

- ML can be used to achieve AI
- Deep learning is a subset of ML with specific architecture called neural network
- Deductive reasoning
  - If given A is in B and B is in C, then can reason that A is in C

## When do we need machine learning?

Lack of human expertise  
(Navigating on Mars)



Involves huge amount of data  
(Genomics)



**Learning is not always useful:**

No need to “learn” to calculate payroll!

My Salary = Days\_of\_work \* Daily Salary + Bonus

## Application of Machine Learning

Task T, Performance P, Experience E

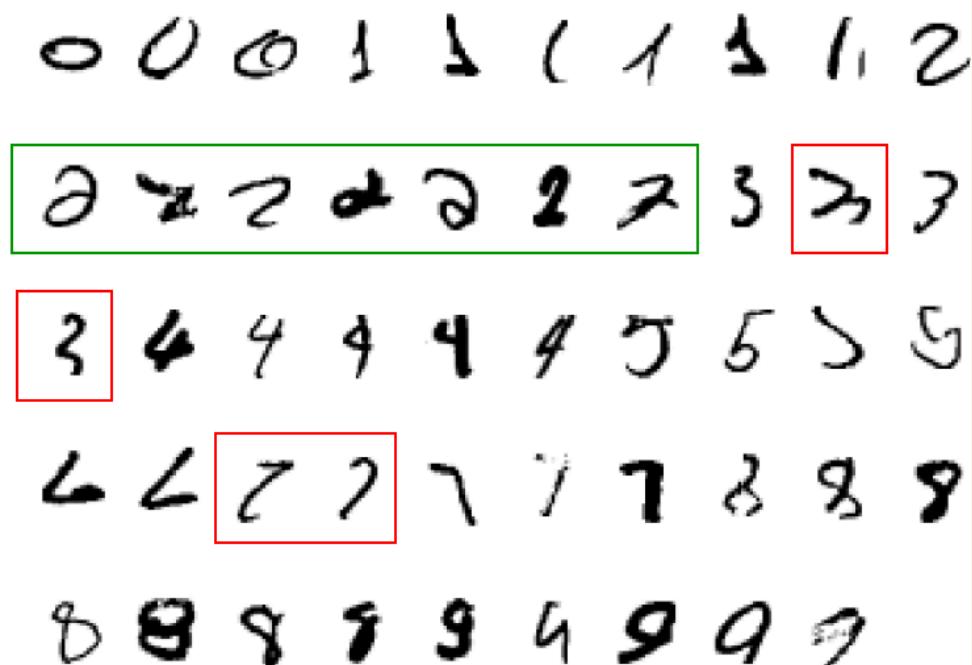
T: Digit Recognition

P: Classification Accuracy

E: Labelled Images

4 “four”

3 “three”



Labels -> Supervision!

- used by USPS, United States Postal Service
- By giving the labelled images the supervision is given to the model

Task T, Performance P, Experience E

T: Email Categorization

P: Classification Accuracy

E: Email Data, Some Labelled



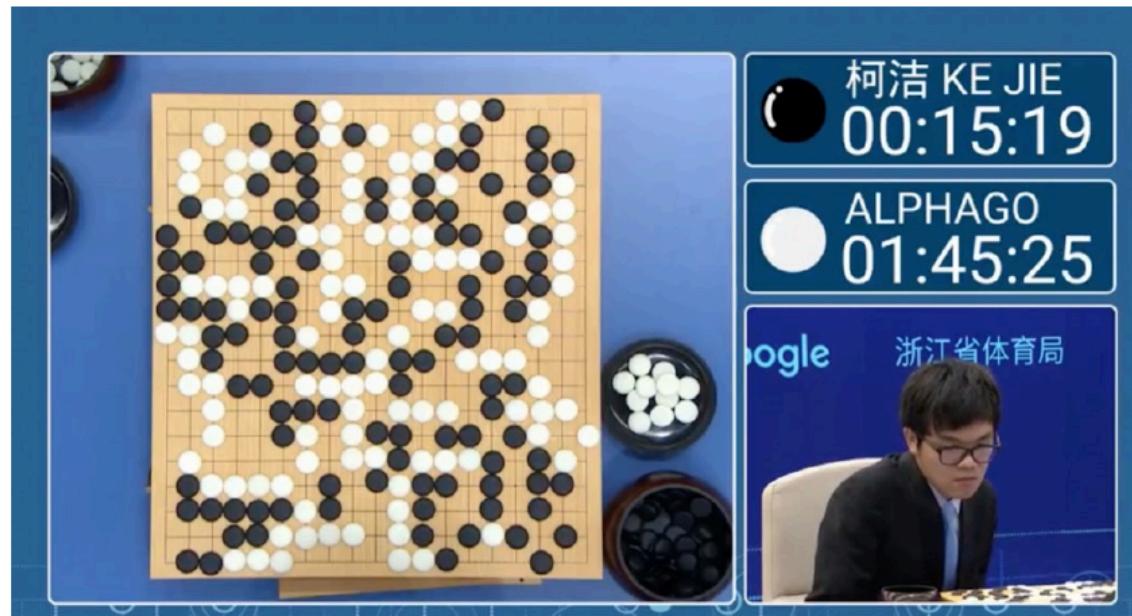
- Only some email data is labelled, user does not label all mails only some

## Task T, Performance P, Experience E

T: Playing Go Game

P: Chances of Winning

E: Records of Past Games



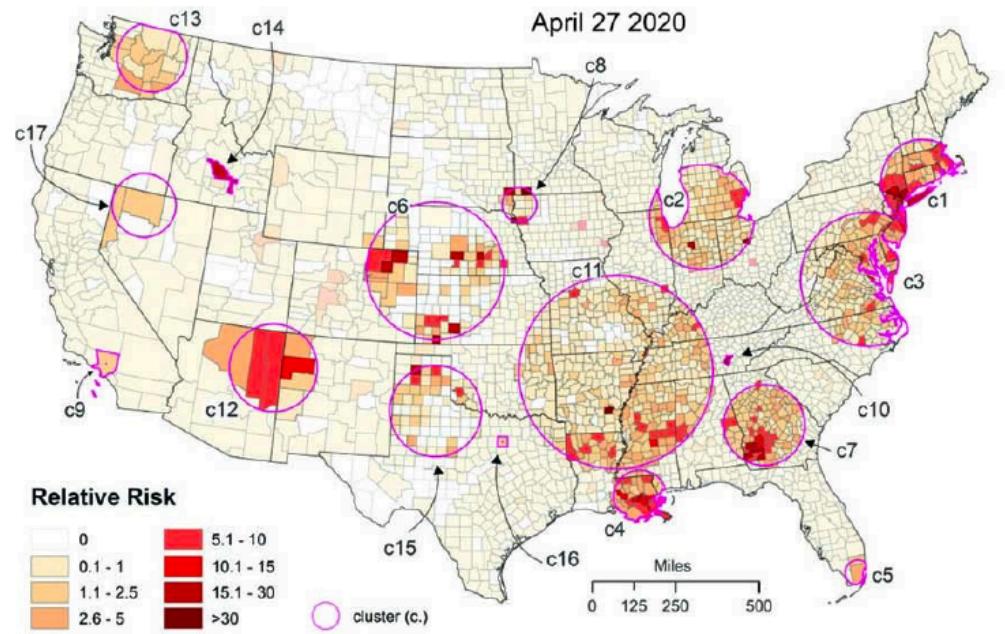
## Task T, Performance P, Experience E

T: Identifying Covid-19 Clusters

P: Small Internal Distances

Larger External Distances

E: Records of Patients



- Since model is finding out clusters

- Between cluster should be big, picking two counties far apart from each other should belong in different cluster
  - small internal distance
- But between neighbouring county should be in one cluster, so picking two counties close to each other they should belong in the same cluster
  - larger external distance
- Cannot be supervised as only have records of patients not cluster

## Types of Machine Learning

### **Supervised Learning**

Input:

1) Training Samples,  
2) Desired Output  
(Teacher/Supervision)

Output:

A rule that maps input to output

### **Unsupervised Learning**

Input:

Samples

Output:

Underlying patterns in data

### **Reinforcement Learning**

Input:

Sequence of States,  
Actions, and  
Delayed Rewards

Output:

Action Strategy: a rule  
that maps the  
environment to action

## Supervised Learning

### **Supervised Learning**

Input:

1) Training Samples,  
2) Desired Output  
(Teacher/Supervision)

Output:

A rule that maps input to output

## **Supervised Learning**

### Data



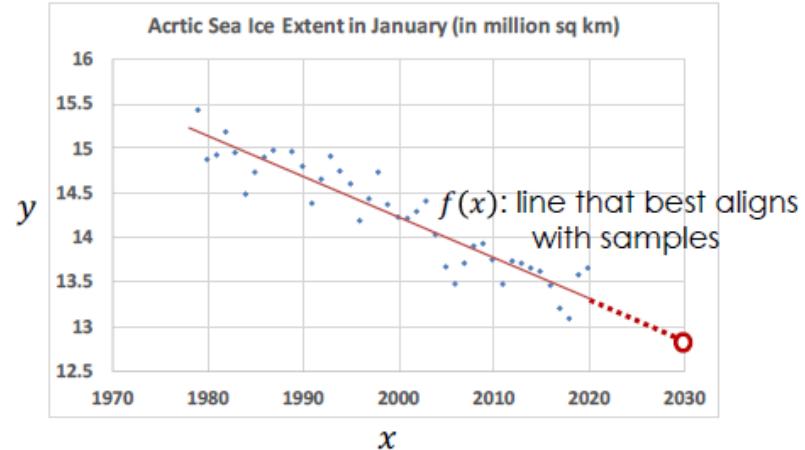
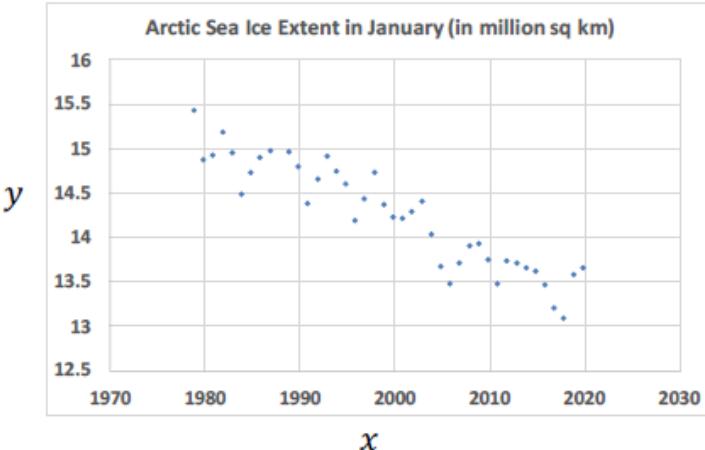
- If  $y$  is continuous, the type of learning is called regression

- if  $y$  is categorical, the type of learning is called classification

## Regression

### Regression

- Given  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$
- Learn a function  $f(\mathbf{x})$  to predict real-valued  $y$  given  $\mathbf{x}$



- if it's a line, then linear regression. If not line, regression but not linear
- to predict to the future
- Regression also belongs to the class of supervised learning methods.

Regression is just like classification except that  $y_i$  are no longer restricted to belong to a finite set. Rather it can take on uncountably many values. In the case of regression, we typically do not say that  $y_i$  is the label.

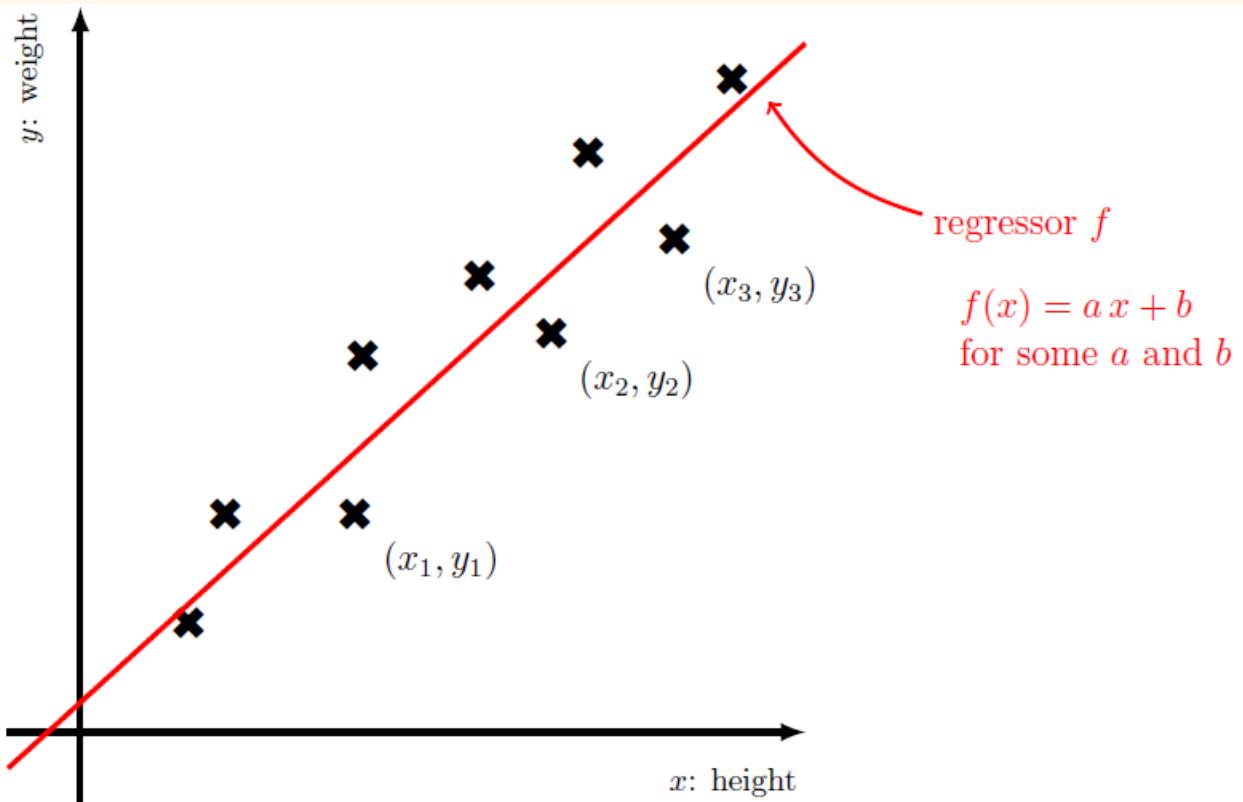


Figure 1.2: Height and weight regression example

Rather we use the term *target variable* or *outcome variable* or *dependent variable* to refer to the  $y_i$ 's. For instance  $y_i$  could take any value in the interval  $[0, 1]$  (which is uncountable) or it could take values in  $\mathbb{R}$ . In either case, the data examples in the dataset  $\mathcal{D}$ , denoted as  $(\mathbf{x}_i, y_i)$  belong to  $\mathbb{R}^d \times \mathbb{R}$ , which means that each feature vector or training sample  $\mathbf{x}_i \in \mathbb{R}^d$  ( $d$ -dimensional Euclidean space) and each target variable  $y_i \in \mathbb{R}$  is a real number. We wish to learn a *regressor*  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  which is a function that brings us from the feature space to the set of real numbers. The regressor  $f$  should be designed such that given a new feature vector  $\mathbf{x}'$ , its corresponding target value  $y'$  is “well predicted”, in some precise mathematical sense, by  $f(\mathbf{x}')$ .

- Say each feature vector  $x_i \in \mathbb{R}$  is scalar (so  $d = 1$ ) and represents the height of a student. The target variable  $y_i$  represents the weight of the same student. Given the data of  $m = 600$  students in EE2211 captured in the dataset  $\mathcal{D} = \{(x_i, y_i) : 1 \leq i \leq 600\}$ , I am tasked to learn a regressor  $f : \mathbb{R} \rightarrow \mathbb{R}$  so that I can use the height of Alice  $x'$  in a new class EE9999 to predict her weight  $y' = f(x')$ . See Fig. 1.2.
- Say each feature vector  $\mathbf{x}_i \in \mathbb{R}^3$  is a 3-dimensional vector and  $x_{i,1}$  represents the air pressure at location  $i$ ,  $x_{i,2}$  represents the amount of rainfall at location  $i$  and  $x_{i,3}$  represents the “amount of greenery” at location  $i$ . Each  $y_i$  corresponds to the temperature at location  $i$ . The temperature can take on uncountably many values—it is a real number. Given the dataset  $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq m\}$ , we would like to learn a regressor  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  such that if I give you the feature vector of new location  $\mathbf{x}'$ , you can tell me the temperature  $y' = f(\mathbf{x}')$ .

## Classification

The most canonical task is classification. This problem takes the following form. We have a *dataset*  $\mathcal{D}$  which consists of a certain number  $m$  of *data examples*  $(\mathbf{x}_i, y_i), i = 1, \dots, m$ . Each data example  $(\mathbf{x}_i, y_i)$  consists of a *feature vector* (also called *training sample* or *training example*)

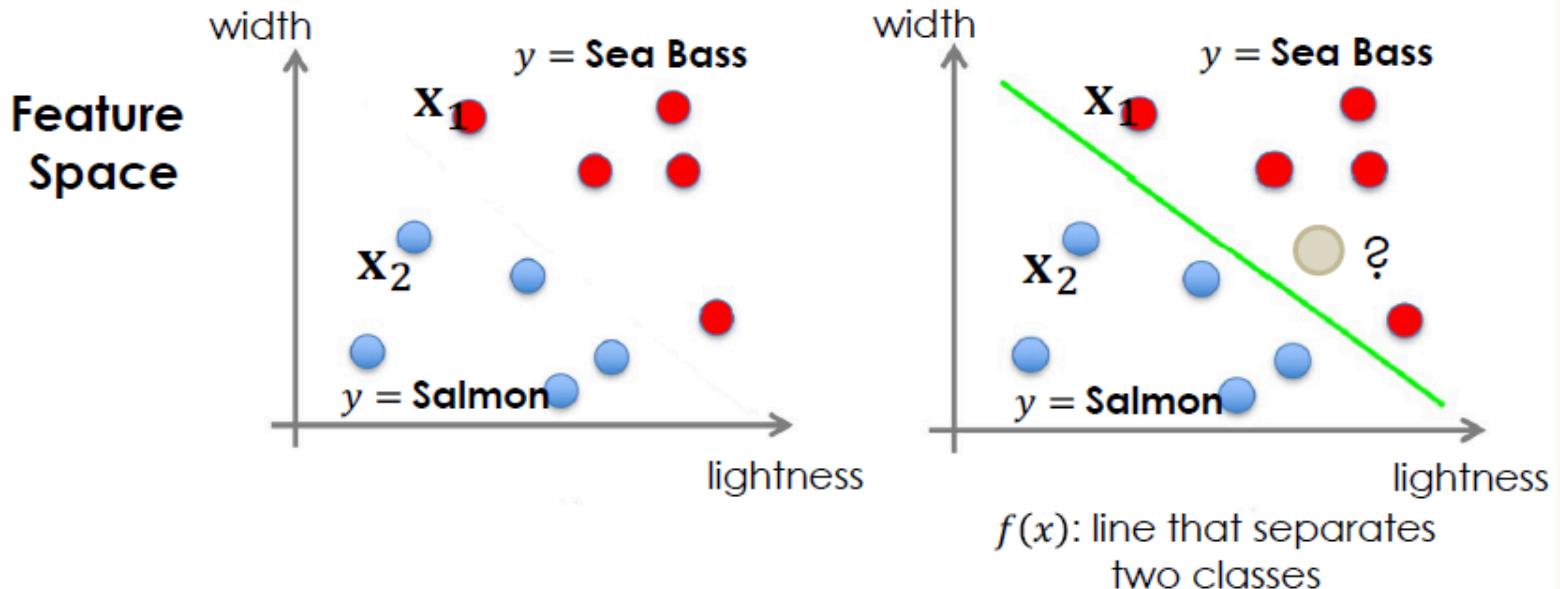
$$\mathbf{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,d} \end{bmatrix} \quad \text{or} \quad \mathbf{x}_i = [x_{i,1} \ x_{i,2} \ \dots \ x_{i,d}]^\top, \quad (1.1)$$

(usually an element of  $d$ -dimensional Euclidean space  $\mathbb{R}^d$ ) and a *label*  $y_i$ . The crux of classification is that the label  $y_i$  can only take on *finitely many values* so  $y_i \in \{1, 2, \dots, c\}$  for some integer  $c \geq 2$ , where  $c$  is the number of classes. We do not allow  $y_i$  to take on infinitely many values.

The classification problem consists in finding a *classifier* which is a function  $f : \mathbb{R}^d \rightarrow \{1, 2, \dots, c\}$  such that it accurately predicts labels given new samples, called *test samples*. This function  $f$  is constructed or learned based on the dataset  $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq m\}$ .

### Classification

- Given  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$
- Learn a function  $f(\mathbf{x})$  to predict categorical  $y$  given  $\mathbf{x}$



- The two axis is the feature of the category, the plane is called feature space
- $f(x)$  learnt is to separate the two category
- Classification belongs to the class of supervised learning methods.
- Example

- We have records of  $m = 100$  emails. Let  $d = 2$ . Thus, there are two components in each  $\mathbf{x}_i$ . Furthermore the first component of  $\mathbf{x}_i$ , denoted as  $x_{i,1}$ , counts the number of times the word “love” appears. The second component of  $\mathbf{x}_i$ , denoted as  $x_{i,2}$ , counts the number of times the word “money” appears. Each label  $y_i \in \{0, 1\}$  where  $y_i = 1$  means that the  $i^{\text{th}}$  email is spam while  $y_i = 0$  means that the  $i^{\text{th}}$  email is non-spam. Based on the dataset  $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq 100\}$  of  $m = 100$  emails, we are tasked to learn a classifier  $f : \mathbb{R}^2 \rightarrow \{0, 1\}$  such that we can accurately predict whether the next email you receive (which is of course not in the training dataset) is spam or not. See Fig. 1.1. In this case in which there are only two classes, we refer to this as *binary classification*.

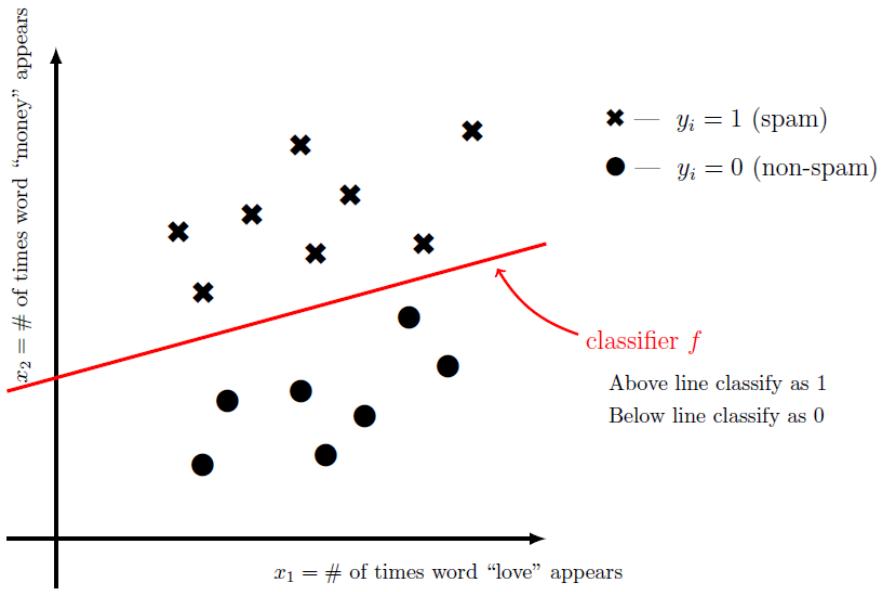


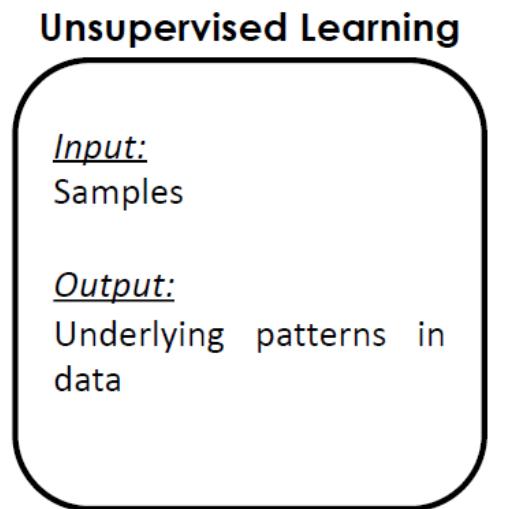
Figure 1.1: Spam classification

- - Consider an image classification problem. We are given a dataset of size  $m = 10^6$ , namely,  $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq 10^6\}$  where each  $\mathbf{x}_i$  represents an image that is vectorized into a vector. For example, each image contains  $5 \times 5 = 25$  pixels and for the sake of simplicity, each pixel value is 0 or 1. Thus  $\mathbf{x}_i \in \{0, 1\}^{5 \times 5} \cong \{0, 1\}^{25}$  and  $d = 25$ . Of course, we can have more complicated images that have multiple quantization levels (not restricted to binary) and colors as well, but the treatment will be the same. To each image, there is a label  $y_i$  which can take on 10 values so  $c = 10$ . These 10 values signify what object is in the image, e.g.,  $y_i \in \{\text{dog}, \text{cat}, \dots, \text{snake}\}$ . We would now like to design a classifier  $f : \{0, 1\}^{25} \rightarrow \{\text{dog}, \text{cat}, \dots, \text{snake}\}$  that “does well” (in some sense) on new images (or test images) that contain one of the 10 animals. Since there are more than two classes in this scenario, we refer to this as *multi-class* classification.
  - Is the following a classification problem? There are a total of  $M \geq 2$  football teams. Each football team  $i \in \{1, 2, \dots, M\}$  plays  $L$  games against every other football team  $j$ . Say there are no draws. The number of times  $i$  beats  $j$  is denoted as  $b_{ij} \in \{0, 1, \dots, L\}$  so necessarily  $b_{ij} + b_{ji} = L$  (convince yourself that this is the case). We set  $b_{ii} = 0$  for all  $i$ . Clearly, if one team wins all its games against other team, it is the best. When this is not case, which is common, it is not so clear what is the best strategy to *rank* the team. More precisely, given the matrix  $\mathbf{B} = [b_{ij}]_{1 \leq i \leq M, 1 \leq j \leq M}$  we would like to rank the teams from top (best) to bottom (worst). This is a non-standard machine learning problem. However, notice that the total number of rankings is  $M!$ , which is finite, so in some sense this is like a classification problem. In what way is it not a classification problem? See [XFTF19] for some discussion of how we can use machine learning to learn the skills of tennis players.

Please select the correct option.

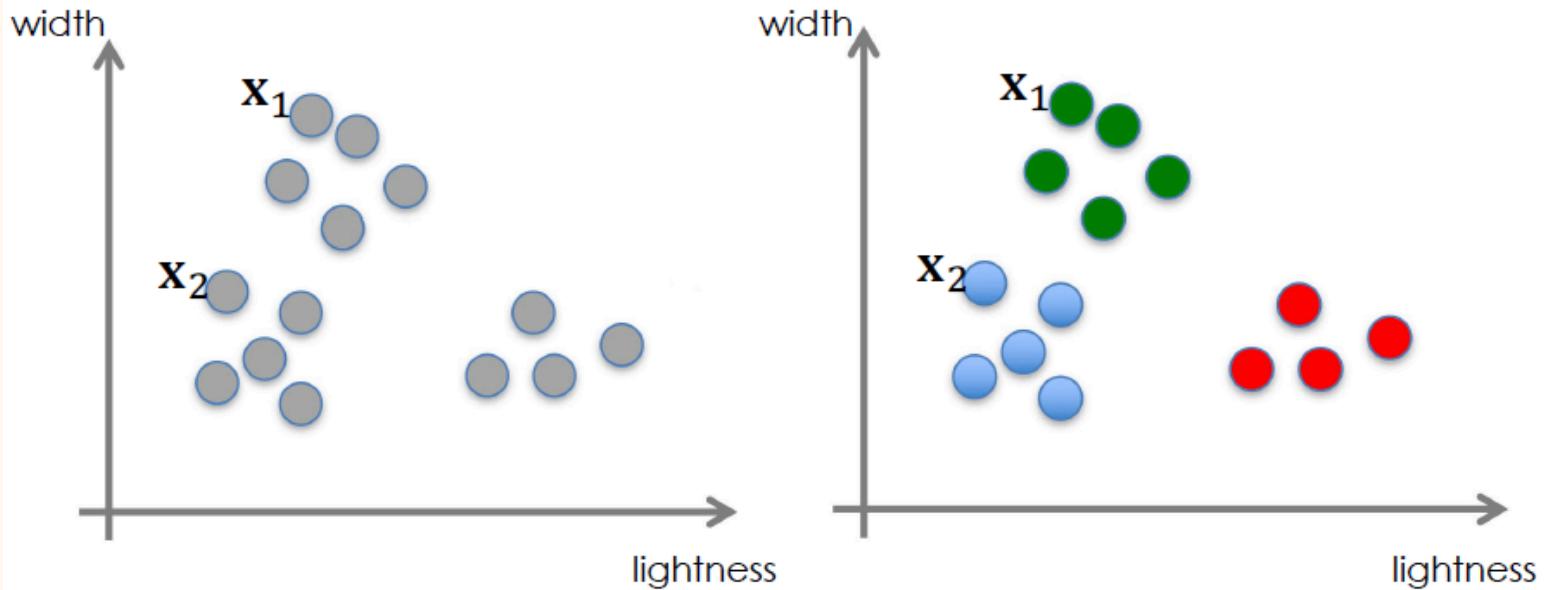
- A. Classification works with labels that are only continuous.
- B. Classification works with labels that only belong to ordinal data.
- C. Classification works with features that are only continuous.
- D. Classification works with features that are only discrete.
- E. Regression works with features that are only discrete.
- F. b) and d)
- G. b) and c)
- ✓H. None of others is correct.

### Unsupervised Learning



## Clustering

- Given  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ , without labels
- Output Hidden Structure Behind



**No Label/Supervision is given!**

- Only clustering in syllabus for unsupervised
  - only categorical in syllabus, continuous exist
- Can only give  $x$ , cannot give  $y$ , there is no label/desire output
- Output hidden structure behind, cluster the groups
- small internal distance and larger external distance

In clustering, the labels or target variables  $y_i$  are no longer available and we *only* have access to the feature vectors  $\mathcal{D} = \{\mathbf{x}_i : 1 \leq i \leq m\}$ . There is often reason to believe that these feature vectors can be grouped or clustered into different clusters.

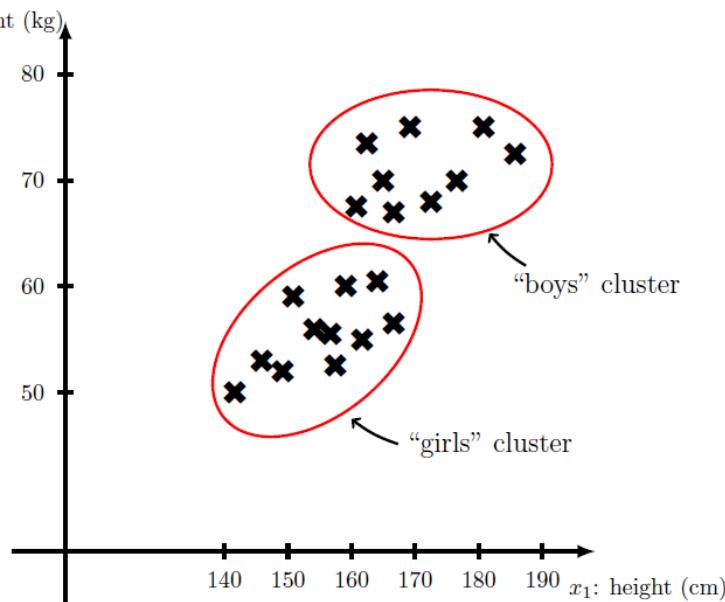


Figure 1.3: Height and weight clustering example

- Say the vector  $\mathbf{x}_i = [x_{i,1}, x_{i,2}]^\top$  represents the height and weight of the  $i^{\text{th}}$  student in this semester's EE2211 cohort. There is reason to believe that girls are shorter and lighter than boys. So given  $\mathcal{D} = \{\mathbf{x}_i : 1 \leq i \leq m\}$ , can we assign each partition this set into 2 groups automatically? See Fig. 1.3. Note that while the clustering algorithm knows that there are 2 clusters, it does not and cannot assign semantic meanings (such as "boys" and "girls") to the clusters it discovers. It can only tell us that  $\mathcal{D}$  is partitioned into two non-empty disjoint subsets  $\mathcal{D}_1$  and  $\mathcal{D}_2$  (i.e.,  $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$  and  $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$ ).
- Say each  $\mathbf{x}_i = [x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}]^\top$  and  $x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}$  respectively represent the number of times the words "winner", "victory", "stock", "prices" appeared in the  $i^{\text{th}}$  article in a bag of  $m$  news articles. There is reason to believe that sports articles would contain more of the first 2 words but finance articles would contain more of the last 2 words. Hence, it seems possible to design a clustering algorithm to group the  $m$  feature vectors into two clusters, one representing sports articles while the other representing finance articles.

## Reinforcement Learning

### Reinforcement Learning

#### Input:

Sequence of States,  
Actions, and  
Delayed Rewards

#### Output:

Action Strategy: a rule  
that maps the  
environment to action

- When task requires a sequence of state or actions

## Breakout Game



**Initial Performance**



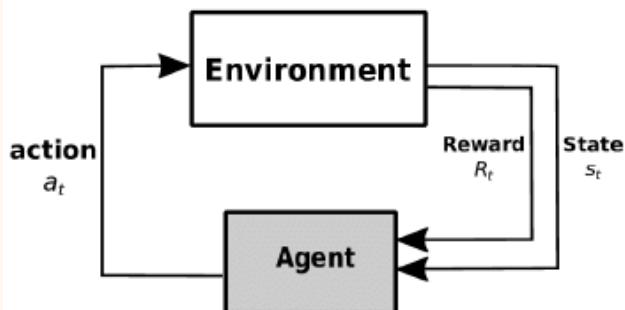
**Training 15 minutes**



**Training 30 minutes**

- It's a gif of the game playing better after training

- Given sequence of states  $S$  and actions  $A$  with (delayed) rewards  $R$
- Output a policy  $\pi(a, s)$ , to guide us what action  $a$  to take in state  $s$



**$S$ :** Ball Location,  
Paddle Location, Bricks

**$A$ :** left, right

**$R$ :**  
positive reward  
Knocking a brick,  
clearing all bricks

negative reward  
Missing the ball

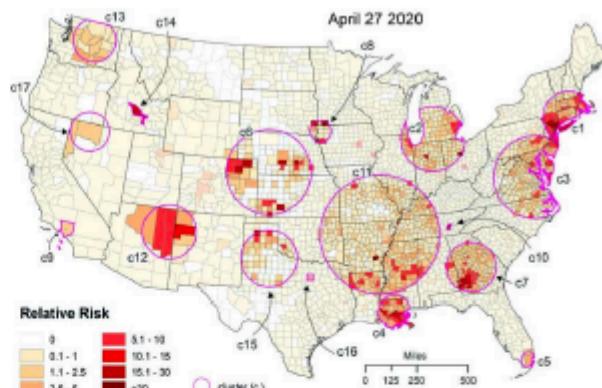
zero reward  
Cases in between

- Intuitively, reinforcement learning with its actions and rewards is like supervision learning. But it's given a different label because the nature is so different

## Example

0 0 0 1 1 1 1 1 1 2  
 2 2 2 2 2 2 3 3 3  
 3 4 4 4 4 5 5 5 5  
 6 6 7 7 7 7 8 8 8  
 9 9 9 9 9 9 9 9 9

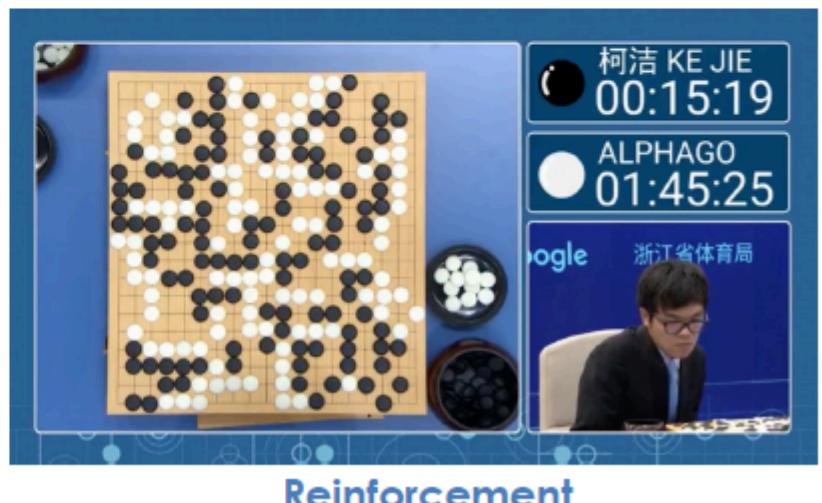
**Supervised**



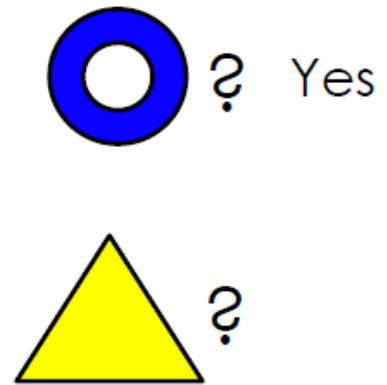
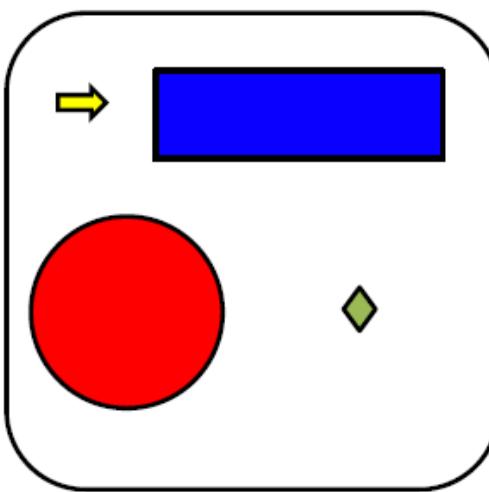
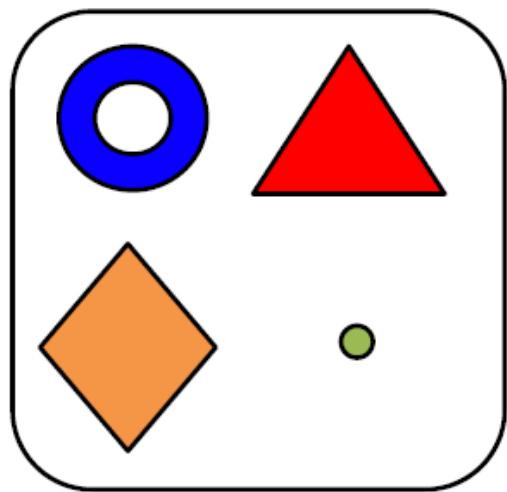
**Unsupervised**



**Supervised**



## Walking Through A Toy Example: Token Classification



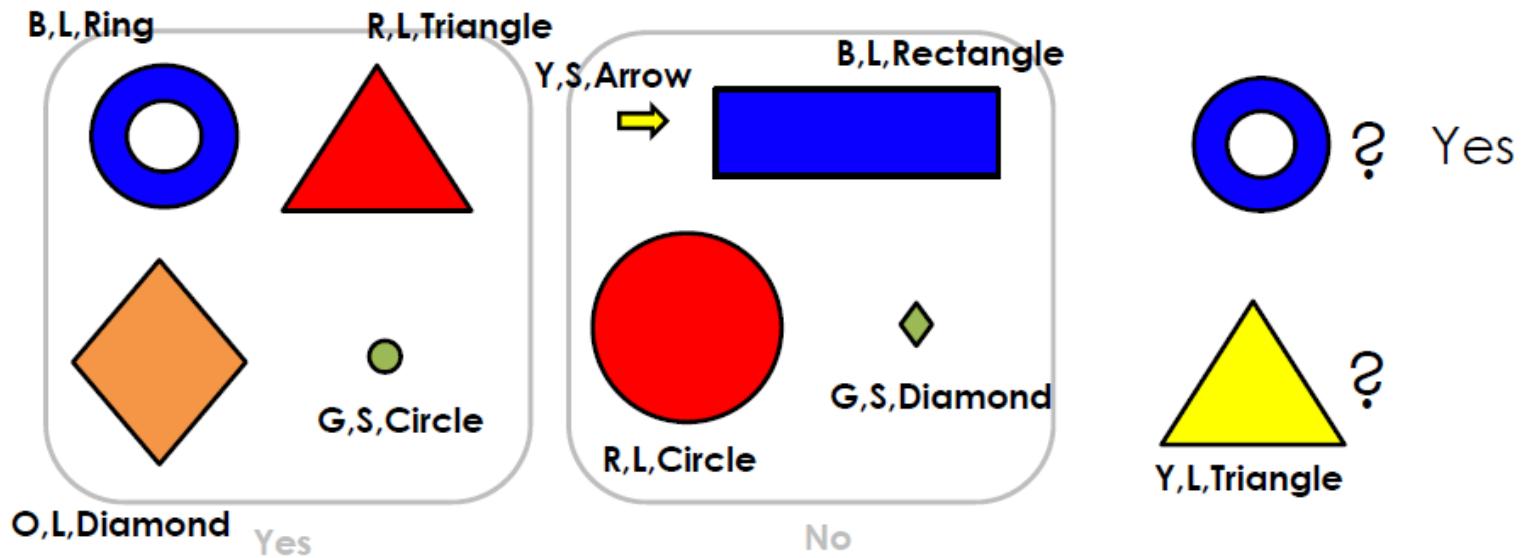
- Issue comes out when the testing sample (yellow triangle) does not overlap with the training sample (the set of all shapes with labels)
- Need to train a classification system

**Step 1: Feature Extraction**  
Extract Attributes of Samples

**Step 2: Sample Classification**  
Decide Label for a Sample

- requires a way to describe each sample, by extracting out their attributes
- based on the feature extracted, conduct the classification
- Sample classification can have many ways to do
  - Nearest Neighbour Classifier is just one way

Step 1: Feature Extraction



- Extract the feature of colour, size, shape (can have more)

**Feature Extraction**

Color	Size	Shape	Label
Blue	Large	Ring	Yes
Red	Large	Triangle	Yes
Orange	Large	Diamond	Yes
Green	Small	Circle	Yes
Yellow	Small	Arrow	No
Blue	Large	Rectangle	No
Red	Large	Circle	No
Green	Small	Diamond	No
Yellow	Large	Triangle	?

- To check the testing sample, check the testing sample against all the other training sample, and build up a Similarity table
- feature extraction = feature selection (for EE2211)

## Step 2: Nearest Neighbour Classifier

**Similarity**

	Color	Size	Shape	Total
	0	1	0	1
	0	1	1	2
	0	1	0	1
	0	0	0	0
	0	0	0	0
	1	0	0	1
	0	1	0	1
	0	1	0	1
	0	0	0	0

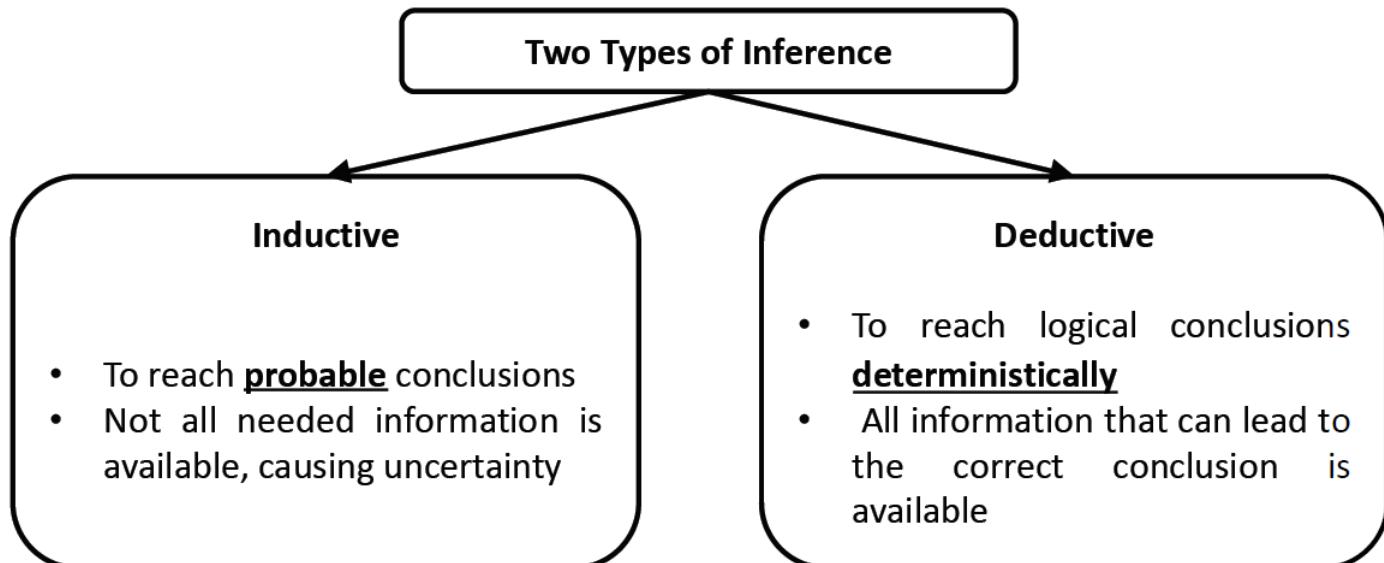
### **Nearest Neighbor Classifier:**

- 1) Find the “nearest neighbor” of a sample in the feature space
- 2) Assign the label of the nearest neighbor to the sample

- Yellow big triangle is the most similar to the red big triangle, hence, given the label of red big triangle. Most similar in the feature space
- Nearest neighbour = more similar
- If attributes is changed the result may differ
  - need to extract as many feature as possible to avoid draw in similarity
  - if has draw choose any
- In EE2211, the newly labelled will not be learn, only the experience from the input/output pair given
  - done in advanced technique
- Nearest Neighbour Classifier does not requires training, it's just comparing against lookup table

## Inductive vs. Deductive Reasoning

- Main Task of Machine Learning: to make inference



### Probability and Statistics



### Rule-based reasoning

NUS is in Singapore, Singapore is in Asia ->  
NUS is in Asia

- Inductive
  - Do not have all the information needed, can only give probable conclusion
  - Result have probability/confidence
  - Has uncertainty
  - If face is far away hard to tell
  - EE2211 ML: is inductive
  - Guessing population from sample, small picture to big picture
- Deductive
  - Have all the information needed
  - Conclusion is deduced from information and hence 100% confidence and certainty
  - Draw conclusion deterministically
  - With the big picture get the small picture

## Summary and Practise Questions

### **Three Components in ML Definition**

Task T, Performance P, Experience E

### **Two Types of Supervised Learning**

Classification, Regression

### **Three Types of in ML**

Supervised Learning

Unsupervised Learning

Reinforcement Learning

### **One Type of Unsupervised Learning**

Clustering

### **Inductive and Deductive**

Inductive: Probable

Deductive: Rule-based

### **Example of a Classifier Model**

Nearest Neighbor Classifier

Which of the following statement is true?

- A. Nearest Neighbor Classifier is an example of unsupervised learning
- B. Nearest Neighbor Classifier is an example of deductive learning
- C. Nearest Neighbor Classifier is an example of feature selection
- D. None of the above is correct.

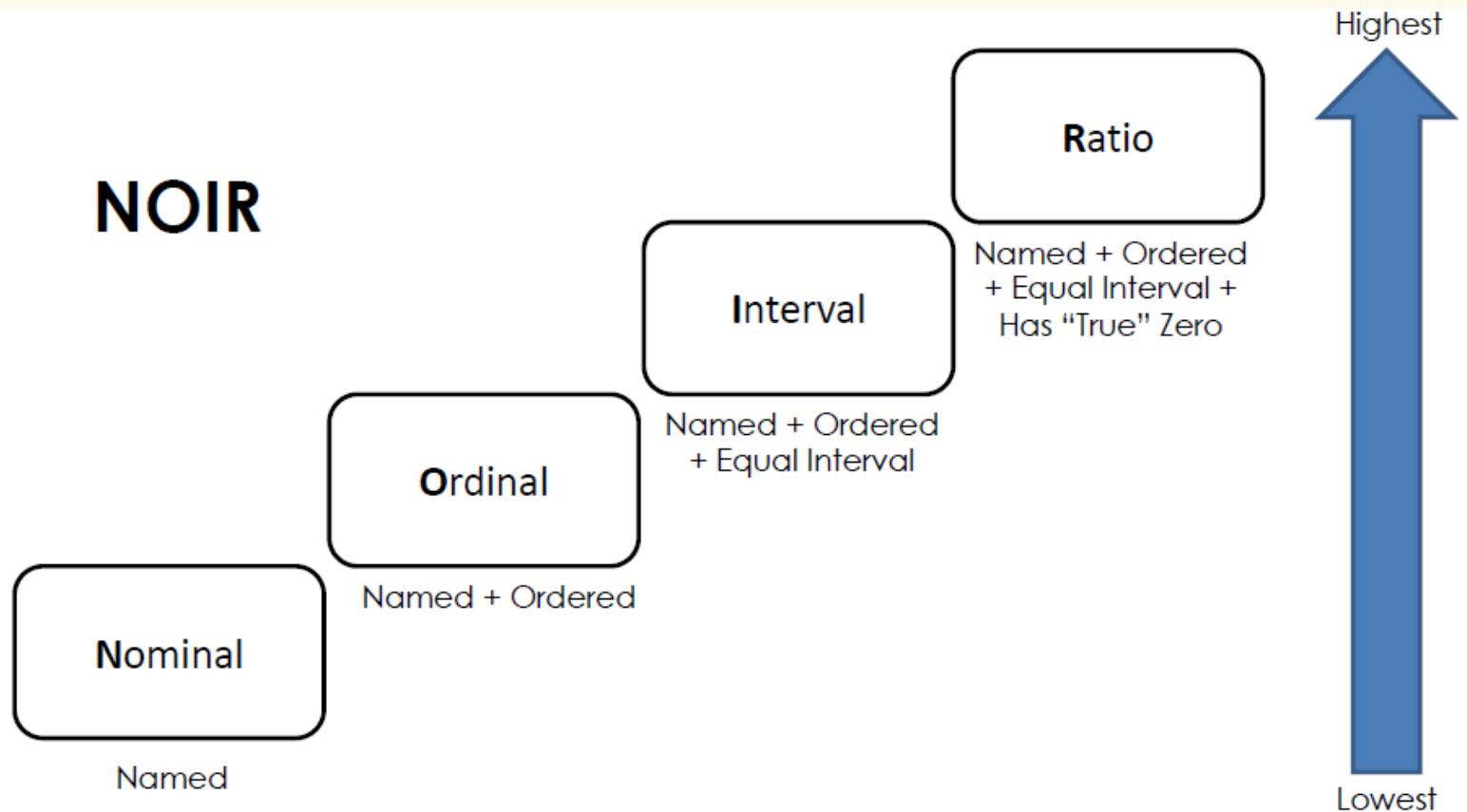
- A is wrong, nearest neighbour classifier is supervised
- B is wrong, nearest neighbour classifier is not with full certainty
- C is wrong, nearest neighbour classifier is the second step, the first step is feature extraction/selection
  - other classification can be used
  - feature extraction = feature selection (for EE2211)

## Lec2: Data Engineering

### Types of Data

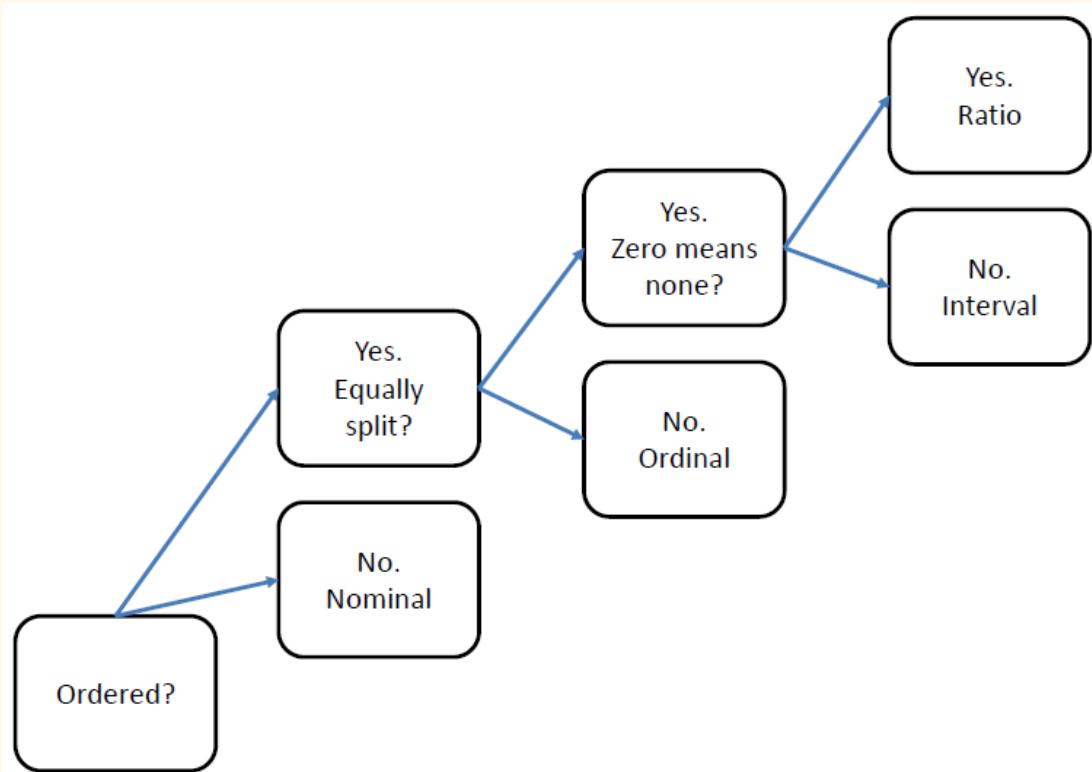
- Based on [Levels/Scales of Measurement](#)
  - Nominal Data
  - Ordinal Data
  - Interval Data
  - Ratio Data
- Based on [Numerical/Categorical](#)
  - Numerical, also known as Quantitative
  - Categorical, also known as Qualitative
- Other aspects
  - Available or Missing Data

### Levels/Scales of Measurement



- Going up means, Ordinal is Named + Ordered

We can estimate	Nominal	Ordinal	Interval	Ratio
Frequency Distribution	Yes	Yes	Yes	Yes
Median	No	Yes	Yes	Yes
Add or subtract	No	No	Yes	Yes
Mean, standard deviation	No	No	Yes	Yes
Ratios	No	No	No	Yes



## Nominal Data

- Lowest Level of Measurement
- Discrete Categories
- **NO** natural order
- Estimating a **mean**, **median**, or **standard deviation**, would be meaningless.
- Possible Measure: **mode**, **frequency distribution**
- Example:

Gender	Occupation
 man	 Doctor
 woman	 Police

Gender	Occupation
 man	 Doctor
 woman	 Police

Gender	Occupation
 man	 Doctor
 woman	 Police

## Ordinal Data

- **Ordered** Categories
- Relative Ranking
- Unknown “distance” between categories: orders matter but not the difference between values
- Possible Measure: **mode**, **frequency distribution** + **median**
- Example:
  - Evaluate the difficulty level of an exam
    - 1: Very Easy, 2: Easy, 3: About Right, 4: Difficult, 5: Very Difficult
  - can sort means can have median

## Interval Data

- Ordered Categories
- Well-defined “unit” measurement:
  - Distances between points on the scale are measurable and well-defined
  - Can measure differences!
- **Equal Interval** (between two consecutive unit points)
- **Zero is arbitrary** (not absolute), in many cases human-defined
  - If the variable equals zero, it does not mean there is none of that variable
- **Ratio is meaningless**
- Possible Measure: mode, frequency distribution + median + mean, standard deviation, addition/subtraction
- Example:
  - Temperature measured in Celsius
    - For instance: 10 degrees C, 28 degrees C
  - Year of someone's birth
    - For instance: 1990, 2005, 2010, 2022
- because zero is arbitrary defined, the ratio of two value is meaningless
- 20 degC is not twice as energetic as 10degC



## Ratio Data

- Most precise and **highest** level of measurement
- Ordered
- Equal Intervals
- **Natural Zeros:**
  - If the variable equals zero, it means there is none of that variable
  - Not arbitrary
- Possible Measure: mode, frequency distribution + median + mean, standard deviation, addition/subtraction + multiplication and division (ratio)
- Example:
  - Weights
    - 10 KG, 20 KG, 30 KG
  - Time
    - 10 Seconds, 1 Hour, 1 Day
- well defined ratio, 20 kg twice as heavy as 10 kg



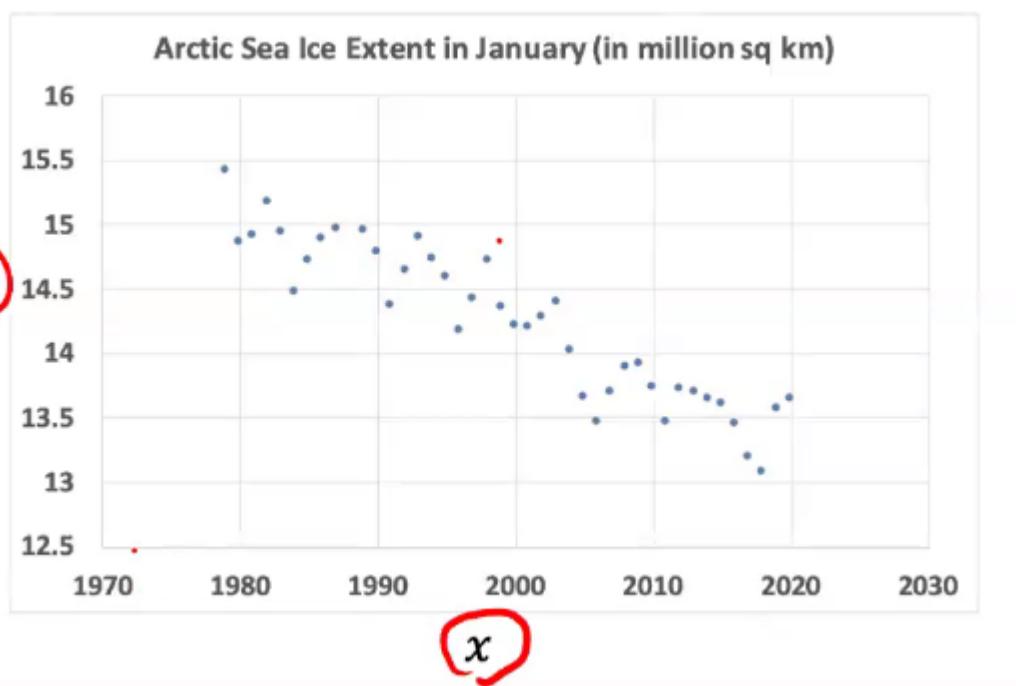
## Example

1. **Favorite Restaurant**
  - Mcdonald's, Burger King, Subway, KFC, ...
2. **Weight of luggage measured in KG**
3. **SAT Scores: note that, SAT ranges is [400, 1600]**
4. **Size of Packed Eggs in supermarkets**
  - Small, Medium, Large, Extra Large, ...
5. **Military rank**
  - General, Major, Captain, ...
6. **Number of people in a household**
  - 1, 2, 3, 4, 5, ...
7. **Credit Score in United States: the range is [300, 850]**



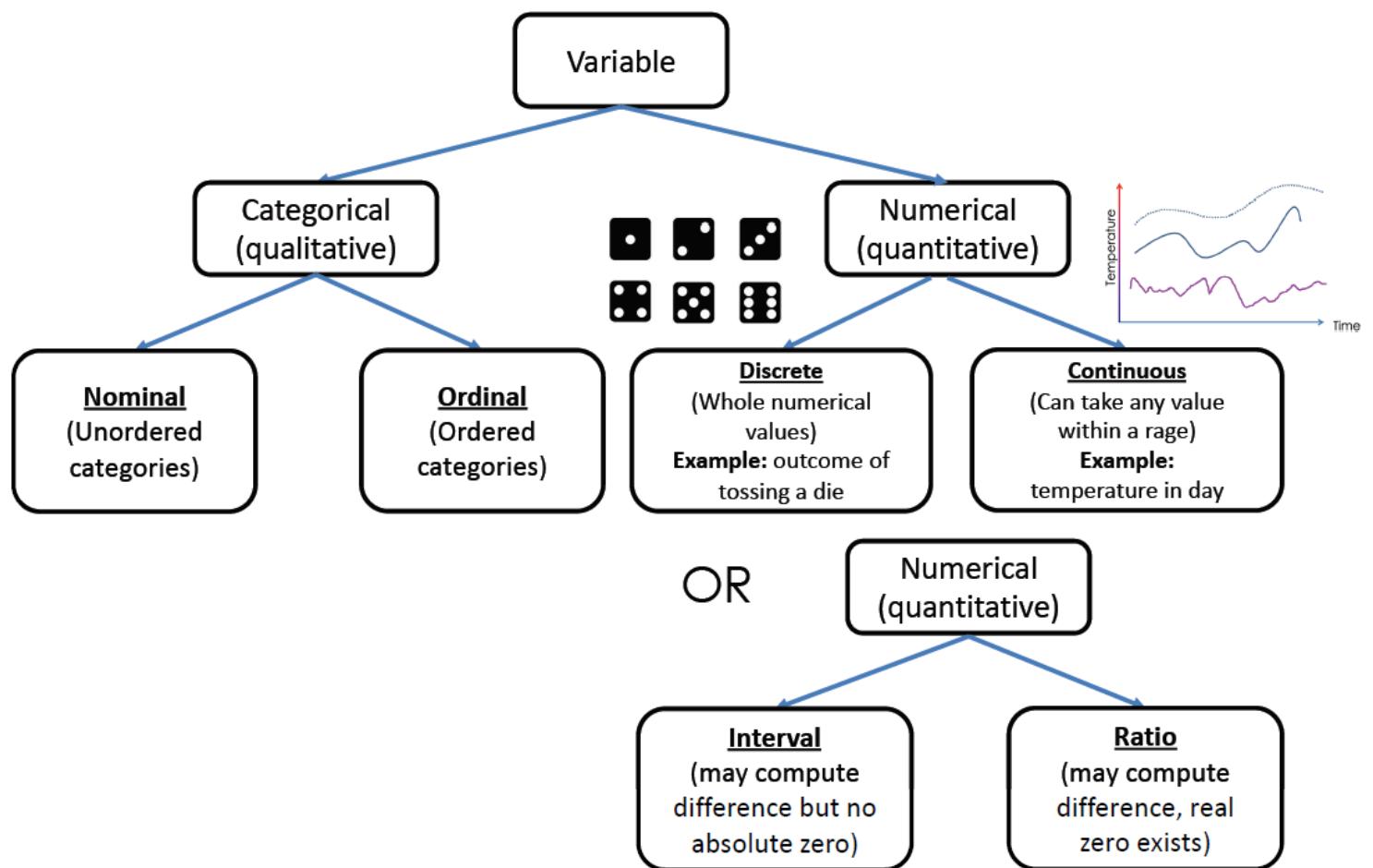
1. Nominal

2. Ratio
3. Interval
4. Ordinal
5. Ordinal
6. Ratio
7. Interval



- x is interval, y is ratio

## Based on Numerical/Categorical



## Other Aspects

- Missing data: data that is missing and you do not know the mechanism.
  - You should use a single common code for all missing values (for example, “NA”), rather than leaving any entries blank.

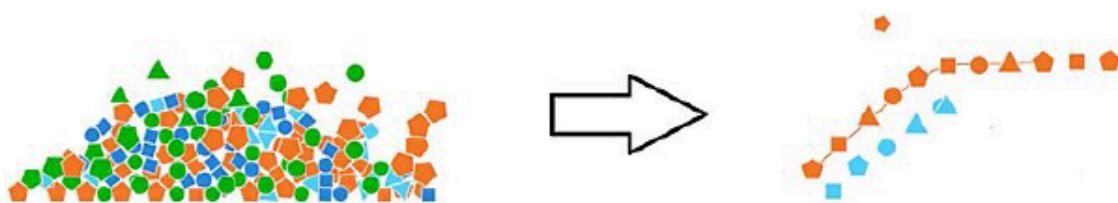
NUS student	Age	Country of birth
Olivia Tan	20	Singapore
Hendra Setiawan	19	Indonesia
John Smith	19	NA

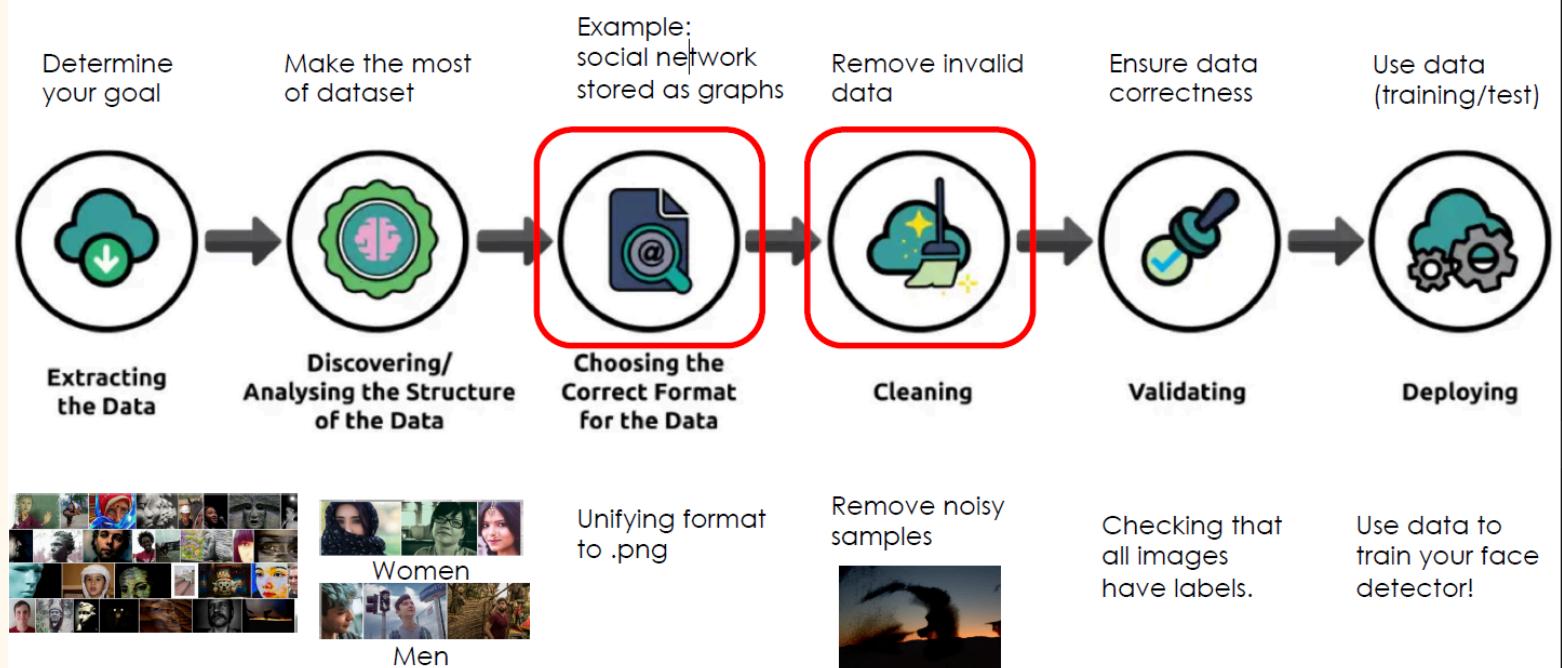
## Data Wrangling and Cleaning (Data Preprocessing)



- Prepare data for ML/feature extraction
- Cleaning is subset of wrangling

- The process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.
- In short, transforms data to gain insight
- It is a general process!





## Collect Human Face Images for Face Detector

Credit: <https://understandingdata.com/what-is-data-wrangling/>

- second step is to group the data

### Formatting Data

#### Binary Coding

- **Binary Coding** to convert categories into binary form
  - One-hot encoding: unify several entities within one vector
    - Example: the color of a pixel can be red, yellow, or green
    - Very common in classification tasks!

$$\text{red} = [1, 0, 0]$$

$$\text{yellow} = [0, 1, 0]$$

$$\text{green} = [0, 0, 1]$$

- Format Labels/output
- One-hot encoding is one type of binary coding
  - Put stuff in one vector

!  $\text{red} = [1, 0, 0]$

$\text{yellow} = [0, 1, 0]$

$\text{green} = [0, 0, 1]$

- o One-hot memes, no inbetween, only one attribute can be present

■  $\boxed{[1, 0, 0]}$

## Normalisation

Often we have feature vectors in which features are on different scales. We let  $m$  be the number of training samples and  $d$  be the number of features as usual. Say the feature vectors are

$$\mathbf{x}_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,d} \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,d} \end{bmatrix}, \quad \dots \quad \text{and} \quad \mathbf{x}_m = \begin{bmatrix} x_{m,1} \\ x_{m,2} \\ \vdots \\ x_{m,d} \end{bmatrix}. \quad (2.2)$$

The first feature  $x_{i,1}$  for  $1 \leq i \leq m$  could be the height of students measured in centimeters so the dynamic range is  $[x_{\min,1}, x_{\max,1}] = [140, 190]$ . The second feature  $x_{i,2}$  for  $1 \leq i \leq m$  could measure the (American) shoe size of the students so it ranges from  $[x_{\min,2}, x_{\max,2}] = [6, 13]$ . So even if both features are deemed equally “important”, unfortunately, any machine learning method would place more importance on the first feature because of its larger values, which is not ideal. Thus, we have to scale or normalize the features so that their dynamic ranges are roughly the same. We can use (at least) two methods to do so:

## • Normalization

- Linear Scaling:

scale each variable to  $[0, 1]$

$$x_i = \frac{x_i^{\text{raw}} - x^{\min}}{x^{\max} - x^{\min}}, \quad i = 1, 2, \dots, M$$

- Z-score standardization:

each independent dimension of data is normally distributed

$$x_i = \frac{x_i^{\text{raw}} - E[X]}{\sigma(X)}, \quad i = 1, 2, \dots, M.$$

- Format data/input

- Linear scaling

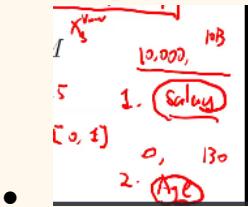
input raw  $1, 2, 3, 5, 8, 9$

$$x_i = \frac{x_i^{\text{raw}} - x^{\min}}{x^{\max} - x^{\min}}, \quad i = 1, 2, \dots, M$$

$$x_3 = \frac{3 - 1}{9 - 1} = \frac{2}{8} = 0.25$$

- o convert into 0-1

- any range to 0-1
- uses for converting two data of very different range into comparable range



- - example is age and salary

$$\bar{x}_{i,1} := \frac{x_{i,1} - x_{\min,1}}{x_{\max,1} - x_{\min,1}}$$

- 

- Z-score

- must be normally distributed

$$\hat{x}_1 = \frac{1}{m} \sum_{i=1}^m x_{i,1}, \quad \text{and} \quad \hat{\sigma}_1 = \sqrt{\frac{1}{m-1} \sum_{i=1}^m (x_{i,1} - \hat{x}_1)^2}$$

- 

$$\bar{x}_{i,1} := \frac{x_{i,1} - \hat{x}_1}{\hat{\sigma}_1}$$

- 

## Data Cleaning

- The process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database.

- Example:

- Clipping outliers



- Handling missing features

Students	Year of Birth	Gender	Height	GPA
Tan Ah Kow	1995	M	1.72	4.2
Ahmad Abdul	X NA	M	1.65	4.1
John Smith	1995	M	1.75	X NA
Chen Lulu	1995	F	X NA	4.0
Raj Kumar	1995	M	1.73	4.5
Li Xiuxiu	1994	F	1.70	3.8

- must make sure outlier is outlier before taking them out, otherwise it's still valuable data
- handling missing features has several ways

# Data Cleaning: Handling missing features

## 1. Removing the examples with missing features from the dataset

- Can be done if the dataset is big enough so we can sacrifice some training examples

## 2. Using a learning algorithm that can deal with missing feature values

- Example: random forest

## 3. Using a data imputation technique

- Only can remove if dataset is big enough
- imputation means fill in the data

### Data Cleaning: Handling missing features: Imputation

- Method 1. Replace the missing value of a feature by an average value of this feature in the dataset:

$$\hat{x}^{(j)} \leftarrow \frac{1}{N} \sum_{i=1}^N x_i^{(j)}$$

- Method 2. Highlight the missing value

- Replace the missing value with a value outside the normal range of values.
- For example, if the normal range is [0, 1], then you can set the missing value to -1.
- Enforce the learning algorithm to learn what is best to do when the feature has a value significantly different from regular values.

- Method 1 is less commonly used, as it's not the best practice
- Method 2, is used to make the algorithm able to learn what to do when encounter -1 in the future
  - -1 is only for the algo, when finding mean and whatnot, ignore the -1

## Data Integrity and Visualisation

### Data Integrity

- Data integrity is the maintenance and the assurance of data accuracy and consistency;
  - A critical aspect to the design, implementation, and usage of any system that stores, processes, or retrieves data.
  - Very broad concept!
- Example:
  - In a dataset, numeric columns/cells should not accept alphabetic data.
  - A binary entry should only allow binary inputs

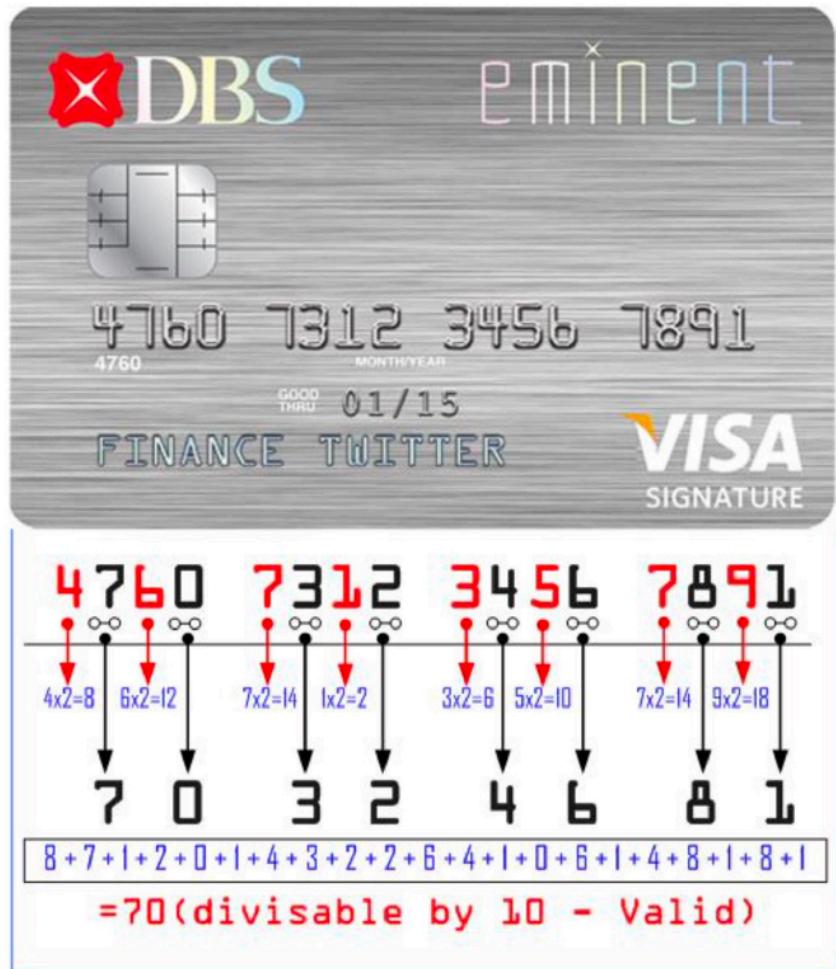
We can only select one of these

Organization	User Type	Is_Emergency ↑	External Profile Entered	Subject Areas		Bid	Relevance	Candidate Suggestion Rank	Tpms Rank	Quota	Number Of Assignments
				Primary	Secondary						
National University of Singapore	Student, >3 times as reviewer for CVPR, ICCV, or ECCV			Machine learning	3D from single images; Adversarial attack and defense; Computer vision theory; Explainable computer vision; Self- & semi-& meta- & unsupervised learning; Transfer/ low-shot/ long-tail learning; Vision + graphics	Not Entered	0.08	1	1434		4
Zhejiang University	Faculty/Researcher, 3-10 times as reviewer for CVPR, ICCV, or ECCV	No		Transfer/ low-shot/ long-tail learning	Efficient learning and inferences; Explainable computer vision; Image and video synthesis and generation; Recognition; detection, categorization,	Not Entered	0.16	7			2



4760 7312 3456 7891

Issuer Number      Bank Number      Account Number      Check Digits



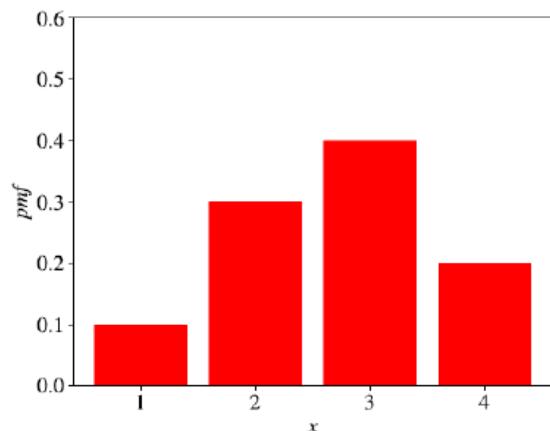


## Chart Types

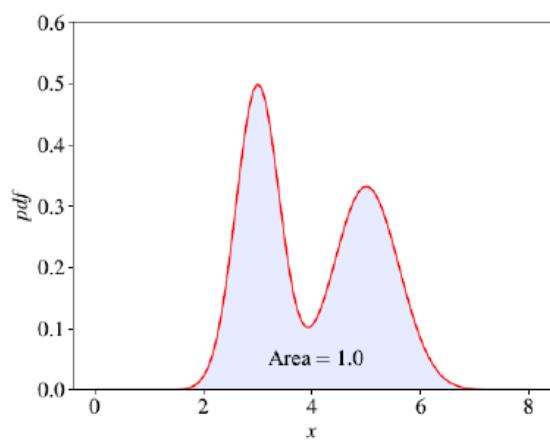


[www.adatis.co.uk](http://www.adatis.co.uk)

## Visualization: Distribution



Probability Mass Function



Probability Density Function

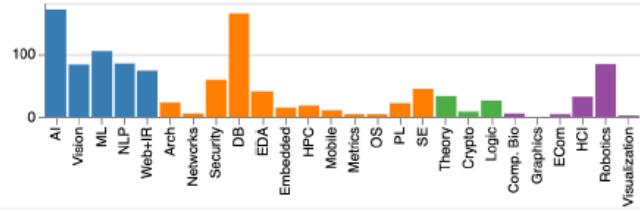
## Visualization: Bars

## CSRankings: Computer Science Rankings

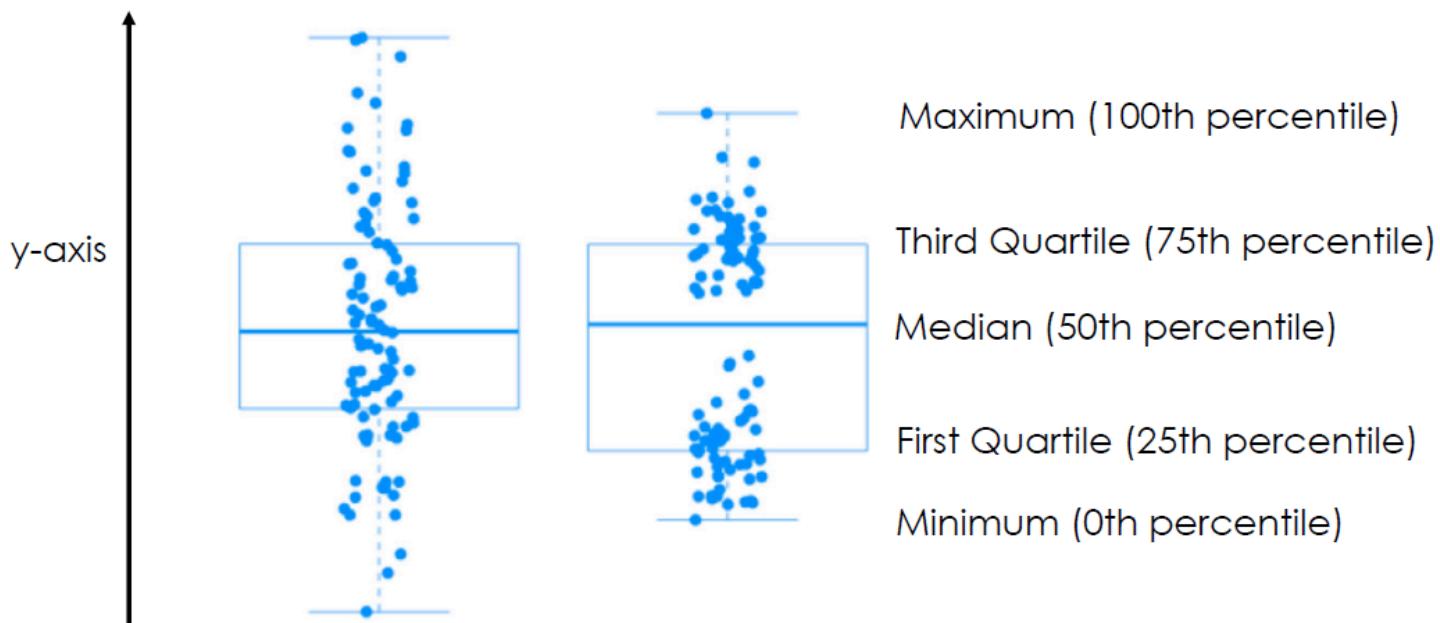
CSRankings is a metrics-based ranking of top computer science institutions around the world. Click on a triangle (▶) to expand areas or institutions. Click on a name to go to a faculty member's home page. Click on a chart icon (the bar chart icon after a name or institution) to see the distribution of their publication areas as a bar chart. Click on a Google Scholar icon (✉) to see publications, and click on the DBLP logo (✳) to go to a DBLP entry. Applying to grad school? Read this first.

Rank institutions in  by publications from  to

12	▶ Georgia Institute of Technology		9.1	94
13	▶ University of Maryland - College Park		8.2	83
14	▶ University of Wisconsin - Madison		7.6	65
15	▶ Columbia University		7.4	55
15	▼ National University of Singapore		7.4	66

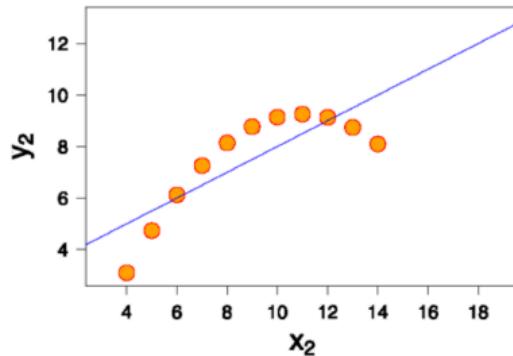
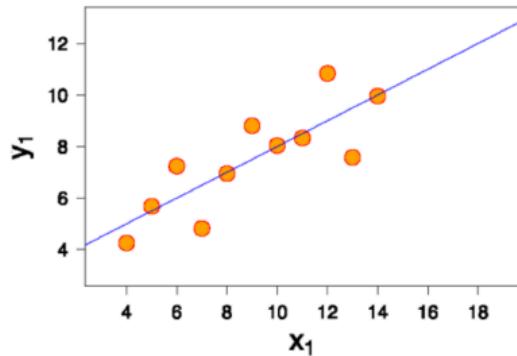


# Visualization: Boxplots

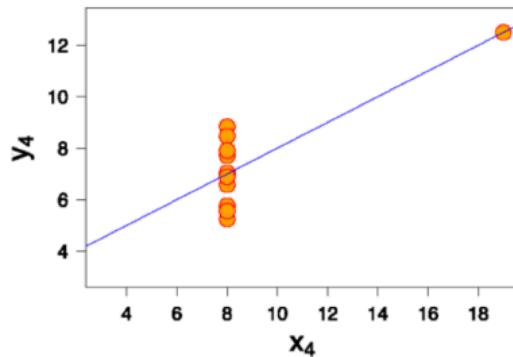
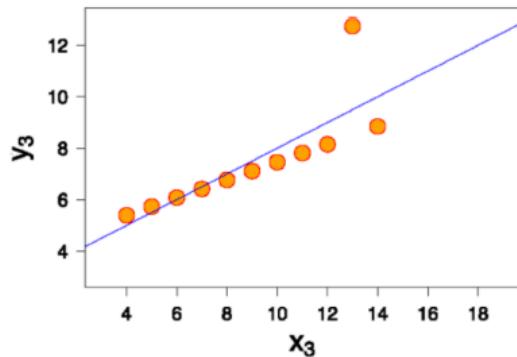


- The first quartile ( $Q_1$ ) is defined as the middle number between the smallest number (i.e., Minimum) and the median of the data set.
- The third quartile ( $Q_3$ ) is the middle number between the median and the highest value (i.e., Maximum) of the data set.

## Why Visualization is Necessary



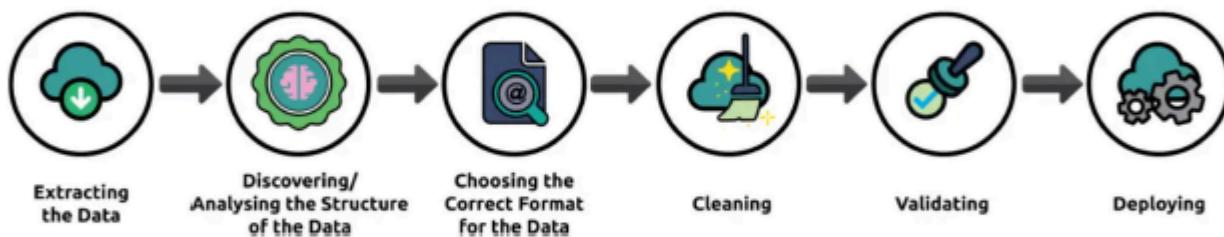
Four datasets with **identical** means, variances and regression lines!



Hence, we need visualization to show their difference!

## Summary and Practice Question

- Types of data
  - NOIR
- Data wrangling and cleaning



- Data integrity and visualization
  - Integrity: Design
  - Visualization: Graphical Representation

Color	Size	Shape
Blue	Large	Ring
Red	Large	Triangle
Orange	Large	Diamond
Green	Small	Circle
Yellow	Small	Arrow
Blue	Large	Rectangle
Red	Large	Circle
Green	Small	Diamond

What are the NOIR data types of *color*, *size*, and *shape* in the table?

- Colour - Nominal
- Size - Ordinal
- Shape - Nominal

### Lec3: Introduction to Linear Algebra, Probability and Statistics

#### Introduction to Linear Algebra

Notations, Vectors, Matrices

- A **scalar** is a simple numerical value, like 15 or  $-3.25$ 
  - Focus on **real** numbers
- **Variables** or **constants** that take scalar values are denoted by an *italic* letter, like  $x$  or  $a$

- A **vector** is an ordered list of scalar values
  - Denoted by a **bold character**, e.g.  $\mathbf{x}$  or  $\mathbf{a}$

- In many books, vectors are written column-wise:

$$\mathbf{a} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ 5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- The three vectors above are two-dimensional, or have two elements

- We denote an **entry** or **attribute** of a vector as an italic value with an index, e.g.  $a^{(j)}$  or  $x^{(j)}$ .
  - The index  $j$  denotes a specific dimension of the vector, the position of an attribute in the list

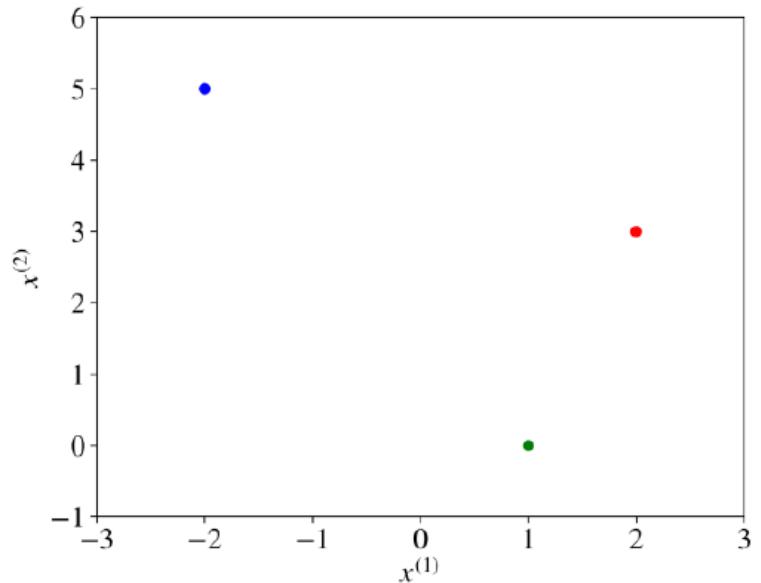
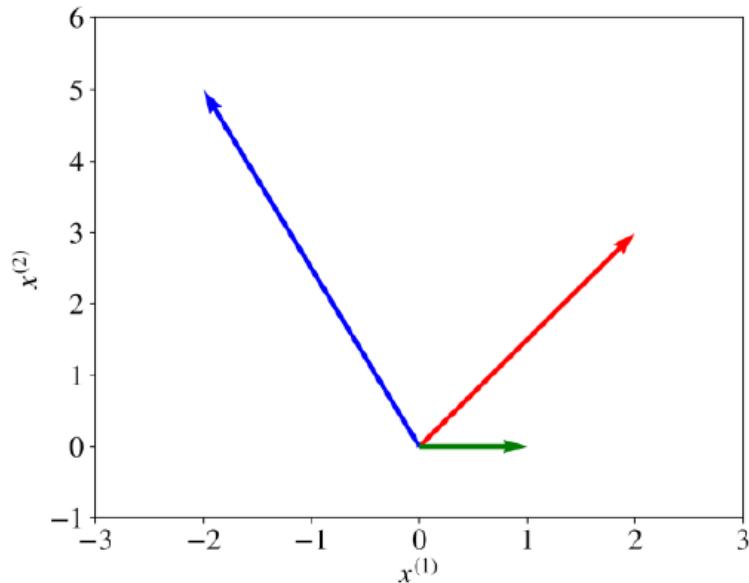
$$\mathbf{a} = \begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \text{or more commonly} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

- Note:

- $x^{(j)}$  is not to be confused with the power operation, e.g.,  $x^2$  (squared)
- Square of an indexed attribute of a vector is denoted as  $(x^{(j)})^2$ .

- **Vectors** can be visualized as, in a multi-dimensional space,
  - arrows that point to some directions, or
  - points

Illustrations of three two-dimensional vectors,  $\mathbf{a} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ ,  $\mathbf{b} = \begin{bmatrix} -2 \\ 5 \end{bmatrix}$ , and  $\mathbf{c} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



- A **matrix** is a rectangular array of numbers arranged in rows and columns
  - Denoted with bold capital letters, such as **X** or **W**
  - An example of a matrix with two rows and three columns:
$$\mathbf{X} = \begin{bmatrix} 2 & 4 & -3 \\ 21 & -6 & -1 \end{bmatrix}$$
- A **set** is an unordered collection of unique elements
  - When an element  $x$  belongs to a set  $S$ , we write  $x \in S$ .
  - A special set denoted **R** includes all real numbers from minus infinity to plus infinity
- Note:
  - For elements in matrix **X**, we shall use the indexing  $x_{1,1}$  where the first and second indices indicate the row and the column position.
  - Usually, for input data, rows represent samples and columns represent features
- set is unordered with unique elements

$S = \{ 3, 2 \}$

or **W** Feature 1 Feature 2 Feature 3

$$\mathbf{X} = \begin{bmatrix} 2 & 4 & -3 \\ 21 & -6 & -1 \end{bmatrix} \begin{matrix} \text{Sample 1} \\ \text{Sample 2} \end{matrix}$$

- **Capital Sigma:** the **summation** over a collection  $\{x_1, x_2, x_3, x_4, \dots, x_m\}$  is denoted by:

$$\sum_{i=1}^m x_i = x_1 + x_2 + \dots + x_{m-1} + x_m$$

- **Capital Pi:** the **product** over a collection  $\{x_1, x_2, x_3, x_4, \dots, x_m\}$  is denoted by:

$$\prod_{i=1}^m x_i = x_1 \cdot x_2 \cdot \dots \cdot x_{m-1} \cdot x_m$$

### Linear dependence and independence

- A collection of  $d$ -vectors  $\mathbf{x}_1, \dots, \mathbf{x}_m$  (with  $m \geq 1$ ) is called **linearly dependent** if

$$\beta_1 \mathbf{x}_1 + \dots + \beta_m \mathbf{x}_m = 0$$

**holds for some**  $\beta_1, \dots, \beta_m$  that are **not all zero**.

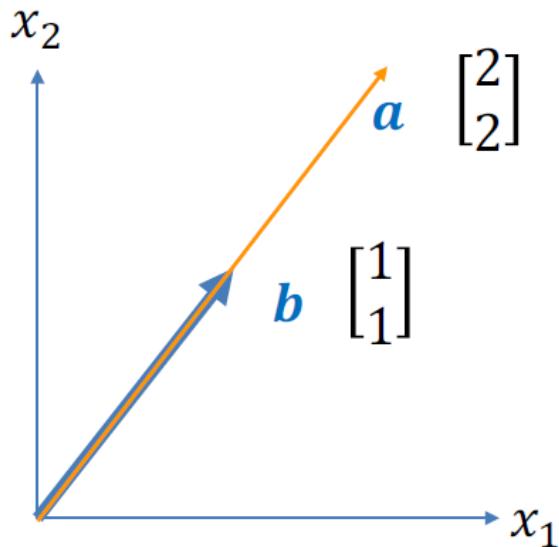
- A collection of  $d$ -vectors  $\mathbf{x}_1, \dots, \mathbf{x}_m$  (with  $m \geq 1$ ) is called **linearly independent** if it is not linearly dependent, which means that

$$\beta_1 \mathbf{x}_1 + \dots + \beta_m \mathbf{x}_m = 0$$

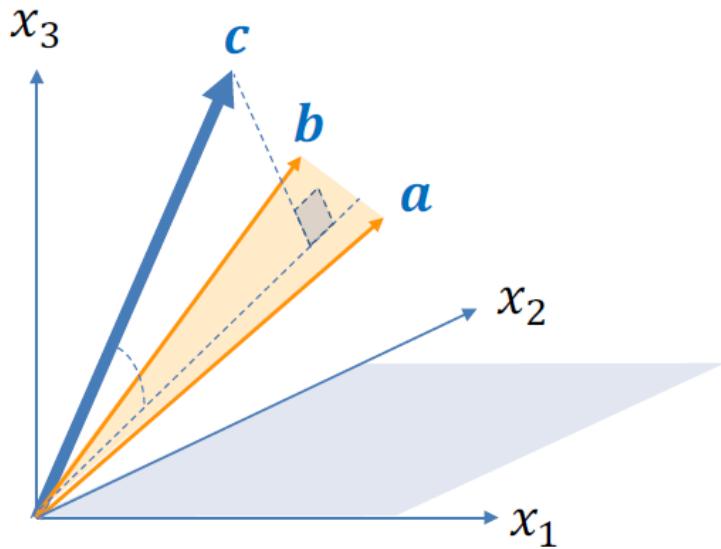
**only holds** for  $\beta_1 = \dots = \beta_m = 0$ .

Note: If all rows or columns of a square matrix  $\mathbf{X}$  are **linearly independent**, then  $\mathbf{X}$  is **invertible**.

# Geometry of dependency and independency



$$\beta_1 \mathbf{a} + \beta_2 \mathbf{b} = 0$$



$$\beta_1 \mathbf{a} + \beta_2 \mathbf{b} \neq \beta_3 \mathbf{c}$$

## Systems of Linear Equations

These equations can be written compactly in matrix-vector notation:

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

Where

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,d} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}.$$

## Note:

- The data matrix  $\mathbf{X} \in \mathcal{R}^{m \times d}$  and the target vector  $\mathbf{y} \in \mathcal{R}^m$  are given
- The unknown vector of parameters  $\mathbf{w} \in \mathcal{R}^d$  is to be learnt
- The rank( $\mathbf{X}$ ) corresponds to the maximal number of linearly independent columns/rows of  $\mathbf{X}$ .

- Learn the solution to  $Xw = y$ , to map  $X$  to  $y$ , hence leaning  $w$
- The principled way for computing rank is to do Echelon Form
  - <https://stattrek.com/matrix-algebra/echelon-transform.aspx#MatrixA>
- For small-size matrices, however, the rank is in many cases easy to estimate
- What is the rank of

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 100 & 100 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 & 3 \\ 0 & -3 & 3 \\ 1 & 1 & 0 \end{bmatrix}$$

## Causality

≡ GEA1000 Notes

- Causality, or causation is:
  - The influence by which one event or process (i.e., **cause**) contributes to another (i.e. **effect**),
  - The **cause** is partly responsible for the **effect**, and the **effect** is partly dependent on the **cause**
- Causality relates to an extremely very wide domain of subjects: philosophy, science, management, humanity.
- Causality research is extremely complex
  - Researcher can never be completely certain that there are **no other factors** influencing the causal relationship,
  - In most cases, we can only say "probably" causal.
- Partially or partly, cannot be completely sure that there is anything else that something is affecting the relationship

- (**Probable**) causal relations or non-causal?

- New web design implemented ? Web page traffic increased Yes
- Your height and weight ? Gets A in EE2211 No
- Uploaded new app store images ? Downloads increased by 2X Yes
- One works hard and attends lectures/tutorials ? Gets A in EE2211 Yes
- Your favorite color ? Your GPA in NUS No

- One popular way to causal data analysis is **Randomized Controlled Trial (RCT)**
  - A study design that randomly assigns participants into an experimental group or a control group.
  - As the study is conducted, the only expected difference between two groups is the outcome variable being studied.
- Example:
  - To decide whether smoking and lung cancer has a causal relation, we put participants into experimental group (people who smoke) and control group (people who don't smoke), and check whether they develop lung cancer eventually.
- RCT is sometimes infeasible to conduct, and also has moral issues.

Causality is very difficult to study, but it's possible

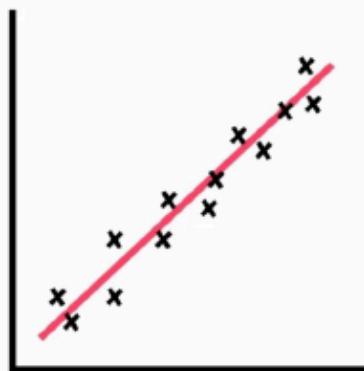
	Experiments	Observational Studies
Assignment	By researchers	Participants assign themselves
Randomisation	Preferable	Not possible
Ethical issues	Possible (if intervention may be harmful)	Unlikely
Confounders	Unlikely (if randomisation is done on enough participants)	Usually exist many
Possible to show causation	Yes (in the ideal case)	Very difficult
Able to show association	Yes	

## Causality is a Statistical Relationship

- Decades of data show a clear causal relationship between smoking and cancer.
- If one smokes, it is a sure thing that his/her risk of cancer will increase.
- But it is not a sure thing that one will get cancer.
- The relationship is not deterministic.

## Correlation vs Causality

- In statistics, **correlation** is any **statistical relationship**, whether causal or not, between two random variables.
- Correlations are useful because they can indicate a **predictive relationship** that can be exploited in practice.



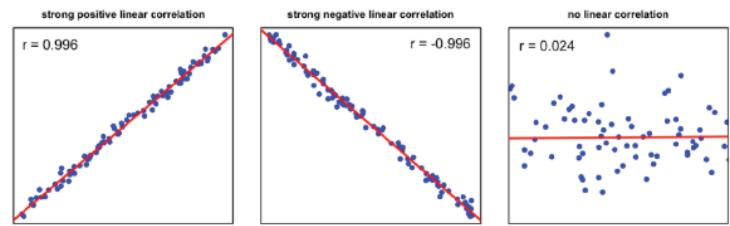
Positive  
Correlation



Negative  
Correlation

- Linear correlation coefficient,  $r$ , which is also known as the Pearson Coefficient.

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{s_{xy}}{s_x s_y},$$



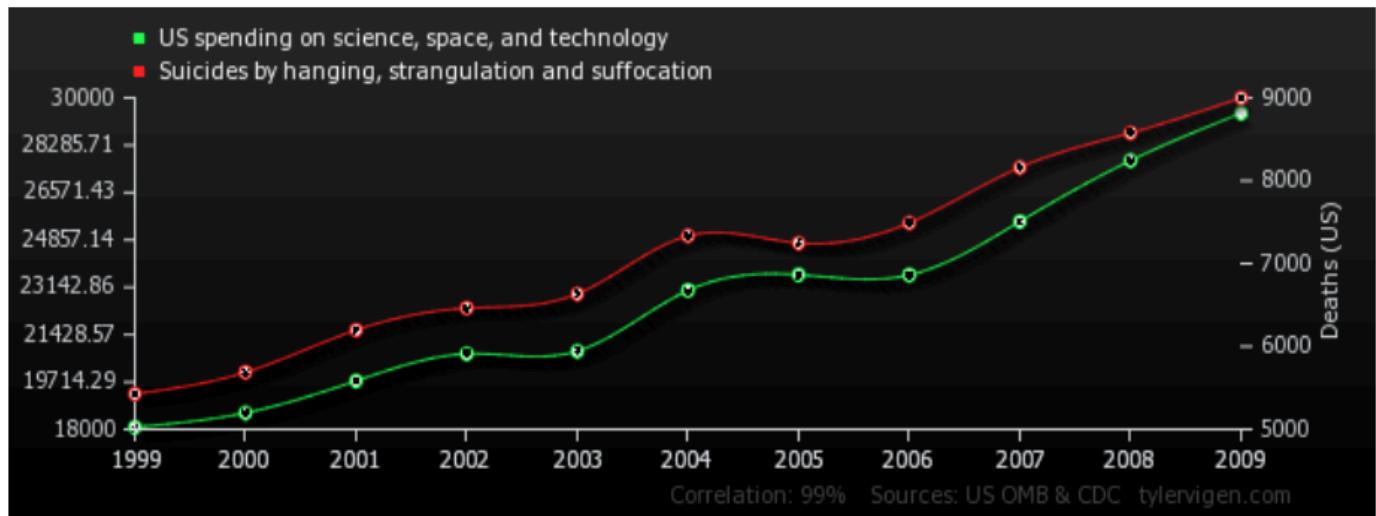
Strong linear relationship	$r > 0.9$
Medium linear relationship	$0.7 < r \leq 0.9$
Weak linear relationship	$0.5 < r \leq 0.7$
No or doubtful linear relationship	$0 < r \leq 0.5$

The same holds for negative values.

<https://www.geo.fu-berlin.de/en/v/soga/>

Correlation does not imply causation!

- Some great examples of correlations that can be calculated but are clearly not causally related appear at <http://tylervigen.com/>

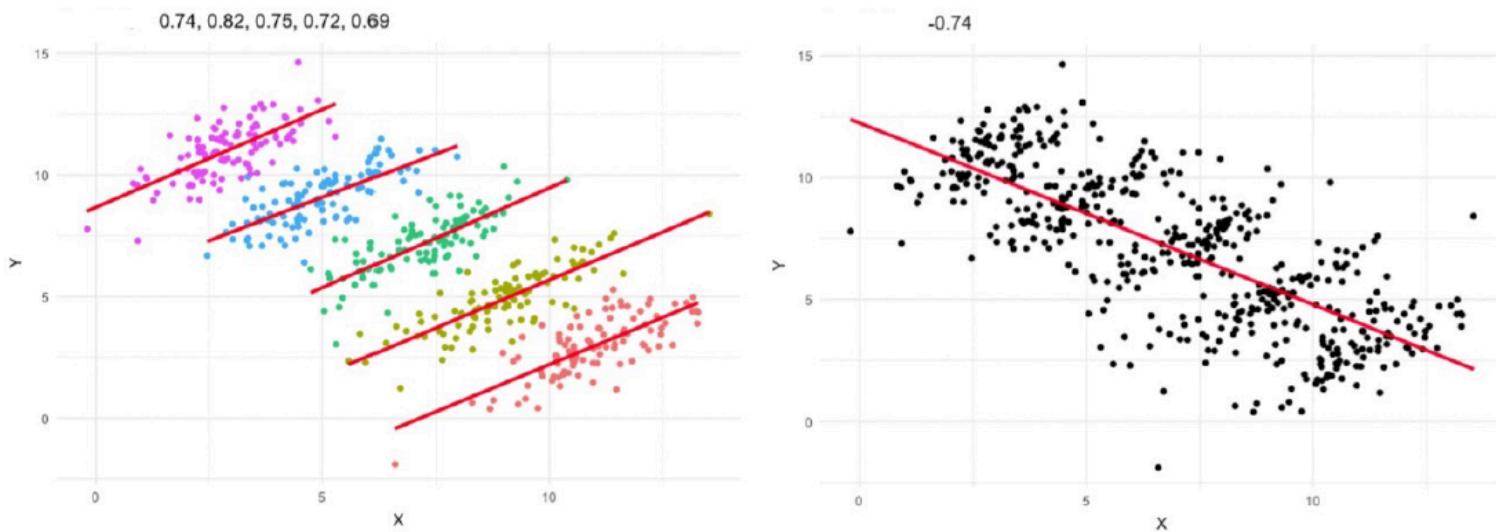


<https://tylervigen.com>

## Simpson's paradox

Simpson's paradox: GEA1000 Notes

- **Simpson's paradox** is a phenomenon in probability and statistics, in which a trend appears in several different groups of data but disappears or reverses when these groups are combined.



The same set of samples!

### Example

- Batting Average in professional baseball game
- Two well-known players, Derek Jeter and David Justice

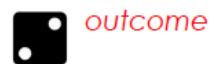
Batter \ Year	1995		1996		Combined	
Batter	# of wins	# of games	# of wins	# of games	# of wins	# of games
Derek Jeter	12/48	.250	183/582	.314	195/630	.310
David Justice	104/411	.253	45/140	.321	149/551	.270

## Random Variable

GEA1000 Notes

EE2012 Notes

- We describe a *random experiment* by describing its procedure and observations of its *outcomes*.
- *Outcomes* are mutual exclusive in the sense that only one outcome occurs in a specific trial of the random experiment.
  - This also means an outcome is not decomposable.
  - All unique outcomes form a *sample space*.
- A subset of sample space  $S$ , denoted as  $A$ , is an *event* in a random experiment  $A \subset S$ , that is meaningful to an application.
  - Example of an event: faces with numbers no greater than 3
- Some books used  $P(\cdot)$  and  $p(\cdot)$  to distinguish between the probability of discrete random variable and the probability of continuous random variables respectively.
- We shall use  $Pr(\cdot)$  for both the above cases



- A **random variable**, usually written as an *italic* capital letter, like  $X$ , is a variable whose possible values are numerical outcomes of a random event.
- There are two types of random variables: **discrete** and **continuous**.

#### Axioms of Probability

Assuming events  $A \subseteq S$  and  $B \subseteq S$ , the probabilities of events related with and must satisfy,

1.  $Pr(A) \geq 0$
2.  $Pr(S) = 1$
3. If  $A \cap B = \emptyset$  , then  $Pr(A \cup B) = Pr(A) + Pr(B)$   
\*otherwise,  $Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B)$

# Two Basic Rules

- Sum Rule

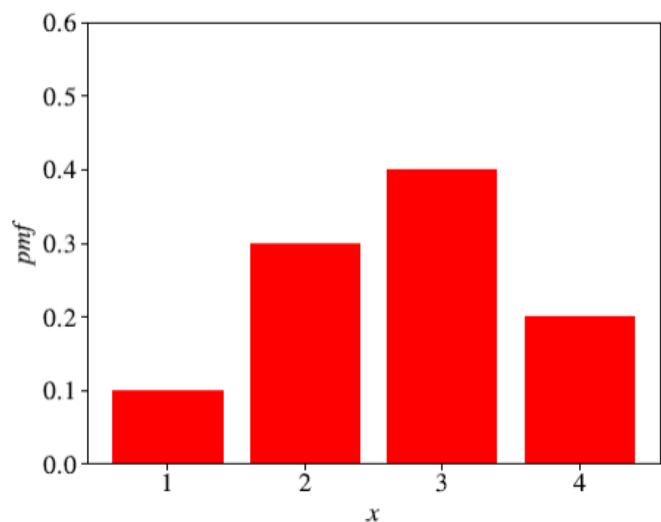
$$\Pr(X = x) = \sum_Y \Pr(X = x, Y = y_i)$$

- Product Rule

$$\Pr(X = x, Y = y) = \Pr(Y = y|X = x) P(X = x)$$

## Discrete Random Variable

- A **discrete random variable (DRV)** takes on only a countable number of distinct values such as **red**, **orange**, **blue** or 1, 2, 3.
- The **probability distribution** of a discrete random variable is described by a list of probabilities associated with each of its possible values.
- This list of probabilities is called a **probability mass function (pmf)**.
  - Like a histogram, except that here the probabilities sum to 1



A probability mass function

- Let a **discrete** random variable  $X$  have  $k$  possible values  $\{x_i\}_{i=1}^k$ .
- The **expectation** of  $X$  denoted as  $E(x)$  is given by,

$$E(x) \stackrel{\text{def}}{=} \sum_{i=1}^k [x_i \cdot \Pr(X = x_i)] \\ = x_1 \cdot \Pr(X = x_1) + x_2 \cdot \Pr(X = x_2) + \cdots + x_k \cdot \Pr(X = x_k)$$

where  $\Pr(X = x_i)$  is the probability that  $X$  has the value  $x_i$  according to the **pmf**.

- The expectation of a random variable is also called the **mean, average** or **expected value** and is frequently denoted with the letter  $\mu$ .

- Another important statistic is the **standard deviation**, defined as,

$$\sigma \stackrel{\text{def}}{=} \sqrt{E[(X - \mu)^2]} .$$

- **Variance**, denoted as  $\sigma^2$  or  $\text{var}(X)$ , is defined as,

$$\sigma^2 = E[(X - \mu)^2]$$

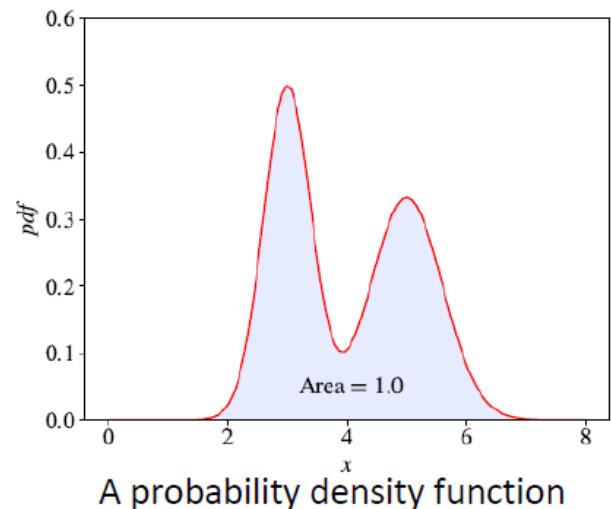
- For a **discrete random variable**, the standard deviation is given by

$$\sigma = \sqrt{\Pr(X = x_1)(x_1 - \mu)^2 + \cdots + \Pr(X = x_k)(x_k - \mu)^2}$$

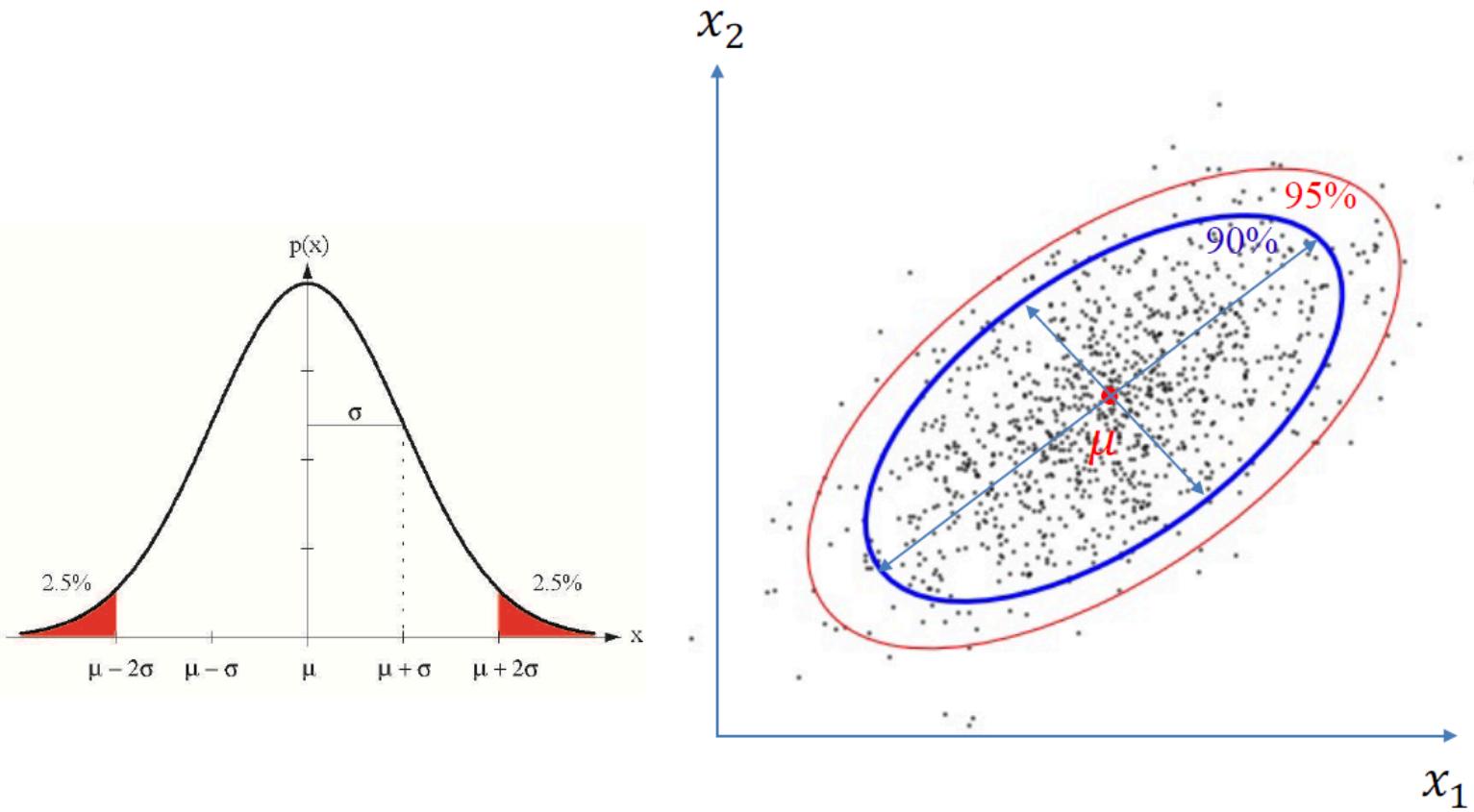
where  $\mu = E(X)$ .

## Continuous Random Variable

- A **continuous random variable (CRV)** takes an infinite number of possible values in some interval.
  - Examples include height, weight, and time.
  - The number of values of a continuous random variable  $X$  is infinite, the probability  $\Pr(X = c)$  for any  $c$  is 0
  - Therefore, instead of the list of probabilities, the probability distribution of a CRV (a continuous probability distribution) is described by a **probability density function (pdf)**.
  - The pdf is a function whose range is **nonnegative** and the **area under the curve is equal to 1**.



- The **expectation** of a continuous random variable  $X$  is given by  $E[x] \stackrel{\text{def}}{=} \int_R x f_X(x) dx$  where  $f_X$  is the **pdf** of the variable  $X$  and  $\int_R$  is the integral of function  $x f_X$ .
- The **variance** of a continuous random variable  $X$  is given by  $\sigma^2 \stackrel{\text{def}}{=} \int_R (X - \mu)^2 f_X(x) dx$
- **Integral** is an equivalent of the **summation** over all values of the function when the function has a continuous domain.
- It equals the **area under the curve** of the function.
- The property of the pdf that the **area under its curve is 1** mathematically means that  $\int_R f_X(x) dx = 1$



## Example 1

- **Independent random variables**
- Consider tossing a fair coin twice, what is the probability of having (H,H)? Assuming a coin has two sides, H=head and T=Tail
  - $\Pr(x=H, y=H) = \Pr(x=H)\Pr(y=H) = (1/2)(1/2) = 1/4$

## Example 2

- **Dependent random variables**
- Given 2 balls with different colors (**Red** and Black), what is the probability of first drawing B and then **R**? Assuming we are drawing the balls **without replacement**.
- The space of outcomes of taking two balls sequentially without replacement:  
 $B-R$ ,  $R-B$ 
  - Thus having **B-R** is  $1/2$ .
- Mathematically:
  - $\Pr(x=B, y=R) = \Pr(y=R | x=B) \Pr(x=B) = 1 \times (1/2) = 1/2$

### Conditional Probability

### Example 3

- **Dependent random variables**
- Given 3 balls with different colors (R, G, B), and we draw 2 balls. What is the probability of first having B and then G, if we draw **without replacement**?
- The space of outcomes of taking two balls sequentially without replacement:

R-G | G-B | B-R

R-B | G-R | B-G      Thus,  $\Pr(y=G, x=B) = 1/6$

- Mathematically:

$$\begin{aligned}\Pr(y=G, x=B) &= \Pr(y=G \mid x=B) \Pr(x=B) \\ &= (1/2) \times (1/3) \\ &= 1/6\end{aligned}$$

### Bayes' Rule

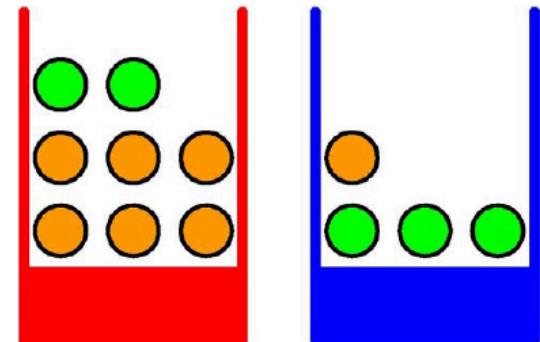
- The conditional probability  $\Pr(Y = y \mid X = x)$  is the probability of the random variable  $Y$  to have a specific value  $y$ , given that another random variable  $X$  has a specific value of  $x$ .
- The **Bayes' Rule** (also known as the **Bayes' Theorem**):

$$\Pr(Y = y \mid X = x) = \frac{\text{likelihood} \quad \text{prior}}{\text{posterior} \quad \text{evidence}} \quad \Pr(X = x \mid Y = y) \Pr(Y = y)$$

### Example

- Drawing a sample of fruit from a box
  - First pick a box, and then draw a sample of fruit from it
  - B: variable for Box, can be  $r$  (red) or  $b$  (blue)
  - F: variable for Fruit, can be  $o$  (orange) or  $a$  (apple)

- $\Pr(B=r)=0.4$  prior
- $\Pr(F=o | B=r)= 0.75$  likelihood
- $\Pr(F=o)= 0.45$  evidence



$$\begin{aligned} \Pr(B=r | F=o) &= \Pr(F=o | B=r) * \Pr(B=r) / \Pr(F=o) \\ &= 0.75 * 0.4 / 0.45 = 2/3 \quad \text{posterior} \end{aligned}$$

### Summary and Practice Question

Suppose the random variable  $X$  has the following probability mass function (pmf) listed in the table below.  $k$  is unknown.

X	1	2	3	4	5
Pr[X]	0.1	0.05	0.05	0.6	$k$

What is the probability that  $X$  takes a value of odd numbers?

$$\begin{aligned} k &= 0.2 \\ \Pr(X = \text{odd}) &= 0.35 \end{aligned}$$

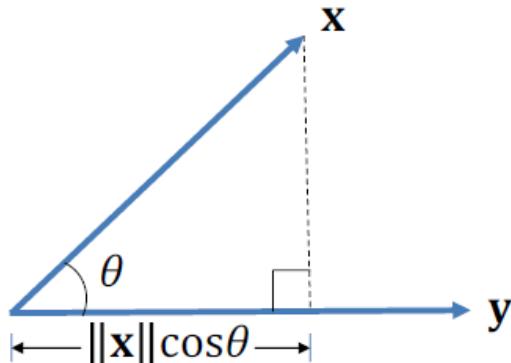
## Lec4: Systems of linear equations

### Vector and Matrix Operation

Dot Product = Inner Product

### Dot Product or Inner Product of Vectors:

$$\begin{aligned} \mathbf{x} \cdot \mathbf{y} &= \mathbf{x}^T \mathbf{y} \\ &= [x_1 \ x_2] \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \\ &= x_1 y_1 + x_2 y_2 \end{aligned}$$



### Geometric definition:

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos\theta$$

where  $\theta$  is the angle between  $\mathbf{x}$  and  $\mathbf{y}$ ,  
and  $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$  is the Euclidean length of vector  $\mathbf{x}$

E.g.  $\mathbf{a} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ ,  $\mathbf{c} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \mathbf{a} \cdot \mathbf{c} = 2*1 + 3*0 = 2$

	Orthogonal Projection of a vector to a subspace
	$S = \{u_1, u_2, \dots, u_k\}$ is a orthonormal basis for a subspace $V \subseteq \mathbb{R}^n$ , if $u_i \in \mathbb{R}^n$ , $u_i \perp u_j$ for $i \neq j$ and $\ u_i\  = 1$
requires the $\Rightarrow$	$W_p = (w \cdot u_1)u_1 + (w \cdot u_2)u_2 + \dots + (w \cdot u_k)u_k$ = projection of $w$ to $V$
orthonormal basis	for any different orthonormal basis for $V$ , projection of $w$ to $V$ is <del>the same</del> unique.
of $V$ to find the projection	Example: $\{(1, 0)\}$ & $\{\frac{1}{\sqrt{2}}(1, 0), \frac{1}{\sqrt{2}}(0, 1)\}$ are orthonormal basis for the $x-y$ plane in $\mathbb{R}^3$
1) get by writing Gram-Schmidt	Let $w = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ then $w_p = ((1) \cdot (1)) \begin{pmatrix} 1 \\ 0 \end{pmatrix} + ((1) \cdot (0)) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
2) least square approximation	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$

- The dot product is the to find the projection of  $y$  to  $x$  or  $x$  to  $y$

# Inverse of Matrix

if $A$ is $n \times n$ , $(A I) \rightarrow (R I)$ $R \neq I$ $\Rightarrow R$ has zero row non pivot col. per steed pivots in non trivial sol	$AB = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$
Inverse of Matrix	invertible } only for square invertible } singular } matrices
if $A$ is non-singular $\rightarrow$	A square matrix $A$ of order $n$ is invertible if there exists a square matrix $B$ of order $n$ such that
if $A$ is invertible	$AB = I_n = BA$ . A square matrix is singular if it is not invertible.
$A\vec{x} = \vec{b}$	Uniqueness of inverse - if $B$ & $C$ are both inverses of a square matrix $A$ , then $B=C$ $\hookrightarrow BA = I = AC \Rightarrow B = BI = BIA(C) = (BA)C = IC = C \Rightarrow B = C \#$
$\vec{x} = A^{-1}\vec{b}$	$\hookrightarrow$ since inverse is unique, it is given a notation $A^{-1}$ for inverse of $A$ . $AA^{-1} = I = A^{-1}A$ Not $\frac{1}{A}$
$\therefore$ a unique solution exists & system is consistent	for non-square matrix, if left inverse exists then there is no right inverse and vice versa. Right & Left inverse need not be unique. $\hookrightarrow$ if both right & left inverse exist then $A$ must be square & right = left inverse and are unique.
Proof	Cancellation Law for matrices - if $A$ is invertible and $\begin{matrix} AB = AC \\ BA = CA \end{matrix}$ then $B = C$ $\because A^{-1}AB = A^{-1}AC \Rightarrow B = C \therefore$ this is true if $A^{-1}$ exists $\therefore$ inverse of symmetric matrix is symmetric
i) $(A^{-1})^{-1} = A$	i) $(A^{-1})^{-1} = A$
ii) $I = I^T = (AA^{-1})^T$ $I = (A^{-1})^T(A^{-1})$	ii) for any non-zero $a \in \mathbb{R}$ , $(aA)$ is invertible with inverse $(aA)^{-1} = \frac{1}{a}A^{-1}$ $A = A^T$ $(A^{-1})^T = (A^T)^{-1}$
$(A^{-1})^T = (A^T)^{-1}$	iii) $A^T$ is invertible with inverse $(A^T)^{-1} = (A^{-1})^T$ , inverse of the transpose is the transpose of the inverse. $A = (A^{-1})^{-1}$ $= (A^{-1})^T$
$(AB)^{-1} = B^{-1}A^{-1}$	iv) $(AB)^{-1} = B^{-1}A^{-1}$ , for $(AB)^{-1}$ to exist $A, B$ must be invertible. $\Rightarrow (A_1A_2 \dots A_k)^{-1} = A_k^{-1} \dots A_2^{-1}A_1^{-1}$ $A^{-1} = (A^{-1})^T$
$(AB)C = I = C(AB)$	<u>Finding Inverse For <math>2 \times 2</math></u> $\hookrightarrow C = AB \text{ is invertible}$ $\hookrightarrow AB \text{ is invertible}$ $\hookrightarrow A, B, C \text{ all invertible}$ $\hookrightarrow AB \text{ must be invertible}$ $\hookrightarrow$ inverse $X$ of $A$ satisfies $AX = I$ $\hookrightarrow (A I) \xrightarrow{\text{REF}} (R B)$ $\therefore A^{-1} = R$ unique solution $\therefore A^{-1} \text{ is symmetric}$ $\hookrightarrow$ no solution
$ABC = I$ $A^{-1}ABC = A^{-1}I$ $B^{-1}A^{-1} = A^{-1}A^{-1}$ $B^{-1}BC = B^{-1}A^{-1}$ $\therefore B^{-1} = A^{-1}A^{-1} \#$	$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ $\hookrightarrow$ can explain only identity matrix
$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$	$\det A = ad - bc$ $A^{-1}$ exists if $ad - bc \neq 0$ $\therefore$ if $R = I$ then $B = A^{-1}$ $\therefore R$ has a zero row & determinant of $A = 0$ , $A^{-1}$ doesn't exist.

**Determinant computation**  $\det(A) = \sum_{j=1}^k (-1)^{i+j} a_{ij} M_{ij}$

Example: 3x3 matrix, use the first row ( $i = 1$ )

$$\begin{aligned}
 |A| &= \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} \square & \square & \square \\ \square & e & f \\ \square & h & i \end{vmatrix} - b \begin{vmatrix} \square & \square & \square \\ d & \square & f \\ g & \square & i \end{vmatrix} + c \begin{vmatrix} \square & \square & \square \\ d & e & \square \\ g & h & \square \end{vmatrix} \\
 &= a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\
 &= a(ei - fh) - b(di - fg) + c(dh - eg)
 \end{aligned}$$

Consider a 3x3 matrix

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}. \quad \text{The minor of } a_{22} = \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix}$$

Its cofactor matrix is

$$\mathbf{C} = \begin{pmatrix} + \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & - \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & + \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \\ - \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & + \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & - \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} \\ + \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} & - \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & + \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{pmatrix}.$$

$\text{adj}(\mathbf{A}) = \mathbf{C}^T$   
 $\det(\mathbf{A}) = \sum_{j=1}^k a_{ij} C_{ij} = (-1)^{i+j} a_{ij} M_{ij}$   
 $\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \text{adj}(\mathbf{A})$

## Systems of Linear Equations

Consider a system of  $m$  linear equations with  $d$  variables or unknowns  $w_1, \dots, w_d$ :

$$\begin{aligned} x_{1,1}w_1 + x_{1,2}w_2 + \cdots + x_{1,d}w_d &= y_1 \\ x_{2,1}w_1 + x_{2,2}w_2 + \cdots + x_{2,d}w_d &= y_2 \\ &\vdots \\ x_{m,1}w_1 + x_{m,2}w_2 + \cdots + x_{m,d}w_d &= y_m. \end{aligned}$$

- $m$  is number of sample
- $d$  is number of feature

These equations can be written compactly in matrix-vector notation:

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

Where

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,d} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}.$$

### Note:

- The data matrix  $\mathbf{X} \in \mathcal{R}^{m \times d}$  and the target vector  $\mathbf{y} \in \mathcal{R}^m$  are given
- The unknown vector of parameters  $\mathbf{w} \in \mathcal{R}^d$  is to be learnt
- For number of feature  $d$ , is  $X$ , and number of sample  $m$
- $w$  is the classifier to map  $R^d \rightarrow R$ , which is  $y$

## Types of System

A set of linear equations can have no solution, one solution, or multiple solutions:

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

Where

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,d} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}.$$

$\mathbf{X}$ is Square	Even-determined	$m = d$	Equal number of equations and unknowns
$\mathbf{X}$ is Tall	Over-determined	$m > d$	More number of equations than unknowns
$\mathbf{X}$ is Wide	Under-determined	$m < d$	Fewer number of equations than unknowns

$\mathbf{X}$ is Square	Even-determined	$m = d$	One unique solution in general	$\hat{\mathbf{w}} = \mathbf{X}^{-1}\mathbf{y}$
$\mathbf{X}$ is Tall	Over-determined	$m > d$	No exact solution in general; An approximated solution	$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ Left-inverse
$\mathbf{X}$ is Wide	Under-determined	$m < d$	Infinite number of solutions in general; Unique constrained solution	$\hat{\mathbf{w}} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y}$ Right-inverse

```

28 X = np.array([[1, 2], [0, 6], [1, 0]])
29 leftinverseX = helper.leftInverse(X)
30 print("leftinverseX =\n", leftinverseX)
31 rightinverseX = helper.rightInverse(X)
32 print("rightinverseX =\n", rightinverseX)
33
34 Xpadded = helper.paddingOfOnes(X)
35 leftinverseX = helper.leftInverse(Xpadded)
36 print("leftinverseX =\n", leftinverseX)
37 rightinverseX = helper.rightInverse(Xpadded)
38 print("rightinverseX =\n", rightinverseX)
39
40 print("inv(Xpadded) =\n", inv(Xpadded))
•

```

```

leftinverseX =
[[ 0.47368421 -0.15789474  0.52631579]
 [ 0.02631579  0.15789474 -0.02631579]]
rightinverseX =
[[ 0.          0.          1.        ]
 [-0.25       0.1875     0.25      ]]
leftinverseX =
[[[-3.   1.   3.  ]
 [ 3.  -1.  -2.  ]
 [ 0.5  0.   -0.5 ]]]
rightinverseX =
[[[-3.000000e+00  1.000000e+00  3.000000e+00]
 [ 3.000000e+00 -1.000000e+00 -2.000000e+00]
 [ 5.000000e-01  4.4408921e-16 -5.000000e-01]]]
inv(Xpadded) =
[[[-3.   1.   3.  ]
 [ 3.  -1.  -2.  ]
 [ 0.5  0.   -0.5 ]]]

```

- If square and invertible, left and right inverse give the same thing
- Left inverse for over determined is actually the least square approximation
  - There is no solution in column space of  $\mathbf{X}$ , hence find the closest one in  $\text{Col}(\mathbf{X})$

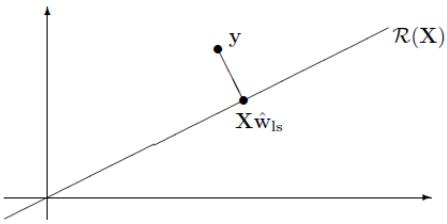


Figure 4.1: The least squares problem consists in finding  $\hat{\mathbf{w}}_{ls}$ , the point such that  $\mathbf{X}\hat{\mathbf{w}}_{ls} \in \mathcal{R}(\mathbf{X})$  is closest to a given point  $\mathbf{y}$ .

- This means  $\mathbf{X}^T\mathbf{X}$  has to have non zero determinant OR  $\mathbf{X}$  is full rank
- Right inverse for under determined is actually the least norm solution
  - There is infinitely many solution, hence find the one closest to origin, having the least norm

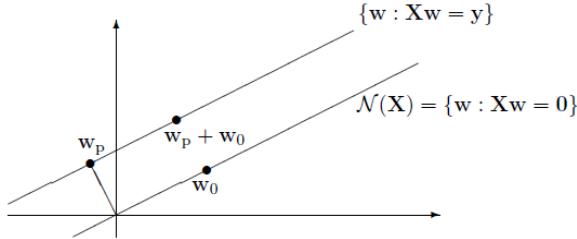


Figure 4.2: The least norm problem consists of finding the particular solution  $\hat{\mathbf{w}}_p$  that minimizes the norm.

- All other solutions  $\hat{\mathbf{w}}_p + \mathbf{w}_0$  where  $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$  have norms that are at least as large as that of  $\hat{\mathbf{w}}_p$ .

- This means  $\mathbf{X}\mathbf{X}^T$  has to have non zero determinant OR  $\mathbf{X}$  is full rank

The matrix  $\mathbf{X} \in \mathbb{R}^{m \times d}$  and vector  $\mathbf{y} \in \mathbb{R}^m$  are given and  $\mathbf{w} \in \mathbb{R}^d$  is to be found. As mentioned, if the matrix  $\mathbf{X}$  is square and full rank,  $\mathbf{X}^{-1}$  exists and so we can solve for  $\mathbf{w}$  by simple matrix inversion and multiplication  $\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$ . However, most of the time in engineering,  $m \neq d$  and more care is needed to discuss the existence and uniqueness of solutions to the linear system in (4.6). For this, we appeal to the Rouché-Capelli Theorem. We need the notion of the *augmented matrix*

$$\tilde{\mathbf{X}} = [\mathbf{X} \quad \mathbf{y}] \in \mathbb{R}^{m \times (d+1)}. \quad (4.8)$$

Note that the augmented matrix  $\tilde{\mathbf{X}}$  has rank at least as large as that of  $\mathbf{X}$ , i.e.,  $\text{rank}(\mathbf{X}) \leq \text{rank}(\tilde{\mathbf{X}})$ . This is because  $\tilde{\mathbf{X}}$  has more columns than  $\mathbf{X}$  so the dimension of its column space must be as large as that of  $\mathbf{X}$ .

**Theorem 4.1** (Rouché-Capelli Theorem). *For the linear system in (4.6), the following hold:*

(i) *The system in (4.6) admits a unique solution if and only if*

$$\text{rank}(\mathbf{X}) = \text{rank}(\tilde{\mathbf{X}}) = d; \quad (4.9)$$

(ii) *The system in (4.6) has no solution if and only if*

$$\text{rank}(\mathbf{X}) < \text{rank}(\tilde{\mathbf{X}}); \quad (4.10)$$

(iii) *The system in (4.6) has infinitely many solutions if and only if*

$$\text{rank}(\mathbf{X}) = \text{rank}(\tilde{\mathbf{X}}) < d. \quad (4.11)$$

*Proof sketch (Only the  $\Leftarrow$  directions).* For part (i), we note that the condition that  $\text{rank}(\mathbf{X}) = \text{rank}(\tilde{\mathbf{X}})$  means that  $\mathbf{y}$  is in the column space of  $\mathbf{X}$ . This means that there exists  $\{w_i\}_{i=1}^d \subset \mathbb{R}$  such that  $\sum_{i=1}^d w_i \underline{x}_i = \mathbf{y}$  where the  $\underline{x}_i$ 's are the  $d$  columns of  $\mathbf{X}$ . Since  $\text{rank}(\mathbf{X}) = d$ ,  $\{\underline{x}_i\}_{i=1}^d$  span  $\mathbb{R}^d$  and the representation  $\sum_{i=1}^d w_i \underline{x}_i = \mathbf{y}$  is unique (see discussion after Definition 4.3), which means there is a unique solution.

For part (ii), the condition that  $\text{rank}(\mathbf{X}) < \text{rank}(\tilde{\mathbf{X}})$  means that  $\mathbf{y}$  is not in the column space of  $\mathbf{X}$  so there is no solution.

For part (iii), since  $\text{rank}(\mathbf{X}) < d$ ,  $\{\underline{x}_i\}_{i=1}^d$  do not span  $\mathbb{R}^d$  and the dimension of the nullspace of  $\mathbf{X}$  is non-zero. This means that if  $\hat{\mathbf{w}}_p$  is a particular solution so is  $\hat{\mathbf{w}}_p + \mathbf{w}_0$  where  $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$ . Hence, are infinitely many solutions.  $\square$

### Square or even-determined system

## 1. Square or even-determined system: $m = d$

- Equal number of equations and unknowns, i.e.,  $\mathbf{X} \in \mathcal{R}^{d \times d}$
- **One unique solution** if  $\mathbf{X}$  is invertible or all rows/columns of  $\mathbf{X}$  are linearly independent
- If all rows or columns of  $\mathbf{X}$  are linearly independent, then  $\mathbf{X}$  is invertible.

Solution:

If  $\mathbf{X}$  is invertible (or  $\mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$ ), then pre-multiply both sides by  $\mathbf{X}^{-1}$

$$\begin{aligned} \mathbf{X}^{-1}\mathbf{X}\mathbf{w} &= \mathbf{X}^{-1}\mathbf{y} \\ \Rightarrow \quad \hat{\mathbf{w}} &= \mathbf{X}^{-1}\mathbf{y} \end{aligned}$$

(Note: we use a *hat* on top of  $\mathbf{w}$  to indicate that it is a specific point in the space of  $\mathbf{w}$ )

- Number of sample,  $m =$  number of features,  $d$

## Example 1

$$\begin{aligned} w_1 + w_2 &= 4 \\ w_1 - 2w_2 &= 1 \end{aligned}$$

(1)

(2)

Two unknowns

Two equations

$$\mathbf{X} \quad \mathbf{w} \quad \mathbf{y}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$\hat{\mathbf{w}} = \mathbf{X}^{-1} \mathbf{y}$$

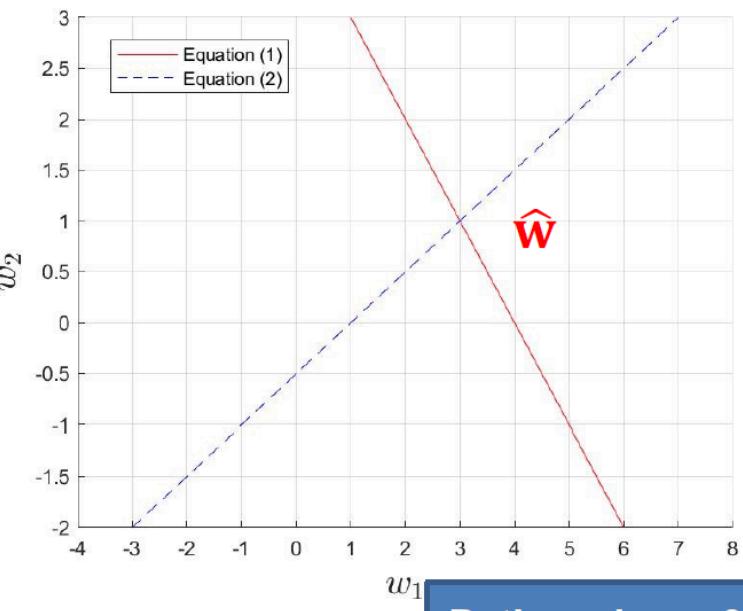
$$= \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix}^{-1} \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$= \frac{-1}{3} \begin{bmatrix} -2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$\mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \text{adj}(\mathbf{A})$$

$$\text{adj}(\mathbf{A}) = \mathbf{C}^T = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\det(\mathbf{A}) = ad - bc$$



Python demo 3

## 2. Over-determined system: $m > d$

- More equations than unknowns
- $X$  is non-square (tall) and hence not invertible
- Has no exact solution in general \*
- An approximated solution is available using the left inverse

If the **left-inverse** of  $X$  exists such that  $X^\dagger X = I$ , then pre-multiply both sides by  $X^\dagger$  results in

$$\begin{aligned} X^\dagger X w &= X^\dagger y \\ \Rightarrow \hat{w} &= X^\dagger y \end{aligned}$$

### Definition:

A matrix  $B$  that satisfies  $B_{d \times m} A_{m \times d} = I$  is called a **left-inverse** of  $A$ .

The **left-inverse** of  $X$ :  $X^\dagger = (X^T X)^{-1} X^T$  given  $X^T X$  is invertible.

Note: \* exception: when  $\text{rank}(X) = \text{rank}([X, y])$ , there is a solution.

- Number of sample,  $m >$  number of features,  $d$
- No exact solution in general, only approximation (least square solution)

Let  $V \subseteq \mathbb{R}^n$  and  $S = \{u_1, u_2, \dots, u_k\}$  be a basis for  $V$ . Then the orthogonal projection of any vector  $w \in \mathbb{R}^n$  onto  $V = A(A^T A)^{-1} A^T w$ , where  $A = (u_1, u_2, \dots, u_k) \quad A_{n \times k} \quad n \geq k$

Since  $S$  is basis, columns of  $A$  are linearly independent ( $\text{rank}(A) = k$ ,  $A$  is full ranked) so  $A^T A$  is invertible

$\therefore \hat{u} = (A^T A)^{-1} A^T w$  is the unique solution to  $A^T A \hat{u} = A^T w$

$\circ$   $\therefore \text{proj}_V w = A \hat{u} = A (A^T A)^{-1} A^T w$

$\circ$  Except for  $\text{rank}(X) = \text{rank}([X, y])$  which just means  $y$  is in the column space of  $X$ , hence there is an exact solution

- Lots of noisy measurement/sample to find something

<u>Case 1</u>	<u>Full Rank Matrices</u>	no of non-zero rows = $R = n$
Rank = no of col	let $A$ be $m \times n$ , $m \geq n$ . Then $A$ has full rank, $\text{rank}(A) = \text{column} = n$ , iff RREF( $A$ ) has the form:	
	$R = \begin{pmatrix} I_n \\ 0_{(m-n) \times n} \end{pmatrix}$	$\leftarrow n$ first columns, $n$ non-zero rows, $m-n$ zero rows
1) $\text{rank}(A) = n$ .		$\rightarrow R$ has $n$ non-zero rows with $n$ entries $\Rightarrow$ spans $\mathbb{R}^n$
2) The rows of $A$ spans $\mathbb{R}^n$ , $\text{Row}(A) = \mathbb{R}^n$ .		All first columns
3) The columns of $A$ are linearly independent.		$\rightarrow A\vec{x} = \vec{0}$
4) Homogeneous system $A\vec{x} = \vec{0}$ has only the trivial solution, $\text{Null}(A) = \{\vec{0}\}$ .		$\leftarrow (A^T A)^{-1} A^T \vec{A} = ((A^T A)^{-1} A^T) \vec{0} = \vec{0}$
5) $A^T A$ is an invertible matrix of order $n$ .		$\rightarrow I_n = \vec{0}$
6) $A$ has a left inverse. $= (A^T A)^{-1} A^T$		$\rightarrow \vec{A} = \vec{0}$
⑤ $\rightarrow$ ⑥: $I = (A^T A)^{-1} A^T A = ((A^T A)^{-1} A^T) A \Rightarrow$ left inverse of $A = (A^T A)^{-1} A^T \neq$		
⑥ $\rightarrow$ ⑤: Suppose $C$ is a left inverse of $A$ , then if $\vec{u} \in \mathbb{R}^n$ such that $A\vec{u} = \vec{0}$ ,		
$\vec{u} = I\vec{u} = C A \vec{u} = C \vec{0} = \vec{0} \Rightarrow$ nullspace of $A$ is trivial		
since $\text{null}(A^T A) = \text{null}(A)$		$\Rightarrow$ nullspace of $A^T A$ is trivial $\Rightarrow A^T A$ is invertible

- to many samples for little features, cannot find inverse
  - $w$  cannot fit the constraint needed
  - so need to be an approximation

## Example 2

$$w_1 + w_2 = 1 \quad (1)$$

$$w_1 - w_2 = 0 \quad (2)$$

$$w_1 = 2 \quad (3)$$

$$\begin{matrix} \mathbf{X} & \mathbf{w} & \mathbf{y} \end{matrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

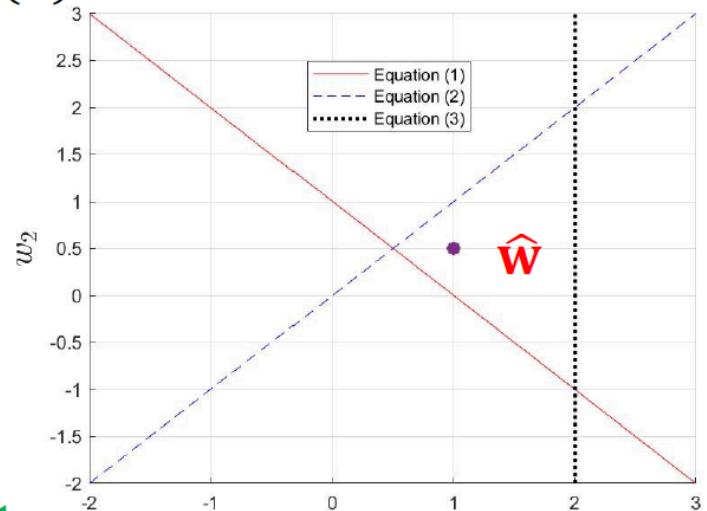
No exact solution

Approximated solution

$$\hat{\mathbf{w}} = \mathbf{X}^\dagger \mathbf{y} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

$$= \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$$

$\mathbf{X}^T \mathbf{X}$  is invertible



Python demo 4

- Consider the following over-determined system in which  $m = 3$  and  $d = 2$ :

$$\mathbf{X} = \begin{bmatrix} 2 & 1 \\ 4 & 3 \\ 5 & 6 \end{bmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}. \quad (4.12)$$

The augmented matrix is

$$\tilde{\mathbf{X}} = [\mathbf{X} \quad \mathbf{y}] = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 2 \\ 5 & 6 & 3 \end{bmatrix}. \quad (4.13)$$

In this case  $\text{rank}(\mathbf{X}) = 2$  and  $\text{rank}(\tilde{\mathbf{X}}) = 3$ . This is case (ii) of the Rouché-Capelli Theorem and there is no solution. This is the usual case for over-determined systems. Note that in Python, you can find the rank of a matrix (2D array)  $\mathbf{A}$  using `np.linalg.matrix_rank(A)`.

Consider the following over-determined system in which  $m = 3$  and  $d = 2$ :

$$\mathbf{X} = \begin{bmatrix} 2 & 1 \\ 4 & 3 \\ 5 & 6 \end{bmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 4 \\ 10 \\ 17 \end{bmatrix}. \quad (4.14)$$

In this case  $\text{rank}(\mathbf{X}) = 2$  and  $\text{rank}(\tilde{\mathbf{X}}) = 2$ . This is case (i) of the Rouché-Capelli Theorem and there is a unique solution even though the system is over-determined. Note that  $\mathbf{y}$  is one times the first column of  $\mathbf{X}$  plus two times the second column of  $\mathbf{X}$ , so it is in the linear span of the columns of  $\mathbf{X}$ .

Consider the following over-determined system in which  $m = 3$  and  $d = 2$ :

$$\mathbf{X} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \\ 6 & 3 \end{bmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 8 \\ 16 \\ 24 \end{bmatrix}. \quad (4.15)$$

In this case  $\text{rank}(\mathbf{X}) = 1$  and  $\text{rank}(\tilde{\mathbf{X}}) = 1$  and both these ranks are less than  $d = 2$ . This is case (iii) of the Rouché-Capelli Theorem and there are infinitely many solutions even though the system is over-determined. Note that the three columns of  $\tilde{\mathbf{X}}$  are collinear.

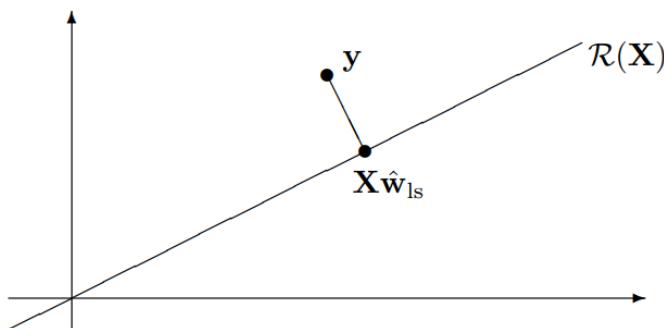


Figure 4.1: The least squares problem consists in finding  $\hat{\mathbf{w}}_{ls}$ , the point such that  $\mathbf{X}\hat{\mathbf{w}}_{ls} \in \mathcal{R}(\mathbf{X})$  is closest to a given point  $\mathbf{y}$ .

### 3. Under-determined system: $m < d$

- More unknowns than equations
- Infinite number of solutions in general \*

If the **right-inverse** of  $X$  exists such that  $XX^\dagger = I$ , then the  $d$ -vector  $w = X^\dagger y$  (one of the infinite cases) satisfies the equation  $Xw = y$ , i.e.,

$$\begin{aligned} Xw &= y \quad \Rightarrow \quad XX^\dagger y = y \\ &\quad \Rightarrow \quad Iy = y \end{aligned}$$

#### Definition:

A matrix  $B$  that satisfies  $A_{m \times d} B_{d \times m} = I$  is called a **right-inverse** of  $A$ . The **right-inverse** of  $X$ :  $X^\dagger = X^T (XX^T)^{-1}$  given  $XX^T$  is invertible.

If  $X$  is right-invertible, we can find a unique constrained solution.

Note: \* exception: no solution if the system is inconsistent  $\text{rank}(X) < \text{rank}([X, y])$

- Number of sample,  $m <$  number of features,  $d$ 
  - Few patients but uses high resolution MRI
- Infinite number of solution in general
  - But can find unique constraint solution

A unique solution is yet possible by constraining the search using  
 $w = X^T a$

If  $XX^T$  is invertible, let  $w = X^T a$ , then

$$\begin{aligned} X^T a &= y \\ \Rightarrow \hat{a} &= (XX^T)^{-1} y \\ \Rightarrow \hat{w} &= X^T \hat{a} = X^T \underbrace{(XX^T)^{-1} y}_{X^\dagger} \end{aligned}$$

**right-inverse**

- By constraint  $w$  to be the form of  $X^T a$ , then from infinite solution to unique solution
  - This is finding the least norm solution
- Or exception when  $\text{rank}(X) < \text{rank}([X, y])$ , meaning  $y$  is linearly independent of the column space of  $X$ , so confirm no solution

### Case 2

Rank = no of rows Let  $A$  be  $m \times n$ ,  $n \geq m$ . Then  $A$  has a full rank,  $\text{rank}(A) = \text{no of rows} = m$ , iff  $R = \text{RREF}$  has the form:

$$R = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \cdots & 0 \end{pmatrix} \leftarrow m \text{ non-zero rows, } m \text{ pivot column, } n-m \text{ non-pivot column} \quad \text{full Rank} = 3$$

\* columns before the 1st pivot column can be zero column also  $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

1)  $\text{rank}(A) = m$ .

$\Rightarrow R$  has  $m$  pivot column with  $m$  entries  $\Rightarrow$  spans  $\mathbb{R}^m$

2) The columns of  $A$  spans  $\mathbb{R}^m$ ,  $\text{Col}(A) = \mathbb{R}^m$

All non-zero

3) The rows of  $A$  are linearly independent.

4) The linear system  $Ax = b$  is consistent for every  $b \in \mathbb{R}^m$ .

5)  $AA^T$  is an invertible matrix of order  $m$ .

6)  $A$  has a right inverse.  $= A^T(AA^T)^{-1}$

③  $\rightarrow$  ④:  $AA^T(AA^T)^{-1} = I \Rightarrow A^T(AA^T)^{-1}$  is the right inverse of  $A$ . For any  $b$ ,  $A(A^T(AA^T)^{-1})^T b = b$ ,  $1_A = A^T(AA^T)^{-1} b$   $\therefore$  consistent &

④  $\rightarrow$  ③: Suppose  $Ax = b$  is consistent for every  $b \in \mathbb{R}^m$ . Let  $b_i$  be a soln to  $Ax = e_i$ ,  $e_i$  =  $i$ -th vector in the standard basis, for  $i=1, \dots, m$ . Let  $B = (b_1 \ b_2 \ \dots \ b_m)$ , then  $AB = A(b_1 \ b_2 \ \dots \ b_m) = e_1 \ e_2 \ \dots \ e_m = I_m \Rightarrow B$  is a right inverse of  $A$  #

To satisfy both case 1 & 2,  $A$  must be a square matrix with  $R = I$ .  $\rightarrow$  RREF (full rank square matrix) =  $I$

Case 1 & 2 are mutually exclusive for non-square matrix  $\Rightarrow$  can have at most a left or a right inverse not both.

↳ if  $r > c$ , then can't have right inverse; if  $A$  has right inverse, then  $\text{rank}(A) = r$  but  $r > c \therefore \text{rank}(A) \neq r$

- Too many features, too little samples
  - o too many possible solution that is possible and most are not good

**Example 3**  $w_1 + 2w_2 + 3w_3 = 2 \quad (1)$

$$w_1 - 2w_2 + 3w_3 = 1 \quad (2)$$

Three unknowns  
Two equations

**X**      **w**      **y**

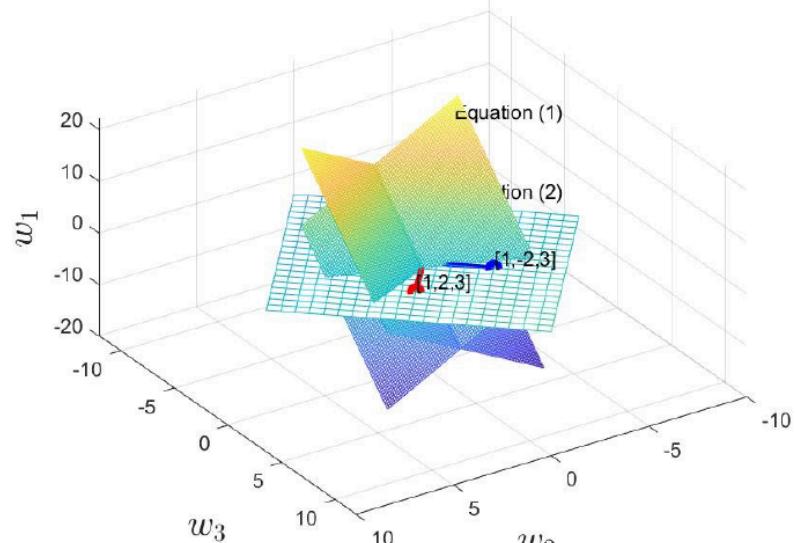
$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & -2 & 3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Infinitely many solutions along the intersection line

Here  $\mathbf{X}\mathbf{X}^T$  is invertible

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

$$= \begin{bmatrix} 1 & 1 \\ 2 & -2 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} 14 & 6 \\ 6 & 14 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.25 \\ 0.45 \end{bmatrix}$$



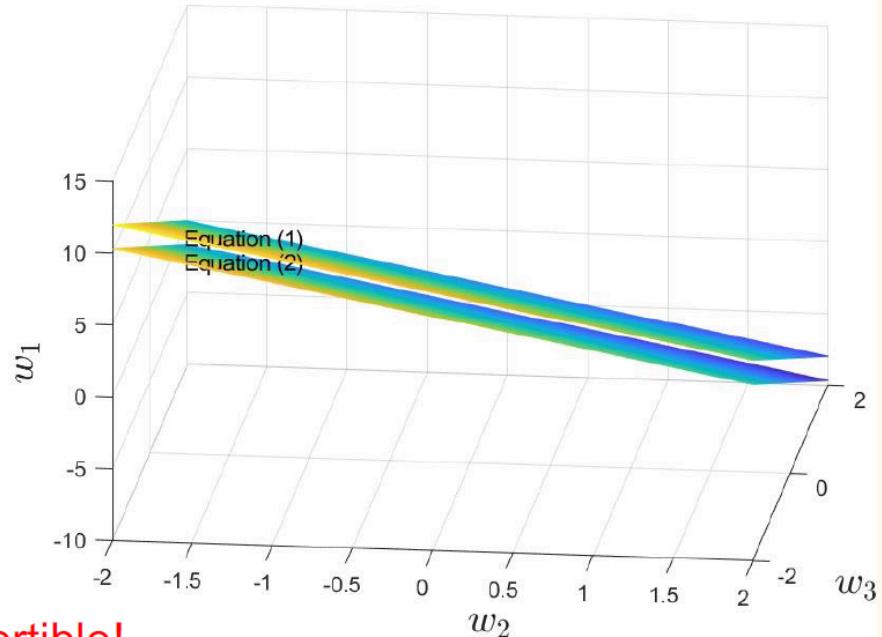
Constrained solution

- The two original planes, intersect in a line, infinite solution
- Putting in the constraint, introduce one more plane and hence only one point of intersection

**Example 4**  $w_1 + 2w_2 + 3w_3 = 2 \quad (1)$       Three unknowns  
 $3w_1 + 6w_2 + 9w_3 = 1 \quad (2)$       Two equations

$$\mathbf{X} \quad \mathbf{w} \quad \mathbf{y}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$



Both  $\mathbf{XX}^T$  and  $\mathbf{X}^T\mathbf{X}$  are not invertible!

There is no solution for the system

Let  $A$  be  $m \times n$ ,  $n \geq m$ . Then  $A$  has a full rank,  $\text{rank}(A) = \text{no of rows} = m$ , iff  $A = \text{REF}$  has the form:

$$A = \begin{pmatrix} 0 & 1 & \dots & 0 & \dots & 0 & \dots & 0 & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & \dots \\ 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & \dots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \end{pmatrix} \leftarrow m \text{ non-zero rows, } m \text{ pivot column, } n-m \text{ non-pivot column}$$

\* columns before the 1st pivot column can be zero column also

- $\mathbf{X}$  is not full rank so cannot find right inverse

Consider the following under-determined system in which  $m = 2$  and  $d = 3$ :

$$\mathbf{X} = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 2 & 5 \end{bmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 10 \\ 7 \end{bmatrix}. \quad (4.16)$$

In this case,  $\text{rank}(\mathbf{X}) = 2$  and  $\text{rank}(\tilde{\mathbf{X}}) = 2$  but  $d = 3$ . This is case (iii) of the Rouché-Capelli Theorem and there are infinitely many solutions. This is the usual case for under-determined systems.

Consider the following under-determined system in which  $m = 2$  and  $d = 3$ :

$$\mathbf{X} = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 2 & 6 \end{bmatrix}, \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}. \quad (4.17)$$

In this case,  $\text{rank}(\mathbf{X}) = 1$  and  $\text{rank}(\tilde{\mathbf{X}}) = 2$  because  $\mathbf{y} \notin \mathcal{R}(\mathbf{X})$ . This is case (ii) of the Rouché-Capelli Theorem and there is no solution. Note that  $\mathbf{y}$  boosts the rank of  $\mathbf{X}$  by 1 in the augmented matrix  $\tilde{\mathbf{X}}$ , i.e.,  $\mathbf{y}$  is not in the column space of  $\mathbf{X}$ , which is the ray  $\{[t, 2t]^\top : t \in \mathbb{R}\}$ .

### Least Norm Solution

## 4.4 Least Norm Solution for $m < d$

Now we consider the case in which  $\mathbf{X}$  is wide ( $m < d$ ) and full rank. This means that  $\text{rank}(\mathbf{X}) = m$ ; equivalently, all rows are linearly independent (full row rank). This *under-determined* situation also occurs a lot in engineering.

**Example 4.3.** For example, in control engineering (a field of study within mechanical and electrical engineering), one often considers the following discrete-time state-space system (e.g., describing the dynamics of a robot operating over a quantized time interval):

$$v_{i+1} = av_i + bw_i, \quad i = 0, 1, \dots, d-1, \quad (4.21)$$

where  $v_i$  is the state of the system at time  $i$  and  $w_i$  is our control. We assume the system starts at the origin  $v_0 = 0$ . We desire to design the  $w_i$ 's such that the terminal state  $v_d = y$  (for some given  $y$ ) while minimizing the cost of the control  $\sum_{i=0}^{d-1} w_i^2$ . After some algebra, this can be rewritten as

$$\text{minimize } \|\mathbf{w}\|^2 \quad \text{subject to} \quad y = [b \ ab \ a^2b \ \dots \ a^{d-1}b] \begin{bmatrix} w_{d-1} \\ w_{d-2} \\ \vdots \\ w_0 \end{bmatrix}. \quad (4.22)$$

This is exactly an under-determined problem if we make the identifications  $\mathbf{X} = [b \ ab \ a^2b \ \dots \ a^{d-1}b]$  (for obvious reasons, this is called the  $d$ -step reachability matrix) and  $\mathbf{w} = [w_{d-1} \ w_{d-2} \ \dots \ w_0]^\top$  and  $y$  is scalar (i.e.,  $m = 1$ ).

We return to the equation  $\mathbf{X}\mathbf{w} = \mathbf{y}$  in which  $m < d$  and the matrix  $\mathbf{X}$  has full row rank. It is clear that  $(\mathbf{X}\mathbf{X}^\top)^{-1}$  exists and

$$\hat{\mathbf{w}}_p = \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} \quad (4.23)$$

is a solution to the equation (check!). The subscript p indicates that this is a *particular* solution to the linear system. From the usual case of the Rouché-Capelli Theorem (case (iii)), we know that there are infinitely many solutions. Where are these infinitely many solutions? Let  $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$  be any vector in the nullspace of  $\mathbf{X}$ . Note that the nullspace has positive dimension because  $\text{rank}(\mathbf{X}) = m < d$  so  $\mathbf{w}_0$  can be chosen to be a non-zero vector. Then  $\hat{\mathbf{w}}_p + \mathbf{w}_0$  is also a solution to (4.6) (check!). In the following, we argue that among all the solutions,  $\hat{\mathbf{w}}_p$  has a special place in our hearts because it is the *least norm solution* to (4.6).

Suppose that  $\mathbf{w}$  is any solution to  $\mathbf{X}\mathbf{w} = \mathbf{y}$ . Then since  $\hat{\mathbf{w}}_p$  is also a solution, we have  $\mathbf{X}(\mathbf{w} - \hat{\mathbf{w}}_p) = \mathbf{0}$  and

$$(\mathbf{w} - \hat{\mathbf{w}}_p)^\top \hat{\mathbf{w}}_p \stackrel{(4.23)}{=} (\mathbf{w} - \hat{\mathbf{w}}_p)^\top \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} = (\mathbf{X}(\mathbf{w} - \hat{\mathbf{w}}_p))^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} = 0. \quad (4.24)$$

This means that  $\mathbf{w} - \hat{\mathbf{w}}_p$  is orthogonal to  $\hat{\mathbf{w}}_p$ . By the Pythagorean theorem,

$$\|\mathbf{w}\|^2 = \|(\mathbf{w} - \hat{\mathbf{w}}_p) + \hat{\mathbf{w}}_p\|^2 = \|\mathbf{w} - \hat{\mathbf{w}}_p\|^2 + \|\hat{\mathbf{w}}_p\|^2 \geq \|\hat{\mathbf{w}}_p\|^2, \quad (4.25)$$

which shows that  $\hat{\mathbf{w}}_p$  is the least norm solution to  $\mathbf{X}\mathbf{w} = \mathbf{y}$ , so we also denote this as

$$\hat{\mathbf{w}}_{ln} = \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} \quad (4.26)$$

See the geometry of the least norm problem in Fig. 4.2.

Finally, we say a few words about the matrix  $\mathbf{X}^\dagger = \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}$ . This matrix is called the *pseudo-inverse* of the full-rank wide matrix  $\mathbf{X}$ . It is also known as the *right-inverse* of  $\mathbf{X}$  (why?). The right-inverse of a matrix with full row rank can be implemented in Python as `np.linalg.pinv(X)`.

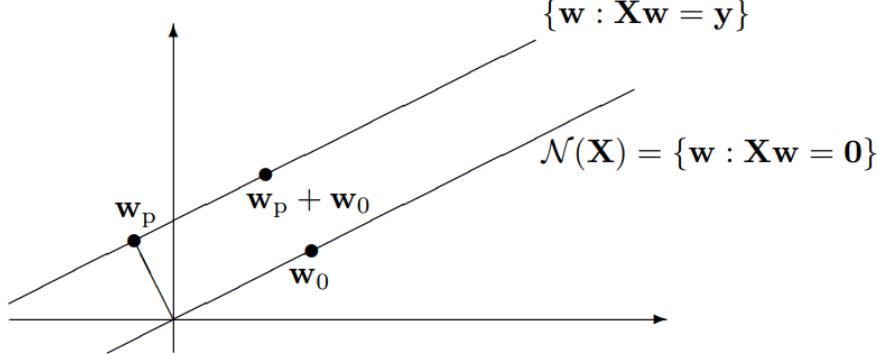


Figure 4.2: The least norm problem consists of finding the particular solution  $\hat{\mathbf{w}}_p$  that minimizes the norm. All other solutions  $\hat{\mathbf{w}}_p + \mathbf{w}_0$  where  $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$  have norms that are at least as large as that of  $\hat{\mathbf{w}}_p$ .

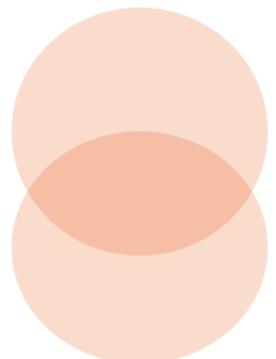
## Set

- A **set** is an **unordered** collection of unique elements
  - Denoted as a calligraphic capital character e.g.,  $\mathcal{S}, \mathcal{R}, \mathcal{N}$  etc
  - When an element  $x$  belongs to a set  $S$ , we write  $x \in S$
- A set of numbers can be **finite** - include a fixed amount of values
  - Denoted using accolades, e.g.  $\{1, 3, 18, 23, 235\}$  or  $\{x_1, x_2, x_3, x_4, \dots, x_d\}$
- A set can be **infinite** and include all values in some interval
  - If a set of real numbers includes all values between  $a$  and  $b$ , **including  $a$  and  $b$** , it is denoted using square brackets as  $[a, b]$
  - If the set **does not include the values  $a$  and  $b$** , it is denoted using parentheses as  $(a, b)$
- Examples:
  - The special set denoted by  $\mathcal{R}$  includes all real numbers from minus infinity to plus infinity
  - The set  $[0, 1]$  includes values like 0, 0.0001, 0.25, 0.9995, and 1.0

- **Intersection** of two sets:

$$\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cap \mathcal{S}_2$$

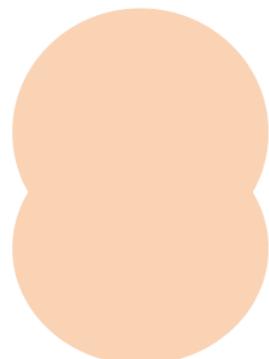
Example:  $\{1, 3, 5, 8\} \cap \{1, 8, 4\} = \{1, 8\}$



- **Union** of two sets:

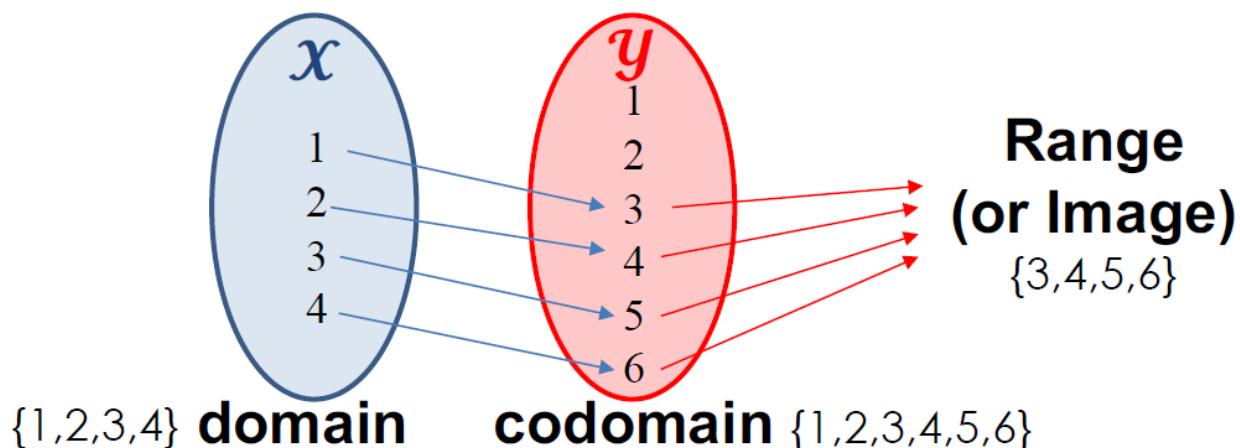
$$\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$$

Example:  $\{1, 3, 5, 8\} \cup \{1, 8, 4\} = \{1, 3, 4, 5, 8\}$



## Function

- A **function** is a relation that associates each element  $x$  of a **set  $X$** , the **domain** of the function, to a single element  $y$  of another **set  $Y$** , the **codomain** of the function
- If the function is called  $f$ , this relation is denoted  $y = f(x)$ 
  - The element  $x$  is the **argument** or **input** of the function
  - $y$  is the value of the function or the **output**
- The symbol used for representing the input is the **variable** of the function
  - $f(x)$   $f$  is a function of the variable  $x$ ;  $f(x, w)$   $f$  is a function of the variable  $x$  and  $w$



- A **scalar function** can have vector argument
  - E.g.  $y = f(\mathbf{x}) = x_1 + x_2 + 2x_3$
- A **vector function**, denoted as  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  is a function that returns a vector  $\mathbf{y}$ 
  - Input argument can be a **vector**  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  or a **scalar**  $y = f(x)$
  - E.g.  $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -x_1 \\ x_2 \end{bmatrix}$
  - E.g.  $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -2x_1 \\ 3x_1 \end{bmatrix}$

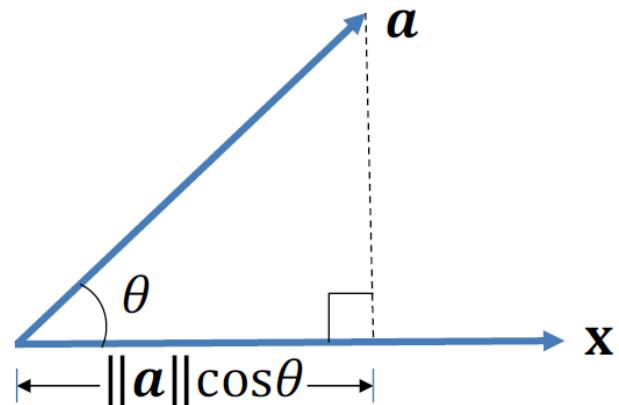
- The notation  $f: \mathcal{R}^d \rightarrow \mathcal{R}$  means that  $f$  is a function that maps **real**  $d$ -vectors to **real** numbers
  - i.e.,  $f$  is a scalar-valued function of  $d$ -vectors
- If  $\mathbf{x}$  is a  $d$ -vector argument, then  $f(\mathbf{x})$  denotes the value of the function  $f$  at  $\mathbf{x}$ 
  - i.e.,  $f(\mathbf{x}) = f(x_1, x_2, \dots, x_d)$ ,  $\mathbf{x} \in \mathcal{R}^d$ ,  $f(\mathbf{x}) \in \mathcal{R}$
- Example: we can define a function  $f: \mathcal{R}^4 \rightarrow \mathcal{R}$  by  

$$f(\mathbf{x}) = x_1 + x_2 - x_4^2$$

### The inner product function

- Suppose  $\mathbf{a}$  is a  $d$ -vector. We can define a scalar valued function  $f$  of  $d$ -vectors, given by  

$$f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} = a_1 x_1 + a_2 x_2 + \dots + a_d x_d \quad (1)$$
 for any  $d$ -vector  $\mathbf{x}$
- The inner product of its  $d$ -vector argument  $\mathbf{x}$  with some (fixed)  $d$ -vector  $\mathbf{a}$
- We can also think of  $f$  as forming a **weighted sum** of the elements of  $\mathbf{x}$ ; the elements of  $\mathbf{a}$  give the weights



- Weighted sum,  $\mathbf{a}$  can be called weight or coefficient
- $\mathbf{a}$  is to be found to map  $\mathbf{x}$  to a scalar

## Linearity and Superposition

A function  $f: \mathcal{R}^d \rightarrow \mathcal{R}$  is **linear** if it satisfies the following two properties:

- **Homogeneity**
  - For any  $d$ -vector  $\mathbf{x}$  and any scalar  $\alpha$ ,  $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$
  - **Scaling** the (vector) argument is the same as scaling the function value
- **Additivity**
  - For any  $d$ -vectors  $\mathbf{x}$  and  $\mathbf{y}$ ,  $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$
  - **Adding** (vector) arguments is the same as adding the function values

## **Superposition and linearity**

- The inner product function  $f(\mathbf{x}) = \mathbf{a}^T \mathbf{x}$  defined in equation (1) (slide 42) satisfies the property

$$\begin{aligned}f(\alpha\mathbf{x} + \beta\mathbf{y}) &= \mathbf{a}^T(\alpha\mathbf{x} + \beta\mathbf{y}) \\&= \mathbf{a}^T(\alpha\mathbf{x}) + \mathbf{a}^T(\beta\mathbf{y}) \\&= \alpha(\mathbf{a}^T\mathbf{x}) + \beta(\mathbf{a}^T\mathbf{y}) \\&= \alpha f(\mathbf{x}) + \beta f(\mathbf{y})\end{aligned}$$

for all  $d$ -vectors  $\mathbf{x}$ ,  $\mathbf{y}$ , and all scalars  $\alpha$ ,  $\beta$ .

- This property is called **superposition**, which consists of **homogeneity** and **additivity**
- A function that satisfies the superposition property is called **linear**
- If a function  $f$  is **linear**, superposition extends to linear combinations of any number of vectors:

$$f(\alpha_1\mathbf{x}_1 + \cdots + \alpha_k\mathbf{x}_k) = \alpha_1 f(\mathbf{x}_1) + \cdots + \alpha_k f(\mathbf{x}_k)$$

for any  $d$  vectors  $\mathbf{x}_1 + \cdots + \mathbf{x}_k$ , and any scalars  $\alpha_1 + \cdots + \alpha_k$ .

## Affine Function

A linear function plus a constant is called an affine function

A linear function  $f: \mathcal{R}^d \rightarrow \mathcal{R}$  is **affine** if and only if it can be expressed as  $f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} + b$  for some  $d$ -vector  $\mathbf{a}$  and scalar  $b$ , which is called the **offset (or bias)**

### Example:

$$f(\mathbf{x}) = 2.3 - 2x_1 + 1.3x_2 - x_3$$

This function is affine, with  $b = 2.3$ ,  $\mathbf{a}^T = [-2, 1.3, -1]$ .

- Linear function plus DC offset constant get non linear function, get affine function

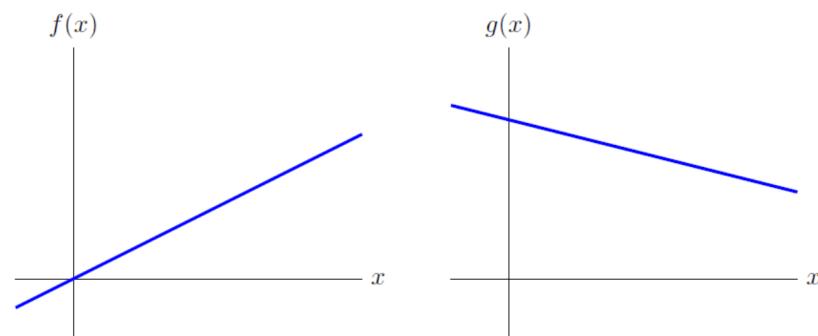


Figure 2.1 Left. The function  $f$  is linear. Right. The function  $g$  is affine, but not linear.

## Class Quiz

100 elderly participated in a research study on brain imaging and cognitive decline in ageing. Each subject has 10 brain measures. If we would like to use these measures to predict cognition, this is an over-determined system.

Yes, overdetermined as no of sample > no of feature

If each of the 100 older adults has 100 brain and body measures, this is an even-determined system. We can usually find a unique solution.

Yes, even determine as no of sample = no of feature

If only a subset of 40 older adults have completed 100 brain and body measures, this is an under-determined system, i.e. more equations than unknowns. To predict cognition, we need to use right inverse.

No, under determined as no of sample < no of feature, find right inverse, but it's less equations than unknown

## Summary

# Linear Functions

A function  $f: \mathcal{R}^d \rightarrow \mathcal{R}$  is **linear** if it satisfies the following **two properties**:

- **Homogeneity**  $f(\alpha \mathbf{x}) = \alpha f(\mathbf{x})$  **Scaling**
- **Additivity**  $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$  **Adding**

## Inner product function

$$f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} = a_1 x_1 + a_2 x_2 + \dots + a_d x_d$$

## Affine function

$$f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} + b \quad \text{scalar } b \text{ is called the offset (or bias)}$$

- Operations on Vectors and Matrices
  - Dot-product, matrix inverse
- Systems of Linear Equations  $\mathbf{Xw} = \mathbf{y}$ 
  - Matrix-vector notation, linear dependency, invertible
  - Even-, over-, under-determined linear systems
- Set and Functions

Assignment 1 (week 7 Wed)  
Tutorial 4

X is Square	Even-determined	$m = d$	One unique solution in general	$\hat{\mathbf{w}} = \mathbf{X}^{-1} \mathbf{y}$
X is Tall	Over-determined	$m > d$	No exact solution in general; An approximated solution	$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ Left-inverse
X is Wide	Under-determined	$m < d$	Infinite number of solutions in general; Unique constrained solution	$\hat{\mathbf{w}} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{y}$ Right-inverse

- Scalar and vector functions
- Inner product function
- Linear and affine functions

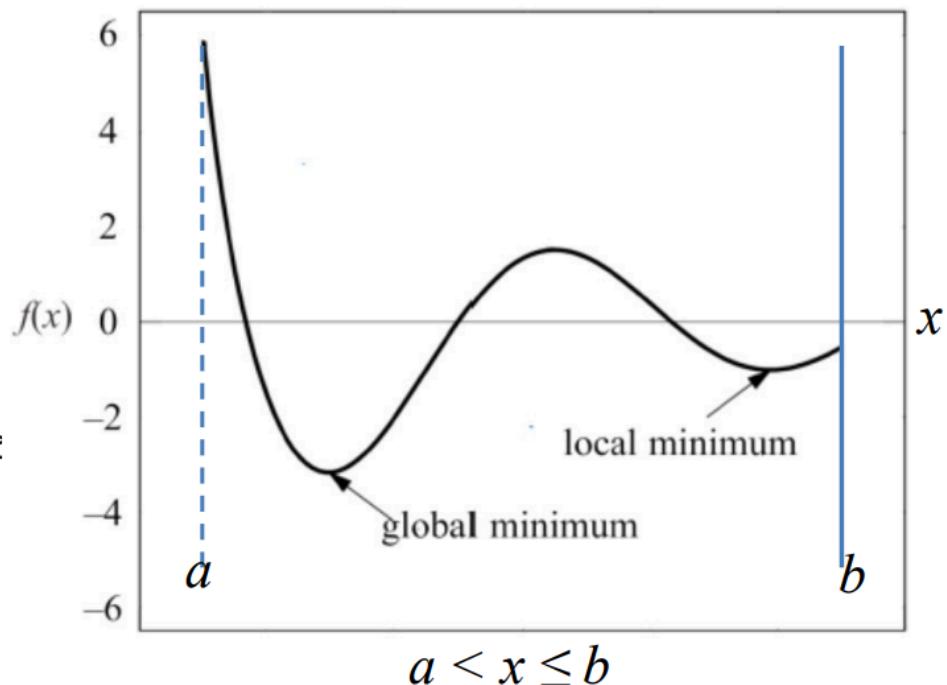
python package *numpy*  
Inverse: *numpy.linalg.inv(X)*  
Transpose: *X.T*

## Lec5: Least Squares and Linear Regression

### Functions: Maximum and Minimum

- $f(x)$  has a **local minimum** at  $x = c$  if  $f(x) \geq f(c)$  for every  $x$  in some open interval around  $x = c$
- $f(x)$  has a **global minimum** at  $x = c$  if  $f(x) \geq f(c)$  for all  $x$  in the domain of  $f$

A local and a global minima of a function



Note: An **interval** is a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set.

An **open interval** does not include its endpoints and is denoted using parentheses. E.g.  $(0, 1)$  means “all numbers greater than 0 and less than 1”.

- optimisation problem

## Max and Arg Max

- Given a set of values  $\mathcal{A} = \{a_1, a_2, \dots, a_m\}$ ,
- The operator  $\max_{a \in \mathcal{A}} f(a)$  returns the highest value  $f(a)$  for all elements in the set  $\mathcal{A}$
- The operator  $\arg \max_{a \in \mathcal{A}} f(a)$  returns the element of the set  $\mathcal{A}$  that maximizes  $f(a)$
- When the set is **implicit** or **infinite**, we can write

$$\max_a f(a) \quad \text{or} \quad \arg \max_a f(a)$$

E.g.  $f(a) = 3a$ ,  $a \in [0,1]$   $\rightarrow \max_a f(a) = 3$  and  $\arg \max_a f(a) = 1$

## Min and Arg Min operate in a similar manner

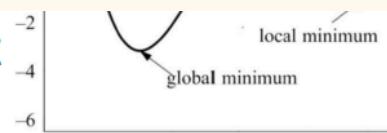
Note: **arg max** returns a value from the **domain** of the function and **max** returns from the **range (codomain)** of the function.

- $f(a)$  is what we want to optimise

## Derivative and Gradient

Optimise in terms of cost functions, so need to find the maximum or minimum of the cost functions

- The **derivative**  $f'$  of a function  $f$  is a function that describes how fast  $f$  grows (or decreases)



- If the derivative is a **constant** value, e.g. 5 or -3
  - The function  $f$  grows (or decreases) constantly at any point  $x$  of its domain
- When the derivative  $f'$  is a function
  - If  $f'$  is **positive** at some  $x$ , then the function  $f$  grows at this point
  - If  $f'$  is **negative** at some  $x$ , then the function  $f$  **decreases** at this point
  - The derivative of zero at  $x$  means that the function's **slope** at  $x$  is **horizontal** (e.g. **maximum** or **minimum** points)
- The process of finding a derivative is called **differentiation**.
- Gradient** is the generalization of derivative for functions that take several inputs (or one input in the form of a vector or some other complex structure).

The gradient of a function is a vector of **partial derivatives**

## Differentiation of a **scalar** function w.r.t. a **vector**

If  $f(\mathbf{x})$  is a **scalar** function of  $d$  variables,  $\mathbf{x}$  is a  $d \times 1$  vector.

Then differentiation of  $f(\mathbf{x})$  w.r.t.  $\mathbf{x}$  results in a  $d \times 1$  vector

$$\frac{df(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$$

$T : \mathbb{R}^3 \rightarrow \mathbb{R}^1$   
 $(x, y, z)$   
 $3 \times 1$

This is referred to as the **gradient** of  $f(\mathbf{x})$  and often written as  $\nabla_{\mathbf{x}}f$ .

E.g.  $f(\mathbf{x}) = ax_1 + bx_2 \quad \nabla_{\mathbf{x}}f = \begin{bmatrix} a \\ b \end{bmatrix}$

Ref: Duda, Hart, and Stork, "Pattern Classification", 2001 (Appendix)

$$\nabla = \left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right)$$

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

- $f$  takes in a vector and returns a scalar
- differentiating  $f$  with respect to the vector it takes in is nothing but the gradient of  $f = \nabla f$
- Imagine  $V(x, y, z)$ , then  $\nabla V = -\mathbf{E}$
- Partial derivative of  $f$  with respect to all the variable in  $\mathbf{x}$

## Differentiation of a vector function w.r.t. a vector

If  $\mathbf{f}(\mathbf{x})$  is a vector function of size  $h \times 1$  and  $\mathbf{x}$  is a  $d \times 1$  vector.  
Then differentiation of  $\mathbf{f}(\mathbf{x})$  results in a  $h \times d$  matrix

$$\frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_h}{\partial x_1} & \dots & \frac{\partial f_h}{\partial x_d} \end{bmatrix}$$

The matrix is referred to as the **Jacobian** of  $\mathbf{f}(\mathbf{x})$

- $\mathbf{f}$  is vector valued function of vector
- equivalent to the matrix multiplication of  $\mathbf{f}$  as a column vector ( $h \times 1$ ) and  $\nabla$  as a row vector ( $1 \times d$ )
  - first row is the partial derivative of the first scalar function in  $\mathbf{f}$

### Properties

#### Derivative and Gradient

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

#### Some Vector-Matrix Differentiation Formulae

$$\frac{d(\mathbf{Ax})}{d\mathbf{x}} = \mathbf{A} \quad \begin{matrix} 2 \times 1 \\ 3 \times 1 \end{matrix} \rightarrow \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \end{bmatrix}$$

$$\frac{d(\mathbf{b}^T \mathbf{x})}{d\mathbf{x}} = \mathbf{b} \quad \frac{d(\mathbf{y}^T \mathbf{Ax})}{d\mathbf{x}} = \underline{\mathbf{A}^T \mathbf{y}}$$

$$\frac{d(\mathbf{x}^T \mathbf{Ax})}{d\mathbf{x}} = (\mathbf{A} + \mathbf{A}^T)\mathbf{x}$$

$$f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} = \underline{a_1 x_1 + a_2 x_2 + \dots + a_d x_d}$$

<https://math.stackexchange.com/questions/312077/differentiate-fx-xtax>

- last one is hard lmao

# Linear Regression

$$\sum_{i=1}^m (f_{\mathbf{w}, b}(\mathbf{x}_i) - y_i)^2$$

- All model-based learning algorithms have a **loss function**
- What we do to find the best model is **to minimize the objective** known as the **cost function**
- **Cost function** is a sum of **loss functions** over training set plus possibly some model complexity penalty (regularization)
- In linear regression, the cost function is given by the *average loss*, also called the **empirical risk** because we do not have all the data (e.g. testing data)
  - The average of all penalties is obtained by applying the model to the training data
- Loss function is the penalty of being inaccurate
- Empirical risk, because, sample may not be representative the real world distribution, the best possible thing doable is to minimise the empirical risk

**Linear regression** is a popular regression learning algorithm that learns a model which is a linear combination of features of the input example.

$$\mathbf{X}\mathbf{w} = \mathbf{y}, \quad \mathbf{X} \in \mathcal{R}^{m \times d}, \mathbf{w} \in \mathcal{R}^{d \times 1}, \mathbf{y} \in \mathcal{R}^{m \times 1}$$

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \dots & x_{m,d} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

**Problem Statement:** To predict the unknown  $y$  for a given  $\mathbf{x}$  (**testing**)

- We have a collection of labeled examples (**training**)  $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ 
  - $m$  is the size of the collection
  - $\mathbf{x}_i$  is the  $d$ -dimensional feature vector of example  $i = 1, \dots, m$  (input)
  - $y_i$  is a real-valued target (1-D)
  - Note:
    - when  $y_i$  is **continuous** valued, it is a **regression problem**
    - when  $y_i$  is **discrete** valued, it is a **classification problem**
- We want to build a model  $f_{\mathbf{w}, b}(\mathbf{x})$  as a linear combination of features of example  $\mathbf{x}$ :  $f_{\mathbf{w}, b}(\mathbf{x}) = \mathbf{x}^T \mathbf{w} + b$   
where  $\mathbf{w}$  is a  $d$ -dimensional vector of parameters and  $b$  is a real number.
- The notation  $f_{\mathbf{w}, b}$  means that the model  $f$  is parametrized by two values:  $\mathbf{w}$  and  $b$

Ref: [Book4] Stephen Boyd and Lieven Vandenberghe, "Introduction to Applied Linear Algebra", Cambridge University Press, 2018 (chp.14)

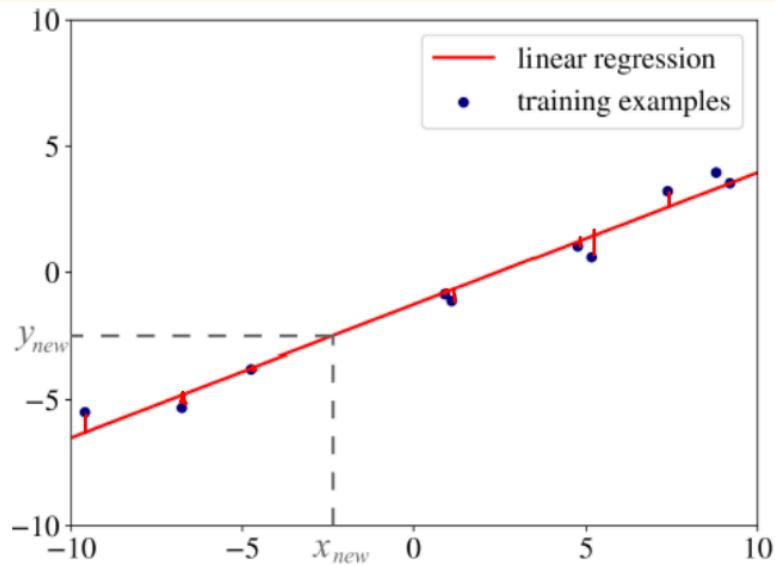
- during training cannot use test data, only training data, the use of test data is called data leakage
- a model is a function that turns  $\mathbf{x}$  into  $y$ 
  - parametrized by  $\mathbf{w}$  and  $b$ , so these are the values to be found

## Learning objective function

- To find the optimal values for  $w^*$  and  $b^*$  which **minimizes** the following expression:

$$\frac{1}{m} \sum_{i=1}^m (f_{w,b}(x_i) - y_i)^2$$

- In mathematics, the expression we minimize or maximize is called an **objective function**, or, simply, an **objective**



$(f_w(x_i) - y_i)^2$  is called the **loss function**: a measure of the difference between  $f_w(x_i)$  and  $y_i$  or a penalty for misclassification of example  $i$ .

- Objective function = cost function, minimise or maximise this to find the best regression, ie, the best value of w and b
  - sum of squared error or mean squared error
  - sum of the loss functions is the objective function
- Compare the value predicted by the model and the ground truth value
- Bias term is needed to shift the line in

## Learning objective function (using simplified notation hereon)

- To find the optimal values for  $\mathbf{w}^*$  which **minimizes** the following expression:

$$\sum_{i=1}^m (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$$

with  $f_{\mathbf{w}}(\mathbf{x}_i) = \mathbf{x}^T \mathbf{w}$ ,

where we define  $\mathbf{w} = [b, w_1, \dots, w_d]^T = [w_0, w_1, \dots, w_d]^T$ ,

and  $\mathbf{x}_i = [1, x_{i,1}, \dots, x_{i,d}]^T = [x_{i,0}, x_{i,1}, \dots, x_{i,d}]^T, i = 1, \dots, m$

- This particular choice of the loss function is called **squared error loss**

Note: The normalization factor  $\frac{1}{m}$  can be omitted as it does not affect the optimization.

- putting b inside w to simplify the expression, adjust the x accordingly put a bunch of 1

## Learning (Training)

- Consider the set of feature vector  $\mathbf{x}_i$  and target output  $y_i$  indexed by  $i = 1, \dots, m$ , a linear model  $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$  can be stacked as

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{X}) &= \mathbf{X}\mathbf{w} & \longleftrightarrow & \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{x}_1^T \mathbf{w} \\ \vdots \\ \mathbf{x}_m^T \mathbf{w} \end{bmatrix} & & \text{Learning target vector} \end{aligned}$$

where  $\mathbf{x}_i^T \mathbf{w} = [1, x_{i,1}, \dots, x_{i,d}] \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$

**Note:** The **bias/offset term** is responsible for translating the line/plane/hyperplane away from the origin.

- shove all the sample together and form matrix X

## Least Squares Regression

In vector-matrix notation, the minimization of the objective function can be written compactly using  $\mathbf{e} = \mathbf{X}\mathbf{w} - \mathbf{y}$  :

$$\begin{aligned} J(\mathbf{w}) &= \mathbf{e}^T \mathbf{e} \\ &= (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= (\mathbf{w}^T \mathbf{X}^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{y}^T \mathbf{y} \\ &= \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{y}^T \mathbf{y}. \end{aligned}$$

**Note:** when  $f_{\mathbf{w}}(\mathbf{X}) = \mathbf{X}\mathbf{w}$ , then

$$\sum_{i=1}^m (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}).$$

- minimise sum of square of error  $\Rightarrow$  least squares regression
- $J(w)$  is a scalar

Differentiating  $J(w)$  with respect to  $w$  and setting the result to  $0$ :

$$\frac{\partial}{\partial w} J(w) = 0$$

$$\begin{aligned}\frac{\partial}{\partial w} (w^T X^T X w - 2y^T X w + y^T y) &= 0 \\ \Rightarrow 2X^T X w - 2X^T y &= 0 \\ \Rightarrow X^T X w &= X^T y\end{aligned}$$

$\Rightarrow$  Any minimizer  $\hat{w}$  of  $J(w)$  must satisfy  $X^T(Xw - y) = 0$

If  $X^T X$  is invertible, then

**Learning/training:**

$$\hat{w} = (X^T X)^{-1} X^T y$$

**Prediction/testing:**

$$\hat{f}_w(X_{new}) = X_{new} \hat{w}$$

- differentiate with respect to  $w$  to minimise the cost, to find the best  $w$

Let  $V \subseteq \mathbb{R}^n$  and  $S = \{u_1, u_2, \dots, u_k\}$  be a basis for  $V$ . Then the orthogonal projection of any vector  $w \in \mathbb{R}^n$  onto  $V = A(A^T A)^{-1} A^T w$ , where  $A = (u_1 \ u_2 \ \dots \ u_k)$   $A \in \mathbb{R}^{n \times k}$

Since  $S$  is basis, columns of  $A$  are linearly independent ( $\text{Rank}(A) = k$ ,  $A$  is full ranked) so  $A^T A$  is invertible

$\therefore u = (A^T A)^{-1} A^T w$  is the unique solution to  $A^T A u = A^T w$

$\therefore$  project of  $w$  onto  $\text{Col}(A) = V = Au = A(A^T A)^{-1} A^T w$

- the least square is the left inverse

- $X^T X$  must be invertible

- note that  $w$  include the b

Example 1: one dimension input, 1 feature

**Example 1**    **Training set**     $\{(x_i, y_i)\}_{i=1}^m$

X	w	y	
$\begin{bmatrix} 1 & -9 \\ 1 & -7 \\ 1 & -5 \\ 1 & 1 \\ 1 & 5 \\ 1 & 9 \end{bmatrix}$	$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$	$\begin{bmatrix} -6 \\ -6 \\ -4 \\ -1 \\ 1 \\ 4 \end{bmatrix}$	$\{x = -9\} \rightarrow \{y = -6\}$ $\{x = -7\} \rightarrow \{y = -6\}$ $\{x = -5\} \rightarrow \{y = -4\}$ $\{x = 1\} \rightarrow \{y = -1\}$ $\{x = 5\} \rightarrow \{y = 1\}$ $\{x = 9\} \rightarrow \{y = 4\}$

$$\begin{bmatrix} 1 & -9 \\ 1 & -7 \\ 1 & -5 \\ 1 & 1 \\ 1 & 5 \\ 1 & 9 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} -6 \\ -6 \\ -4 \\ -1 \\ 1 \\ 4 \end{bmatrix}$$

This set of linear equations has **no exact solution**

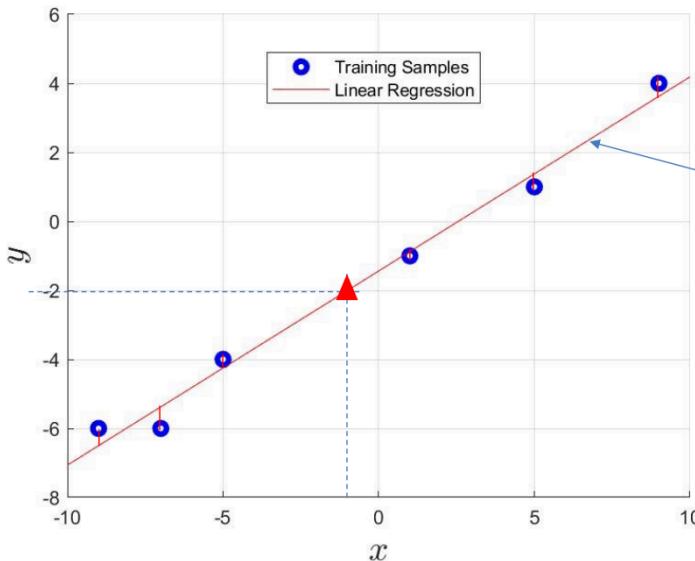
However,  $X^T X$  is invertible

**Least square approximation**

$$\hat{w} = X^\dagger y = (X^T X)^{-1} X^T y$$

$$= \begin{bmatrix} 6 & -6 \\ -6 & 262 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -9 & -7 & -5 & 1 & 5 & 9 \end{bmatrix} \begin{bmatrix} -6 \\ -6 \\ -4 \\ -1 \\ 1 \\ 4 \end{bmatrix} = \begin{bmatrix} -1.4375 \\ 0.5625 \end{bmatrix}$$

- remember to add 1 to get the bias in (augmentation)
- then shove the x into X along with the padded 1
- Overdetermined system, likely to have left inverse
  - no exact solution, has least square solution



$$\hat{y} = \mathbf{X}\hat{\mathbf{w}}$$

$$= \mathbf{X} \begin{bmatrix} -1.4375 \\ 0.5625 \end{bmatrix}$$

$$y = -1.4375 + 0.5625x$$

### Prediction:

#### Test set

$$\{x = -1\} \rightarrow \{y = ?\}$$

$$\hat{y} = [1 \quad -1] \begin{bmatrix} -1.4375 \\ 0.5625 \end{bmatrix}$$

$$= -2$$

Linear Regression on one-dimensional samples

### Python demo 1

```

1 # EE2211 Lecture 5 Demo 1 linear regression
2 import pandas as pd
3 #import matplotlib.pyplot as plt
4 import numpy as np
5 from numpy.linalg import inv
6 from sklearn.metrics import mean_squared_error
7 ##
8 X = np.array([[1, -9], [1, -7], [1, -5], [1, 1], [1, 5], [1, 9]])
9 Y = np.array([[ -6], [-6], [-4], [-1], [1], [4]])
10 w = inv(X.T @ X) @ X.T @ Y
11 print(w)
12
13 Xnew = np.array([1, -1])
14 Ynew = Xnew@w
15 print(Ynew)
16
17 ## difference
18 print("Mean squared error between Y and Xw")
19 Ytest=X@w
20 MSE = np.square(np.subtract(Ytest,Y)).mean()
21 print(MSE)
22 MSE = mean_squared_error(Ytest,Y)
23 print(MSE)
24

```

- manually do MSE or use built in function
- remember to pad 1 for X

```

[[[-1.4375]
 [ 0.5625]]
 [-2.]
Mean squared error between Y and XW
0.1666666666666666
0.1666666666666666

```

Example 2: 3 dimension input, 3 features

<b>Example 2</b>	$\{(\mathbf{x}_i, y_i)\}_{i=1}^m$	$\{x_1 = 1, x_2 = 1, x_3 = 1\} \rightarrow \{y = 1\}$
<b>Training set</b>		$\{x_1 = 1, x_2 = -1, x_3 = 1\} \rightarrow \{y = 0\}$
		$\{x_1 = 1, x_2 = 1, x_3 = 3\} \rightarrow \{y = 2\}$
		$\{x_1 = 1, x_2 = 1, x_3 = 0\} \rightarrow \{y = -1\}$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 3 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \end{bmatrix}$$

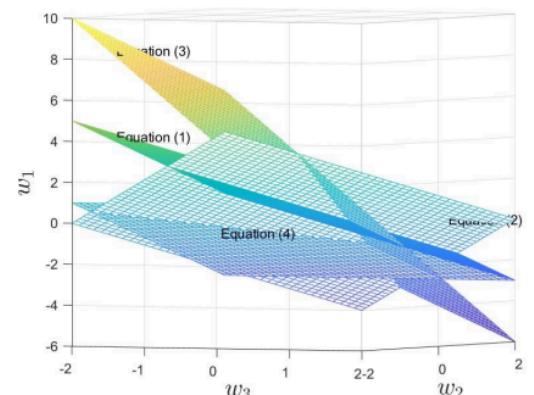
This set of linear equations has no exact solution

However,  $\mathbf{X}^T \mathbf{X}$  is invertible

$$\hat{\mathbf{w}} = \mathbf{X}^\dagger \mathbf{y} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

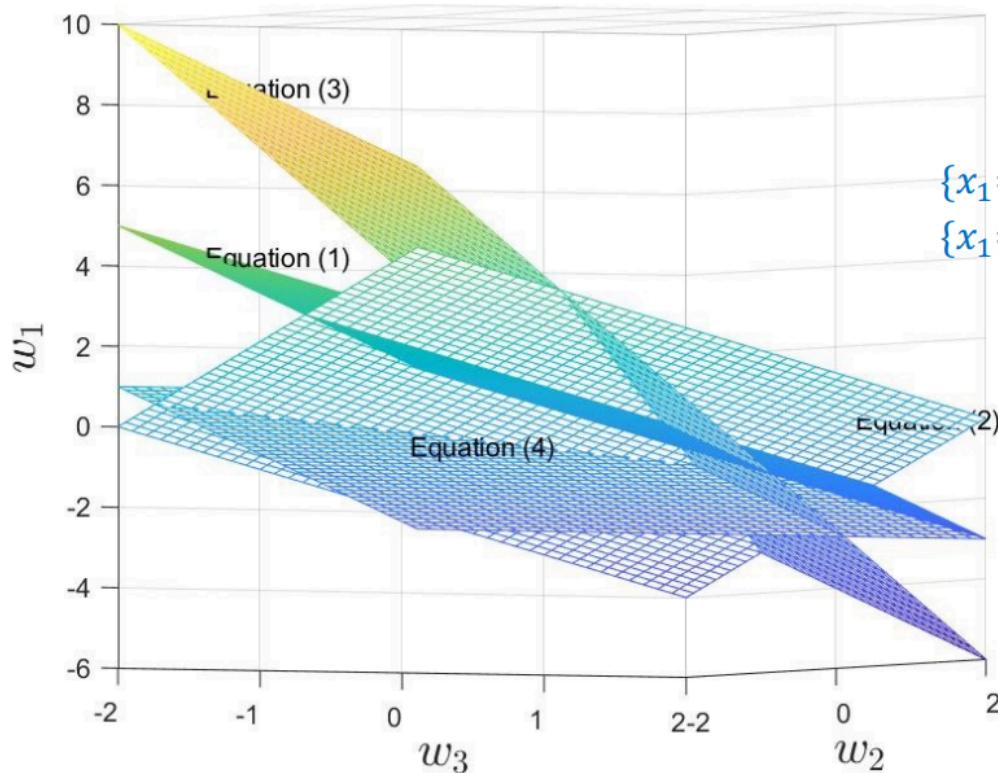
$$= \begin{bmatrix} 4 & 2 & 5 \\ 2 & 4 & 3 \\ 5 & 3 & 11 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.7500 \\ 0.1786 \\ 0.9286 \end{bmatrix}$$

**Least square approximation**



- in this case the first feature is all one (or a constant), so no need to pad ones
  - so  $w_1$  used as the offset

## The four linear equations



## Prediction:

### Test set

$$\{x_1 = 1, x_2 = 6, x_3 = 8\} \rightarrow \{y = ?\}$$

$$\{x_1 = 1, x_2 = 0, x_3 = -1\} \rightarrow \{y = ?\}$$

$$\hat{\mathbf{y}} = \hat{\mathbf{f}}_{\mathbf{w}}(\mathbf{X}_{new}) = \mathbf{X}_{new}\hat{\mathbf{w}}$$

$$\begin{aligned}\hat{\mathbf{y}} &= \begin{bmatrix} 1 & 6 & 8 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} -0.7500 \\ 0.1786 \\ 0.9286 \end{bmatrix} \\ &= \begin{bmatrix} 7.7500 \\ -1.6786 \end{bmatrix}\end{aligned}$$

### Learning of Vectored Function (Multiple Outputs)

For one sample: a linear model  $\mathbf{f}_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T \mathbf{W}$  Vector function

For  $m$  samples:  $\mathbf{F}_{\mathbf{w}}(\mathbf{X}) = \mathbf{X}\mathbf{W} = \mathbf{Y}$

$$\begin{array}{l} \text{Sample 1} \longrightarrow \mathbf{x}_1^T \\ \vdots \\ \text{Sample } m \longrightarrow \mathbf{x}_m^T \end{array} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix} \mathbf{W} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m,1} & \dots & x_{m,d} \end{bmatrix} \underbrace{\begin{bmatrix} w_{0,1} & \dots & w_{0,h} \\ w_{1,1} & \dots & w_{1,h} \\ \vdots & \ddots & \vdots \\ w_{d,1} & \dots & w_{d,h} \end{bmatrix}}_h$$

$$\begin{array}{l} \text{Sample 1's output} \longrightarrow \begin{bmatrix} y_{1,1} & \dots & y_{1,h} \end{bmatrix} \\ \vdots \\ \text{Sample } m \text{'s output} \longrightarrow \underbrace{\begin{bmatrix} y_{m,1} & \dots & y_{m,h} \end{bmatrix}}_h \end{array} \underbrace{m}_h$$

$$\mathbf{X} \in \mathcal{R}^{m \times (d+1)}, \mathbf{W} \in \mathcal{R}^{(d+1) \times h}, \mathbf{Y} \in \mathcal{R}^{m \times h}$$

- w is no longer a vector, it's a matrix W
- it's solving the same system of equation but with different y, the features are the same, to predict the different output, the weights must necessarily be different
- But now must be least squares for all the output predictions

**Objective:**  $\sum_{i=1}^m (\mathbf{f}_w(\mathbf{x}_i) - \mathbf{y}_i)^2 = \mathbf{E}^T \mathbf{E}$

## Least Squares Regression of Multiple Outputs

In matrix notation, the sum of squared errors cost function can be written compactly using  $\mathbf{E} = \mathbf{XW} - \mathbf{Y}$ :

$$\begin{aligned} J(\mathbf{W}) &= \text{trace}(\mathbf{E}^T \mathbf{E}) \\ &= \text{trace}[(\mathbf{XW} - \mathbf{Y})^T (\mathbf{XW} - \mathbf{Y})] \end{aligned}$$

- With error matrix  $\mathbf{E} = \mathbf{XW} - \mathbf{Y}$ ,  $\mathbf{E}$  is  $m \times h$
- Minimising  $J$  is hard, trace of  $J$  is easier
- $\mathbf{E}^T \mathbf{E}$  is  $h \times h$  with the trace of  $\mathbf{E}^T \mathbf{E}$  being the sum of error squared

### Least Squares Regression of Multiple Outputs

$$\begin{aligned} J(\mathbf{W}) &= \text{trace}(\mathbf{E}^T \mathbf{E}) \\ &= \text{trace}\left(\begin{bmatrix} \mathbf{e}_1^T \\ \vdots \\ \mathbf{e}_h^T \end{bmatrix} [\mathbf{e}_1 \ \mathbf{e}_2 \ \dots \ \mathbf{e}_h]\right) \\ &= \text{trace}\left(\begin{bmatrix} \mathbf{e}_1^T \mathbf{e}_1 & \mathbf{e}_1^T \mathbf{e}_2 & \dots & \mathbf{e}_1^T \mathbf{e}_h \\ \mathbf{e}_2^T \mathbf{e}_1 & \mathbf{e}_2^T \mathbf{e}_2 & \dots & \mathbf{e}_2^T \mathbf{e}_h \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{e}_h^T \mathbf{e}_1 & \mathbf{e}_h^T \mathbf{e}_2 & \dots & \mathbf{e}_h^T \mathbf{e}_h \end{bmatrix}\right) = \sum_{k=1}^h \mathbf{e}_k^T \mathbf{e}_k \end{aligned}$$

○

If  $\mathbf{X}^T \mathbf{X}$  is invertible, then

**Learning/training:**  $\widehat{\mathbf{W}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$

**Prediction/testing:**  $\widehat{\mathbf{F}}_w(\mathbf{X}_{new}) = \mathbf{X}_{new} \widehat{\mathbf{W}}$

- - least square estimate for multiple outputs is the similar as the for single output

- The regression is the same as the outputs are independent, and can just do the same regression for the output separately. But doing together can save computation resource

Example 3:

**Training set**  $\{x_1 = 1, x_2 = 1, x_3 = 1\} \rightarrow \{y_1 = 1, y_2 = 0\}$

$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^m \quad \{x_1 = 1, x_2 = -1, x_3 = 1\} \rightarrow \{y_1 = 0, y_2 = 1\}$

$\{x_1 = 1, x_2 = 1, x_3 = 3\} \rightarrow \{y_1 = 2, y_2 = -1\}$

$\{x_1 = 1, x_2 = 1, x_3 = 0\} \rightarrow \{y_1 = -1, y_2 = 3\}$

**X**                    **W**                    **Y**

$$\text{Bias} \rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 3 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & -1 \\ -1 & 3 \end{bmatrix}$$

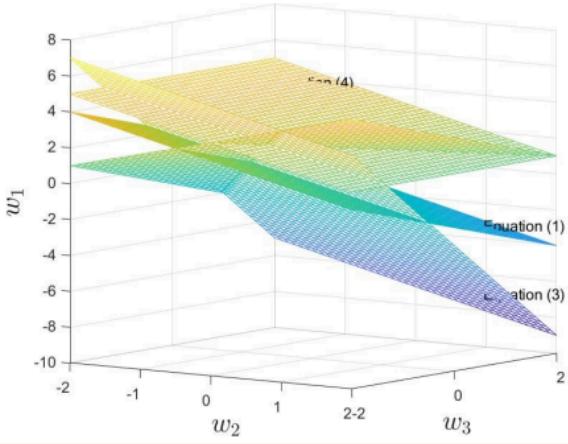
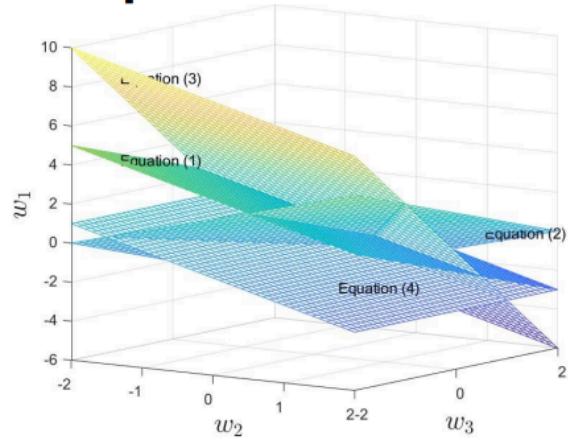
This set of linear equations has NO exact solution

$$\hat{\mathbf{W}} = \mathbf{X}^\dagger \mathbf{Y} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \quad \mathbf{X}^T \mathbf{X} \text{ is invertible}$$

**Least square approximation**

$$= \begin{bmatrix} 4 & 2 & 5 \\ 2 & 4 & 3 \\ 5 & 3 & 11 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & -1 \\ -1 & 3 \end{bmatrix} = \begin{bmatrix} -0.75 & 2.25 \\ 0.1786 & 0.0357 \\ 0.9286 & -1.2143 \end{bmatrix}$$

## Example 3



## Prediction:

**Test set:** two new samples

$$\{x_1 = 1, x_2 = 6, x_3 = 8\} \rightarrow \{y_1 = ?, y_2 = ?\}$$

$$\{x_1 = 1, x_2 = 0, x_3 = -1\} \rightarrow \{y_1 = ?, y_2 = ?\}$$

$$\hat{\mathbf{Y}} = \mathbf{X}_{new} \hat{\mathbf{W}}$$

$$\text{Bias} \rightarrow \begin{bmatrix} 1 & 6 & 8 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} -0.75 & 2.25 \\ 0.1786 & 0.0357 \\ 0.9286 & -1.2143 \end{bmatrix} = \begin{bmatrix} 7.75 & -7.25 \\ -1.6786 & 3.4643 \end{bmatrix}$$

Python demo 2

```

27 # EE2211 Lecture 5 Demo 2
28 import pandas as pd
29 import matplotlib.pyplot as plt
30 import numpy as np
31 from numpy.linalg import inv
32 from sklearn.metrics import mean_squared_error
33 ##
34 X = np.array([[1, 1, 1], [1, -1, 1], [1, 1, 3], [1, 1, 0]])
35 Y = np.array([[1, 0], [0, 1], [2, -1], [-1, 3]])
36 w = inv(X.T @ X) @ X.T @ Y
37 print("the estimated w")
38 print(w)
39
40 Xnew = np.array([[1, 6, 8], [1, 0, -1]])
41 Ynew = Xnew@w
42 print("testing Ynew")
43 print(Ynew)
44
45 ## difference
46 print("Mean squared error between Y and Xw")
47 Ytest=X@w
48 MSE = np.square(np.subtract(Ytest,Y)).mean()
49 print(MSE)
50 MSE = mean_squared_error(Ytest,Y)
51 print(MSE)
52

```

```

the estimated w
[[ -0.75      2.25      ]
 [  0.17857143  0.03571429]
 [  0.92857143 -1.21428571]]
testing Ynew
[[ 7.75      -7.25      ]
 [ -1.67857143  3.46428571]]
Mean squared error between Y and Xw
0.3035714285714286
0.30357142857142855

```

Example 4

The values of feature  $x$  and their corresponding values of multiple outputs target  $y$  are shown in the table below.

Based on the least square regression, what are the values of  $w$ ?  
 Based on the current mapping, when  $x = 2$ , what is the value of  $y$ ?

$x$	[3]	[4]	[10]	[6]	[7]
$y$	[0, 5]	[1.5, 4]	[-3, 8]	[-4, 10]	[1, 6]

$$\hat{W} = X^{\dagger}Y = (X^T X)^{-1} X^T Y = \begin{bmatrix} 1.9 & 3.6 \\ -0.4667 & 0.5 \end{bmatrix}$$

Python demo 3

$$\widehat{Y}_{new} = X_{new} \hat{W} = [1 \quad 2] \hat{W} = [0.9667 \quad 4.6]$$

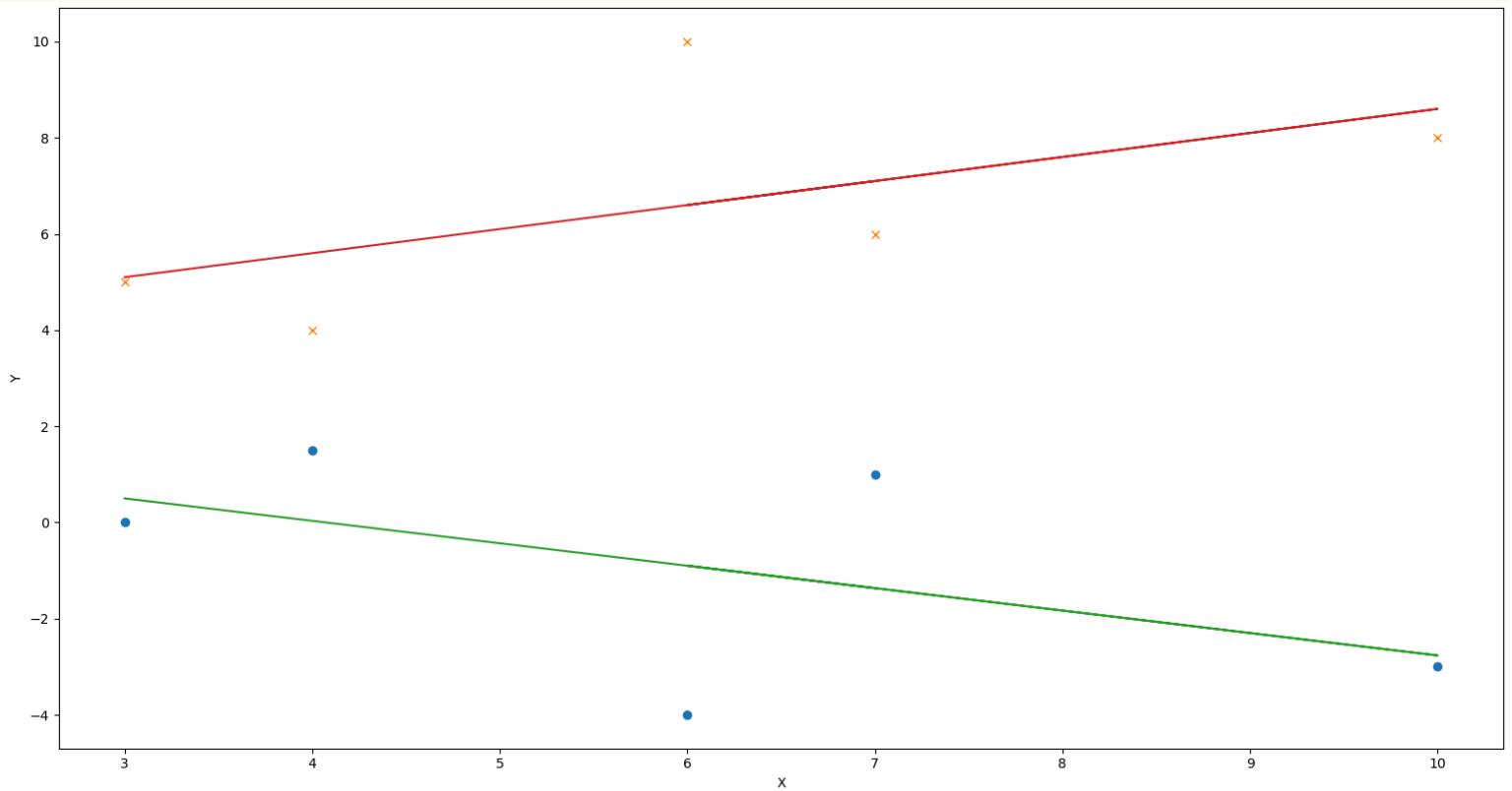
Prediction

```

53 # EE2211 Lecture 5 Demo 3
54 import pandas as pd
55 import matplotlib.pyplot as plt
56 import numpy as np
57 from numpy.linalg import inv
58 from sklearn.metrics import mean_squared_error
59 ##
60 X = np.array([[1, 3], [1, 4], [1, 10], [1, 6], [1, 7]])
61 Y = np.array([[0, 5], [1.5, 4], [-3, 8], [-4, 10], [1, 6]])
62 w = inv(X.T @ X) @ X.T @ Y
63 print('W')
64 print(w)
65
66 Xnew = np.array([[1, 2]])
67
68 Ynew = Xnew@w
69
70 print('Ynew')
71 print(Xnew)
72 print(Xnew.shape)
73 print(Ynew)
74
75 Ytest= X@w
76 print('Ytest')
77 print(Ytest)
78 plt.plot(X[:,1], Y[:,0], 'o', label = 'Y1')
79 plt.plot(X[:,1], Y[:,1], 'x', label = 'Y2')
80 plt.plot(X[:,1], Ytest[:,0])
81 plt.plot(X[:,1], Ytest[:,1])
82 plt.xlabel('X')
83 plt.ylabel('Y')
84 plt.show()

```

W	$\begin{bmatrix} 1.9 & 3.6 \\ -0.46666667 & 0.5 \end{bmatrix}$
Ynew	$\begin{bmatrix} [1 2] \\ (1, 2) \end{bmatrix}$
Ytest	$\begin{bmatrix} [0.96666667 4.6] \\ [[0.5 5.1] [0.03333333 5.6] [-2.76666667 8.6] [-0.9 6.6] [-1.36666667 7.1]] \end{bmatrix}$



- green is  $Y_1$  and red is  $Y_2$

### Class Quiz

Suppose  $y = 3x + 5$ , this is a linear function.

Jin Yueming 42

True

69%

False

31%

- its affine

Suppose  $g(\mathbf{x})$  is a scalar function of  $d$  variables where  $\mathbf{x}$  is a  $d \times 1$  vector, the outcome of differentiation of  $g(\mathbf{x})$  w.r.t.  $\mathbf{x}$  is a  $d \times 1$  vector.

43

True

91%

False

9%

# Summary

- Notations, Vectors, Matrices
- Operations on Vectors and Matrices
  - Dot-product, matrix inverse
- Systems of Linear Equations  $f_w(\mathbf{X}) = \mathbf{Xw} = \mathbf{y}$ 
  - Matrix-vector notation, linear dependency, invertible
  - Even-, over-, under-determined linear systems
- Functions, Derivative and Gradient
  - Inner product, linear/affine functions
  - Maximum and minimum, partial derivatives, gradient
- Least Squares, Linear Regression
  - Objective function, loss function
  - Least square solution, training/learning and testing/prediction
  - Linear regression with multiple outputs

Midterm (L1 to L5)  
Trial quiz

**Learning/training**  $\hat{\mathbf{w}} = (\mathbf{X}_{train}^T \mathbf{X}_{train})^{-1} \mathbf{X}_{train}^T \mathbf{y}_{train}$

**Prediction/testing**  $\mathbf{y}_{test} = \mathbf{X}_{test} \hat{\mathbf{w}}$

- Classification
- Ridge Regression
- Polynomial Regression

Python packages: numpy, pandas, matplotlib.pyplot, numpy.linalg, and sklearn.metrics (for mean\_squared\_error), numpy.linalg.pinv

## Lec6: Ridge regression & Polynomial regression

### Linear Regression for Classification

- We have a collection of labeled examples
  - $m$  is the size of the collection
  - $\mathbf{x}_i$  is the  $d$ -dimensional feature vector of example  $i = 1, \dots, m$
  - $y_i$  is discrete target label (e.g.,  $y_i \in \{-1, +1\}$  or  $\{0, 1\}$  for binary classification problems)
  - Note:
    - when  $y_i$  is **continuous** valued  $\rightarrow$  a **regression problem**
    - when  $y_i$  is **discrete** valued  $\rightarrow$  a **classification problem**
- Linear model:  $f_{\mathbf{w}, b}(\mathbf{x}) = \mathbf{x}^T \mathbf{w} + b$  or in compact form  $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$  (having the offset term absorbed into the inner product)

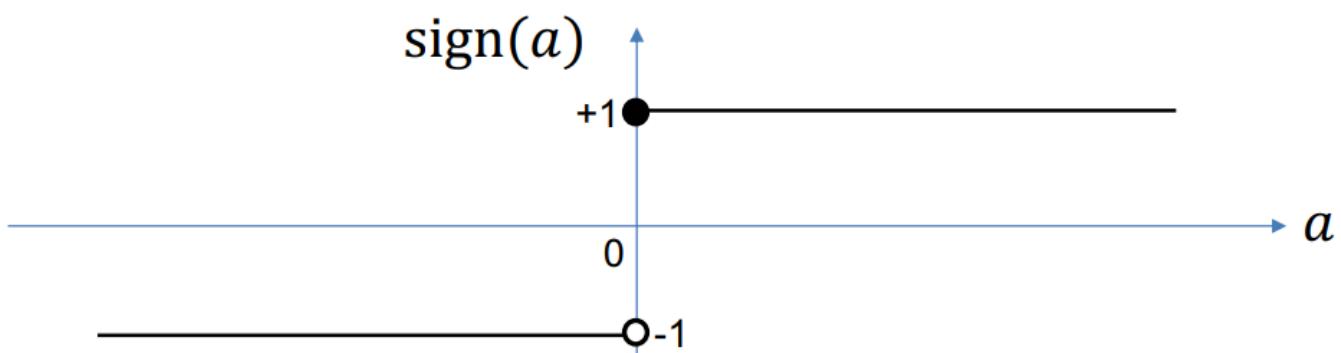
### Binary Classification

If  $\mathbf{X}^T \mathbf{X}$  is invertible, then

**Learning:**  $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ ,  $y_i \in \{-1, +1\}, i = 1, \dots, m$

**Prediction:**  $\hat{f}_{\mathbf{w}}^c(\mathbf{x}_{new}) = \text{sign}(\mathbf{x}_{new}^T \hat{\mathbf{w}})$  for each row  $\mathbf{x}_{new}^T$  of  $\mathbf{X}_{new}$

$$\text{sign}(a) = +1 \text{ for } a \geq 0 \text{ and } -1 \text{ for } a < 0$$



- If classified by 0, 1, take threshold as 0.5
  - or use -1 +1
  - don't use larger number incase large number overshadows smaller number
- Use sign function for binary

Example

**Example 1** Training set  $\{x_i, y_i\}_{i=1}^m$

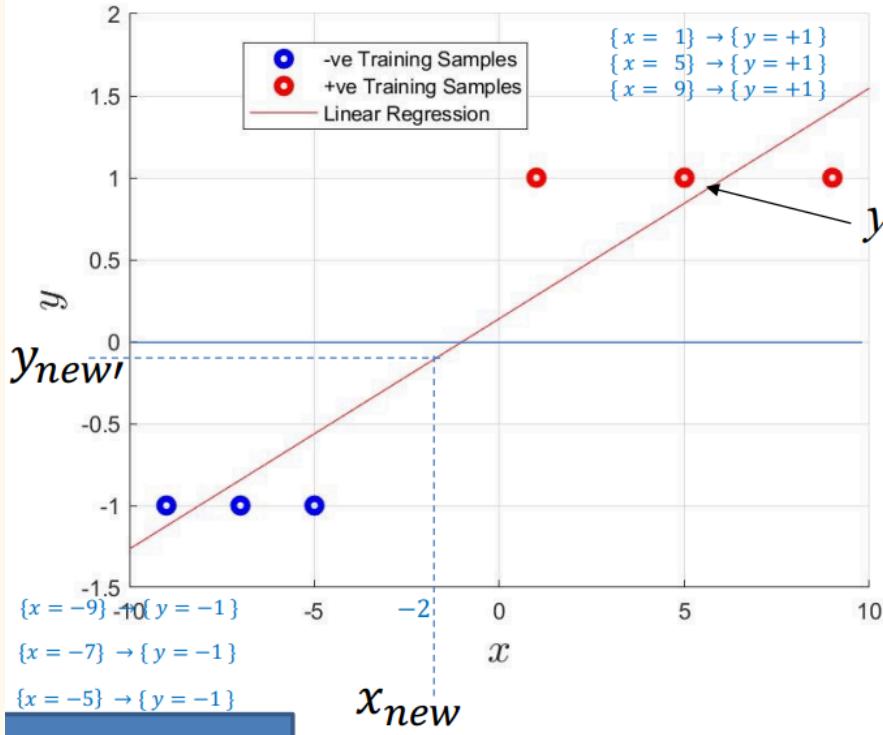
Bias	X	w	y	
	1 - 9 1 - 7 1 - 5 1 1 1 5 1 9	$w_0$ $w_1$	$-1$ $-1$ $-1$ $1$ $1$ $1$	$\{x = -9\} \rightarrow \{y = -1\}$ $\{x = -7\} \rightarrow \{y = -1\}$ $\{x = -5\} \rightarrow \{y = -1\}$ $\{x = 1\} \rightarrow \{y = +1\}$ $\{x = 5\} \rightarrow \{y = +1\}$ $\{x = 9\} \rightarrow \{y = +1\}$

This set of linear equations has NO exact solution

$$\hat{\mathbf{w}} = \mathbf{X}^\dagger \mathbf{y} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad \mathbf{X}^T \mathbf{X} \text{ is invertible}$$

$$= \begin{bmatrix} 6 & -6 \\ -6 & 262 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -9 & -7 & -5 & 1 & 5 & 9 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.1406 \\ 0.1406 \end{bmatrix} \quad \text{Least square approximation}$$

# Example 1



$$\begin{aligned}\hat{y} &= \text{sign}(\mathbf{X}\hat{\mathbf{w}}) \\ &= \text{sign}(\mathbf{X} \begin{bmatrix} 0.1406 \\ 0.1406 \end{bmatrix}) \\ y' &= \mathbf{X}\hat{\mathbf{w}} = 0.1406 + 0.1406x\end{aligned}$$

## Prediction:

Test set  $\{x = -2\} \rightarrow \{y = ?\}$

$$\begin{aligned}y_{new} &= \hat{f}_{\mathbf{w}}^c(\mathbf{x}_{new}) = \text{sign}(\mathbf{x}_{new}\hat{\mathbf{w}}) \\ &\quad \text{Bias} \\ &= \text{sign}(\begin{bmatrix} 1 & -2 \end{bmatrix} \begin{bmatrix} 0.1406 \\ 0.1406 \end{bmatrix}) \\ &= \text{sign}(-0.1406) = -1\end{aligned}$$

## Python demo 1

Linear Regression for one-dimensional classification

```
# EE2211 Lecture 6 Demo 1 Binary classification
import numpy as np
from numpy.linalg import inv

X = np.array([[1,-9], [1,-7], [1,-5], [1,1], [1,5], [1, 9]])
y = np.array([-1, -1, -1, 1, 1, 1])
## Linear regression for classification
w = inv(X.T @ X) @ X.T @ y
print("Estimated w")
print(w)
xt = np.array([[-2]])
y_predict = xt @ w
print("Predicted y")
print(y_predict)
y_class_predict = np.sign(y_predict)
print("Predicted y class")
print(y_class_predict)
```

```
Estimated w
[[0.140625]
 [0.140625]]
Predicted y
[[-0.140625]]
Predicted y class
[[-1.]]
```

If  $\mathbf{X}^T \mathbf{X}$  is invertible, then

$$\text{Learning: } \widehat{\mathbf{W}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}, \quad \mathbf{Y} \in \mathbf{R}^{m \times C}$$

**Prediction:**  $\hat{f}_w^c(x_{new}) = \arg \max_{k=1,\dots,c} \left( x_{new}^T \widehat{\mathbf{W}}(:, k) \right)$  for each  $x_{new}^T$  of  $\mathbf{X}_{new}$

Each row (of  $i = 1, \dots, m$ ) in  $\mathbf{Y}$  has an **one-hot** encoding/assignment:

e.g., target for class-1 is labelled as  $y_i^T = [1, 0, 0, \dots, 0]$  for the  $i$ th sample,

target for class-2 is labelled as  $\mathbf{y}_i^T = [0, 1, 0, \dots, 0]$  for the  $i$ th sample,

target for class-C is labelled as  $\mathbf{y}_m^T = \underbrace{[0, 0, \dots, 0]}_C, 1$  for the  $m$ th sample.

- Use one hot encoding and multiple output linear regression
  - One column for each class
  - Then for prediction, the column with the biggest value to determine which class it is
    - biggest value means the most positive, not the magnitude

<b>Training set</b>	$\{x_1 = 1, x_2 = 1\} \rightarrow \{y_1 = 1, y_2 = 0, y_3 = 0\}$	Class 1
$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^m$	$\{x_1 = -1, x_2 = 1\} \rightarrow \{y_1 = 0, y_2 = 1, y_3 = 0\}$	Class 2
	$\{x_1 = 1, x_2 = 3\} \rightarrow \{y_1 = 1, y_2 = 0, y_3 = 0\}$	Class 1
	$\{x_1 = 1, x_2 = 0\} \rightarrow \{y_1 = 0, y_2 = 0, y_3 = 1\}$	Class 3

$$\text{Bias} \rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 3 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This set of linear equations has NO exact solution.

$$\hat{\mathbf{W}} = \mathbf{X}^\dagger \mathbf{Y} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \quad \text{if } \mathbf{X}^T \mathbf{X} \text{ is invertible}$$

## Least square approximation

$$= \begin{bmatrix} 4 & 2 & 5 \\ 2 & 4 & 3 \\ 5 & 3 & 11 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.2857 & -0.5 & 0.2143 \\ 0.2857 & 0 & -0.2857 \end{bmatrix}$$

**Test set  $\mathbf{X}_{new}$**      $\{x_1 = 6, x_2 = 8\} \rightarrow \{\text{class 1, 2, or 3?}\}$   
 $\{x_1 = 0, x_2 = -1\} \rightarrow \{\text{class 1, 2, or 3?}\}$

$$\widehat{\mathbf{Y}} = \mathbf{X}_{new} \widehat{\mathbf{W}} = \begin{bmatrix} 1 & 6 & 8 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.2857 & -0.5 & 0.2143 \\ 0.2857 & 0 & -0.2857 \end{bmatrix}$$

**Category prediction:**

$$\begin{aligned} \widehat{f}_w^c(\mathbf{X}_{new}) &= \arg \max_{k=1, \dots, C} (\widehat{\mathbf{Y}}(:, k)) \\ &= \arg \max_{k=1, \dots, C} \left( \begin{bmatrix} 4 & -2.50 & -0.50 \\ -0.2587 & 0.50 & 0.7857 \end{bmatrix} \right) \\ &= \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \begin{array}{l} \xrightarrow{\text{Class 1}} \\ \xrightarrow{\text{Class 3}} \end{array} \end{aligned}$$

For each row of  $\mathbf{Y}$ , the **column position** of the **largest** number (across all columns for that row) determines the **class label**.

E.g. in the first row, the maximum number is 4 which is in column 1. Therefore, the resulting predicted class is 1.

## Python demo 2

```

# EE2211 Lecture 6 Demo 2 Multi-class classification
import numpy as np
from numpy.linalg import inv
from sklearn.preprocessing import OneHotEncoder
X = np.array([[1, 1, 1], [1, -1, 1], [1, 1, 3], [1, 1, 0]])
y_class = np.array([1, 2, 1, 3])
y_onehot = np.array([[1, 0, 0], [0, 1, 0], [1, 0, 0], [0, 0, 1]])
print("One-hot encoding manual")
print(y_class)
print(y_onehot)

print("One-hot encoding function")
onehot_encoder=OneHotEncoder(sparse=False)
print(onehot_encoder)
Ytr_onehot = onehot_encoder.fit_transform(y_class)
print(Ytr_onehot)

#reshaped = y_class.reshape(len(y_class), 1)
#print(reshaped)
#Ytr_onehot = onehot_encoder.fit_transform(reshaped)

```

```

## Linear Classification
print("Estimated W")
W = inv(X.T @ X) @ X.T @ Ytr_onehot
print(W)
X_test = np.array([[1, 6, 8], [1, 0, -1]])
yt_est = X_test@W;
print("Test")
print(yt_est)
yt_class = [[1 if y == max(x) else 0 for y in x] for x in yt_est ]
print("class label test")
print(yt_class)

```

```
One-hot encoding manual
```

```
[[1]
 [2]
 [1]
 [3]]
[[1 0 0]
 [0 1 0]
 [1 0 0]
 [0 0 1]]
```

```
One-hot encoding function
```

```
OneHotEncoder(sparse=False)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

```
Estimated W
```

```
[[ 1.11022302e-16  5.00000000e-01  5.00000000e-01]
 [ 2.85714286e-01 -5.00000000e-01  2.14285714e-01]
 [ 2.85714286e-01  2.77555756e-17 -2.85714286e-01]]
```

```
Test
```

```
[[ 4.          -2.5          -0.5         ]
 [-0.28571429  0.5          0.78571429]]
```

```
class label test
```

```
[[1, 0, 0], [0, 0, 1]]
```

## Ridge Regression

### Recall Linear regression

**Objective:**  $\hat{\mathbf{w}} = \operatorname{argmin} \sum_{i=1}^m (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 = (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$

The learning computation:  $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

We cannot guarantee that the matrix  $\mathbf{X}^T \mathbf{X}$  is invertible

**Ridge regression:** shrinks the regression coefficients  $w$  by imposing a penalty on their size

**Objective:**  $\hat{\mathbf{w}} = \operatorname{argmin} \sum_{i=1}^m (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 + \lambda \sum_{j=1}^d w_j^2$   
 $= \operatorname{argmin} (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$

Here  $\lambda \geq 0$  is a complexity parameter that controls the amount of shrinkage: the larger the value of  $\lambda$ , the greater the amount of shrinkage.

Note:  $m$  samples &  $d$  parameters

- Least square solution for linear regression requires  $\mathbf{X}^T \mathbf{X}$  must be invertible, but this is not guarantee for any given sample
- Ridge regression introduce additional objective term to shrink the  $w$ 
  - first part is minimise square error, second part is to minimise size of  $w$
  - reduce size of weights
- Ridge regression to avoid overfitting issue
- Multiple output ridge regression is out of scope

## Using a linear model:

$$\min_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

### Solution:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} ((\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}) &= \mathbf{0} \\ \Rightarrow 2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{y} + 2\lambda \mathbf{w} &= \mathbf{0} \\ \Rightarrow \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda \mathbf{w} &= \mathbf{X}^T \mathbf{y} \\ \Rightarrow (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})\mathbf{w} &= \mathbf{X}^T \mathbf{y} \end{aligned}$$

where  $\mathbf{I}$  is the  $d \times d$  identity matrix

Here on, we shall focus on single column of output  $\mathbf{y}$  in derivations in the sequel

Learning:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

- Adding a small value along the diagonal means even if  $\mathbf{X}^T \mathbf{X}$  is not invertible,  $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}$  is.

### Primal and Dual Form

- dual and prime give the same answer but one is computationally less expensive
- Always use the one that requires to find the inverse of the smaller square matrix

```
## dual solution m < d (without ridge)
w_dual = P.T @ inv(P @ P.T) @ y
print("Under-determined system")
print("Unique constrained solution, no ridge")
print(w_dual)

print("*****")
print("Approximation with dual ridge regression")
print(P.shape)
reg_L2 = 0.0001*np.identity(P.shape[0]) #number of rows of P = Dual I
print(reg_L2)
w_dual_ridge = P.T @ (inv(P @ P.T + reg_L2)) @ y
print(w_dual_ridge)

print("*****")
## primal ridge
print("Approximation with primal ridge regression")
print(P.shape)
reg_L = 0.0001*np.identity(P.shape[1]) #number of columns of P = Primal I
print(reg_L)
w_primal_ridge = inv(P.T @ P + reg_L) @ P.T @ y
print(w_primal_ridge)
```

<pre>Under-determined system Unique constrained solution, no ridge [[ -1.]  [ 1.]  [ 1.]  [ 1.]  [-4.]  [ 1.]]</pre>	<pre>***** Approximation with dual ridge regression (4, 6) [[ 0.0001  0.    0.    0.    0.    ]  [ 0.    0.0001  0.    0.    0.    ]  [ 0.    0.    0.0001  0.    0.    ]  [ 0.    0.    0.    0.0001  0.    ]  [ 0.    0.    0.    0.    0.0001 ]]</pre>
<pre>***** Approximation with primal ridge regression (4, 6) [[ 0.0001  0.    0.    0.    0.    0.    ]  [ 0.    0.0001  0.    0.    0.    0.    ]  [ 0.    0.    0.0001  0.    0.    0.    ]  [ 0.    0.    0.    0.0001  0.    0.    ]  [ 0.    0.    0.    0.    0.0001  0.    ]  [ 0.    0.    0.    0.    0.    0.0001 ]]</pre>	<pre>[[ -0.9993004 ]  [  0.99940033]  [  0.99940033]  [  0.99940033]  [ -3.9790115]  [  0.99940033]]</pre>

# Ridge Regression in Primal Form (when $m > d$ )

$(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})$  is invertible for  $\lambda > 0$ ,

Learning:  $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$

Prediction:  $\hat{f}_{\mathbf{w}}(\mathbf{X}_{new}) = \mathbf{X}_{new} \hat{\mathbf{w}}$

- Primal form used for  $m > d$ , over determined
  - always find inverse in smaller in dimension
- $\mathbf{X}$  is  $m \times d$
- $\mathbf{X}^T \mathbf{X}$  is  $d \times d$
- $\mathbf{I}$  is  $d \times d$

# Ridge Regression in Dual Form (when $m < d$ )

$(\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})$  is invertible for  $\lambda > 0$ ,

Learning:  $\hat{\mathbf{w}} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$

Prediction:  $\hat{f}_{\mathbf{w}}(\mathbf{X}_{new}) = \mathbf{X}_{new} \hat{\mathbf{w}}$

- Dual form used for  $m < d$ , under determined
  - always find inverse in smaller in dimension
- $\mathbf{X}$  is  $m \times d$
- $\mathbf{X}^T \mathbf{X}$  is  $m \times m$
- $\mathbf{I}$  is  $m \times m$

Derivation as homework (see tutorial 6).

Hint: start off with  $(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}) \mathbf{w} = \mathbf{X}^T \mathbf{y}$  and make use of  $\mathbf{w} = \mathbf{X}^T \mathbf{a}$  and  $\mathbf{a} = \lambda^{-1}(\mathbf{y} - \mathbf{X} \mathbf{w})$ ,  $\lambda > 0$

Given  $(X^T X + \lambda I) w = X^T y$

derive  $\hat{w} = X^T (X X^T + \lambda I)^{-1} y$

Hint:  $w = X^T a$  where

$$a = \lambda^{-1} (y - Xw), \lambda > 0$$

$$\begin{aligned} : X^T X w + \lambda I w &= X^T y \\ \lambda w &= X^T y - X^T X w \\ &= X^T (y - Xw) \end{aligned}$$

$$\begin{aligned} w &= \lambda^{-1} X^T (y - Xw) \\ &= X^T a \quad \text{--- (1)} \end{aligned}$$

$$: a = \lambda^{-1} (y - Xw)$$

$$\begin{aligned} \lambda a &= y - Xw \\ &= y - X X^T a \end{aligned}$$

$$X X^T a + \lambda a = y$$

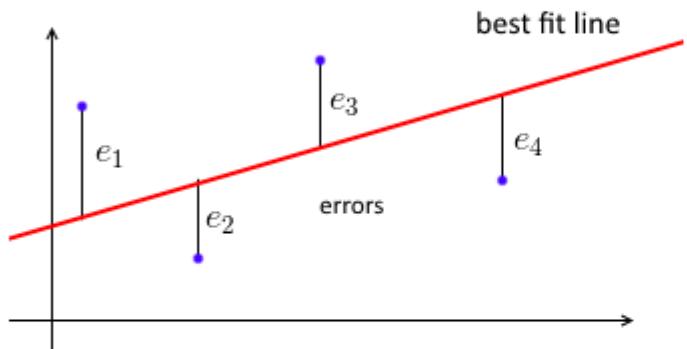
$$(X X^T + \lambda) a = y$$

$$a = (X X^T + \lambda I)^{-1} y \quad \text{--- (2)}$$

Short answer: no difference between Primal and Dual - it's only about the way of arriving to the solution. Kernel ridge regression is essentially the same as usual ridge regression, but uses the kernel trick to go non-linear.

## Linear Regression

First of all, a usual Least Squares Linear Regression tries to fit a straight line to the set of data points in such a way that the sum of squared errors is minimal.



We parametrize the best fit line with  $\mathbf{w}$  and for each data point  $(\mathbf{x}_i, y_i)$  we want  $\mathbf{w}^T \mathbf{x}_i \approx y_i$ . Let  $e_i = y_i - \mathbf{w}^T \mathbf{x}_i$  be the error - the distance between predicted and true values. So our goal is to

minimize the sum of squared errors  $\sum e_i^2 = \|\mathbf{e}\|^2 = \|X\mathbf{w} - \mathbf{y}\|^2$  where  $X = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}$  - a data matrix with each  $\mathbf{x}_i$  being a row, and  $\mathbf{y} = (y_1, \dots, y_n)$  a vector with all  $y_i$ 's.

Thus, the objective is  $\min_{\mathbf{w}} \|X\mathbf{w} - \mathbf{y}\|^2$ , and the solution is  $\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$  (known as "Normal Equation").

For a new unseen data point  $\mathbf{x}$  we predict its target value  $\hat{y}$  as  $\hat{y} = \mathbf{w}^T \mathbf{x}$ .

## Ridge Regression

When there are many correlated variables in linear regression models, the coefficients  $\mathbf{w}$  can become poorly determined and have lots of variance. One of the solutions to this problem is to restrict weights  $\mathbf{w}$  so they don't exceed some budget  $C$ . This is equivalent to using  $L_2$ -regularization, also known as "weight decay": it will decrease the variance at the cost of sometimes missing the correct results (i.e. by introducing some bias).

The objective now becomes  $\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2$ , with  $\lambda$  being the regularization parameter.

By going through the math, we obtain the following solution:

$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$ . It's very similar to the usual linear regression, but here we add  $\lambda$  to each diagonal element of  $\mathbf{X}^T \mathbf{X}$ .

Note that we can re-write  $\mathbf{w}$  as  $\mathbf{w} = \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$  (see [here](#) for details). For a new unseen data point  $\mathbf{x}$  we predict its target value  $\hat{y}$  as  $\hat{y} = \mathbf{x}^T \mathbf{w} = \mathbf{x}^T \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$ . Let  $\boldsymbol{\alpha} = (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$ . Then  $\hat{y} = \mathbf{x}^T \mathbf{X}^T \boldsymbol{\alpha} = \sum_{i=1}^n \alpha_i \cdot \mathbf{x}^T \mathbf{x}_i$ .

## Ridge Regression Dual Form

We can have a different look at our objective - and define the following quadratic program problem:

$$\min_{\mathbf{e}, \mathbf{w}} \sum_{i=1}^n e_i^2 \text{ s.t. } e_i = y_i - \mathbf{w}^T \mathbf{x}_i \text{ for } i = 1 \dots n \text{ and } \|\mathbf{w}\|^2 \leq C.$$

It's the same objective, but expressed somewhat differently, and here the constraint on the size of  $\mathbf{w}$  is explicit. To solve it, we define the Lagrangian  $\mathcal{L}_p(\mathbf{w}, \mathbf{e}; C)$  - this is the primal form that contains primal variables  $\mathbf{w}$  and  $\mathbf{e}$ . Then we optimize it w.r.t.  $\mathbf{e}$  and  $\mathbf{w}$ . To get the dual formulation, we put  $\mathbf{e}$  and  $\mathbf{w}$  back to  $\mathcal{L}_p(\mathbf{w}, \mathbf{e}; C)$ .

So,  $\mathcal{L}_p(\mathbf{w}, \mathbf{e}; C) = \|\mathbf{e}\|^2 + \beta^T (\mathbf{y} - X\mathbf{w} - \mathbf{e}) - \lambda (\|\mathbf{w}\|^2 - C)$ . By taking derivatives w.r.t.  $\mathbf{w}$  and  $\mathbf{e}$ , we obtain  $\mathbf{e} = \frac{1}{2} \beta$  and  $\mathbf{w} = \frac{1}{2\lambda} X^T \beta$ . By letting  $\alpha = \frac{1}{2\lambda} \beta$ , and putting  $\mathbf{e}$  and  $\mathbf{w}$  back to  $\mathcal{L}_p(\mathbf{w}, \mathbf{e}; C)$ , we get dual Lagrangian

$\mathcal{L}_d(\alpha, \lambda; C) = -\lambda^2 \|\alpha\|^2 + 2\lambda \alpha^T \mathbf{y} - \lambda \|X^T \alpha\| - \lambda C$ . If we take a derivative w.r.t.  $\alpha$ , we get  $\alpha = (X X^T - \lambda I)^{-1} \mathbf{y}$  - the same answer as for usual Kernel Ridge regression. There's no need to take a derivative w.r.t  $\lambda$  - it depends on  $C$ , which is a regularization parameter - and it makes  $\lambda$  regularization parameter as well.

Next, put  $\alpha$  to the primal form solution for  $\mathbf{w}$ , and get  $\mathbf{w} = \frac{1}{2\lambda} X^T \beta = X^T \alpha$ . Thus, the dual form gives the same solution as usual Ridge Regression, and it's just a different way to come to the same solution.

<https://stats.stackexchange.com/questions/92672/difference-between-primal-dual-and-kernel-ridge-regression>

## Polynomial Regression

### Motivation: nonlinear decision surface

- Based on the sum of products of the variables
- E.g. when the input dimension is  $d=2$ ,

a polynomial function of degree = 2 is:

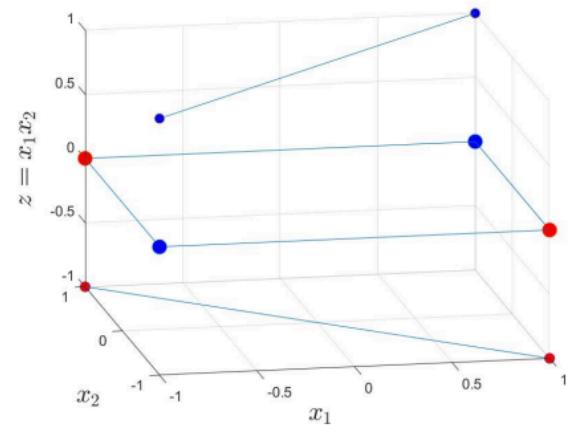
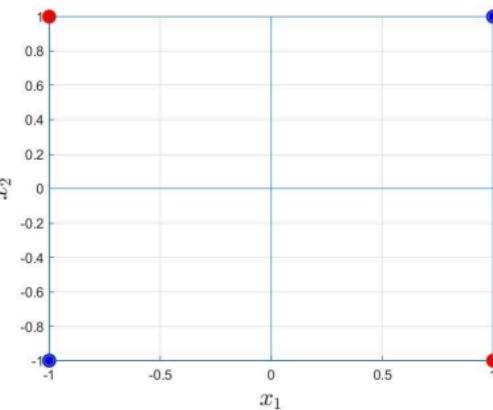
$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2.$$

## Example

### XOR problem

$$\begin{aligned} \mathbf{x}_1 &= [+1 \quad +1]^T & y_1 &= +1 \\ \mathbf{x}_2 &= [-1 \quad +1]^T & y_2 &= -1 \\ \mathbf{x}_3 &= [+1 \quad -1]^T & y_3 &= -1 \\ \mathbf{x}_4 &= [-1 \quad -1]^T & y_4 &= +1 \end{aligned}$$

$$f_{\mathbf{w}}(\mathbf{x}) = x_1 x_2$$



- Must need non linear regression to fit the situation, not possible to split into non linear, not linearly separable
- Pic on left shows how the axis of  $x_1 x_2$  separate the blue and red dots

### Polynomial Expansion

- The linear model  $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$  can be written as

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) &= \mathbf{x}^T \mathbf{w} \\ &= \sum_{i=0}^d x_i w_i, \quad x_0 = 1 \\ &= w_0 + \sum_{i=1}^d x_i w_i. \end{aligned}$$

- By including additional terms involving the products of pairs of components of  $\mathbf{x}$ , we obtain a quadratic model:

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^d w_i x_i + \sum_{i=1}^d \sum_{j=1}^d w_{ij} x_i x_j.$$

2<sup>nd</sup> order:  $f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2$

3<sup>rd</sup> order:  $f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2 + \sum_{i=1}^d \sum_{j=1}^d \sum_{k=1}^d w_{ijk} x_i x_j x_k, \quad d = 2$

$$\begin{aligned} C(n, r) \\ = \frac{(n+r-1)!}{r! (n-1)!} \end{aligned}$$

Ref: Duda, Hart, and Stork, "Pattern Classification", 2001 (Chp.5)

- let  $d$  be the number of feature and  $k$  be the order of polynomial to be generated from the features

$$\text{number of terms in polynomial} = {}^{k+d}C_d = {}^{k+d}C_k$$

Formula:

$$\binom{(d+1)+k-1}{k} \quad \begin{matrix} \text{no. of terms} \\ \uparrow \\ \text{no. chosen} \end{matrix}$$

$$= \binom{d+k}{k} \quad \begin{matrix} k \\ \leftarrow \text{no. chosen} \end{matrix}$$

- 2nd order has 6 parameter/weight
  - $x_1x_2$  is the same as  $x_2x_1$  need to combine
  - $(2+2)C(2) = 6$
- 3rd order has 10 parameter/weight
  - same as 2nd order, + 4
  - $(2+3)C(3) = 10$

## Polynomial Regression



### Generalized Linear Discriminant Function

- In general:

$$f_w(\mathbf{x}) = w_0 + \sum_{i=1}^d w_i x_i + \sum_{i=1}^d \sum_{j=1}^d w_{ij} x_i x_j + \sum_{i=1}^d \sum_{j=1}^d \sum_{k=1}^d w_{ijk} x_i x_j x_k + \dots$$

$$\begin{aligned} & 1 \\ & C(3,1) \\ & C(3,2) \\ & C(3,3) \end{aligned}$$

**Weierstrass Approximation Theorem:** Every continuous function defined on a closed interval  $[a, b]$  can be uniformly approximated as closely as desired by a polynomial function.

- Suppose  $f$  is a continuous real-valued function defined on the real interval  $[a, b]$ .
  - For every  $\varepsilon > 0$ , there exists a polynomial  $p$  such that for all  $x$  in  $[a, b]$ , we have  $|f(x) - p(x)| < \varepsilon$ .
- (Ref: [https://en.wikipedia.org/wiki/Stone%20Weierstrass\\_theorem](https://en.wikipedia.org/wiki/Stone%20Weierstrass_theorem))

### Notes:

- For high dimensional input features (large  $d$  value) and high *polynomial order*, the number of polynomial terms becomes explosive! (i.e., grows exponentially)
- For high dimensional problems, polynomials of order larger than 3 is seldom used.

$$f_{\mathbf{w}}(\mathbf{x}) = \textcolor{blue}{w_0} + \sum_{i=1}^d w_i x_i + \sum_{i=1}^d \sum_{j=1}^d w_{ij} x_i x_j + \sum_{i=1}^d \sum_{j=1}^d \sum_{k=1}^d w_{ijk} x_i x_j x_k + \dots$$

$$\mathbf{f}_{\mathbf{w}}(\mathbf{x}) = \mathbf{P}\mathbf{w} \quad (\text{Note: } \mathbf{P} \triangleq \mathbf{P}(\mathbf{X}) \text{ for symbol simplicity})$$

$$= \begin{bmatrix} \mathbf{p}_1^T \mathbf{w} \\ \vdots \\ \mathbf{p}_m^T \mathbf{w} \end{bmatrix}$$

where  $\mathbf{p}_l^T \mathbf{w} = [1, x_{l,1}, \dots, x_{l,d}, \dots, x_{l,i}x_{l,j}, \dots, x_{l,i}x_{l,j}x_{l,k}, \dots]$

$l = 1, \dots, m$ ;  $d$  denotes the dimension of input features;  $m$  denotes the number of samples

$$\begin{bmatrix} \textcolor{blue}{w_0} \\ w_1 \\ \vdots \\ w_d \\ \vdots \\ w_{ij} \\ \vdots \\ w_{ijk} \end{bmatrix}$$

# Example 3

Training set  $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^m$

2<sup>nd</sup> order polynomial model

$$\{x_1 = 0, x_2 = 0\} \rightarrow \{y = -1\}$$

$$\{x_1 = 1, x_2 = 1\} \rightarrow \{y = -1\}$$

$$\{x_1 = 1, x_2 = 0\} \rightarrow \{y = +1\}$$

$$\{x_1 = 0, x_2 = 1\} \rightarrow \{y = +1\}$$

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2$$

$$= [1 \ x_1 \ x_2 \ x_1 x_2 \ x_1^2 \ x_2^2] \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_{12} \\ w_{11} \\ w_{22} \end{bmatrix}$$

Python  
demo 3

Stack the 4 training samples as a matrix

$$\mathbf{P} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & x_{1,1}x_{1,2} & x_{1,1}^2 & x_{1,2}^2 \\ 1 & x_{2,1} & x_{2,2} & x_{2,1}x_{2,2} & x_{2,1}^2 & x_{2,2}^2 \\ 1 & x_{3,1} & x_{3,2} & x_{3,1}x_{3,2} & x_{3,1}^2 & x_{3,2}^2 \\ 1 & x_{4,1} & x_{4,2} & x_{4,1}x_{4,2} & x_{4,1}^2 & x_{4,2}^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

- Number of rows in P = number of samples
- Number of columns in P = number of terms in the polynomial expansion of with d feature up till order O
- Sub in the value of x1 and x2, and solve the linear system  $\mathbf{y} = \mathbf{P}\mathbf{w}$

# Example 3 (cont'd)

## Training set

2<sup>nd</sup> order polynomial model

$$\mathbf{P} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & x_{1,1}x_{1,2} & x_{1,1}^2 & x_{1,2}^2 \\ 1 & x_{2,1} & x_{2,2} & x_{2,1}x_{2,2} & x_{2,1}^2 & x_{2,2}^2 \\ 1 & x_{3,1} & x_{3,2} & x_{3,1}x_{3,2} & x_{3,1}^2 & x_{3,2}^2 \\ 1 & x_{4,1} & x_{4,2} & x_{4,1}x_{4,2} & x_{4,1}^2 & x_{4,2}^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\hat{\mathbf{w}} = \mathbf{P}^T (\mathbf{P} \mathbf{P}^T)^{-1} \mathbf{y}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 6 & 3 & 3 \\ 1 & 3 & 3 & 1 \\ 1 & 3 & 1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -4 \\ 1 \\ 1 \end{bmatrix}$$

Python  
demo 3

# Example 3 (cont'd) Prediction

## Test set

- Test point 1:  $\{x_1 = 0.1, x_2 = 0.1\} \rightarrow \{y = \text{class } -1 \text{ or } +1?\}$
- Test point 2:  $\{x_1 = 0.9, x_2 = 0.9\} \rightarrow \{y = \text{class } -1 \text{ or } +1?\}$
- Test point 3:  $\{x_1 = 0.1, x_2 = 0.9\} \rightarrow \{y = \text{class } -1 \text{ or } +1?\}$
- Test point 4:  $\{x_1 = 0.9, x_2 = 0.1\} \rightarrow \{y = \text{class } -1 \text{ or } +1?\}$

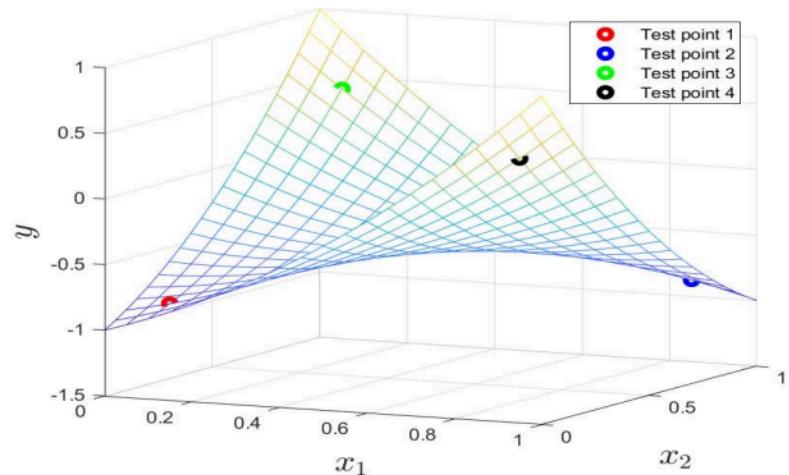
$$\hat{\mathbf{y}} = \mathbf{P}_{new} \hat{\mathbf{w}}$$

$$= \begin{bmatrix} 1 & 0.1 & 0.1 & 0.01 & 0.01 & 0.01 \\ 1 & 0.9 & 0.9 & 0.81 & 0.81 & 0.81 \\ 1 & 0.1 & 0.9 & 0.09 & 0.01 & 0.81 \\ 1 & 0.9 & 0.1 & 0.09 & 0.81 & 0.01 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 1 \\ -4 \\ 1 \\ 1 \end{bmatrix}$$

$$[1 \ x_1 \ x_2 \ x_1x_2 \ x_1^2 \ x_2^2]$$

$$\hat{f}_w^c(\mathbf{P}(\mathbf{X}_{new})) = \text{sign}(\hat{\mathbf{y}}) = \text{sign}\left(\begin{bmatrix} -0.82 \\ -0.82 \\ 0.46 \\ 0.46 \end{bmatrix}\right)$$

$$= \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix} \begin{array}{l} \text{-----} \rightarrow \text{Class } -1 \\ \text{-----} \rightarrow \text{Class } -1 \\ \text{-----} \rightarrow \text{Class } +1 \\ \text{-----} \rightarrow \text{Class } +1 \end{array}$$



- non linear decision surface, each value of  $x_1$  and  $x_2$  mapped onto the surface, then compare to threshold value 0 to get which class it is

```
#EE2211 Lecture 6 Demo 3 Polynomial regression
import numpy as np
from numpy.linalg import inv
from numpy.linalg import matrix_rank
from sklearn.preprocessing import PolynomialFeatures
X = np.array([[0, 0], [1, 1], [1, 0], [0, 1]])
y = np.array([[-1], [-1], [1], [1]])
## Generate polynomial features
order = 2
poly = PolynomialFeatures(order)
print(poly)
P = poly.fit_transform(X)
print("matrix P")
print(P)

print("*****")
#print(matrix_rank(P))
#PY = np.vstack((P.T, y.T))
#print(matrix_rank(PY.T))
```

```
PolynomialFeatures()
matrix P
[[1. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 1. 0. 0.]
 [1. 0. 1. 0. 0. 1.]]
*****
```

- input to polynomial fit transform doesn't need to pad 1, it automatically does it

```

## dual solution m < d (without ridge)
w_dual = P.T @ inv(P @ P.T) @ y
print("Under-determined system")
print("Unique constrained solution, no ridge")
print(w_dual)

print("*****")
print("Approximation with dual ridge regression")
print(P.shape)
reg_L2 = 0.0001*np.identity(P.shape[0]) #number of rows of P = Dual I
print(reg_L2)
w_dual_ridge = P.T @ (inv(P @ P.T + reg_L2)) @ y
print(w_dual_ridge)

print("*****")
## primal ridge
print("Approximation with primal ridge regression")
print(P.shape)
reg_L = 0.0001*np.identity(P.shape[1]) #number of columns of P = Primal I
print(reg_L)
w_primal_ridge = inv(P.T @ P + reg_L) @ P.T @ y
print(w_primal_ridge)

```

```

Under-determined system
Unique constrained solution, no ridge
[[-1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [-4.]
 [ 1.]]
*****
Approximation with dual ridge regression
(4, 6)
[[0.0001 0.    0.    0.    0.    ]
 [0.    0.0001 0.    0.    0.    ]
 [0.    0.    0.0001 0.    0.    ]
 [0.    0.    0.    0.0001 0.    ]
 [0.    0.    0.    0.    0.0001]]
[[-0.9993004 ]
 [ 0.99940033]
 [ 0.99940033]
 [ 0.99940033]
 [-3.99790115]
 [ 0.99940033]]
*****
Approximation with primal ridge regression
(4, 6)
[[0.0001 0.    0.    0.    0.    0.    ]
 [0.    0.0001 0.    0.    0.    0.    ]
 [0.    0.    0.0001 0.    0.    0.    ]
 [0.    0.    0.    0.0001 0.    0.    ]
 [0.    0.    0.    0.    0.0001 0.    ]
 [0.    0.    0.    0.    0.    0.0001]]
[[-0.9993004 ]
 [ 0.99940033]
 [ 0.99940033]
 [ 0.99940033]
 [-3.99790115]
 [ 0.99940033]]

```

```

#EE2211 Lecture 6 Demo 3 Testing/prediction
import numpy as np
from numpy.linalg import inv
from numpy.linalg import matrix_rank
from sklearn.preprocessing import PolynomialFeatures
X = np.array([[0, 0], [1, 1], [1, 0], [0, 1]])
y = np.array([-1, -1, 1, 1])
## Generate polynomial features
order = 2
poly = PolynomialFeatures(order)
print(poly)
P = poly.fit_transform(X)
print("matrix P")
print(P)
print("Under-determined system")
#print(matrix_rank(P))
#PY = np.vstack((P.T, y.T))
#print(matrix_rank(PY.T))

## dual solution m < d (without ridge)
w_dual = P.T @ inv(P @ P.T) @ y
print("Unique constrained solution, no ridge")
print(w_dual)

#testing
print("Prediction")
Xnew= np.array([ [0.1, 0.1], [0.9, 0.9], [0.1, 0.9], [0.9, 0.1]])
Pnew = poly.fit_transform(Xnew)
Ynew=Pnew@w_dual
print(Ynew)
print(np.sign(Ynew))

```

```

PolynomialFeatures()
matrix P
[[1.  0.  0.  0.  0.  0.]
 [1.  1.  1.  1.  1.  1.]
 [1.  1.  0.  1.  0.  0.]
 [1.  0.  1.  0.  0.  1.]]
Under-determined system
Unique constrained solution, no ridge
[[-1.]
 [ 1.]
 [ 1.]
 [ 1.]
 [-4.]
 [ 1.]]
Prediction
[[ -0.82]
 [ -0.82]
 [  0.46]
 [  0.46]]
[[-1.]
 [-1.]
 [ 1.]
 [ 1.]]

```

## Ridge Regression in Primal Form ( $m > d$ )

For  $\lambda > 0$ ,

$$\text{Learning: } \hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

$$\text{Prediction: } \hat{f}_{\mathbf{w}}(\mathbf{P}(\mathbf{X}_{new})) = \mathbf{P}_{new} \hat{\mathbf{w}}$$

## Ridge Regression in Dual Form ( $m < d$ )

For  $\lambda > 0$ ,

$$\text{Learning: } \hat{\mathbf{w}} = \mathbf{P}^T (\mathbf{P} \mathbf{P}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

$$\text{Prediction: } \hat{f}_{\mathbf{w}}(\mathbf{P}(\mathbf{X}_{new})) = \mathbf{P}_{new} \hat{\mathbf{w}}$$

Note: Change  $\mathbf{X}$  to  $\mathbf{P}$  with reference to slides 15/16;  $m$  &  $d$  refers to the size of  $\mathbf{P}$  (not  $\mathbf{X}$ )

## For Regression Applications

- Learn **continuous** valued  $y$  using either primal form or dual form
- Prediction:  $\hat{f}_{\mathbf{w}}(\mathbf{P}(\mathbf{X}_{new})) = \mathbf{P}_{new} \hat{\mathbf{w}}$

## For Classification Applications

- Learn **discrete** valued  $y$  ( $y \in \{-1, +1\}$ ) or  $\mathbf{Y}$  (one-hot) using either primal form or dual form
- Binary Prediction:  $\hat{f}_{\mathbf{w}}^c(\mathbf{P}(\mathbf{X}_{new})) = \text{sign}(\mathbf{P}_{new} \hat{\mathbf{w}})$
- Multi-Category Prediction:  $\hat{f}_{\mathbf{w}}^c(\mathbf{P}(\mathbf{X}_{new})) = \arg \max_{k=1,\dots,C} (\mathbf{P}_{new} \hat{\mathbf{W}}(:, k))$

## In Class Quiz

Suppose we would like to build up one full third-order polynomial model for only one input feature and single output, the polynomial model has three parameters to learn.

Jin Yueming  
15

True



False



- One variable (feature) 3rd order means  $y = c + w_1x + w_2x^2 + w_3x^3$
- Four parameters/weights

If there are four data samples with two input features each, the full second-order polynomial model is an over-determined system.

Jin Yueming  
14

True



False



- $f_w(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_{12}x_1x_2 + w_{11}x_1^2 + w_{22}x_2^2$
- 6 columns, 4 rows, under determined

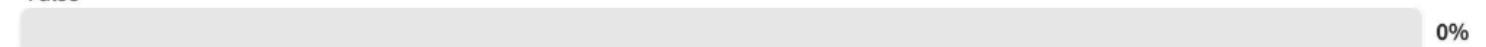
The polynomial model can be used to solve problems with nonlinear decision boundary.

Jin Yueming

True



False



## Lec7: Over-fitting, bias/variance trade-off

- Linear is special case of polynomial => use “**P**” instead of “**X**” from now on
- Training/Learning (primal) on training set

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

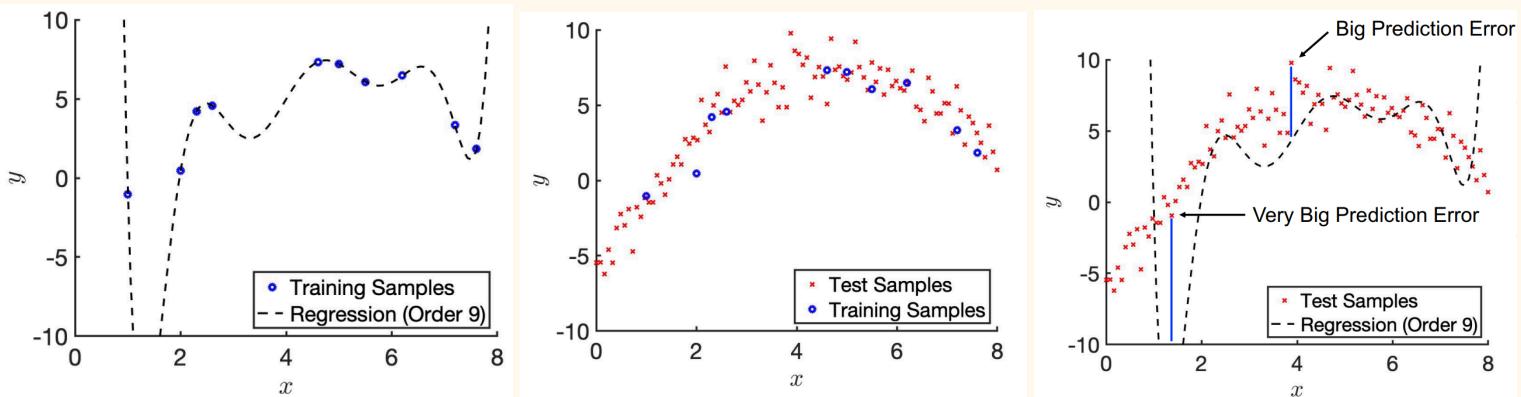
- Prediction/Testing/Evaluation on test set

$$\hat{\mathbf{y}}_{test} = \mathbf{P}_{test} \hat{\mathbf{w}}$$

- There should be **zero** overlap between training & test sets
- Important goal of regression: prediction on **new unseen data**, i.e., test set
- Why is test set important for evaluation?
  - Ideally training set >> test set

## Overfitting, Underfitting & Model Complexity

### Overfitting



- Regression fits the training data well but not the test data
- has very big prediction error when using test data (MSE is big)
- Because training data set is small while trying to fit a very complicated model of order 9, a lot parameter
- Training good, test bad  $\Rightarrow$  over fitting

- **Overfitting** occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly

- **Reason 1**

- Model is too complex for the data
- Previous example: Fit order 9 polynomial to 10 data points

- **Reason 2**

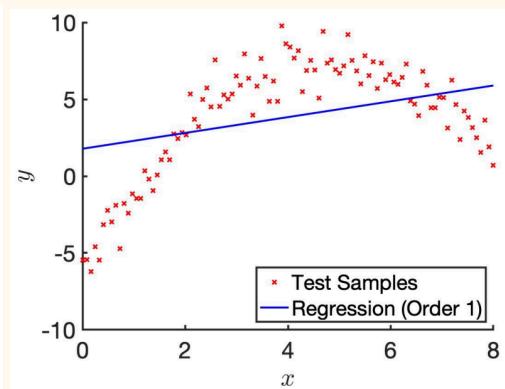
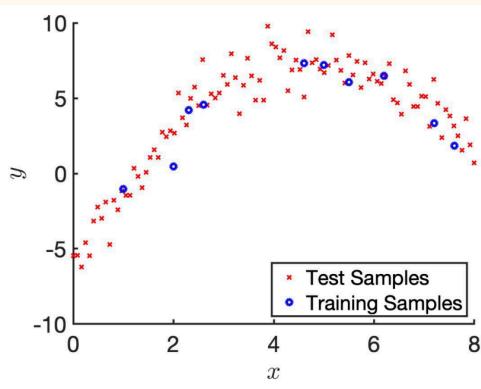
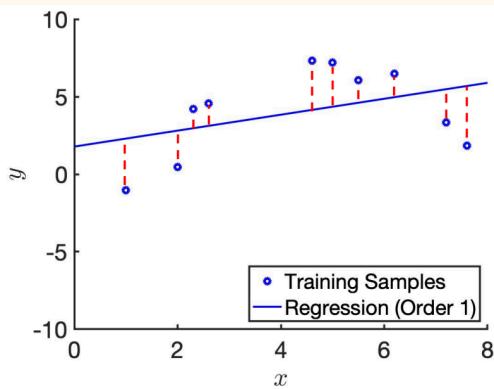
- Too many features but number of training samples too small
- Even linear model can overfit, e.g., linear model with 9 input features (i.e.,  $w$  is 10-D) and 10 data points in training set => data might not be enough to estimate 10 unknowns well

- **Solutions**

- Use **simpler models** (e.g., lower order polynomial)
- Use **regularization** (see next part of lecture)

- Can use less features also

## Underfitting



- Regression is bad for the training data and the test data => under fitting
- Means model is too simple

- **Underfitting** is the inability of trained model to predict the targets in the training set

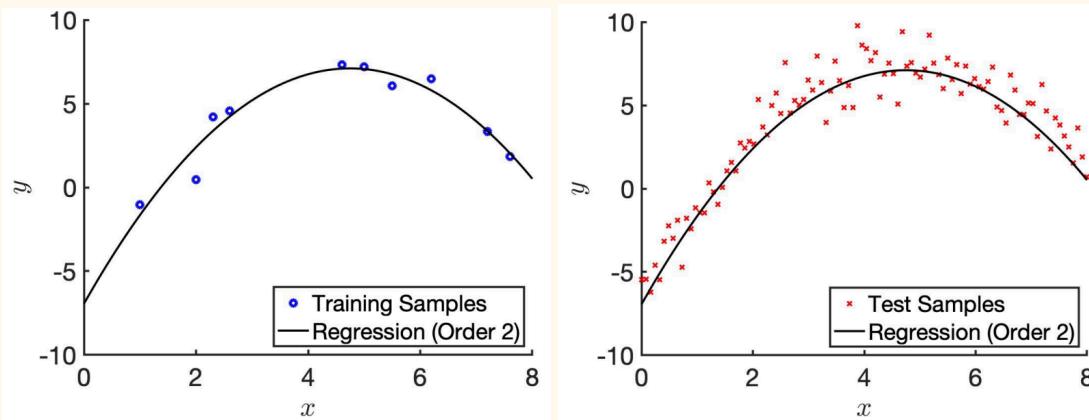
- **Reason 1**

- Model is too simple for the data
- Previous example: Fit order 1 polynomial to 10 data points that came from an order 2 polynomial
- **Solution:** Try more complex model

- **Reason 2**

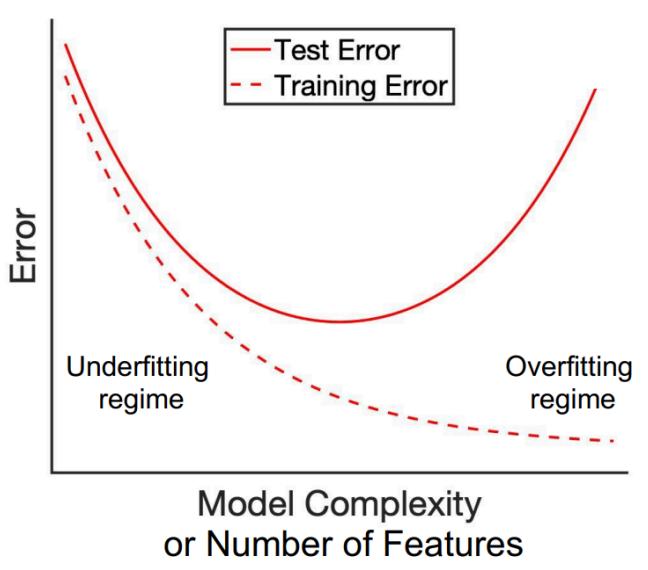
- Features are not informative enough
- **Solution:** Try to develop more informative features
  - Need more features, or have features that doesn't tell much
  - Be better at feature selection

## Good Fit



- Regression does not pass through all the training data, the MSE is actually larger than the 9 order overfitting case
- But this is due to quadratic only having 3 parameters, so the regression is also still good
- The MSE to the test data is also very good

## Overfitting / Underfitting Schematic



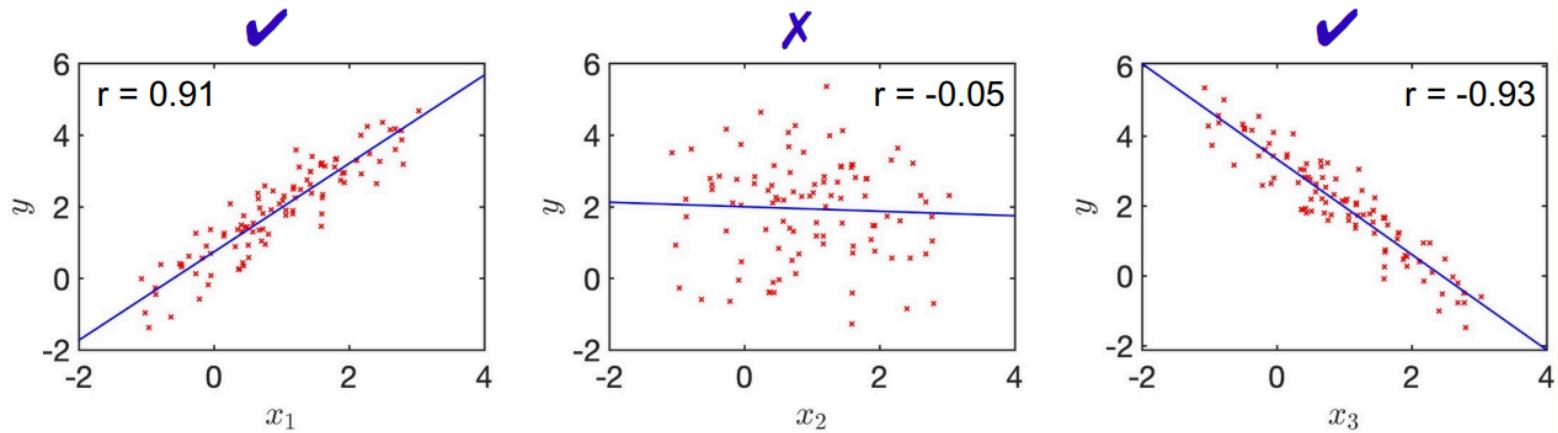
- When both training and test error are high, is when model complexity is too low or number of features is not enough  $\Rightarrow$  underfitting
- When training error is low but test error is high, is when model complexity is too high or number of features is too much  $\Rightarrow$  over fitting
  - if the model is very complicated and has a lot of parameter, the parameter can be chosen to fit the training data very well
- note the test error has a u-shape indicating there is a sweet spot between under and over fitting

## Feature Selection

- Less features might reduce overfitting
    - Want to discard useless features & keep good features, so perform feature selection
  - Feature selection procedure
    - Step 1: feature selection in **training** set
    - Step 2: fit model using selected features in **training** set
    - Step 3: evaluate trained model using **test** set
  - Very common mistake
    - Feature selection with test set (or full dataset) leads to inflated performance
    - Do not perform feature selection with test data
- Do not train with test set
  - Do not feature select from test set
  - Both will lead to information leakage
  - If need to compare two models, give both the same training set and test with the same test set

## Selecting Features With Pearson's r

- Given features  $x$ , we want to predict target  $y$
- Assume  $x$  &  $y$  both continuous
- Compute Pearson's correlation coefficient between each feature & target  $y$  in the training set
  - Pearson's correlation  $r$  measures linear relationship between two variables



- Two options
  - Option 1: Pick K features with largest absolute correlations
  - Option 2: Pick all features with absolute correlations  $> C$
  - K & C are “magic” numbers set by the ML practitioner
- Other metrics besides Pearson's correlation are possible
  - choose the highest magnitude for the strongest correlation w

## Regularisation

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- **Motivation 1:** Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- **Motivation 2:** Reduce overfitting
- For example, in previous lecture, we added  $\lambda \mathbf{w}^T \mathbf{w}$ :

$$\operatorname{argmin}_{\mathbf{w}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to  $\mathbf{w}$ , primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}^T \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^T \mathbf{y}$$

- For  $\lambda > 0$ , matrix becomes invertible (**Motivation 1**)
- $\hat{\mathbf{w}}$  might also perform better in test set, i.e., reduces overfitting (**Motivation 2**) – will show example later
  - ill posed problem means trying to solve under determine system
  - with  $\lambda$ ,  $\mathbf{w}$  reduces fitting by making sure features that does not contribute much have low weights, so can fix over fitting
- Consider minimization from previous slide

$$\operatorname{argmin}_{\mathbf{w}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Cost function quantifying data  
fitting error in training set

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

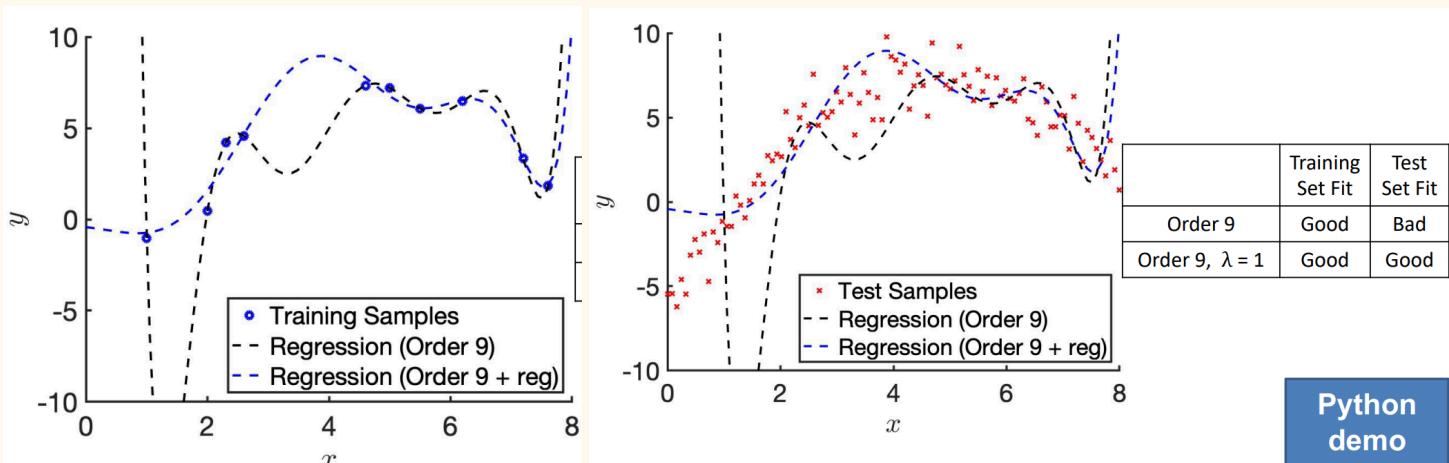
- $\mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \dots + w_d^2$  L2 - Regularization

- Encourage  $w_0, \dots, w_d$  to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}} \text{Data-Loss}(\mathbf{w}) + \lambda \text{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters  $\mathbf{w}$ : smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models
  - Because its squared, its called L2 - Regularization
  - Bigger  $\lambda$  means regularisation dominates, forces model to be less complex at the cost of increasing MSE to training data

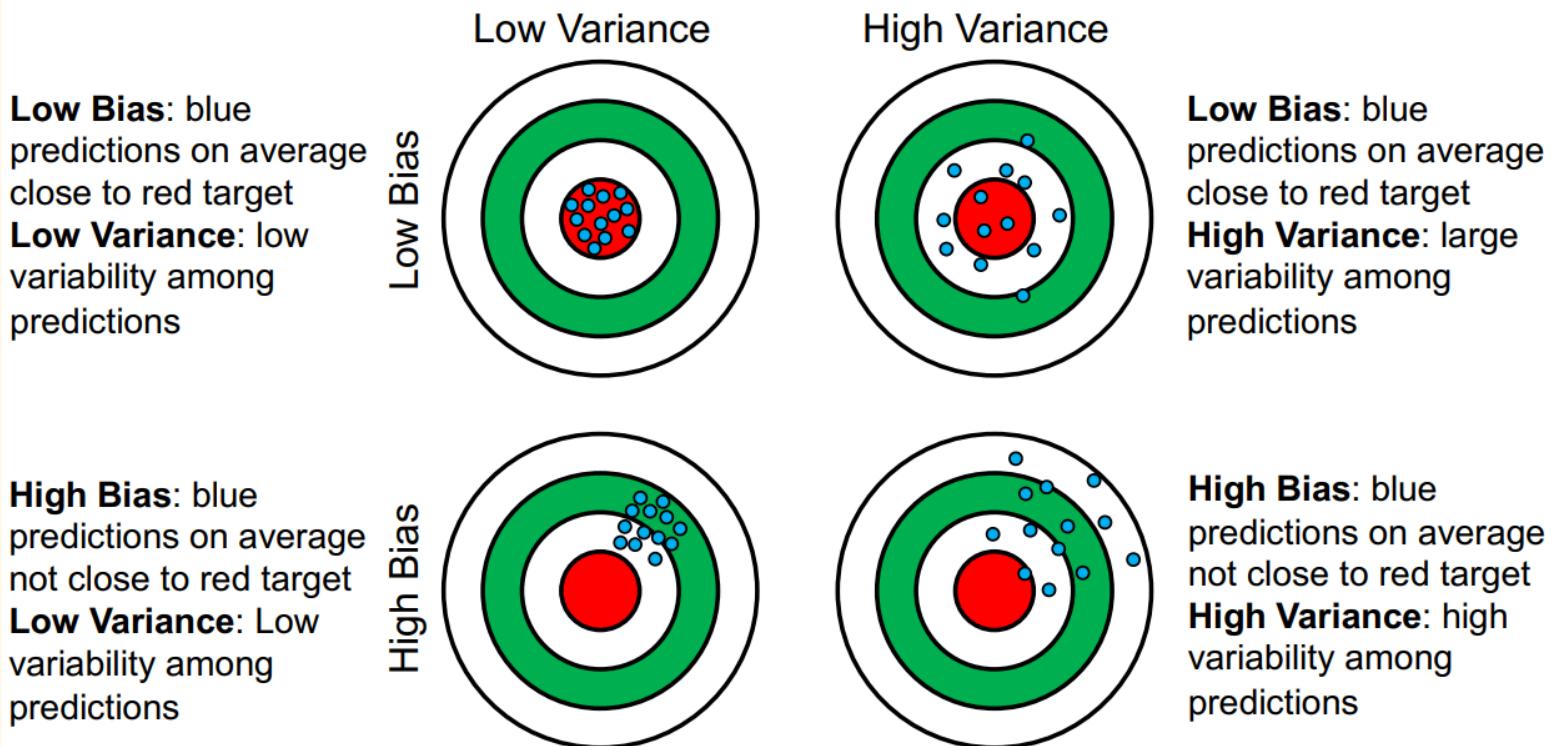
### Regularisation Example



- Can see that with Ridge regression, the blue curve does not fit the training data as well anymore. But it fit the overall data much better

## Bias-Variance Trade-off

- Suppose we are trying to predict red target below:

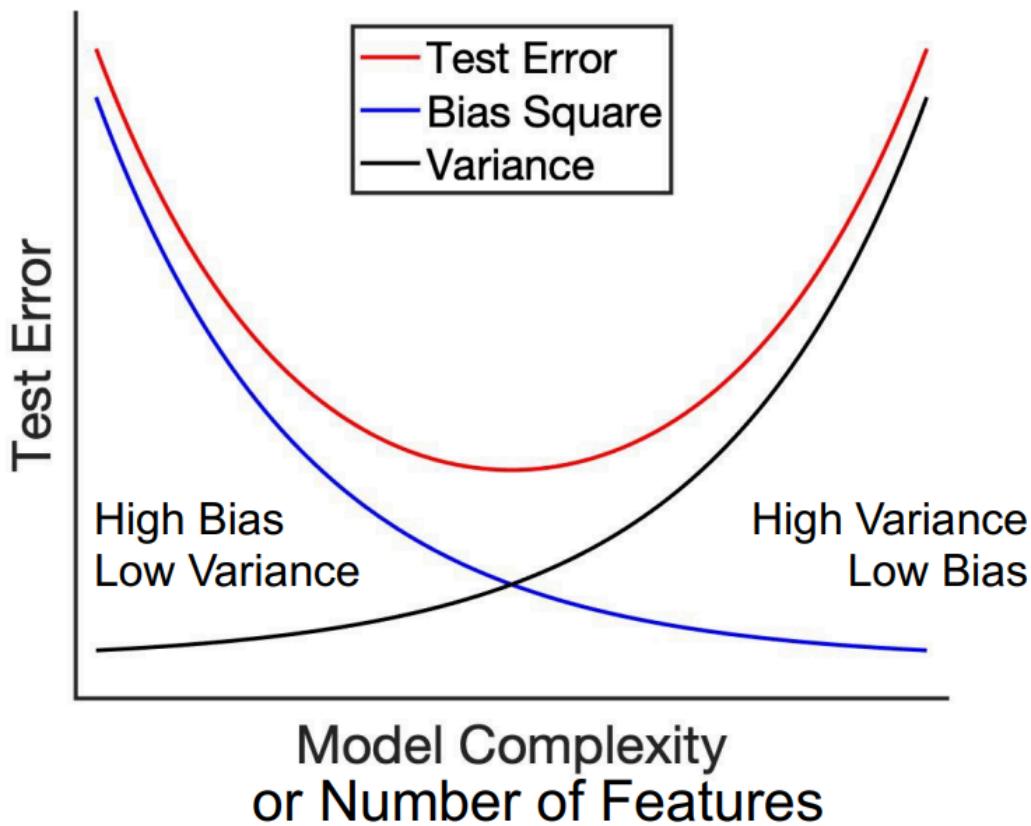


- Both bias and variance contribute to prediction error

## Bias-Variance Decomposition Theorem

- **Test error** = Bias Squared + Variance + Irreducible Noise
  - Mathematical details in optional uploaded material (won't be tested)
- “**Variance**” refers to variability of prediction models across different training sets
  - In previous example, every time the training set of 10 samples changes, the trained model changes
  - “Variance” quantifies variability across trained models
- “**Bias**” refers to how well an average prediction model will perform
  - In previous example, every time the training set of 10 samples changes, the trained model changes
  - If we average the trained models, how well will this average trained model perform?
- “**Irreducible Noise**” reflects the fact that even if we are perfect modelers, it might not be possible to predict target  $y$  with 100% accuracy from feature(s)  $x$

- Test error = Bias Squared + Variance + Irreducible Noise

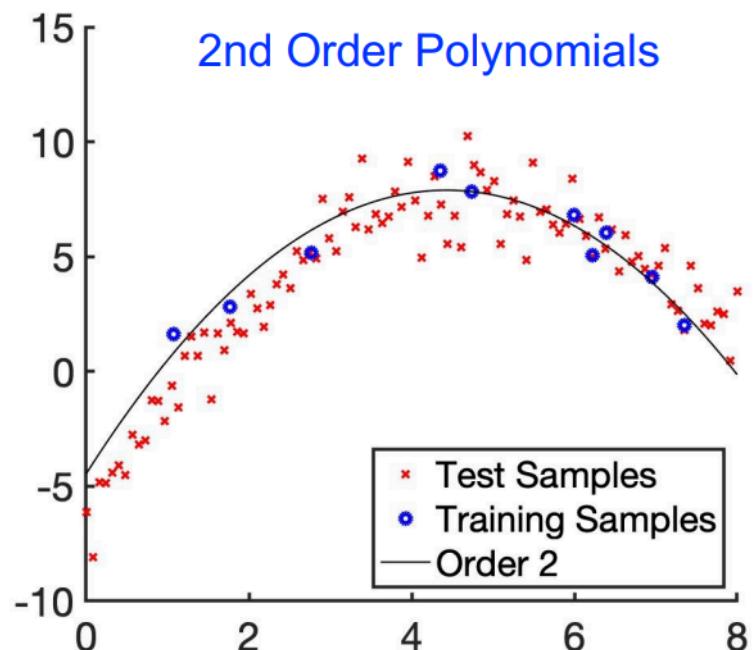
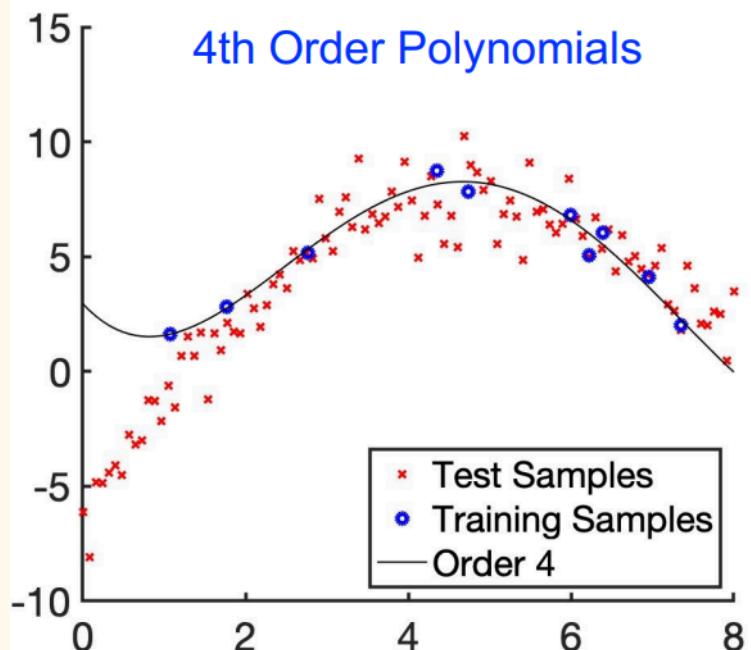


- test error with u shape can be explained here
- With simple models, predictions has high bias but low variance
- With complex models, predictions has low bias but high variance
- This is the bias variance trade off

### Example

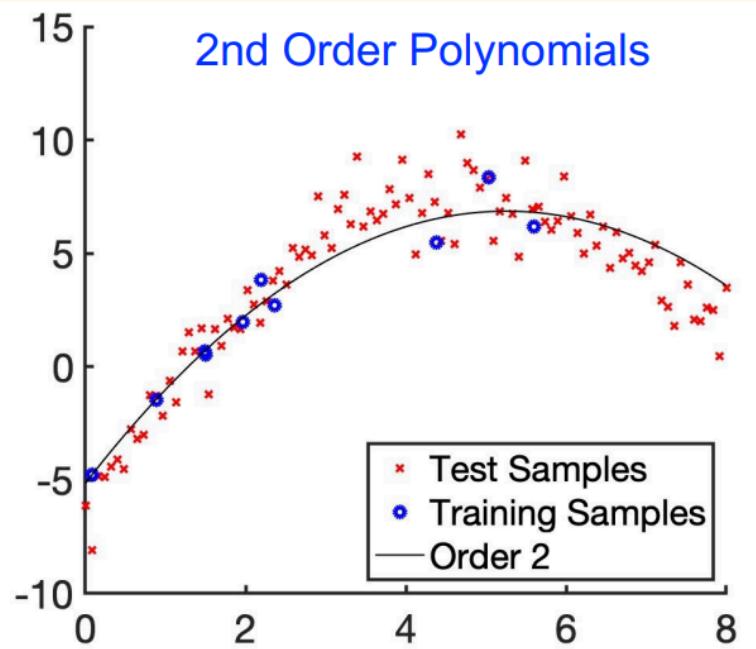
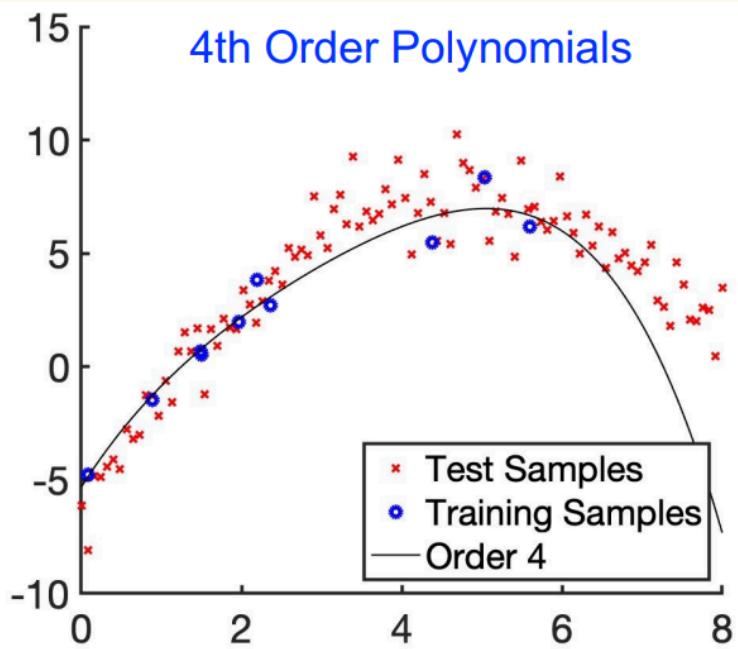
- Simulate data from order 2 polynomial (+ noise)
- Randomly sample 10 training samples each time
- Fit with order 2 polynomial
- Fit with order 4 polynomial
- The data points are the same for all
- The same two models is trained on the same data set each time the experiment is repeated
  - each time the sample is randomly chosen
- Note the variance the 4th order one has

## Test 1



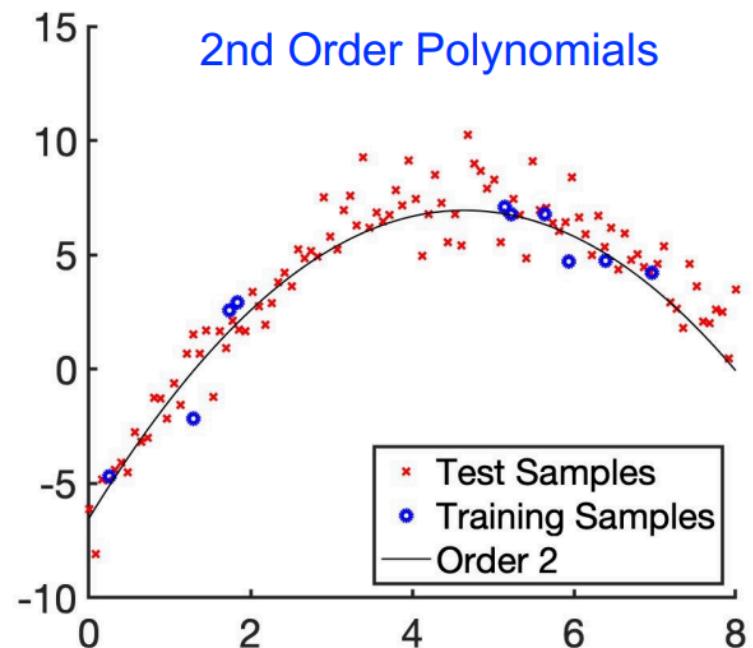
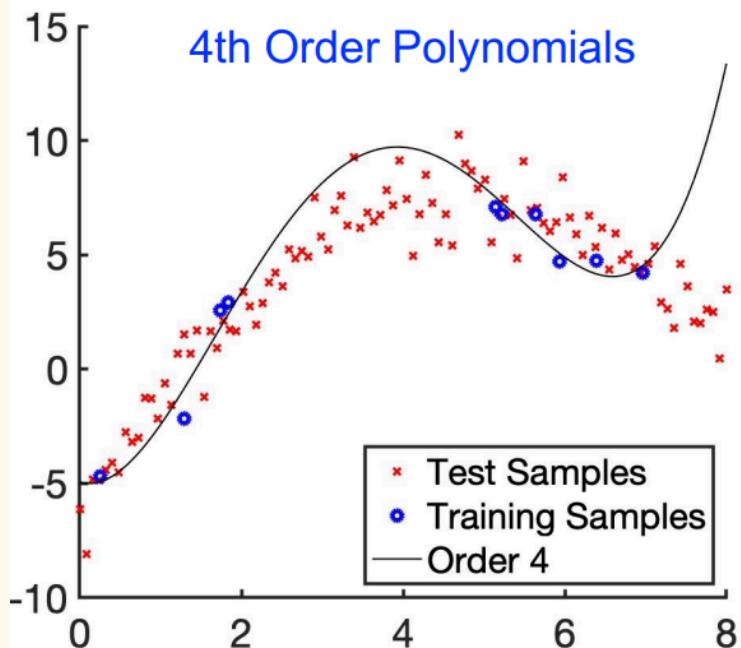
- 4th order has big error on LHS
- 2nd order fit training and test well
- Both fit training quite well

## Test 2



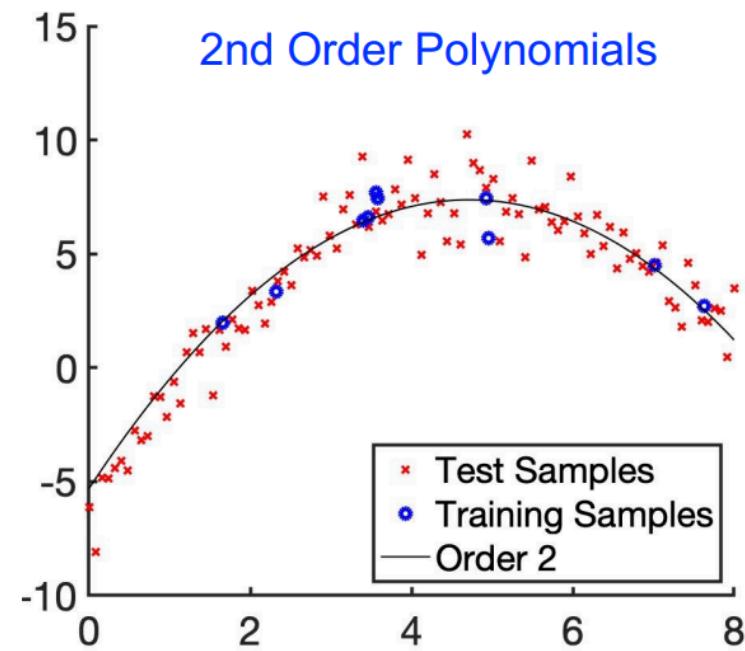
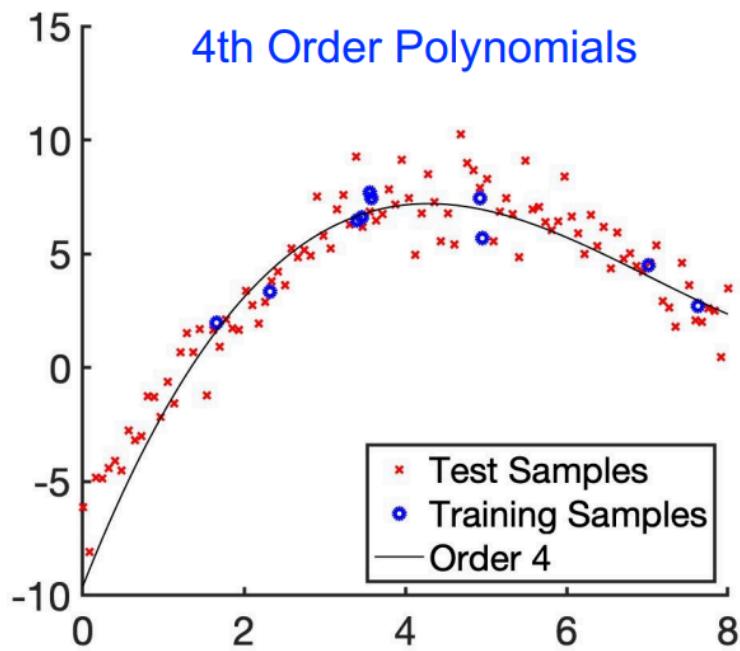
- 4th order has big error on RHS
- 2nd order fit training and test well
- Both fit training quite well

### Test 3



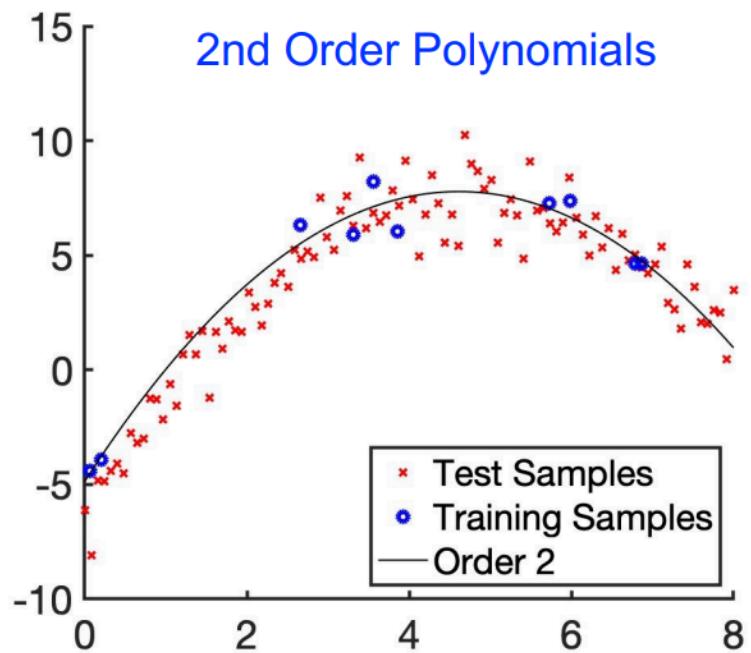
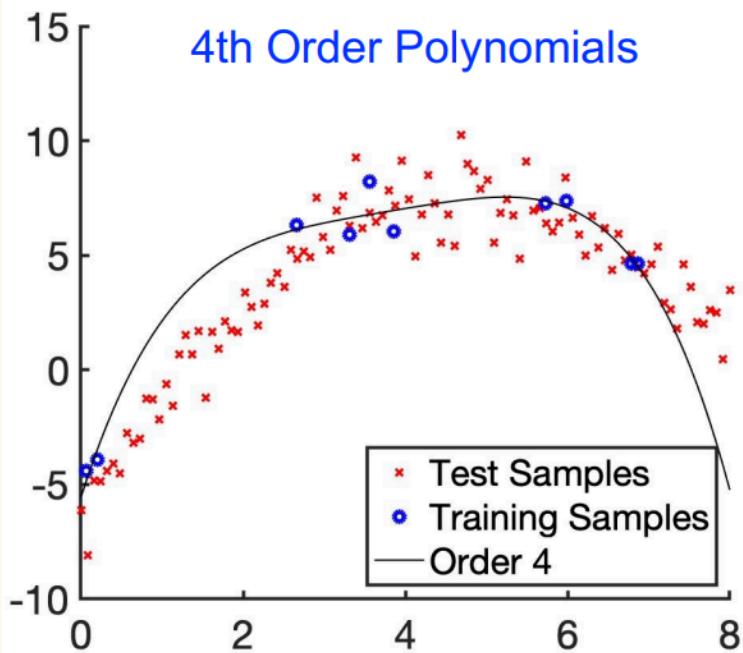
- 4th order has big error on RHS
- 2nd order fit training and test well
- Both fit training quite well

### Test 4



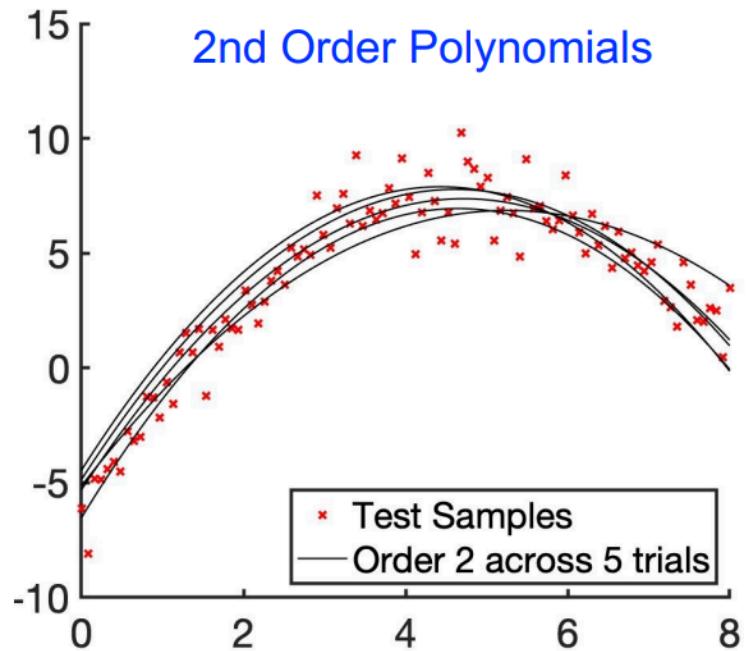
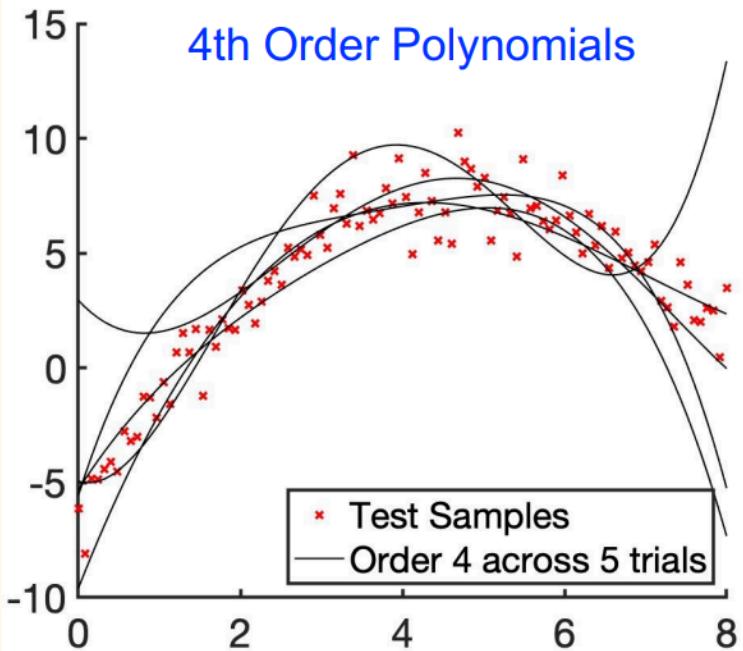
- Both fit training and test quite well

## Test 5



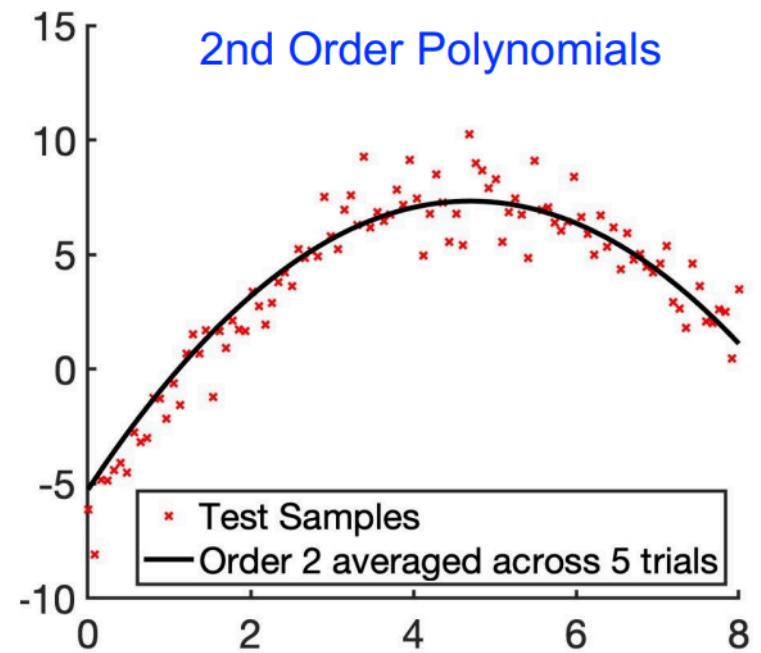
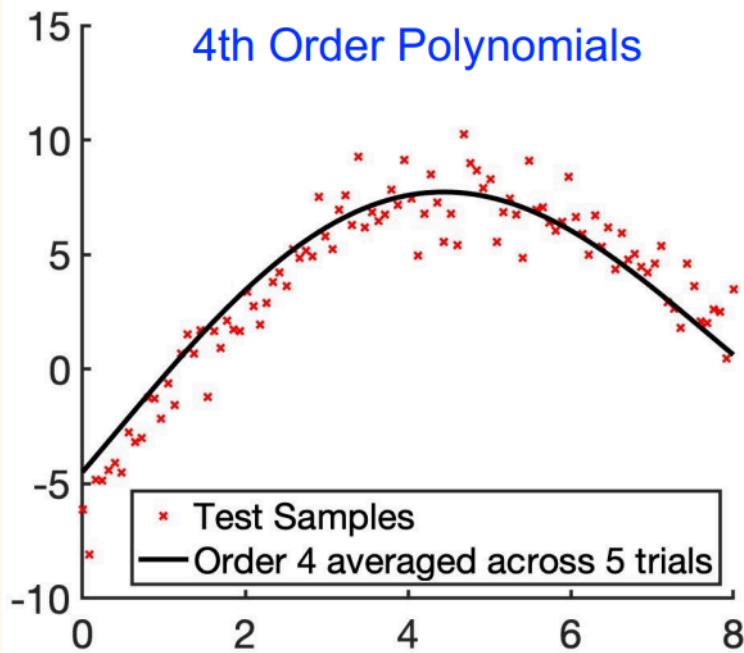
- 4th order has big error on LHS
- 2nd order fit training and test well
- Both fit training quite well

## All together



- 4th varies a lot, high variance
- 2nd all quite similar, low variance

## Average



- Both averages follows the test data well
- Both has low bias

## Conclusion

- Fit with order 2 polynomial: low variance, low bias
- Fit with order 4 polynomial: high variance, low bias
- 2nd order performs better and has lower test error

Order 2  
Achieves Lower  
Test Error

## In Class Quiz

An algorithm achieves 10% error in the training set and 50% error in the test set. This situation is described as

26

Overfitting

81%

Underfitting

19%

## Lec8: Optimization, Gradient descent

### Loss Function

#### Review

- Supervised learning: given feature(s)  $x$ , we want to predict target  $y$
- Most supervised learning algorithms can be formulated as the following optimization problem

$$\operatorname{argmin}_{\mathbf{w}} \mathbf{Data-Loss}(\mathbf{w}) + \lambda \mathbf{Regularization}(\mathbf{w})$$

- **Data-Loss( $\mathbf{w}$ )** quantifies fitting error to training set given parameters  $\mathbf{w}$ : smaller error => better fit to training data
- **Regularization( $\mathbf{w}$ )** penalizes more complex models
- For example, in the case of polynomial regression (previous lectures):

$$\operatorname{argmin}_{\mathbf{w}} \underbrace{(\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y})}_{\mathbf{Data-Loss}(\mathbf{w})} + \underbrace{\lambda \mathbf{w}^T \mathbf{w}}_{\mathbf{Reg}(\mathbf{w})}$$

- review: finding a  $\mathbf{w}$  that minimises data loss and regularisation term, minimises the cost function
  - For polynomial regression (previous lectures)

$$\operatorname{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname{argmin}_{\mathbf{w}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

$$= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m (\mathbf{p}_i^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

$\mathbf{p}_i^T \mathbf{w}$  is prediction of  $i$ -th training sample

$y_i$  is target of  $i$ -th training sample

-

- Linear regression with 2 features,  $\mathbf{p}_i = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}_i$ 
  - 1 ← Bias/Offset
  - $x_1$  ← Feature 1 of i-th sample
  - $x_2$  ← Feature 2 of i-th sample

- Quadratic regression with 1 feature,  $\mathbf{p}_i = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}_i$ 
  - 1 ← Bias/Offset
  - $x$  ←  $x$  is feature of i-th sample

## Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\begin{aligned}\operatorname{argmin}_{\mathbf{w}} C(\mathbf{w}) &= \operatorname{argmin}_{\mathbf{w}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w} \\ &= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m (\mathbf{p}_i^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}\end{aligned}$$

- Let  $f(\mathbf{x}_i, \mathbf{w})$  be the prediction of target  $y_i$  from features  $\mathbf{x}_i$  for  $i$ -th training sample. For example, suppose  $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$ , then above becomes

$$\operatorname{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

- Let  $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$  be the penalty for predicting  $f(\mathbf{x}_i, \mathbf{w})$  when true value is  $y_i$ . For example, suppose  $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ , then above becomes

$$\operatorname{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- From previous slide

$$\operatorname{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\operatorname{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^m L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

Cost Function →  $\mathbf{w}$   
 Loss Function →  $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$   
 Learning Model →  $f(\mathbf{x}_i, \mathbf{w})$   
 Regularization →  $R(\mathbf{w})$

- **Learning model**  $f$  reflects our belief about the relationship between the features  $\mathbf{x}_i$  & target  $y_i$
- **Loss function**  $L$  is the penalty for predicting  $f(\mathbf{x}_i, \mathbf{w})$  when the true value is  $y_i$
- **Regularization**  $R$  encourages less complex models
- **Cost function**  $C$  is the final optimization criterion we want to minimize
- **Optimization routine** to find solution to cost function
  - General expression with function that maps input to output with a given weight
  - Make it more general by using the  $L$  function for the penalty of being incorrect
  - Make it more general by using  $R$  for regularisation
  - Learning model,  $f$ , are the ridge, linear or polynomial regression
    - it is what we think the nature of the relationship is
  - Above general expression is the building blocks of ML algorithms, different algorithm uses different building blocks

## Gradient Descent

### Motivation for Gradient Descent

- Different learning function  $f$ , loss function  $L$  & regularization  $R$  give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal  $\mathbf{w}$  can be written with the following “closed-form” formula (primal solution):

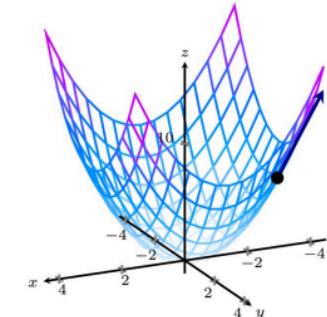
$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- For other learning function  $f$ , loss function  $L$  & regularization  $R$ , optimizing  $C(\mathbf{w})$  might not be so easy
  - Usually have to estimate  $\mathbf{w}$  iteratively with some algorithm
  - Optimization workhorse for modern machine learning is gradient descent
- 
- In real application, or some other model, not possible to have closed form formula (no left and right inverse, only for linear regression), need to do iteratively and numerically
  - Calculating gradient is easy but solving for minimum is hard
  - Use gradient descent to do

## Gradient Descent Algorithm

- Suppose we want to minimize  $C(\mathbf{w})$  with respect to  $\mathbf{w} = [w_1, \dots, w_d]^T$

- Gradient  $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$



- $\nabla_{\mathbf{w}} C(\mathbf{w})$  is vector & function of  $\mathbf{w}$
- $\nabla_{\mathbf{w}} C(\mathbf{w})$  is direction at  $\mathbf{w}$  where  $C$  is increasing most rapidly, so  $-\nabla_{\mathbf{w}} C(\mathbf{w})$  is direction at  $\mathbf{w}$  where  $C$  is decreasing most rapidly

- Gradient Descent:

Initialize  $\mathbf{w}_0$  and learning rate  $\eta$ ;

**while** true **do**

Compute  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$   
**if** converge **then**  
| **return**  $\mathbf{w}_{k+1}$   
**end**

**end**

According to multi-variable calculus, if eta is not too big, then  $C(\mathbf{w}_{k+1}) < C(\mathbf{w}_k) \Rightarrow$  we get better  $\mathbf{w}$  after each iteration

- $C$  is a scalar function of vector  $\mathbf{w}$
- Value of  $\nabla_{\mathbf{w}} C(\mathbf{w})$  is the derivative of  $C$  wrt  $\mathbf{w}$ , at any point, the gradient points to the direction where  $C$  is increasing the fastest
- The -ve of the gradient points to the direction where  $C$  is decreasing the fastest
  - this is from directional derivative

	<i>Directional derivative of a surface f in the direction of unit vector u, (<math> u =1</math>) else <math>g = (\frac{1}{ u }u)</math>, at point (x,y)</i>
$u \cdot \nabla f =  \nabla f   u  \cos \theta$ , $D_u f(x,y) = \nabla f(x,y) \cdot u$	$\Rightarrow  u =1 \Rightarrow D_u f(x,y) \max \text{ when } \cos \theta = 1, \theta = 0^\circ, \text{ if}$
$=  \nabla f(x,y)  \cos \theta \text{ where } \theta = \angle \nabla f \& u \Rightarrow \nabla f(x,y) = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \quad D_u f(x,y) \min \text{ when } \cos \theta = -1, \theta = 180^\circ, \text{ if}$	

- So each iteration move in the direction of -ve gradient

Initialize  $\mathbf{w}_0$  and learning rate  $\eta$ ;

**while** true **do**

Compute  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$   
**if** converge **then**  
| **return**  $\mathbf{w}_{k+1}$   
**end**

**end**

- minus the derivative to move into the direction to decrease  $C$
- $\eta$  is learning rate = how far to move each time = step size
- tl;dr: move to down hill until local minimum
- If  $\eta$  is not too big, each time will get better, where the cost gets smaller, if  $\eta$  too big, may overshoot

- Gradient Descent:

Initialize  $\mathbf{w}_0$  and learning rate  $\eta$ ;

**while** true **do**

Compute  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$

**if** converge **then**

**return**  $\mathbf{w}_{k+1}$

**end**

**end**

- Possible convergence criteria

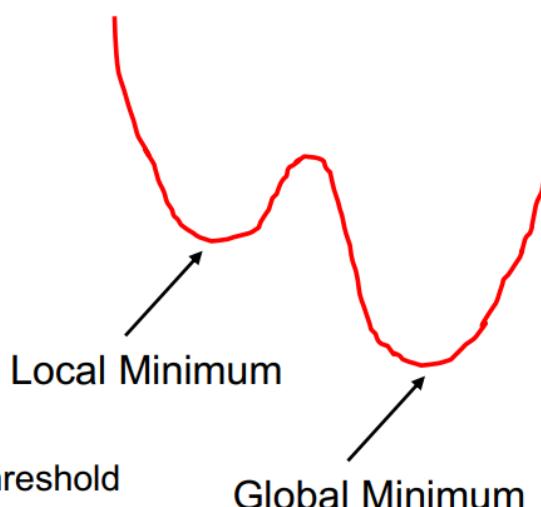
- Set maximum iteration  $k$
- Check percentage or absolute change in  $C$  below a threshold
- Check percentage or absolute change in  $\mathbf{w}$  below a threshold

- Gradient descent can only find local minimum

- Because gradient = 0 at local minimum, so  $\mathbf{w}$  won't change after that

- Many variations of gradient descent, e.g., change how gradient is computed or learning rate  $\eta$  decreases with increasing  $k$

- Can only find the local minimum, does not guarantee the global minimum

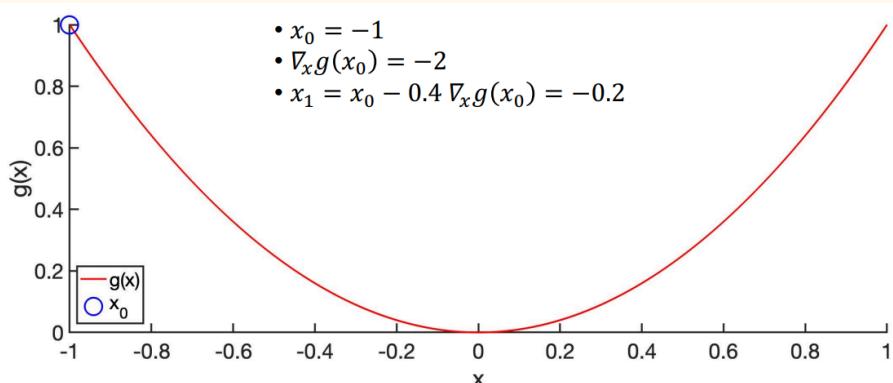


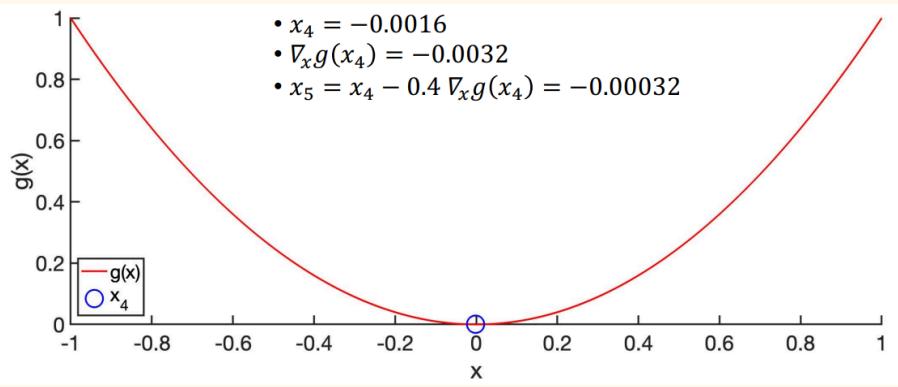
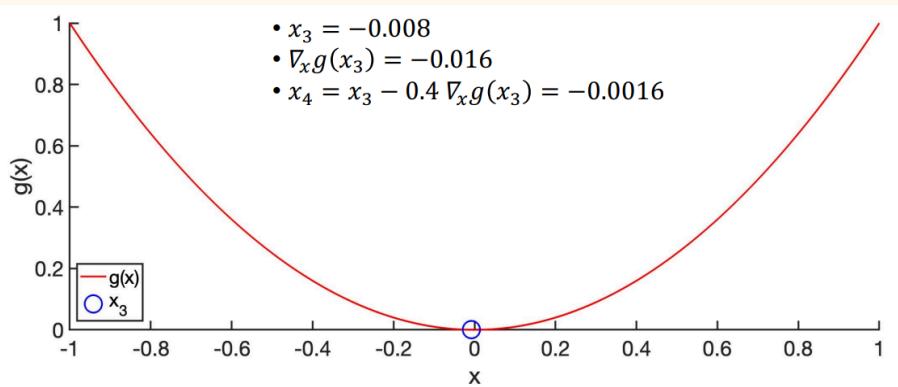
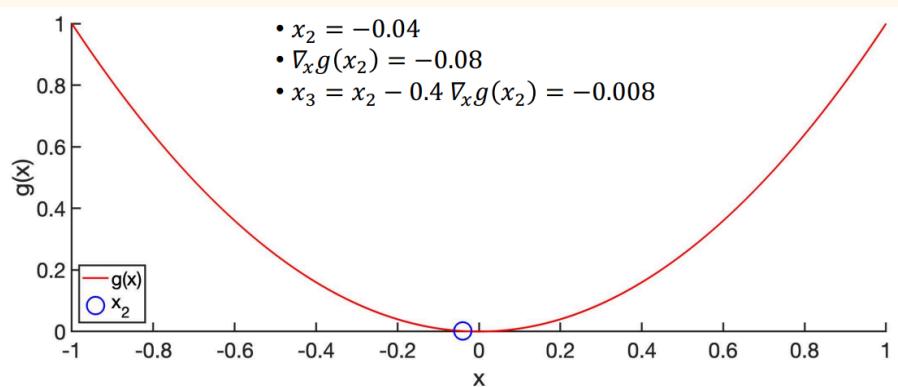
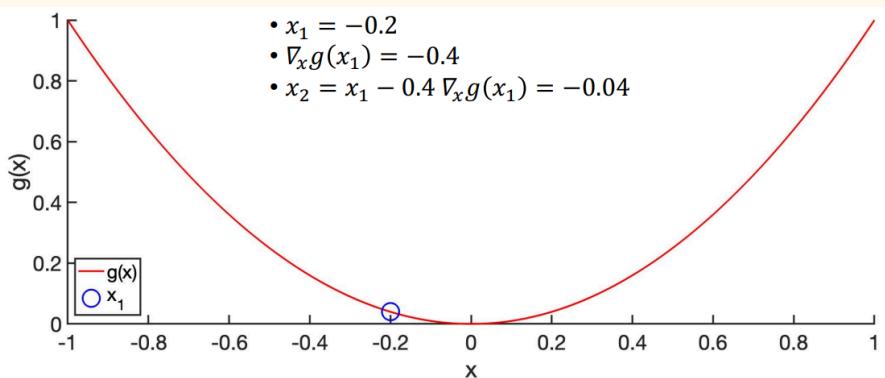
### Example 1: Find $x$ to minimise $g(x) = x^2$

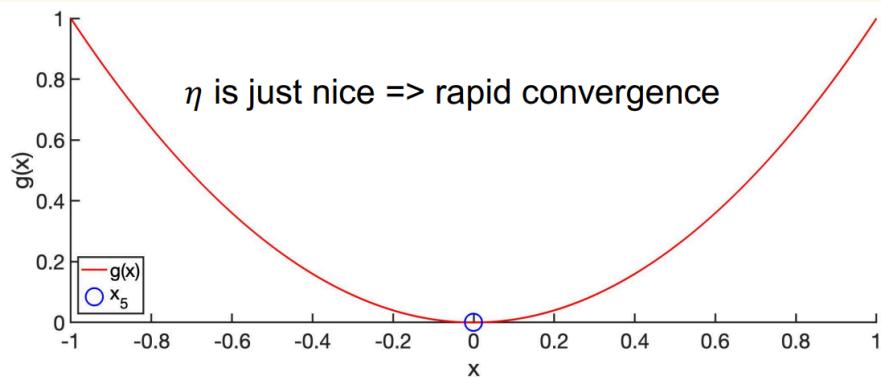
- Obviously minimum corresponds to  $x = 0$ , but let's do gradient descent

- Gradient  $\nabla_x g(x) = 2x$
- Initialize  $x_0 = -1$ , learning rate  $\eta = 0.4$
- At each iteration,  $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

- in this case,  $x$  is the w



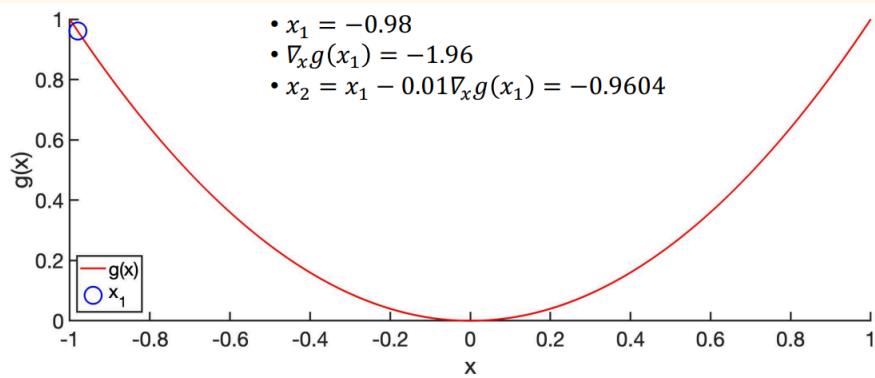
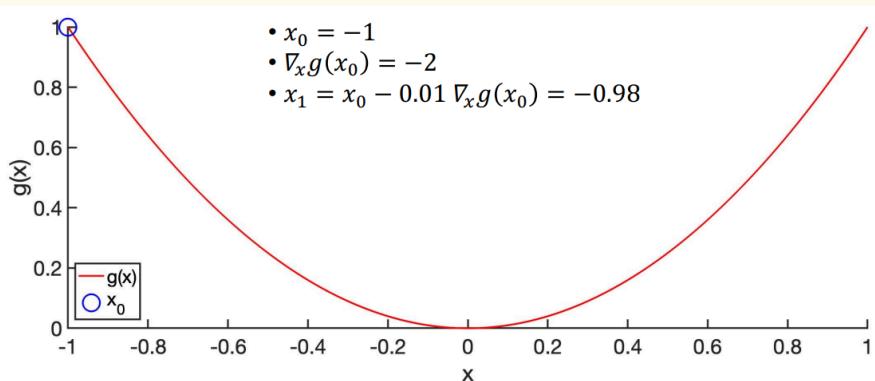


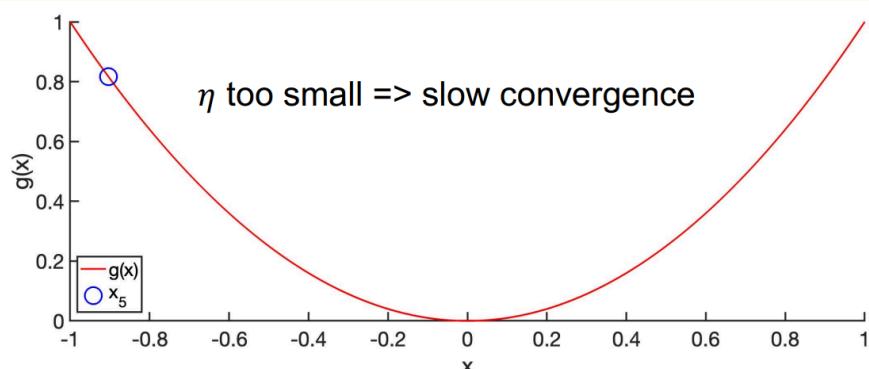
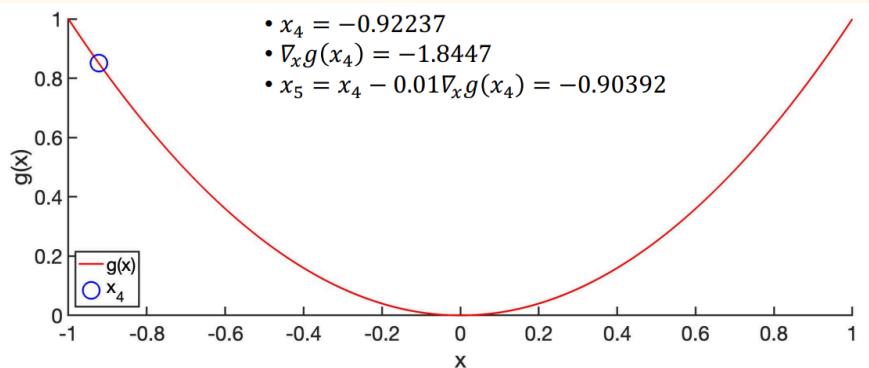
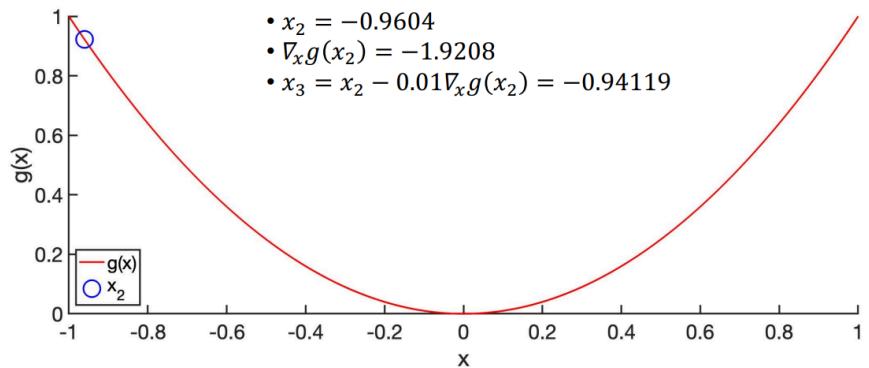


- after 5 iterations, can get very close to 0

### Example 2: What if the learning rate, $\eta$ , is too small?

- Obviously minimum corresponds to  $x = 0$ , but let's do gradient descent
  - Gradient  $\nabla_x g(x) = 2x$
  - Initialize  $x_0 = -1$ , learning rate  $\eta = 0.01$
  - At each iteration,  $x_{k+1} = x_k - \eta \nabla_x g(x_k)$

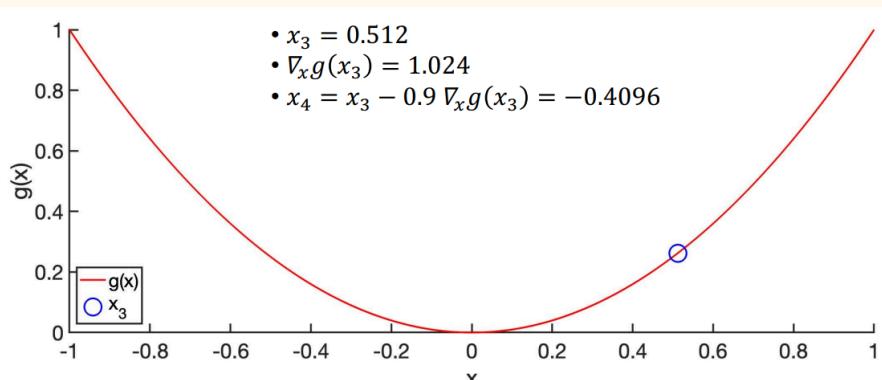
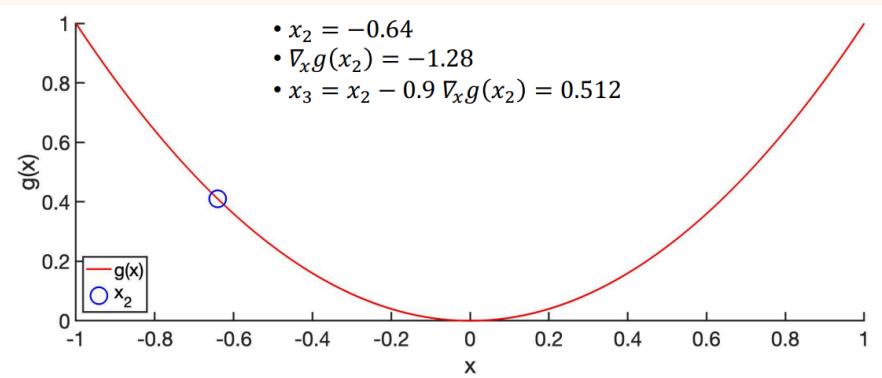
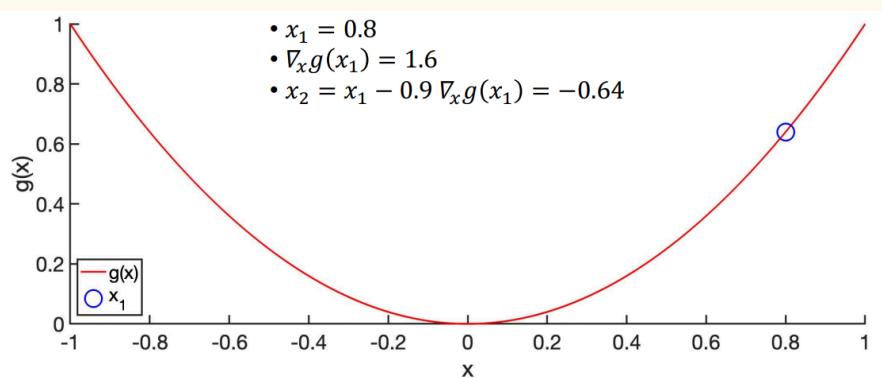
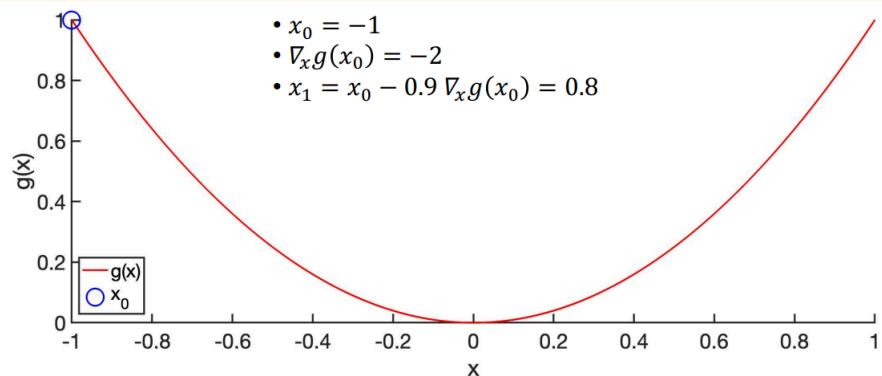


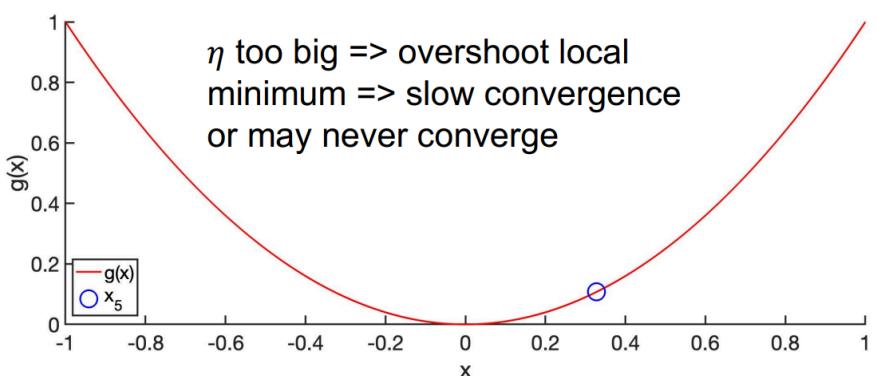
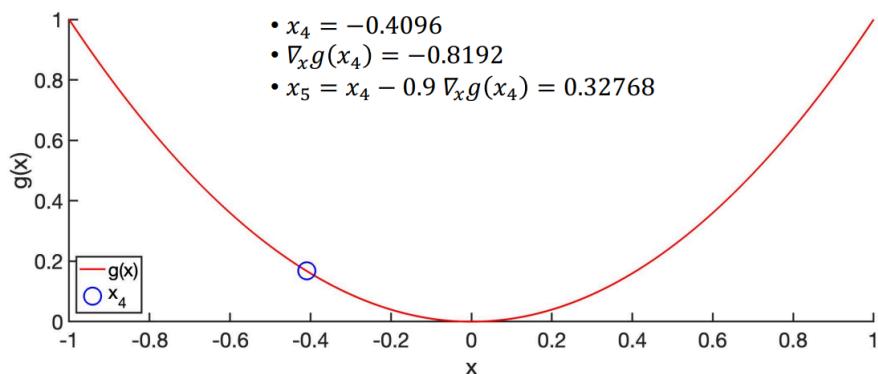


- After 4 iterations, still very far from 0, learning speed is too slow, doesn't mean it won't converge, just converges too slowly

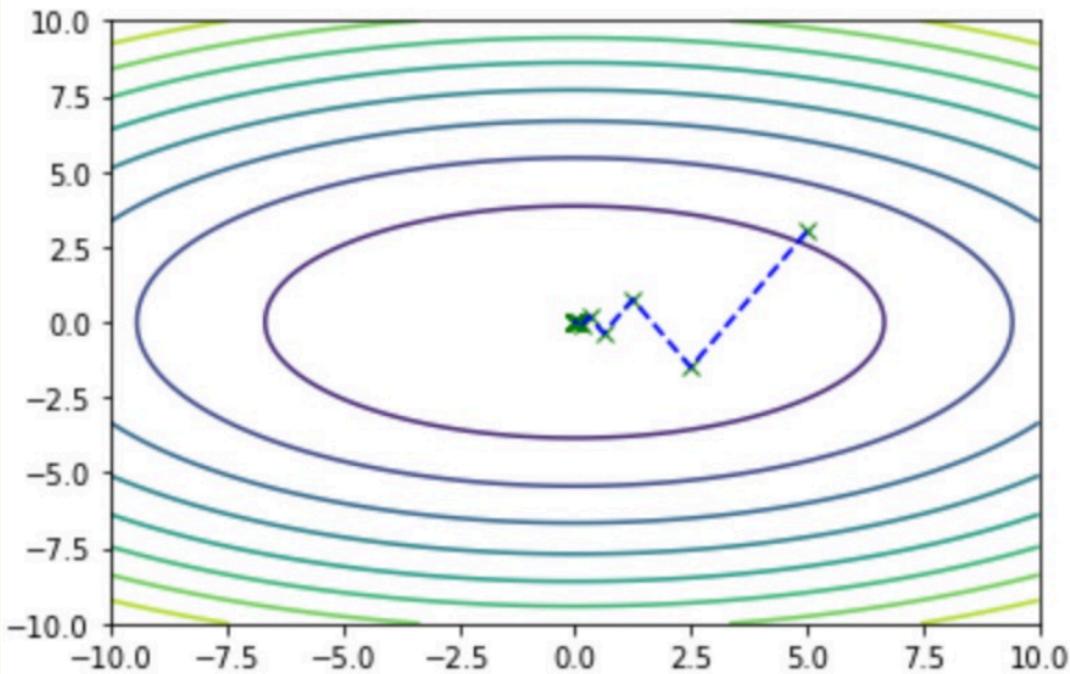
Example 3: What if the learning rate,  $\eta$ , is too big?

- Obviously minimum corresponds to  $x = 0$ , but let's do gradient descent
  - Gradient  $\nabla_x g(x) = 2x$
  - Initialize  $x_0 = -1$ , learning rate  $\eta = 0.9$
  - At each iteration,  $x_{k+1} = x_k - \eta \nabla_x g(x_k)$





- Keeps overshooting the local minimum, causing slow convergence
- In some cases, the overshooting may keep happening and never converge



## Convergence to the foot of the valley

- Cost can increase also

## Python Demo

```
9  # Main function is y=x**2
10 # Gradient is 2x
11 # Initialization & Parameters
12 x = -1
13 eta = 0.4 #just nice
14 print('----- eta = '+str(eta)+' -----')
15 for i in range(0,5):
16     x= x-2*eta*x
17     print(x)
18
19
20 x = -1
21 eta = 0.01 #too small
22 print('----- eta = '+str(eta)+' -----')
23 for i in range(0,5):
24     x= x-2*eta*x
25     print(x)
26
27
28 x = -1
29 eta = 0.9 #too big
30 print('----- eta = '+str(eta)+' -----')
31 for i in range(0,5):
32     x= x-2*eta*x
33     print(x)
```

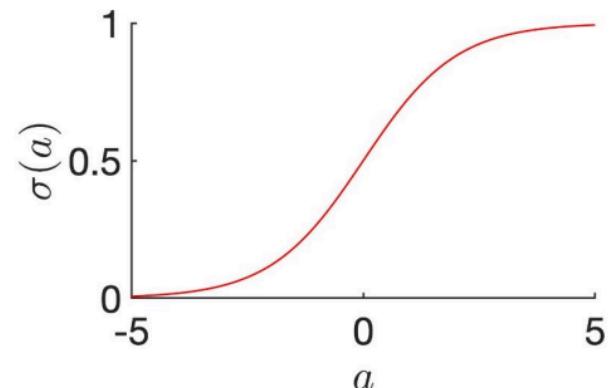
```
----- eta = 0.4 -----
-0.1999999999999996
-0.0399999999999998
-0.00799999999999993
-0.001599999999999981
-0.0003199999999999954
----- eta = 0.01 -----
-0.98
-0.9604
-0.941192
-0.92236816
-0.9039207968
----- eta = 0.9 -----
0.8
-0.6400000000000001
0.5120000000000002
-0.4096000000000002
0.3276800000000002
```

## Different Learning Models

- Different learning models  $f(\mathbf{x}_i, \mathbf{w})$  reflect our beliefs about the relationship between the features  $\mathbf{x}_i$  and target  $y_i$ 
  - For example,  $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T \mathbf{w}$  assumes polynomial relationship between features and target
- Suppose we are performing classification (rather than regression), so  $y_i$  is class  $-1$  or class  $1$ 
  - $\mathbf{p}_i^T \mathbf{w}$  is number between  $-\infty$  to  $\infty$ .
  - Can use sigmoid function to map  $\mathbf{p}_i^T \mathbf{w}$  to between 0 and 1:

$$f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$$

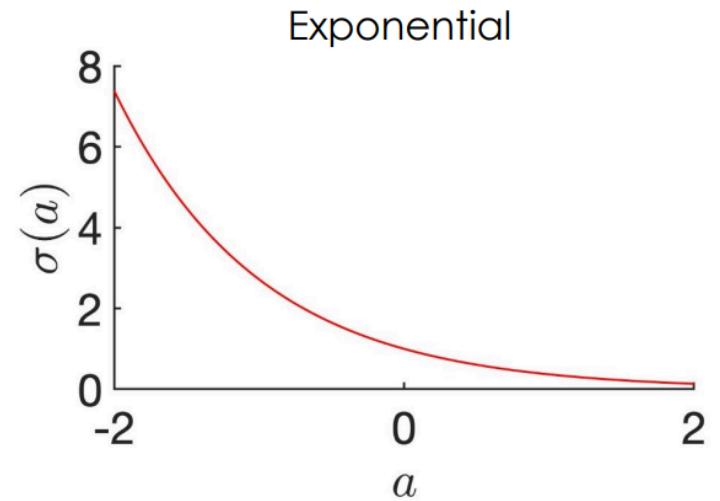
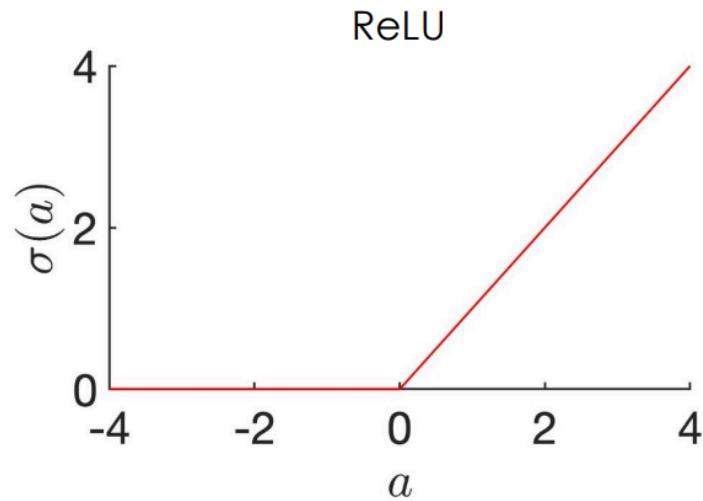
$$\sigma(a) = \frac{1}{1 + e^{-a}}$$



- If  $f(\mathbf{x}_i, \mathbf{w})$  is closer to 0 (or 1), we predict class  $-1$  (or class  $1$ )

- More generally, in one layer neural network:  $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$ , where activation function  $\sigma$  can be sigmoid or some other functions &  $\mathbf{p}$  is linear
  - in neural network, the  $\sigma$  function is called activation function
  - Sigmoid is for mapped big values down to 0-1 to normalised, for probability

- $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$ , where  $\sigma$  can be different functions:
- Rectified linear unit (ReLU):  $\sigma(a) = \max(0, a)$
- Exponential:  $\sigma(a) = \exp(-a)$

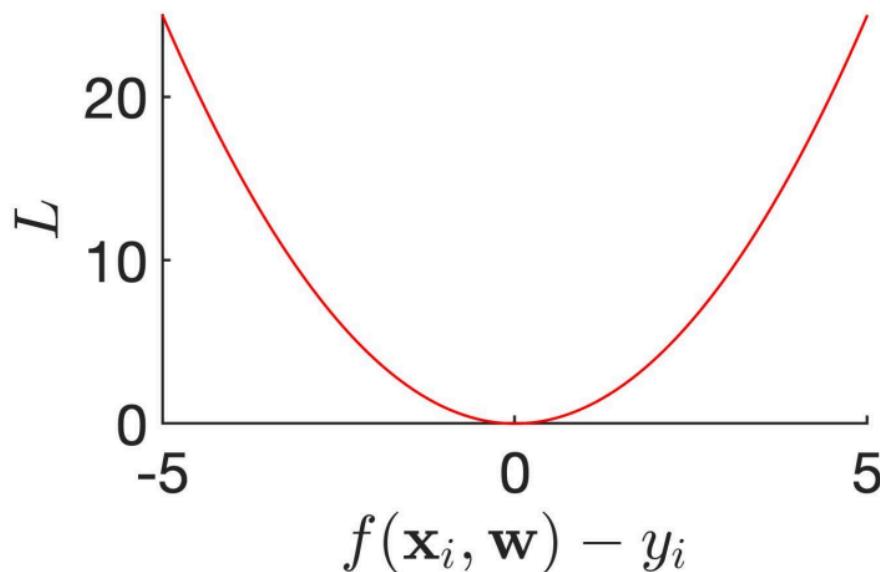


- ReLU, = 0 for negative input, else = input
  - ignores contribution of samples that points the opposite way as w

## Different Loss Functions

### Square Error Loss for Regression

- Different loss functions  $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$  encodes the penalty when we predict  $f(\mathbf{x}_i, \mathbf{w})$  but the true value is  $y_i$ 
  - $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$  is called the square error loss



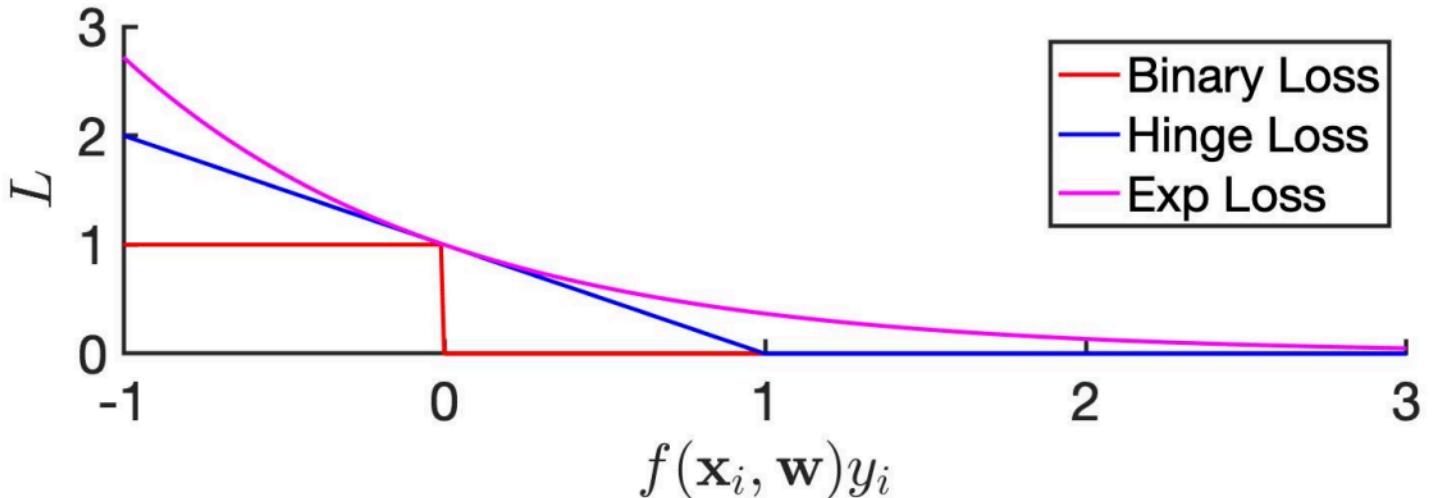
### Binary Loss for Classification

- Suppose we are performing classification (rather than regression), so  $y_i$  is class  $-1$  or class  $1$ , then square error loss makes less sense. Instead, we can use
  - Binary loss (or 0–1 loss):  $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w}) = y_i \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w}) \neq y_i \end{cases}$
  - In practice, hard to constrain  $f(\mathbf{x}_i, \mathbf{w})$  to be exactly  $-1$  or  $1$ , so we can declare “victory” if  $f(\mathbf{x}_i, \mathbf{w})$  &  $y$  have the same sign:

$$L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i < 0 \end{cases}$$

- for classification, use Binary loss
- Equal or use same sign

- Binary loss, where  $y_i$  is class  $-1$  or class  $1$  &  $f(\mathbf{x}_i, \mathbf{w})$  is a number between  $-\infty$  and  $\infty$ :  $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i < 0 \end{cases}$
- Binary loss not differentiable, so two other possibilities
  - Hinge loss:  $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \max(0, 1 - f(\mathbf{x}_i, \mathbf{w})y_i)$
  - Exponential loss:  $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \exp(-f(\mathbf{x}_i, \mathbf{w})y_i)$



- Binary loss cannot be differentiated to find gradient descent

### In Class Quiz

Let us minimize  $f(x) = x^2$ . Assume learning rate = 0.1 & initialize  $x$  to be 2. What is the value of  $x$  after the 1st iteration?



## Summary

- Building blocks of machine learning algorithms
  - Learning model: reflects our belief about relationship between features & target we want to predict
  - Loss function: penalty for wrong prediction
  - Regularization: penalizes complex models
  - Optimization routine: find minimum of overall cost function
- Gradient descent algorithm
  - At each iteration, compute gradient & update model parameters in direction opposite to gradient
  - If learning rate  $\eta$  is too big  $\Rightarrow$  may not converge
  - If learning rate  $\eta$  is too small  $\Rightarrow$  converge very slowly
- Different learning models, e.g., linear, polynomial, sigmoid, ReLU, exponential, etc
- Different loss functions, e.g., square error, binary, logistic, etc

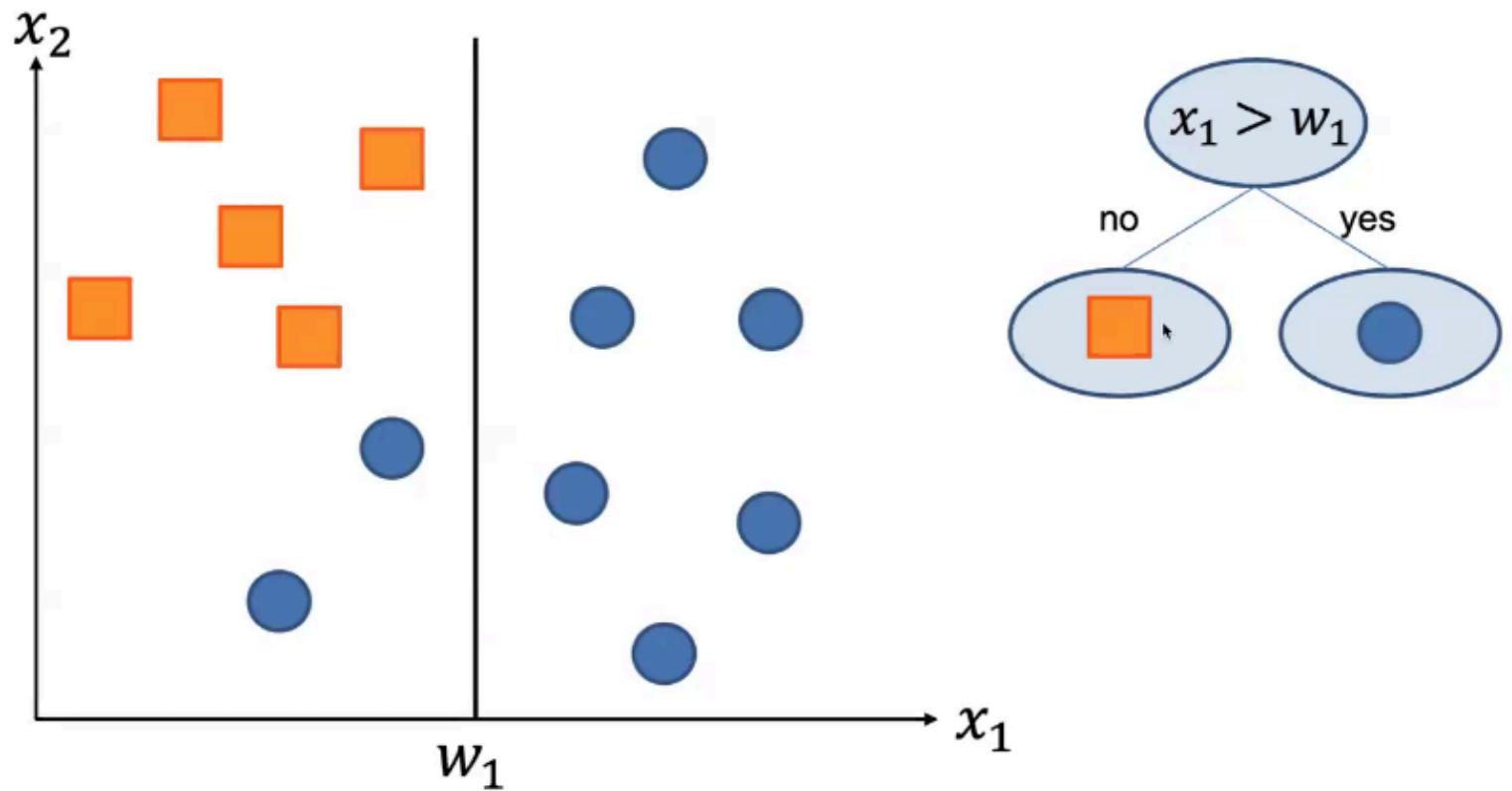
## Lec 9: Decision Trees & Random Forest

- Supervised learning: given feature(s)  $x$ , we want to predict target  $y$  to be some  $f(x)$ 
  - If  $y$  is continuous, problem is called “regression”
  - If  $y$  is discrete, problem is called “classification”
- Previous lectures used linear models for regression (and classification)
  - Nonlinearity added by using polynomial regression or other learning models
- New approach today: trees

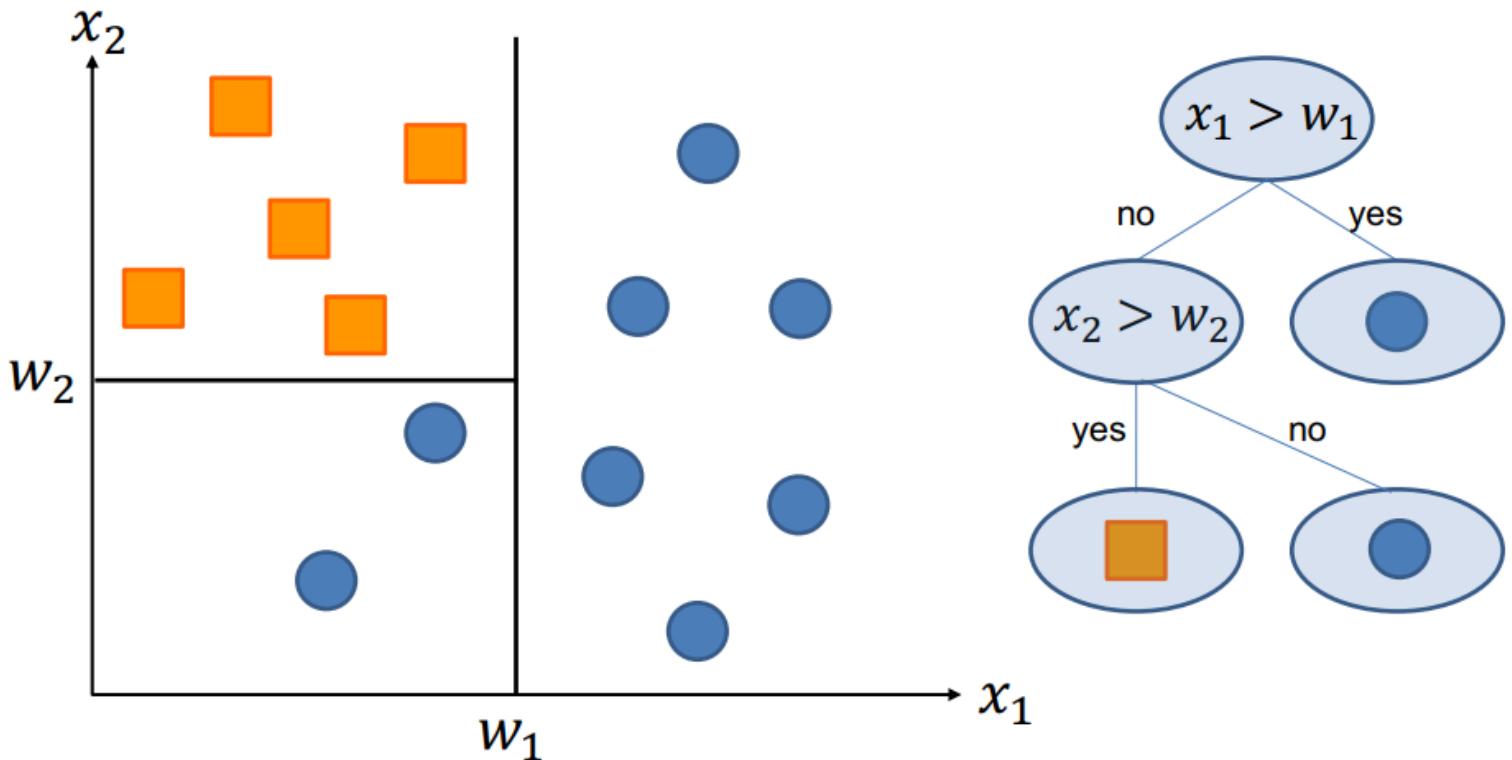
## Decision Tree Classification

### Example

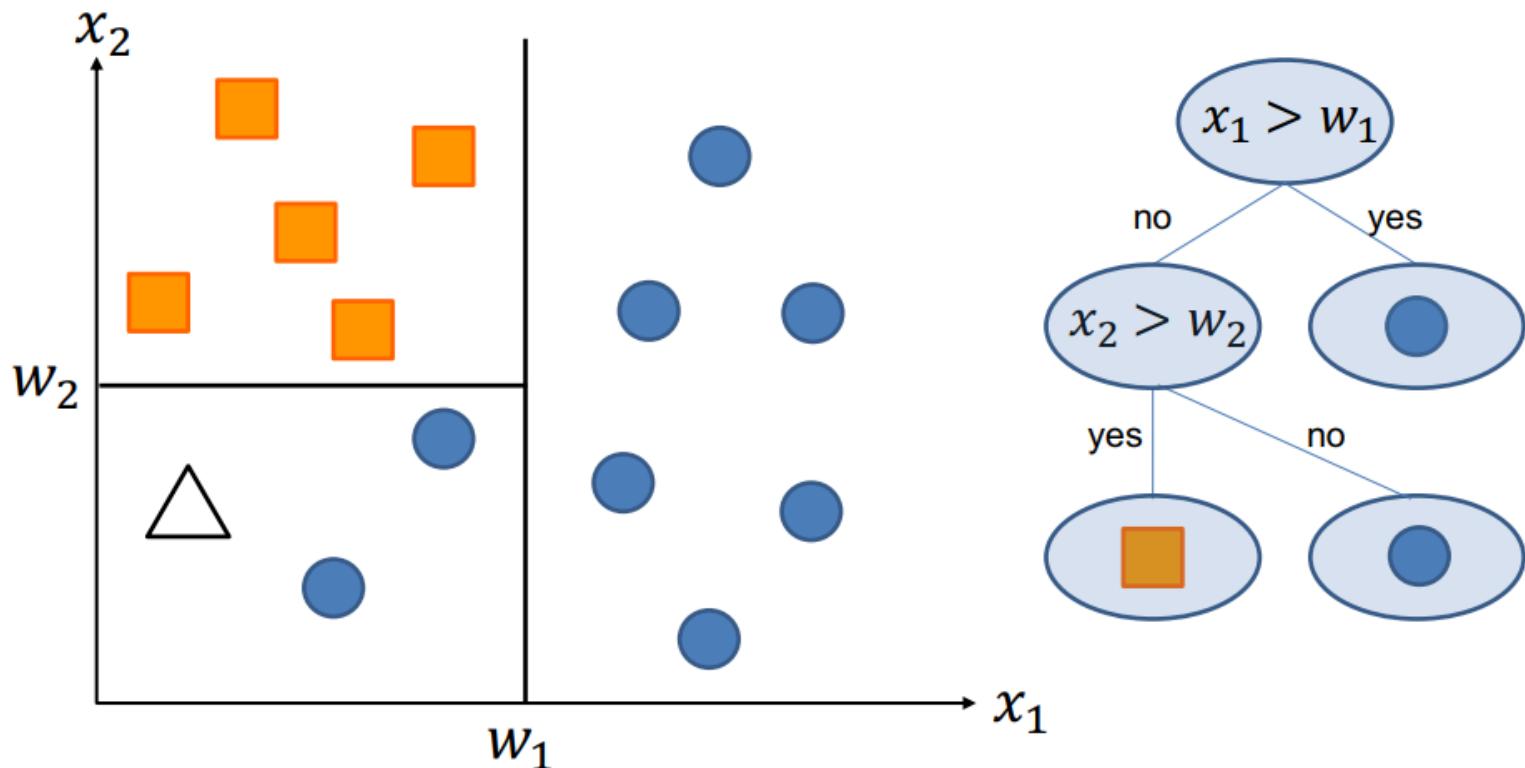
- Goal: predict class labels using two features  $x_1$  &  $x_2$



- Improved by adding a threshold with another variable
- Right hand side is the decision tree skeleton

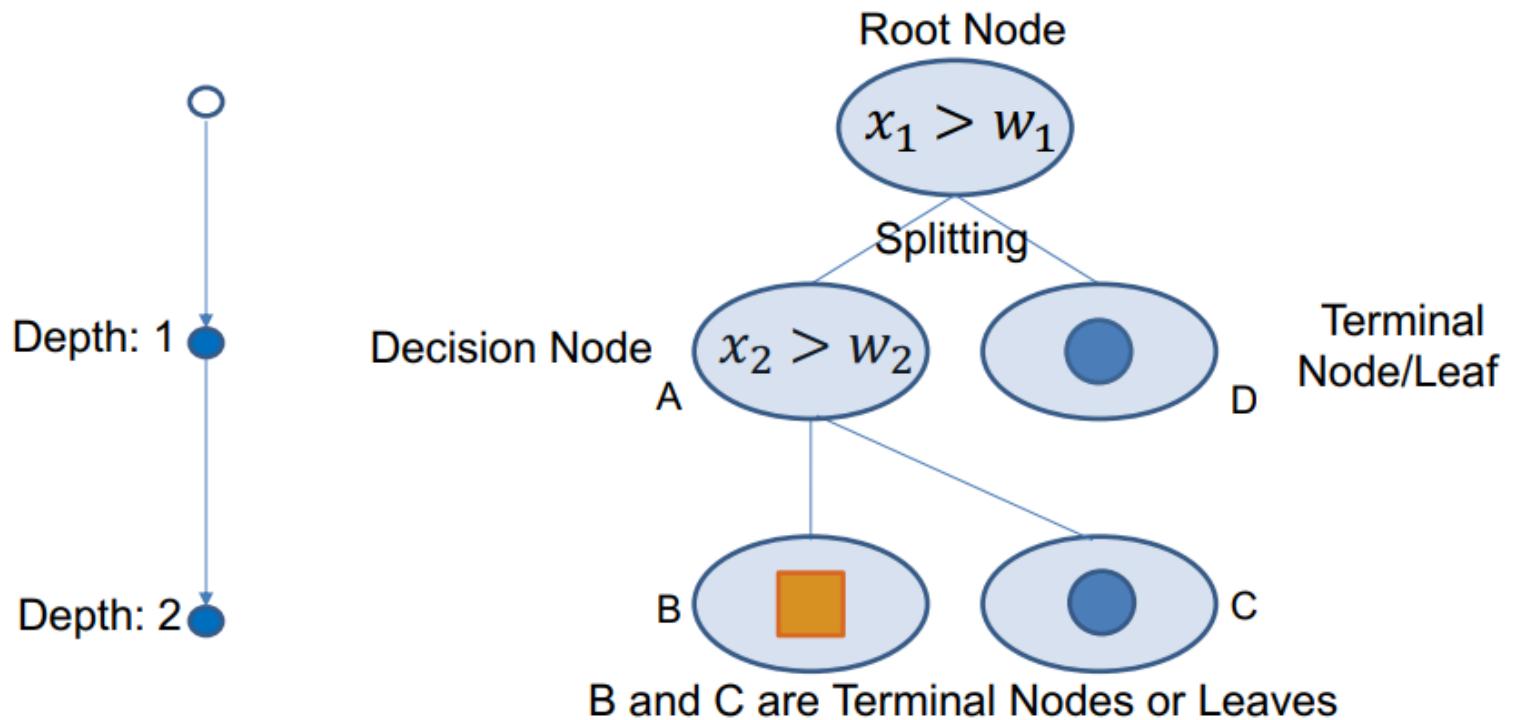


- In our test set, we observe datapoint  $\Delta$  shown below. How would the decision tree classify this point?



- Using the decision tree, the triangle is classified as the blue circle

## Basic Terminologies



A-B-C forms a **sub-tree** or **branch**.

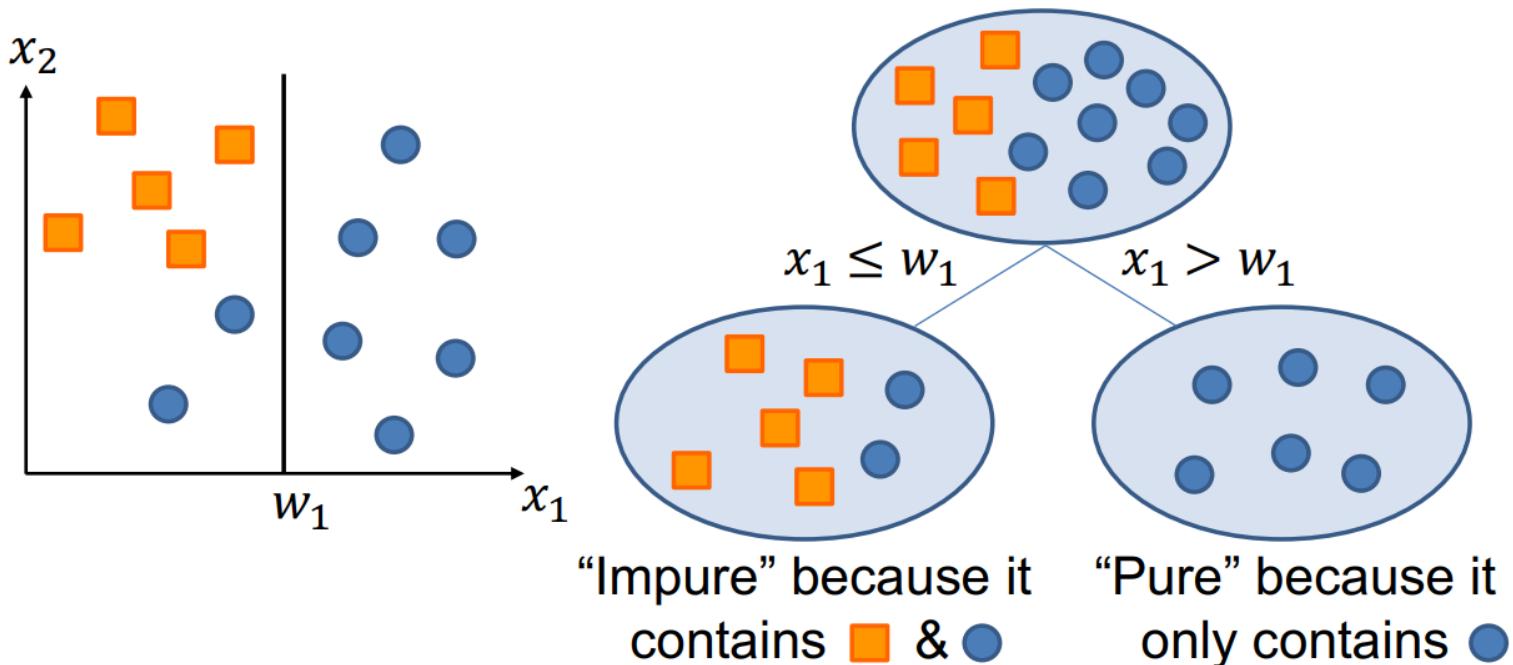
A is **parent node** of B and C; B and C are **children nodes** of A.

- At each decision node, a decision is made
- Decision causes splitting of branch
  - root node is and must be a decision node
- First node is the root node, at depth 0
  - each decision is down one depth
  - each terminal after decision is one depth below the decision
- Node with no decision at the end is the terminal node or leaf
- ABC sub-tree, A is the root node

## Building a Classification Decision Tree

- Classification tree learning (or tree induction or tree growing) is the construction of a classification tree given training data
- For a given training set, there can be many trees with 0 training error, so we prefer less “complex” trees
- Complexity can be defined as number of nodes in the tree
- Finding smallest tree is computationally hard, so we typically use some greedy algorithm not guaranteed to find the best tree
- We need to first define the concept of node impurity
  - There could be many possible trees
  - More complicated tree may overfit data

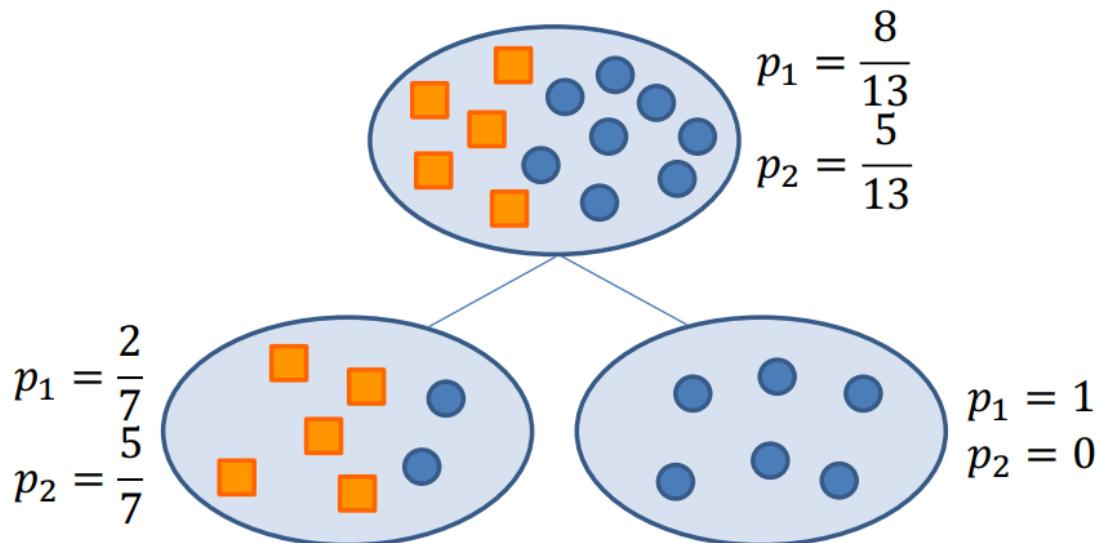
## Node Impurity



- “Purity” is desirable because if a node contains only training data from one class, then prediction for training data in the node is perfect
  - Pure node performs perfectly for training data
  - Purity and MSE always decrease down the depth

## Node Impurity Measures

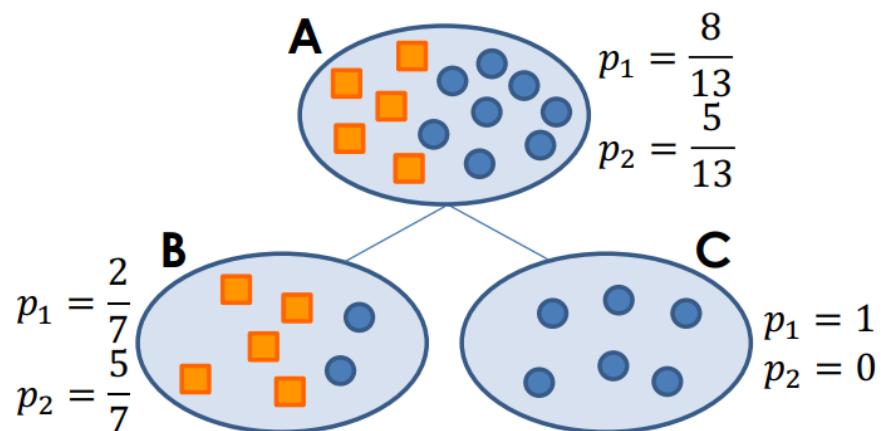
- Let ● be class 1 & ■ be class 2
- For particular node  $m$ , let  $p_i$  be the fraction (or probability) of data samples in node  $m$  belonging to class  $i$
- Let  $Q_m$  be impurity of node  $m$
- 3 impurity measures: Gini, entropy, misclassification rate



## Gini Impurity

- Let  $K = \# \text{ classes}$ , define  $Q_m = 1 - \sum_{i=1}^K p_i^2 = 1 - p_1^2 - p_2^2$
- Node A:  $Q_A = 1 - (8/13)^2 - (5/13)^2 = 0.4734$
- Node B:  $Q_B = 1 - (2/7)^2 - (5/7)^2 = 0.4082$
- Node C:  $Q_C = 1 - 1^2 - 0^2 = 0$
- Overall Gini (depth 1) = fraction of data samples in node B  $\times Q_B +$  fraction of data samples in node C  $\times Q_C$ 
  - $\left(\frac{7}{13}\right) \times 0.4082 + \left(\frac{6}{13}\right) \times 0 = 0.2198$
- Observe lower impurity at depth 1 compared with root
- Same Gini formula for more than 2 classes:

$$Q_m = 1 - \sum_{i=1}^K p_i^2$$



- node B is purer than node A and C
- To make sure the decision is making progress, check the Overall Gini at that depth.

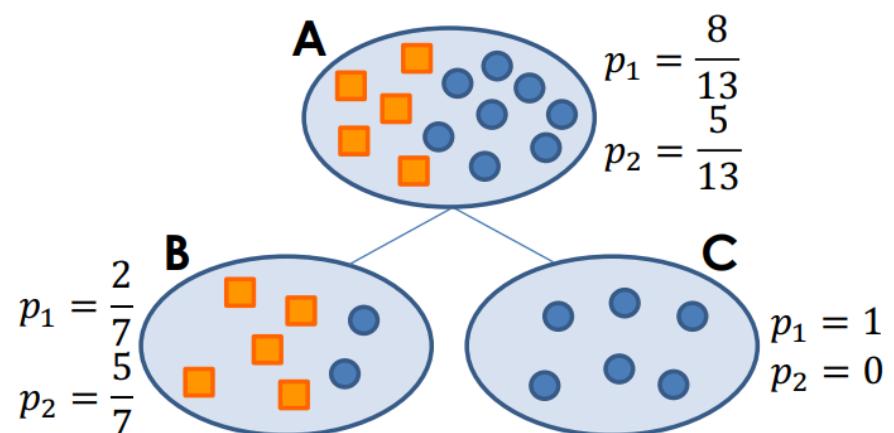
**Overall Gini (depth 1) = fraction of data samples in node B  $\times Q_B +$  fraction of data samples in node C  $\times Q_C$**

- weighted by the amount of each class
- Can see the above example, Overall Gini at depth 1 is lower Gini of node A, means its making progress
- Cost at depth is just the weighted average of the cost of nodes
- Higher Gini, more impure
- Gini = 0, purest pure
- will not  $> 1$
- $p_i = \text{proportion of class } i \text{ in node}, 0 < p_i < 1$
- worst case Gini is when the class is evenly distributed in a node, cannot tell apart

## Entropy

- Let  $K = \# \text{ classes}$ , define  $Q_m = -\sum_{i=1}^K p_i \log_2 p_i = -p_1 \log_2 p_1 - p_2 \log_2 p_2$
- Node A:  $Q_A = -(8/13)\log_2(8/13) - (5/13)\log_2(5/13) = 0.9612$
- Node B:  $Q_B = -(2/7)\log_2(2/7) - (5/7)\log_2(5/7) = 0.8631$
- Node C:  $Q_C = -1 \log_2 1 - 0 \log_2 0 = 0$
- Overall entropy (depth 1) = fraction of data samples in node B  $\times Q_B$   
+ fraction of data samples in node C  $\times Q_C$ 
  - $\left(\frac{7}{13}\right) \times 0.8631 + \left(\frac{6}{13}\right) \times 0 = 0.4648$
- Observe lower impurity at depth 1 compared with root
- Same entropy formula for more than 2 classes:

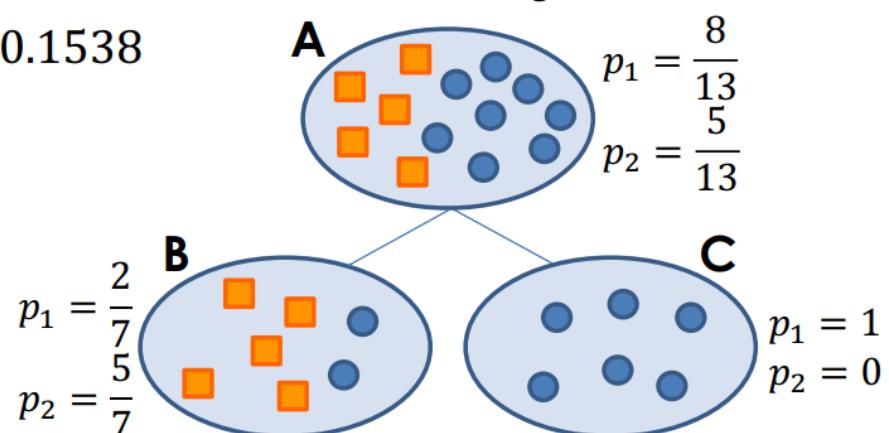
$$Q_m = -\sum_{i=1}^K p_i \log_2 p_i$$



- High entropy, more impure
- Entropy = 0, purest pure
- possible to  $> 1$
- $\log 0$  is undefined, but multiplied by 0

## Misclassification rate

- Let  $K = \# \text{ classes}$ , define  $Q_m = 1 - \max_i p_i = 1 - \max(p_1, p_2)$
- Node A:  $p_1 > p_2$ , so best classification = class 1  $\Rightarrow Q_A = 1 - 8/13 = 5/13$
- Node B:  $p_2 > p_1$ , so best classification = class 2  $\Rightarrow Q_B = 1 - 5/7 = 2/7$
- Node C:  $p_1 > p_2$ , so best classification = class 1  $\Rightarrow Q_C = 1 - 1 = 0$
- Overall misclassification rate (depth 1) = fraction of data samples in node B  $\times Q_B +$  fraction of data samples in node C  $\times Q_C$ 
  - $\left(\frac{7}{13}\right) \times \left(\frac{2}{7}\right) + \left(\frac{6}{13}\right) \times 0 = 0.1538$
- Observe lower impurity at depth 1 compared with root
- Same misclassification rate formula for more than 2 classes:  $Q_m = 1 - \max_i p_i$



- higher means more impure
- Overall misclass is also total misclassification at the depth/total data point
  - total misclassification at the depth = 2
  - total = 13
  - 2/13

---

## Algorithm: Classification Tree Learning

---

**Input:** Impurity measure  $Q$ , parameter  $\text{max\_depth}$  & training set

**Output:** Tree

- 1 root  $\leftarrow$  all training samples
  - 2 **for**  $d \leftarrow 1$  **to**  $\text{max\_depth}$  **do**
  - 3     **for** each leaf node  $m$  at depth  $d - 1$  **do**
  - 4         Find best feature & best threshold, so splitting node  $m$  into two reduces the most impurity
  - 5         Use decision rule to distribute training samples from node  $m$  across two new leaf nodes
  - 6 **return** tree
- 

- For each new depth, will go through all the nodes at the depth before

Find best feature & best threshold, so splitting node  $m$  into two reduces the most impurity

- - this means at a node, find the feature and the threshold such that impurity measured by an agreed upon  $Q$ , reduces the most
- Use decision rule to distribute training samples from node  $m$  across two new leaf nodes
  - once the above decision rule is determined, use it to distribute the node to the new depth
- When comparing cost to choose the best threshold, the impurity is the node and the two new nodes, not of other nodes

## Advantages

- Easy to visualize & understand tree
- Can work with a mix of continuous and discrete data
- Less data cleaning required
- Makes less assumptions about the relationship between features & target

## Disadvantages

- Trees can become overly complex resulting in overfitting
- Trees can be unstable, e.g., small changes in training data can result very different trees
- Less data cleaning required means less affected by outliers
- Unstable as any change to a node will affect all the nodes after it, variation to training data affects training outcomes a lot (feature or threshold, improve by random forest)  $\Rightarrow$  high variance

### To reduce overfitting...



- One or more of the following can help reduce overfitting
  - Set maximum depth for the tree
  - Set minimum number of samples for splitting a leaf node, e.g., if leaf node has less than 10 samples, then do not split node
  - Set minimum decrease in impurity, e.g., if selecting the best feature & threshold does not improve impurity by at least 1%, then do not split the leaf node
  - Instead of looking at all features when considering how to split a leaf node, we can randomly look at a subset (e.g., square root of the total number of features)

## Regression Trees

- Classification trees seek to predict discrete variables (i.e., classification)
- Regression trees seek to predict **continuous** variables (i.e., regression)
- Can use same approach as before, but instead of minimizing impurity, we can try to **minimize mean square error (MSE)**
- Suppose there are  $J_m$  training samples in a leaf node  $m$  of the regression tree with target values  $y_1, y_2, \dots, y_{J_m}$ 
  - We can predict  $\hat{y}_m = \frac{1}{J_m} \sum_{j=1}^{J_m} y_j$
  - Then MSE of node  $m$  is given by  $S_m = \frac{1}{J_m} \sum_{j=1}^{J_m} (y_j - \hat{y}_m)^2$
  - Across all leaf nodes, **total MSE  $S = \sum_m \frac{J_m}{N} S_m$** , where  $N$  is the total number of data samples
- Since its all continuous, cannot get the majority vote, just find the average
- Prediction = average of the Ytrain
- MSE = sum of error squared of Ytrain and Yave
  - applicable at any depth of node
  - root will be Yave of all the value
- At a whole depth is also the weighted average of MSE

- Algorithm is basically the same as classification tree learning
- Various approaches to reduce overfitting also apply here

---

**Algorithm:** Regression Tree Learning

---

**Input:** parameter *max\_depth* & training set

**Output:** Tree

```
1 root ← all training samples
2 for  $d \leftarrow 1$  to max_depth do
3   for each leaf node  $m$  at depth  $d - 1$  do
4     Find best feature & best threshold, so splitting
      node  $m$  into two reduces MSE the most
5     Use decision rule to distribute training samples
      from node  $m$  across two new leaf nodes
6 return tree
```

---

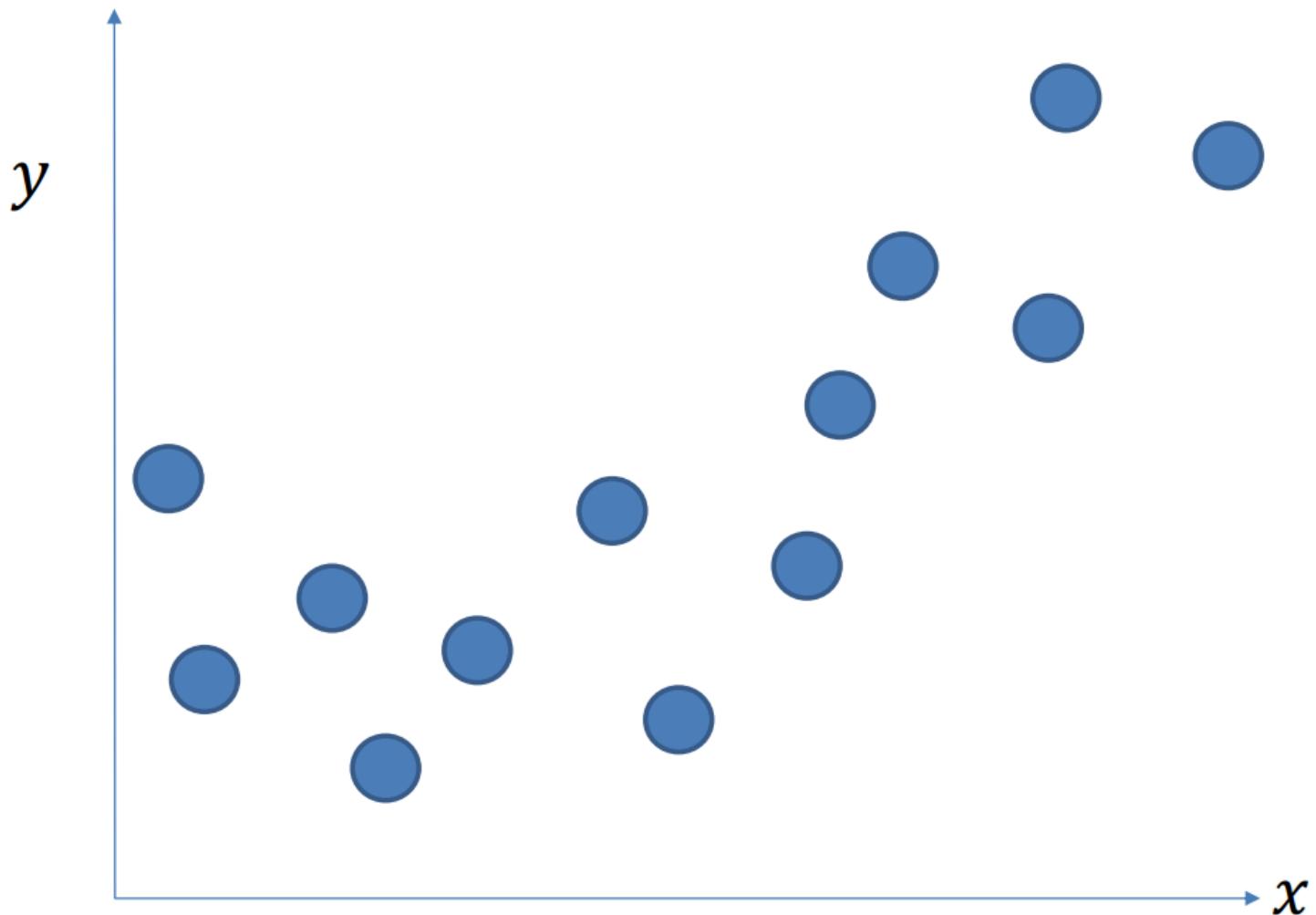
- Change performance metrics to MSE

## To reduce overfitting...

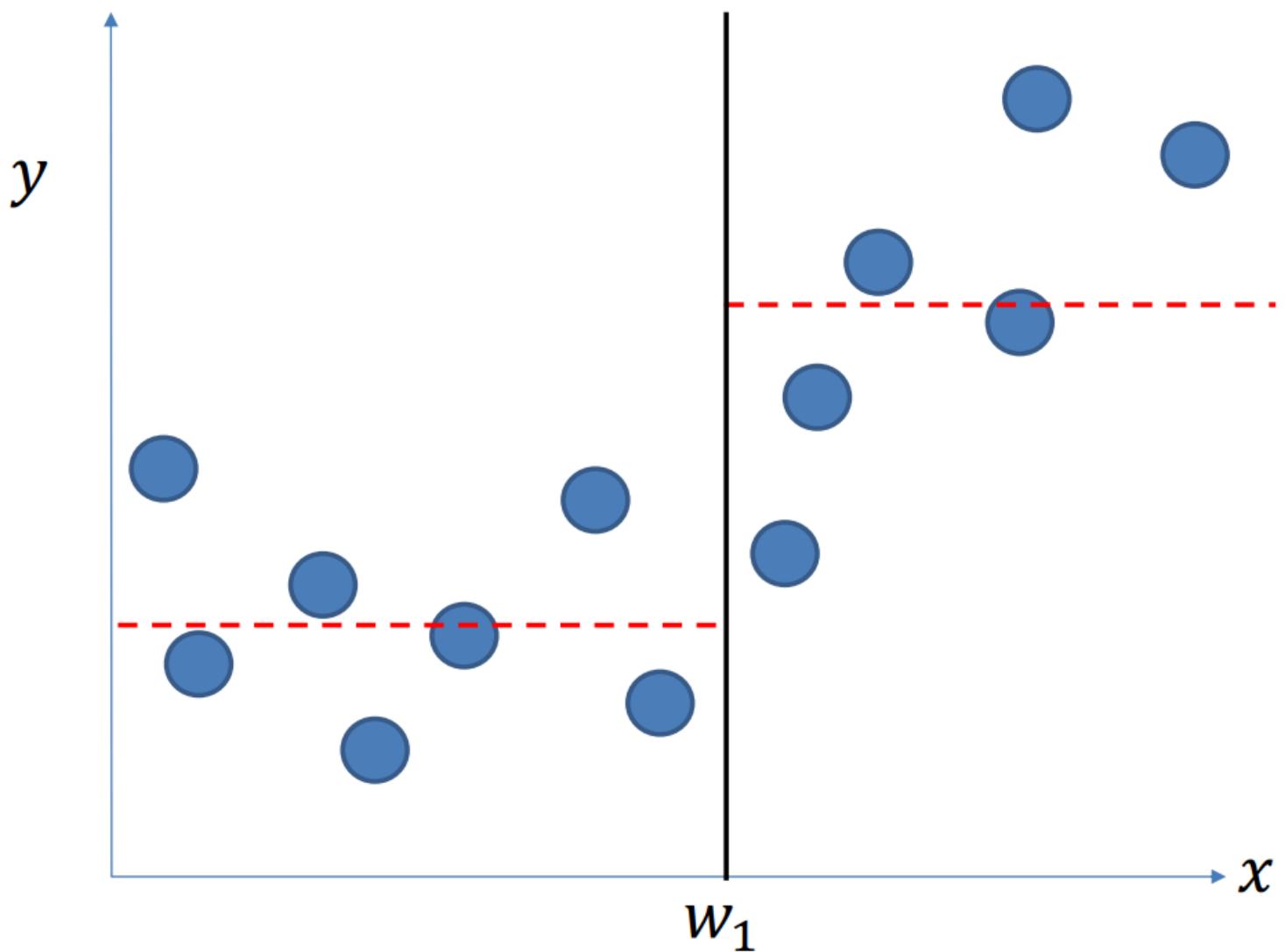


- One or more of the following can help reduce overfitting
  - Set maximum depth for the tree
  - Set minimum number of samples for splitting a leaf node, e.g., if leaf node has less than 10 samples, then do not split node
  - Set minimum decrease in impurity, e.g., if selecting the best feature & threshold does not improve impurity by at least 1%, then do not split the leaf node
  - Instead of looking at all features when considering how to split a leaf node, we can randomly look at a subset (e.g., square root of the total number of features)

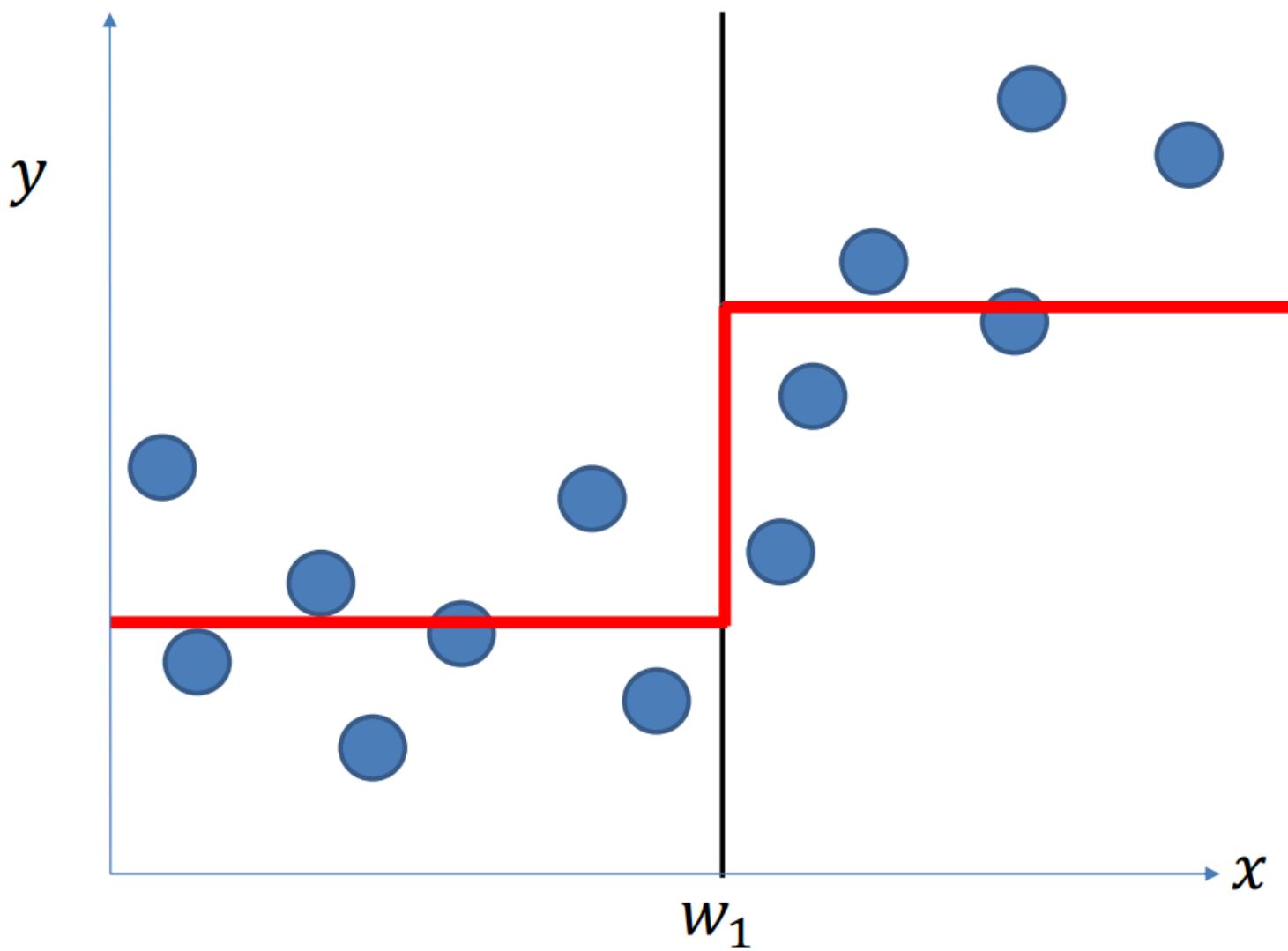
### Example 1

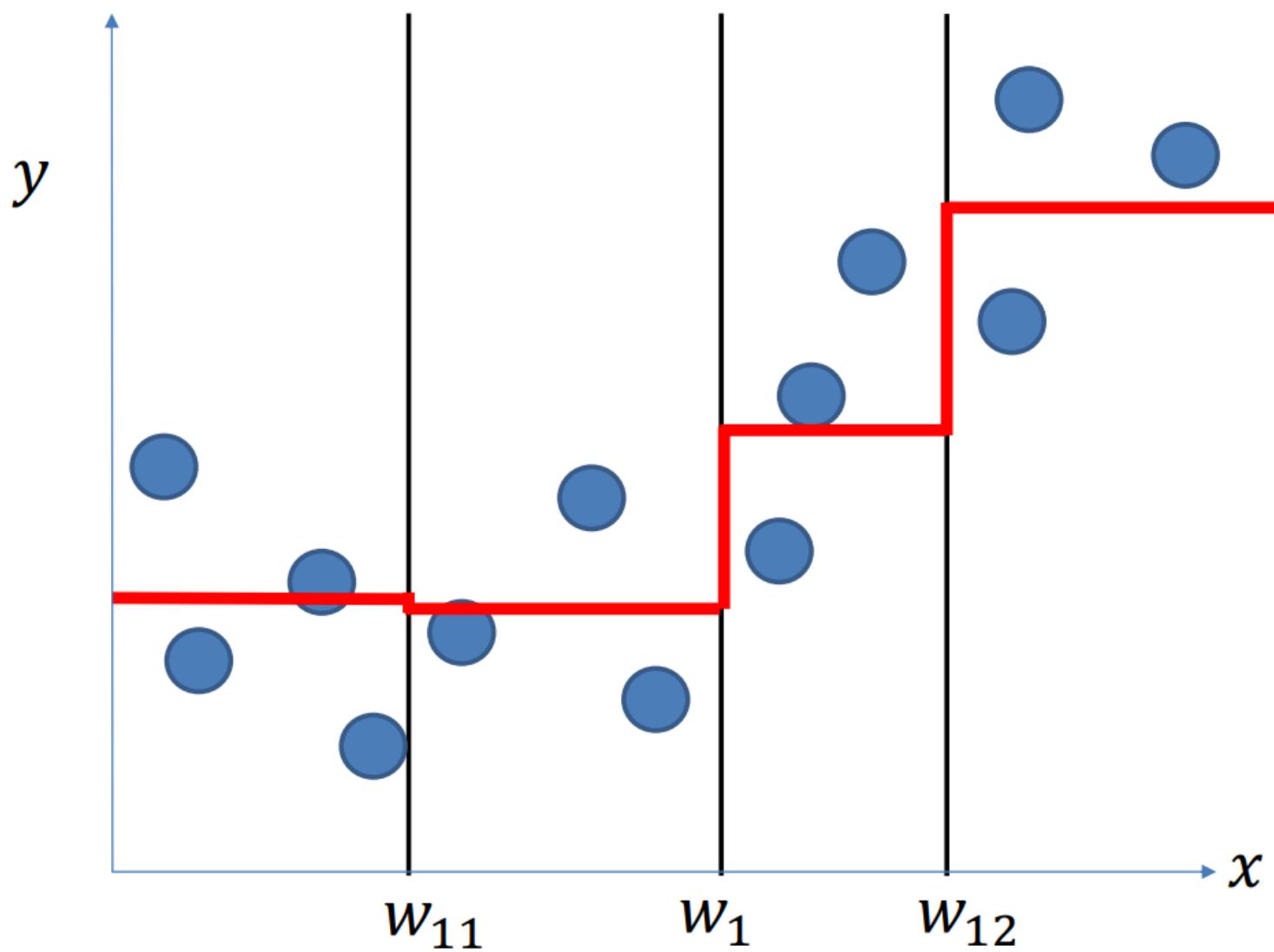


- Using  $x$  to predict  $y$



- the best prediction is the average of the individual target value, shown by the red line
- With feature  $x$  and threshold  $w_1$
- The average is the  $y_m \hat{y}_m$





## Example 2

- Consider house prices in Singapore.
- Target variable is Price  $P$ .
- Attributes are House Size  $S$  and Number of Rooms  $R$ .

	<b>House Size ('000 sq ft)</b>	<b>Num of Rooms</b>	<b>Price ('000,000 SGD)</b>
1	0.5	2	0.19
2	0.6	1	0.23
3	1.0	3	0.28
4	2.0	5	0.42
5	3.0	4	0.53
6	3.2	6	0.75
7	3.8	7	0.80

- Note that I have arranged the data points in increasing order of  $P$ , which so happens to be increasing order of  $S$  as well. However, this is not the same order as that of  $R$ .

# Calculation of MSE for House Size Split

- Focus first on the House Size attribute  $S$ . If we set the threshold at  $\tau = 0.75$ , then the targets of the two classes are  $\{0.19, 0.23\}$  and  $\{0.28, 0.42, 0.53, 0.75, 0.80\}$ . The individual conditional MSEs are

$$\text{MSE}_{P|S<0.75} = 4 \times 10^{-4} \quad \text{and} \quad \text{MSE}_{P|S \geq 0.75} = 0.0385.$$

- Thus, the averaged conditional MSE with a split of  $S$  at 0.75 is

$$\text{MSE}_{P|S(0.75)} = \frac{2}{7}\text{MSE}_{P|S<0.75} + \frac{5}{7}\text{MSE}_{P|S \geq 0.75} = 0.0276.$$

- Sweep through all possible thresholds  $\tau$  to determine the best threshold for attribute  $S$ .

$\text{MSE}_{P S(0.55)}$	$\text{MSE}_{P S(0.75)}$	$\text{MSE}_{P S(1.5)}$	$\text{MSE}_{P S(2.5)}$	$\text{MSE}_{P S(3.1)}$	$\text{MSE}_{P S(3.5)}$
0.0402	0.0276	0.0145	0.0102	0.0116	0.0325

$$S_m = \frac{1}{J_m} \sum_{j=1}^{J_m} (y_j - \hat{y}_m)^2 \quad \text{MSE } S = \sum_m \frac{J_m}{N} S_m$$

## Calculation of MSE for # Rooms Split

- Rearrange the target variables in order of the house sizes. Doing so we get  $(0.23, 0.19, 0.28, 0.53, 0.42, 0.75, 0.80)$ . Now we sweep through all possible thresholds  $\tau$  for  $R$  to get the following averaged conditional MSEs.
- We get the following table.

$MSE_{P R(1.5)}$	$MSE_{P R(2.5)}$	$MSE_{P R(3.5)}$	$MSE_{P R(4.5)}$	$MSE_{P R(5.5)}$	$MSE_{P R(6.5)}$
0.0435	0.0276	0.0145	0.0222	0.0116	0.0325



Choose the threshold with minimum MSE

# Where is the First Split?

- We should first split the dataset into two branches, the left branch indicating  $S < 2.5$  and the right with  $S \geq 2.5$ .
- Split the dataset into two sub-datasets and we may decide to stop or split the  $R$  feature.
- If we decide to stop, then for any new/test house with a house size of  $< 2.5$ , we will predict that its price is the average of the houses in our training set whose size is  $< 2.5$ , i.e.,

$$(0.19 + 0.23 + 0.28 + 0.42)/4 = 0.28.$$

- For a new/test house with a house size of  $\geq 2.5$ , we will predict that its price is the average of the houses in our training set whose size is  $\geq 2.5$ , i.e.,

$$(0.53 + 0.75 + 0.80)/3 = 0.6933.$$

Python  
demo

## Python for Decision Tree Regression

```
2 # Lecture 9 (demo - decision tree & housing price)
3
4 import numpy as np
5 from sklearn.tree import DecisionTreeRegressor
6 import matplotlib.pyplot as plt
7
8 def our_own_tree(y):
9
10    # split data at first level
11    # L stands for left, R stands for right
12    yL, yR = find_best_split(y)
13
14    # compute prediction
15    yL_pred = np.mean(yL)*np.ones(len(yL))
16    yR_pred = np.mean(yR)*np.ones(len(yR))
17
18    y_pred = np.concatenate([yL_pred, yR_pred])
19
20    return y_pred
```

```

22 #Go through all possible thresholds to determine the best split based on MSE
23 def find_best_split(y):
24
25     # index represents last element in the below threshold node
26     sq_err_vec = np.zeros(len(y)-1)
27     meansq_err_vec = np.zeros(len(y)-1)
28     for index in range(0, len(y)-1):
29
30         # split the data
31         data_below_threshold = y[:index+1]
32         data_above_threshold = y[index+1:]
33
34         # Compute estimate
35         mean_below_threshold = np.mean(data_below_threshold)
36         mean_above_threshold = np.mean(data_above_threshold)
37
38         # Compute total square error
39         # Note that MSE = total square error divided by number of data points
40         below_sq_err = np.sum(np.square(data_below_threshold - mean_below_threshold))
41         above_sq_err = np.sum(np.square(data_above_threshold - mean_above_threshold))
42         sq_err_vec[index] = below_sq_err + above_sq_err
43         meansq_err_vec[index] = sq_err_vec[index]/len(y)
44
45     #print out MSE
46     print('MSE list for house size')
47     print(meansq_err_vec)
48     best_index = np.argmin(meansq_err_vec)
49     yL = y[:best_index+1]
50     yR = y[best_index+1:]
51     print('Minimum MSE = '+str(meansq_err_vec[best_index])+' at threshold index '+str(best_index+1))
52     return yL, yR
53

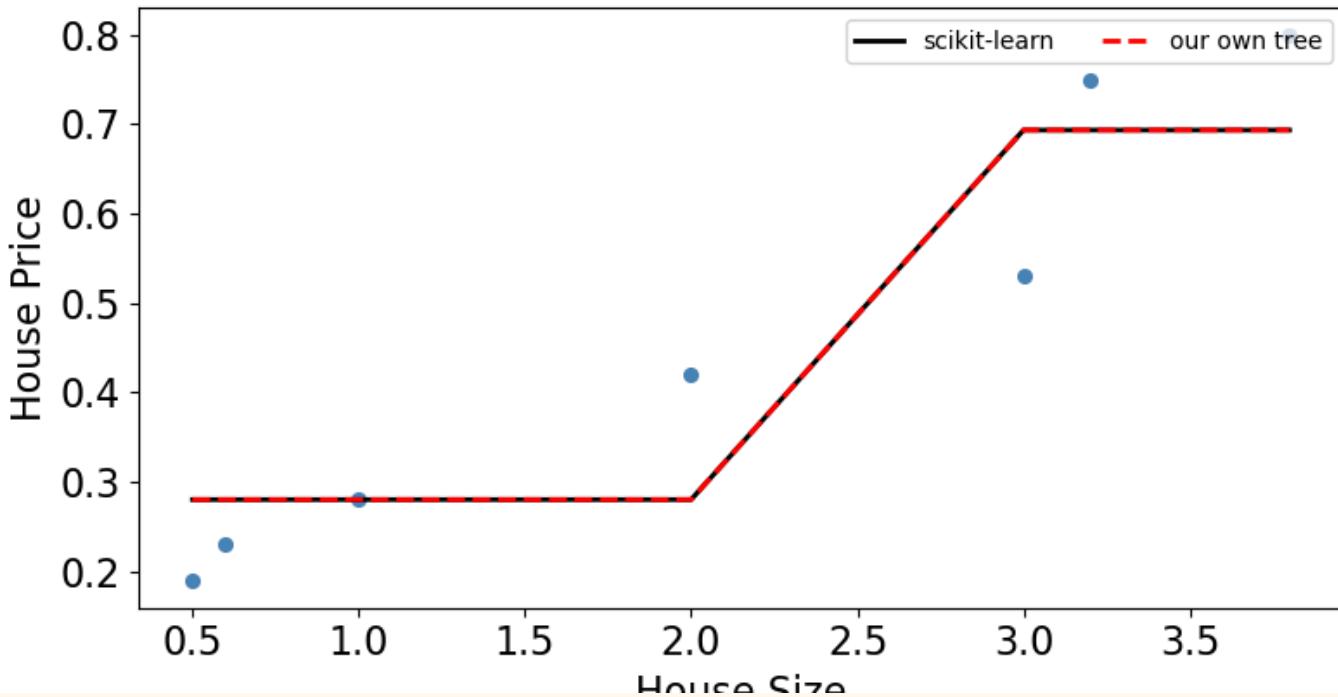
```

```

55 #main
56 #Input (house size and rooms) and output (price)
57 S = np.array([0.5, 0.6, 1.0, 2.0, 3.0, 3.2, 3.8])
58 R = np.array([2,1,3,5,4,6,7])
59 P = np.array([0.19, 0.23, 0.28, 0.42, 0.53, 0.75, 0.80])
60
61 #sort
62 sort_index = S.argsort()
63 S = S[sort_index]
64 R = R[sort_index]
65 P = P[sort_index]
66 print(S)
67
68 # scikit decision tree regressor
69 scikit_tree = DecisionTreeRegressor(criterion='squared_error', max_depth=1)
70 # Focus on House Size
71 scikit_tree.fit(S.reshape(-1,1), P) # reshape necessary because tree expects 2D array
72 scikit_tree_predict = scikit_tree.predict(S.reshape(-1,1))
73
74 # Our own tree regressor
75 tree_predict = our_own_tree(P)
76
77 # Plot
78 plt.figure(0, figsize=[9,4.5])
79 plt.rcParams.update({'font.size': 16})
80 plt.scatter(S, P, c='steelblue', s=30)
81 plt.plot(S, scikit_tree_predict, color='black', lw=2, label='scikit-learn')
82 plt.plot(S, tree_predict, color='red', linestyle='--', lw=2, label='our own tree')
83 plt.xlabel('House Size')
84 plt.ylabel('House Price')
85 plt.legend(loc='upper right', ncol=3, fontsize=10)
86 plt.savefig('Fig_Lec9_decisiontree.png')
87

```

- PS C:\Users\YTTRIUM> & C:/Users/YTTRIUM/anaconda3/envs/ee2211/python.exe
 [0.5 0.6 1. 2. 3. 3.2 3.8]
 MSE list for house size
 [0.04015476 0.02761714 0.01448095 0.01020952 0.01163571 0.03245714]
 Minimum MSE = 0.010209523809523809 at threshold index 4



- Slope is not slope, because it's flat after the next sample
- The slope just shows the split, which is at 2.5

### Reducing Tree Instability

- For both classification & regression trees, small perturbations to data can result in very different trees => low bias, high variance
- To reduce variance, we can perturb training set to generate M perturbed training sets
  - Train one tree for each perturbed training set
  - Average predictions across the M trees?
- For example, if we have 100 regression trees (trained from 100 perturbed training sets)
  - Given features  $x$  from new test sample, the  $i$ -th tree predicts  $f_i(x)$
  - Then final prediction is  $\frac{1}{100} \sum_{i=1}^{100} f_i(x)$
- For example, if we have 100 classification trees (trained from 100 perturbed training sets)
  - Given features  $x$  from new test sample, the  $i$ -th tree predicts  $g_i(x)$
  - Then final prediction is the most frequent class among 100 predictions  $g_1(x), g_2(x), \dots, g_{100}(x)$

- Tree has low bias, high variance
- Apply small change to data, get new tree, get many trees to and final prediction = average prediction of the trees for continuous or the most frequency class predicted for classification
- By making the tree more stable, it will be more generalisable to test data later
- Only add noise to input features, never change the output (regression or classification)
  - has the same number of sample each time it's perturbed
  - Decision trees not for categorical input so no issue with that

## Random Forest

- To perturb data, we can apply “bootstrapping” to the training set to create a new “bootstrapped” dataset
- Bootstrapping is procedure in which we sample data with replacement
  - For example, given training set with 3 data samples  $\{x_1, y_1\}$ ,  $\{x_2, y_2\}$ ,  $\{x_3, y_3\}$ , a bootstrapped training set might comprise sample 1  $\{x_2, y_2\}$ , sample 2  $\{x_2, y_2\}$  and sample 3  $\{x_1, y_1\}$
  - Bootstrapped dataset is the same size as original dataset
  - Bootstrapped dataset might contain repeated samples
  - Bootstrapped dataset might not contain some samples from original dataset
- Bootstrapping is also applied outside of random forest
- Sample training data set with replacement
- Such perturbation does not affect the “correct” mapping for input to output like multiplying noise
- has the same number of sample each time

---

## **Algorithm:** Random Forest Learning

---

**Input:** parameter  $max\_trees$  &  $N$  training samples

**Output:** Forest

```
1 for  $t \leftarrow 1$  to  $max\_trees$  do
2     dataset  $\leftarrow$  bootstrap( $N$  training samples)
3     trees[t]  $\leftarrow$  TreeLearning(dataset)
4 forest  $\leftarrow$  average(trees)
5 return forest
```

---

- To increase randomness, when training the trees, instead of looking at all features when considering how to split a node, we can randomly look at a subset (e.g., square root of the total number of features)
  - Looking at fewer features reduces overfitting also

### Random forest vs Tree

For bias, if we compare these two methods- I would say both a single decision tree and a Random Forest can have similar levels of bias, particularly when the trees in the forest are allowed to grow deep.

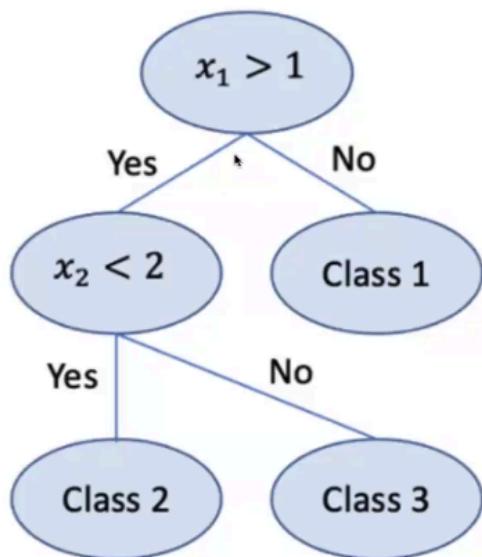
Bias is a measure of the error introduced by approximating a real-world problem (which may be extremely complex) by a much simpler model. Since each individual tree in a Random Forest is typically allowed to grow deep (unless explicitly pruned), they are, individually, as likely to capture the complex patterns in the data as a single decision tree, thereby having a similar squared bias. The forest does not reduce bias because it's the average of the biases of its trees, and each tree has a similar bias as a well-trained single decision tree.

## Summary and In Class Quiz

- Decision tree: series of binary decisions to arrive at prediction of target variable
- Classification tree predicts discrete target class
- Classification tree learning
  - Impurity measures: Gini, Entropy, Misclassification rate
  - For each leaf node, find best feature and threshold to split node to minimize impurity
- Regression tree predicts continuous target
  - Minimize MSE instead of impurity
- Random forest
  - Generate multiple bootstrapped training sets
  - Train on each bootstrapped training set & average

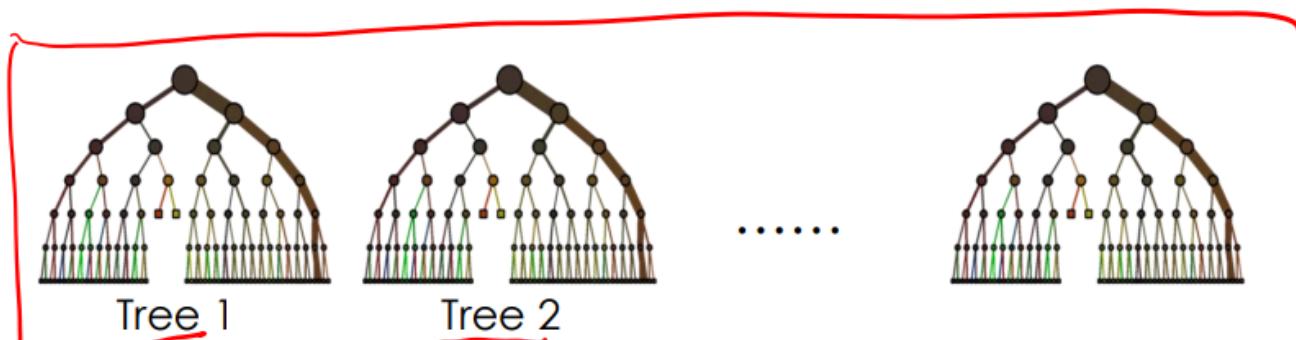
Given the decision tree below, what would a test data point with features ( $x_1 = 2$ ,  $x_2 = 4$ ) be classified as?

Jin Yueming  10



## Lec 10: Performance Issues

- We would like to train a Random Forest for face classification (i.e., to tell an image is a human face or not)
    - We will have one dataset to train the Random Forest
    - We will have tunable (hyper)parameters for the Random Forest.  
For example, ***the number of trees*** in the Random Forest
      - Shall we use 100 trees?      parameter candidate # 1
      - Shall we use 200 trees?      .....      .....      # 2
      - ...                                500
- We need to decide on the parameter
- parameter selection



- Need to decide how many trees to use, this is a parameter (or Hyper parameter)
- Also called parameter selection and it has to be done before running training
- hyper parameter
  - polynomial regression: polynomial order
  - Random forest: number of trees

- We would like to train a Random Forest for face classification (i.e., to tell an image is a human face or not)
    - We will have one dataset to train the Random Forest
    - We will have tunable (hyper)parameters for the Random Forest.  
For example, **the number of trees** in the Random Forest
      - Shall we use 100 trees?
      - Shall we use 200 trees?
      - ...
- We need to decide on the parameter #2 parameter selection
- Once we decide the number of trees, we will apply the Random Forest with the selected parameter on unseen test data.

## Test Data



Yes!

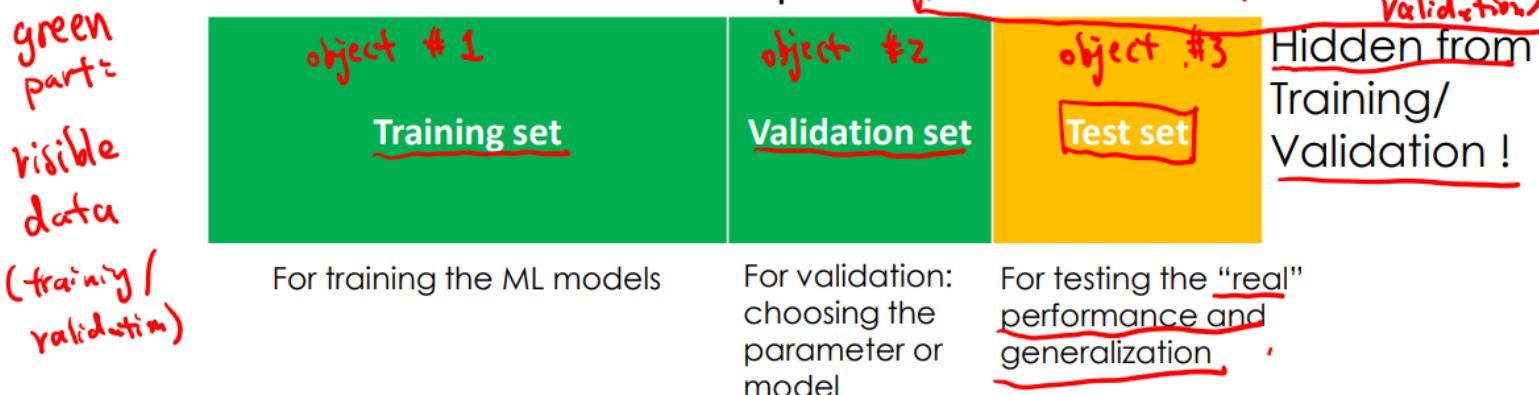


No!

- after selecting a hyper parameter, train then test with unseen test data

## Dataset Partition: Training/Validation/Testing

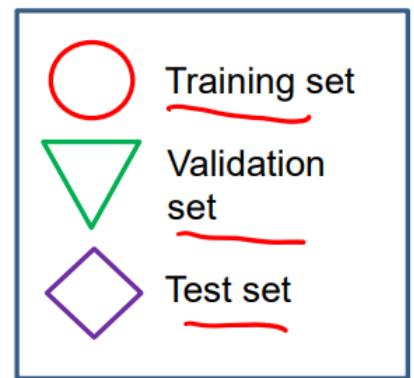
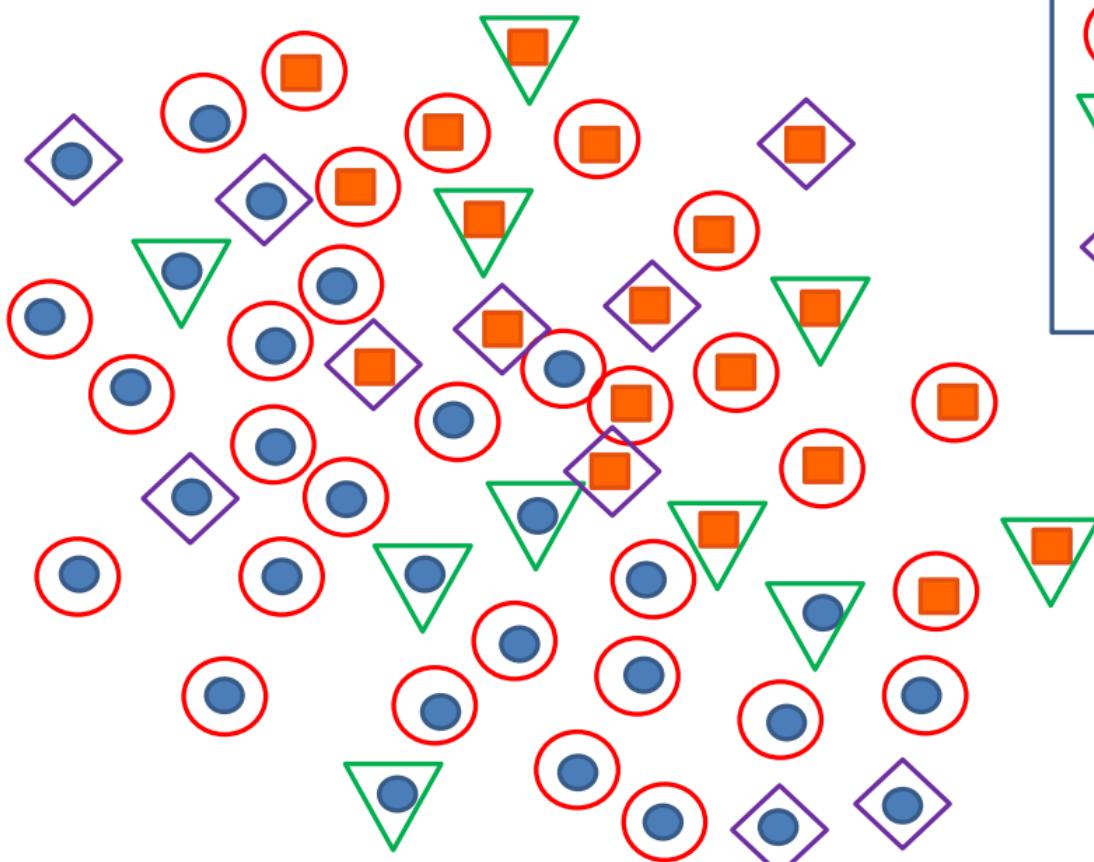
- In real-world application,
  - We don't have test data, since they are unseen
  - Imagine you develop a face detector app, you don't know whom you will test on
- In lab practice,
  - We divide the dataset into three parts



– NEVER touch [test data] during training/validation!!!

- To mimic actual usage in lab, need to divide the data into three part
  - training set - visible during training, use for training
  - Validation set - visible during training, used to check the parameter selected
  - Test set - cannot be seen in training, only used after the model is trained, used to imitate the real world performance

## Example





For training the ML models

For validation:  
choosing the  
parameter or  
model

For testing the "real"  
performance and  
generalization

1- Fold CV

Example: Assume I want to build a Random Forest. I have a parameter to decide: shall I have

- 100 Trees? Parameter Candidate #1
- 200 Trees? #2

What we do next is to use the training set to train two classifiers,

1)  $C_1$ : Random Forest with 100 trees, and 2)  $C_2$ : Random Forest with 200 trees

They have the following accuracy:

1.  $C_1$ : Random Forest with 100 trees: validation accuracy 90%
2.  $C_2$ : Random Forest with 200 trees: validation accuracy 88%

apply  $C_1$  on validity set.  
apply  $C_2$  on validity set.

Which one to choose for real application, i.e., testing?

The one with higher validation accuracy, i.e., Random Forest with 100 trees!

- With the two parameter candidates, C1 and C2, random forest with 100 trees and 200 trees
- then training the model twice based on C1 and C2
- Then test the parameter candidates with the validation test
  - validation is not seen during initial training
- Then the final parameter to be chosen should be the one with higher performance during validation test
- The final achievement is the optimal parameter chosen

150

# Python Demo: lec10.ipynb



For training the ML models

For validation:  
choosing the  
parameter or  
modelFor testing the "real"  
performance and  
generalization

Load Dataset Split it into Train:Val:Test = 100:25:25

```
[76]: ##---- load data from scikit ----##
import numpy as np
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import sklearn
print("scikit-learn version: {}".format(sklearn.__version__))
from sklearn.datasets import load_iris

## Set Seed
seed = 20

## Load dataset
iris_dataset = load_iris()
X = np.array(iris_dataset['data'])
y = np.array(iris_dataset['target'])

## one-hot encoding
Y = list()
for i in y:
    letter = [0, 0, 0]
    letter[i] = 1
    Y.append(letter)
Y = np.array(Y)

## Random shuffle data and train-test split
test_Idx = np.random.RandomState(seed=seed).permutation(Y.shape[0])
X_test = X[test_Idx[:25]]
Y_test = Y[test_Idx[:25]]
X = X[test_Idx[25:]]
Y = Y[test_Idx[25:]]
```

- Problem Setup

- Dataset used: IRISdataset
  - Link: [https://scikit-learn.org/stable/datasets/toy\\_dataset.html#iris-dataset](https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-dataset)
- Training/Validation/Test: 100/25/25
- Machine Learning Task and Model: Polynomial regression
- Parameters to select: Order 1 to 10

In the Final, no coding questions for Xinchao's part!

- Proportion of partition is based on choice
- Parameter for polynomial regression is the order of polynomial

## k-fold Cross Validation

- A more generalised version for validation
- k is chosen by user
- to avoid unlucky outlier when splitting training validation test set, repeated the splitting as fold

### Step 1

- In practice, we do the **k-fold cross validation**

Test

4-fold cross validation

k: set by user

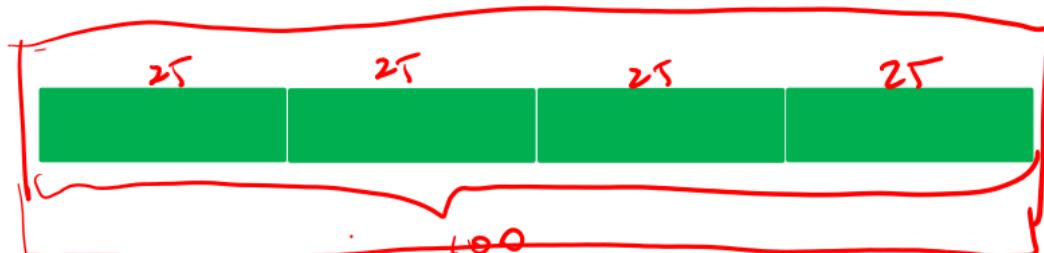
Step 1: take out **test set** from the dataset

## Step 2

- In practice, we do the **k-fold cross validation**

4-fold cross validation  
 $K=4$

Test



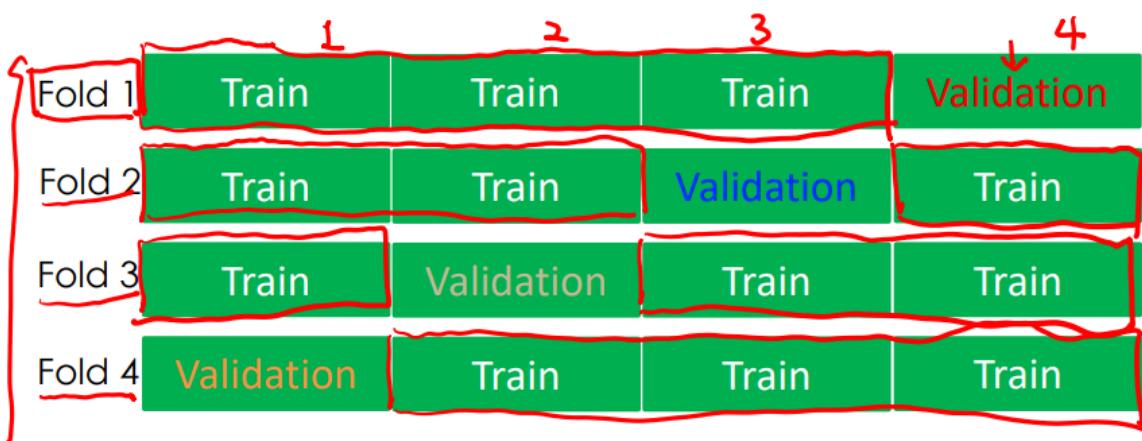
Step 2: We partition the *remaining part of the dataset* (after taking out the test set), into  $k$  equal parts (equal in terms of number of samples).

- Split by  $k$  parts

## Step 3

Test

4-fold cross validation



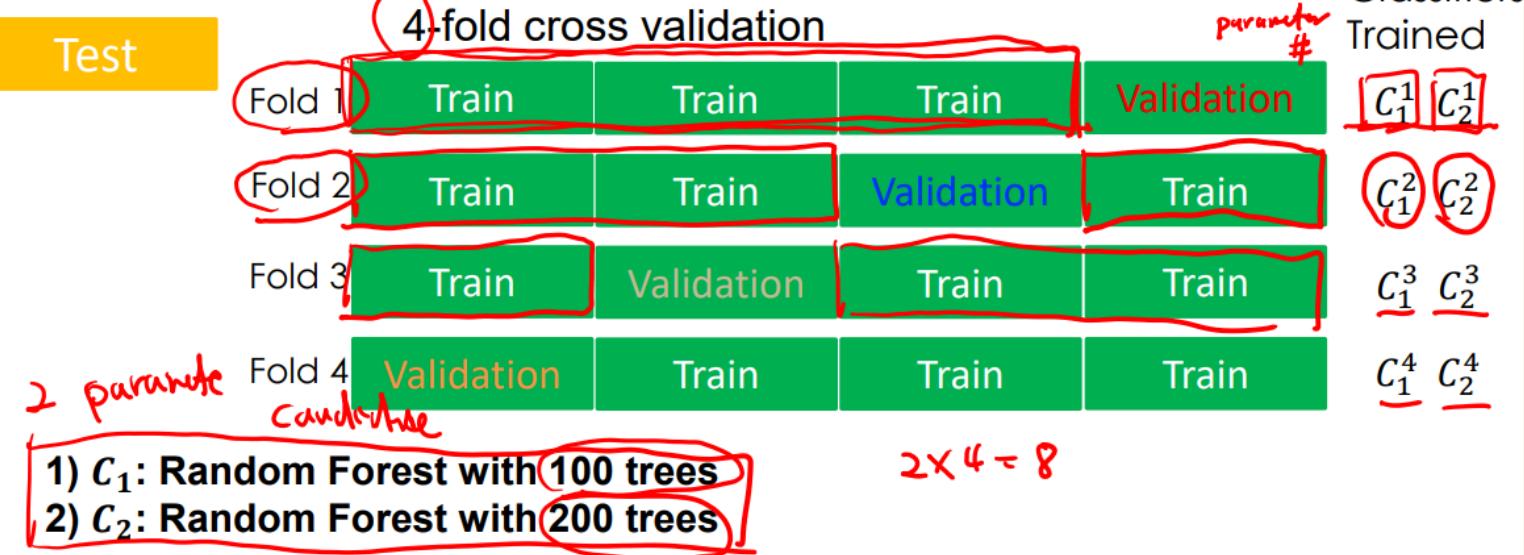
Step 3: We run  $k$  *folds* (i.e.,  $k$  times) of experiments.

Within each fold, we use one part as validation set, and the  $k-1$  remaining parts as training set. We use different validation sets for different folds.

- Use one part out of  $k$  parts for validation, the rest for training
- run for  $k$  times
- Simple case is 1-fold

# k-fold Cross Validation

- In practice, we do the k-fold cross validation

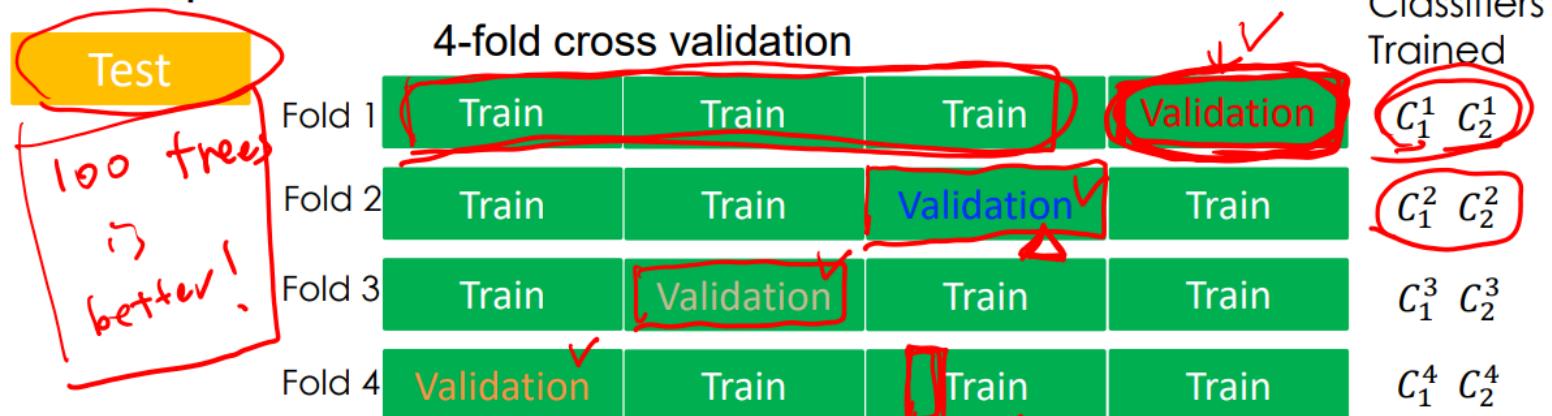


Step 3.1: Within each fold, if we have  $n$  parameter/model candidates, we will train  $n$  models, and we check their validation performance.

- for k in fold
  - for n in parameter candidates
    - do training
- $C_n^k$  mean applying parameter  $C_n$  to fold  $k$
- Total run for  $k * n$  times

## Step 4

- In practice, we do the **k-fold cross validation**



Example: which one to select for test?

one sample used once for validation  
for training

	Fold 1 Accuracy on Validation Set 1	Fold 2 Accuracy on Validation Set 2	Fold 3 Accuracy on Validation Set 3	Fold 4 Accuracy on Validation Set 4	Average Accuracy on All Validation Sets
Classifier with Param1 (e.g. 100 trees)	88% ✓	89% ✓	93% ✓	92% ✓	$\frac{1}{4}(88\% + 89\% + 93\% + 92\%) = 90.5\%$ ✓
Classifier with Param2 (e.g. 200 trees)	90% ✓	88% ✓	91% ✓	91% ✓	90%

Step 4: We select the parameter/model with best average validation performance over k folds.

17

- Final goal is to find the best parameter
- Find mean of accuracy to find the best parameter
- Then after finding the best parameter, train a new model with all the green data so no longer having training and validation set, this has more training data so model should be better
- Each sample in the whole data set is used once for validation and  $k - 1$  times for training

## Other common partitioning:

- 10-Fold CV
- 5-Fold CV
- 3-Fold CV

We may decide on the size of the test set, for example, 15%, 20%, 30% of the whole dataset, the rest for training/validation.

- Partition k and test set proportion does not affect much and set by user

- The **test set** contains the examples that the learning algorithm has **never seen before**,
- So **test performance** shows how well our model **generalizes**.

Example:

Xinchao uses **k-fold** cross validation to obtain an optimal parameter for his model (e.g., **decision tree**). This parameter, on the test set, achieves accuracy of 0.8.

Yueming uses **k-fold** cross validation to obtain an optimal parameter for her model (e.g., **random forest**). This parameter, on the test set, achieves accuracy of 0.9.

We can say, Yueming's model generalizes better than Xinchao's model.

Take home message:

Validation performance -> **Selecting parameters!**

Test performance -> The "real" performance of a **model with selected parameter!**  
*(final)*

- After finding best parameter, training with whole green data set then test with test set
- Then whichever model performs better in the test set, generalise better

- Validation is however not always used:
  - Validation is used when you need to pick parameters or models
  - If you have no models or parameters to compare, you may consider partition the data into only training and test

## Evaluation Metrics

Evaluating the quality of a trained machine learning system

### Evaluating Regression

## Regression

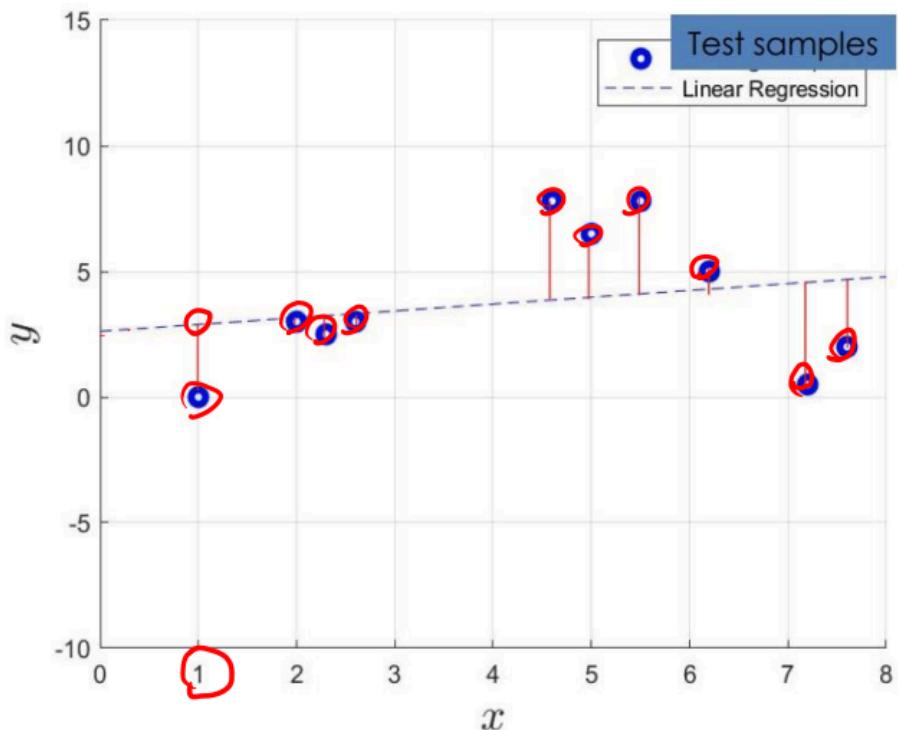
### ✓ Mean Square Error

$$(MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n})$$

### ✓ Mean Absolute Error

$$(MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n})$$

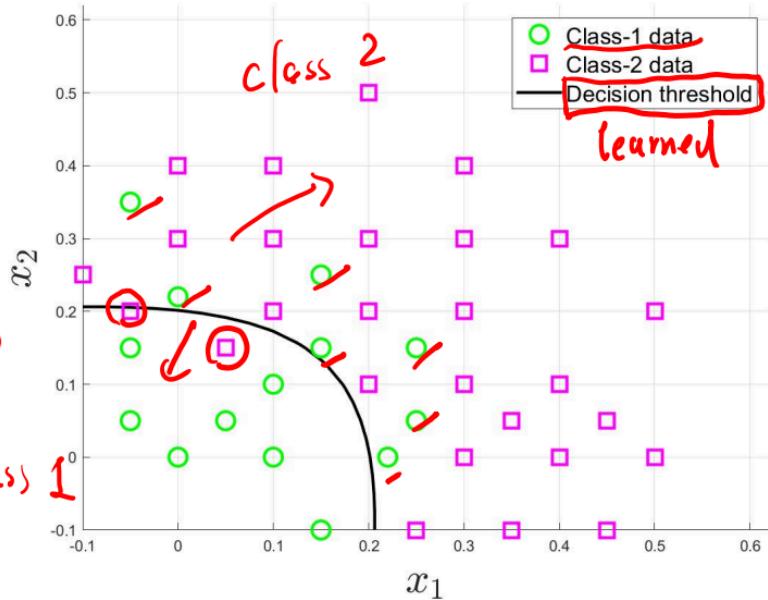
where  $y_i$  denotes the target output and  $\hat{y}_i$  denotes the predicted output for sample i.



- Predicted vs actual

# Classification

	Class-1: Positive Class Class-2: Negative Class
<b>Confusion Matrix</b>	
Class-1 (predicted)	Class-1 (predicted)
Class-1 (actual)	7 (TP) 7 (FN)
Class-2 (actual)	2 (FP) 25 (TN)



TP: True Positive  
 FN: False Negative (i.e., **Type II Error**)  
 FP: False Positive (i.e., **Type I Error**)  
 TN: True Negative

- By default class 1 is positive class and class 2 is negative class
- True means correctly classified, False means misclassified
- Confusion matrix to evaluate the classification

# Evaluation Metrics

## Classification

- How many samples in the dataset have the real label of Class-2?

$$2 + 25 = 27$$

- How many samples are there in total?

$$7 + 7 + 2 + 25 = 41$$

- How many samples are correctly classified? How many are incorrectly classified?

$$7 + 25 = 32$$

	Class-1 (predicted)	Class-2 (predicted)
Class-1 (actual)	7 (TP) 7 (FN)	2 (FP) 25 (TN)
Class-2 (actual)		

## Confusion Matrix for Binary Classification

	$\hat{P}$ (predicted)	$\hat{N}$ (predicted)	
P (actual)	TP	FN	Recall $TP/(TP+FN)$
N (actual)	FP	TN	Precision $TP/(TP+FP)$
			Accuracy $(TP+TN)/(TP+TN+FP+FN)$
			# correctly classified sample / total # of sample

- Higher Recall, Precision and Accuracy better

## Cost Matrix for Binary Classification

	$\hat{P}$ (predicted)	$\hat{N}$ (predicted)
P (actual)	$C_{p,p} * \boxed{\text{TP}}$	$C_{p,n} * \boxed{\text{FN}}$
N (actual)	$C_{n,p} * \boxed{\text{FP}}$	$C_{n,n} * \boxed{\text{TN}}$

**Total cost:**  
 ~~$C_{p,p} * \text{TP}$~~  +  
 ~~$C_{p,n} * \text{FN}$~~  +  
 ~~$C_{n,p} * \text{FP}$~~  +  
 ~~$C_{n,n} * \text{TN}$~~

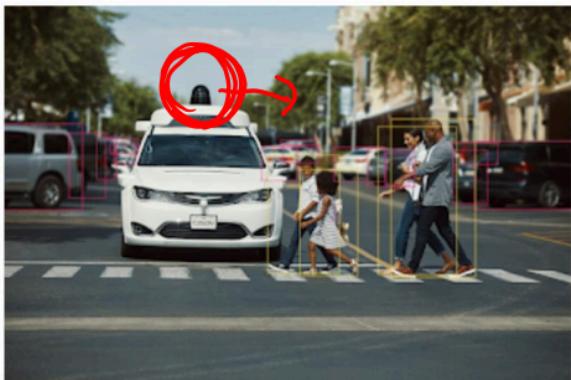
Main Idea: To assign different **penalties** for different entries. Higher **penalties** for more severe results.

Usually,  $C_{p,p}$  and  $C_{n,n}$  are set to 0;  $C_{n,p}$  and  $C_{p,n}$  may and may not equal

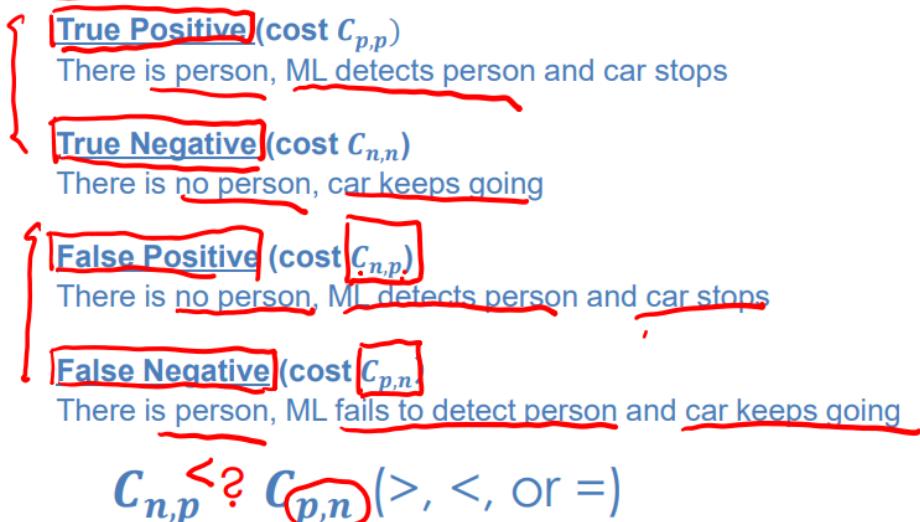
- Cnp and Cpn needs to be chosen based on the application
- Total cost is a value not for training, but to check the model

- Example of cost matrix

- Assume we would like to develop a self-driving car system
- We have an ML system that detects the pedestrians using camera, by conducting a binary classification
  - When it detects a person (positive class), the car should stop
  - When no person is detected (negative class), the car keeps going



Credit: automotiveworld.com



1 10

27

- False positive is less severe than false negative hence  $C_{np} < C_{pn}$
- Give higher penalty to more severe result

- Handling unbalanced data

- Assume we have 1000 samples, of which 10 are positive and 990 are negative

- Accuracy =  $990/1000 = 0.99!$

- Yet, half of the Class-1 are Classified to Class-2!

	Class-1 (predicted)	Class-2 (predicted)
Class-1 (actual)	5 (TP)	5 (FN)
Class-2 (actual)	5 (FP)	985 (TN)

The goal is to highlight the problems of the results!

In this case, we shall

- 1) Use cost matrix, assign different costs for each entry
- 2) Use Precision and Recall! Precision = 0.5 and Recall = 0.5

- Precision and recall are low, Even though accuracy is high
- Use Precision and recall for biased data
- Can also give high cost of FP to give more attention to class 1, make it more cost sensitive

(True Positive Rate)  $\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$  Recall  
 (False Negative Rate)  $\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}}$

(True Negative Rate)  $\text{TNR} = \frac{\text{TN}}{\text{FP} + \text{TN}}$   
 (False Positive Rate)  $\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$

$\text{TPR} + \text{FNR} = 1$  (100% of positive-class data)  
 $\text{TNR} + \text{FPR} = 1$  (100% of negative-class data)

$\hat{P}$ (predicted)	$\hat{N}$ (predicted)	
P (actual)	TP	FN
N (actual)	FP	TN

Confusion Matrix Depends on Threshold for Classification

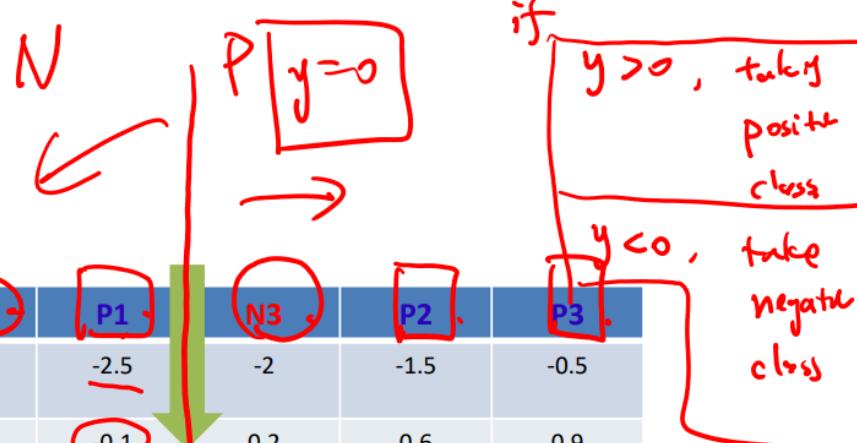
## Classification

Prediction function  $y = f(x)$

sorted by y

sample	N1	N2	P1	N3	P2	P3
input x	-4	-3	-2.5	-2	-1.5	-0.5
Prediction y	-1.1	-0.5	-0.1	0.2	0.6	0.9
Actual Label	-1	-1	1	-1	1	1

If threshold set to be  $y=0$ ,  
 N3, P2, P3 will be taken as +1  
 P1, N2, N1 will be taken as -1



$\hat{P}$ (predicted)	$\hat{N}$ (predicted)	
P (actual)	$P_2, P_3$ $TP = 2$	$P_1$ $FN = 1$
N (actual)	$N_3$ $FP = 1$	$N_1, N_2$ $TN = 2$

- The numbers in confusion matrix is based on some threshold for classification model

## Classification

Prediction function  $y = f(x)$



sample	N1	N2	P1	N3	P2	P3
input x	-4	-3	-2.5	-2	-1.5	-0.5
Prediction y	-1.1	-0.5	-0.1	0.2	0.6	0.9
Actual Label	-1	-1	1	-1	1	1

If threshold set to be  $y=0.4$ ,  
P2, P3 will be taken as +1  
N3, P1, N2, N1 will be taken as -1

$\hat{P}$ (predicted)	$\hat{N}$ (predicted)
P (actual)	$TP = 2$
N (actual)	$FP = 0$

- Changing the threshold will give different result

## Classification:

TP, FP, FN, TN will change wrt thresholds!

If threshold set to be  $y=0$ ,  
 N3, P2, P3 will be taken as +1  
 P1, N2, N1 will be taken as -1

$\hat{P}$ (predicted)	$\hat{N}$ (predicted)	
P (actual)	$TP = 2$	$FN = 1$
N (actual)	$FP = 1$	$TN = 2$

If threshold set to be  $y=0.4$ ,  
 P2, P3 will be taken as +1  
 N3, P1, N2, N1 will be taken as -1

$\hat{P}$ (predicted)	$\hat{N}$ (predicted)	
P (actual)	$TP = 2$	$FN = 1$
N (actual)	$FP = 0$	$TN = 3$

## Classification

*C classes*  
*CxC*

### Confusion Matrix for Multicategory Classification

	$P_{\widehat{1}}$ (predicted)	$P_{\widehat{2}}$ (predicted)		$P_{\widehat{C}}$ (predicted)
$P_1$ (actual)	$P_{1,\widehat{1}}$	$P_{1,\widehat{2}}$	...	$P_{1,\widehat{C}}$
$P_2$ (actual)	$P_{2,\widehat{1}}$	$P_{2,\widehat{2}}$	...	$P_{2,\widehat{C}}$
:	:	:	...	:
$P_C$ (actual)	$P_{C,\widehat{1}}$	$P_{C,\widehat{2}}$		$P_{C,\widehat{C}}$

- Will be  $c * c$  matrix for  $c$  classes

### Other Issues

- Computational speed and memory consumptions are also important factors
  - Especially for mobile or edge devices
- Other factors
  - Parallelable, Modularity, Maintainability
- Not focus of this module

## Practice Question

Suppose we have a dataset of 550 samples. We take out  $n$  samples as test set, and run k-fold cross validation on the remaining samples.



Within each fold, we know that, the number of training samples is three times as large as the number of validation samples, and two times as large as the number of test samples.  $1.5X$

1. What is  $k$ ?
2. What is  $n$ ?

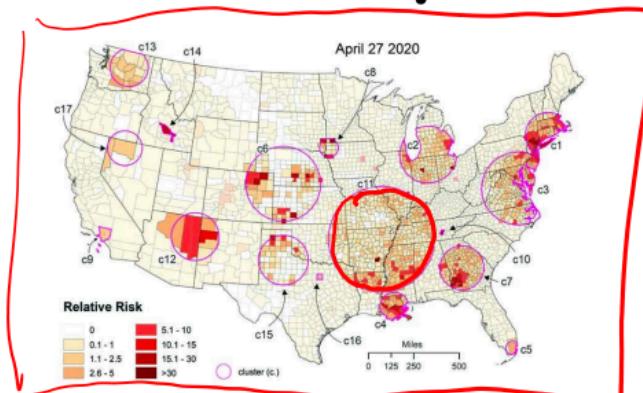
$$k = 4$$

$$4X + 1.5X = 5.5X = 550$$

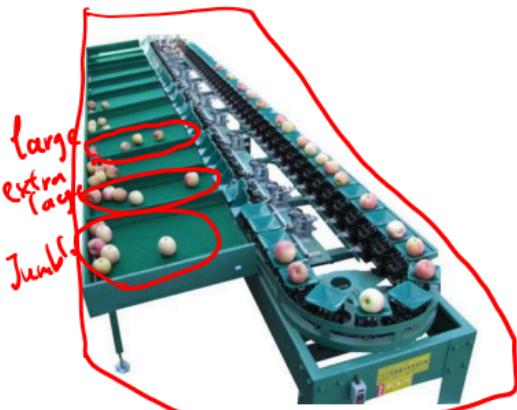
$$X = 100$$

$$3X + X + 1.5X = n = 1.5X = 150. \quad = 550.$$

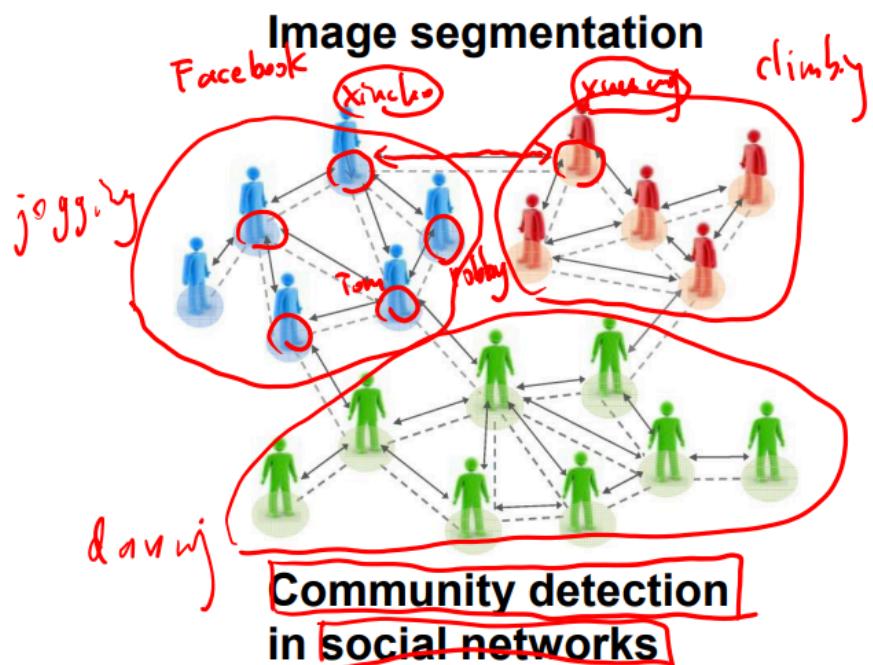
# Unsupervised Learning



Discovering Covid clusters



Business analysis



- Cluster and group are the same thing, cluster into groups, group into cluster
- Unsupervised means no actual output only input
- Covid clusters, colour of pixels for land and sea, size of apples, community detection by grouping social networks (by interests)

**Motivation:** we do not always have labeled data.

In **unsupervised learning**, the dataset is a collection of **unlabeled examples**  $\{\mathbf{x}_i\}_{i=1}^M$ . sample

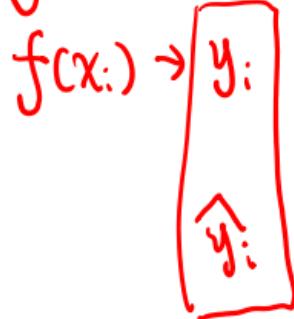
supervised learning

$$(\underline{x}_1, \underline{y}_1)$$

$$(\underline{x}_2, \underline{y}_2)$$

!

$$(\underline{x}_n, \underline{y}_n)$$



unsupervised learning

$$\underline{x}_1$$

$$\underline{x}_2$$

!

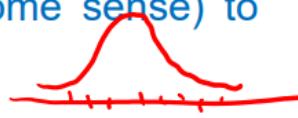
$$\underline{x}_n$$

Evaluation of unsupervised learning is hard:

- The absence of labels representing the desired behavior for your model means the absence of a solid reference point to judge the quality of your model.

# Main Approaches

- **Clustering** is not the only unsupervised learning task.
  - ✓ Groups a set of objects in such a way that objects in the same group (called a **cluster**) are **more similar** (in some sense) to each other than to those in other groups (clusters).
- **Density Estimation**
  - ✓ Models the probability density function (pdf) of the unknown probability distribution from which the dataset has been drawn.
- **Component Analysis**
  - ✓ Breaks down the data from the perspective of signal analysis.
- **Unsupervised Neural Networks**
  - ✓ **Autoencoder**



principle component analysis (PCA)



- Only clustering in EE2211
- Density estimation is to recover the PDF from the sample

## K-means Clustering

- The most popular clustering technique
- Hard clustering

## Basic/Naïve K-means Clustering

Looping between Assignment and Centroid Update

w

c

1. First, we choose  $K$  — the number of clusters. Then we randomly select  $K$  feature vectors, called centroids, to the feature space.
2. Next, compute the distance from each example  $x$  to each centroid  $c$  using some metric, like the Euclidean distance. Then we assign the closest centroid to each example (like if we labeled each example with a centroid id as the label).
3. For each centroid, we calculate the average feature vector of the examples labeled with it. These average feature vectors become the new locations of the centroids.
4. We recompute the distance from each example to each centroid, modify the assignment and repeat the procedure until the assignments don't change after the centroid locations are recomputed.
5. Finally, we conclude the clustering with a list of assignments of centroids IDs to the examples.

- Initial starting centroid can be on the sample or not
- Basic/naive is only one method of k mean clustering

```
# Define the k-means function
def kmeans_step(data, k, centroids):

  # Assign each data point to the closest centroid
  distances = np.sqrt(((data - centroids[:, np.newaxis])**2).sum(axis=2))
  labels = np.argmin(distances, axis=0)

  # Update centroids to be the mean of the data points assigned to them
  new_centroids = np.zeros_like(centroids)
  for j in range(k):
    new_centroids[j] = np.mean(data[labels == j], axis=0)

  # End if centroids no longer change
  if np.linalg.norm(new_centroids - centroids) < tolerance:
    print("End Clustering, Centroids no change.")
    # Return the original centroids and labels, and set end to True
    return centroids, labels, True
  else:
    # Return the centroids and labels, and set end to False
    return new_centroids, labels, False
```

- See Python code:

– lec11.ipynb

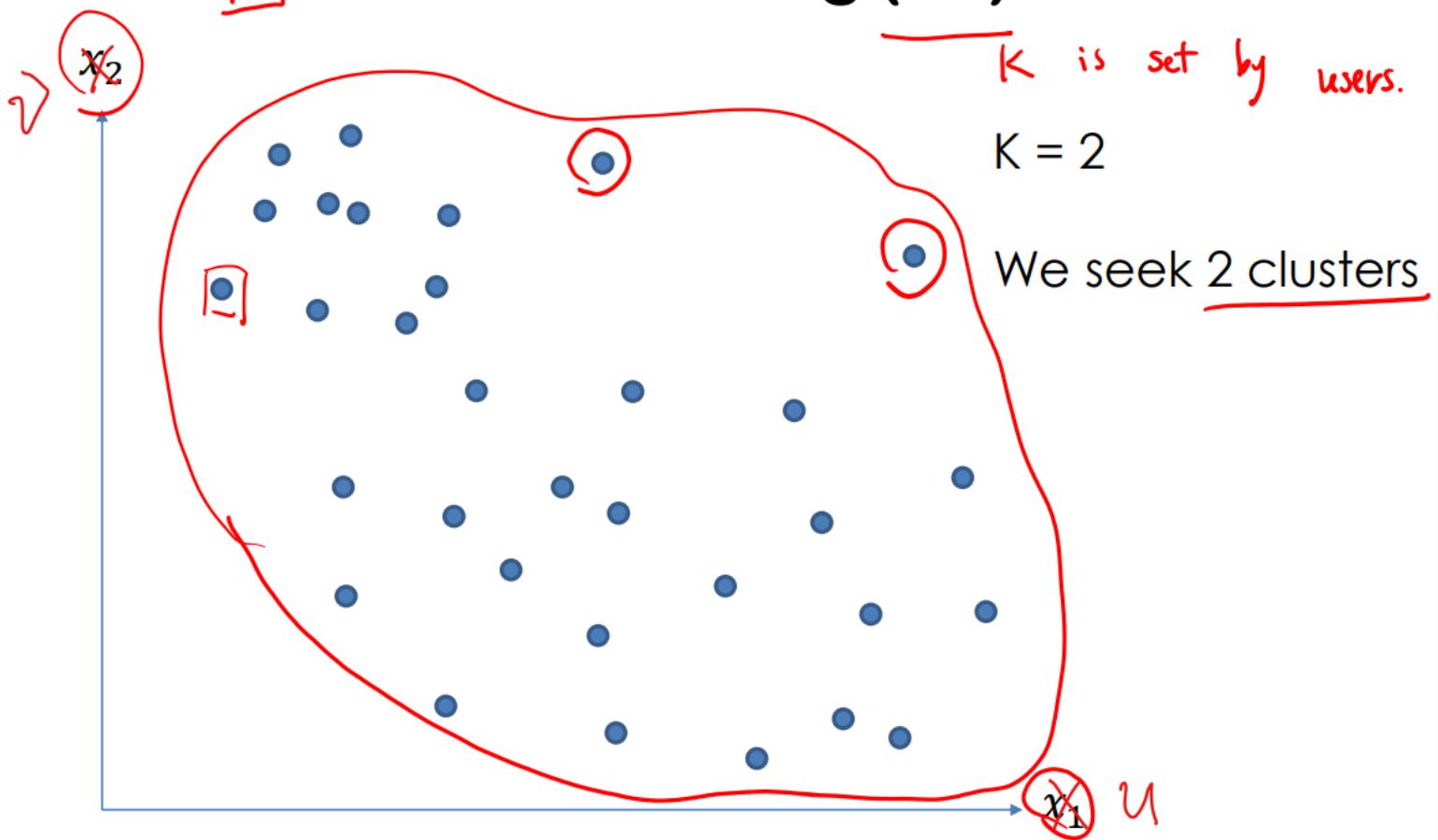
- See live demo at:

– lec11\_kmeans.html

All available in Canvas  
 Files\For Students\ Lecture Notes

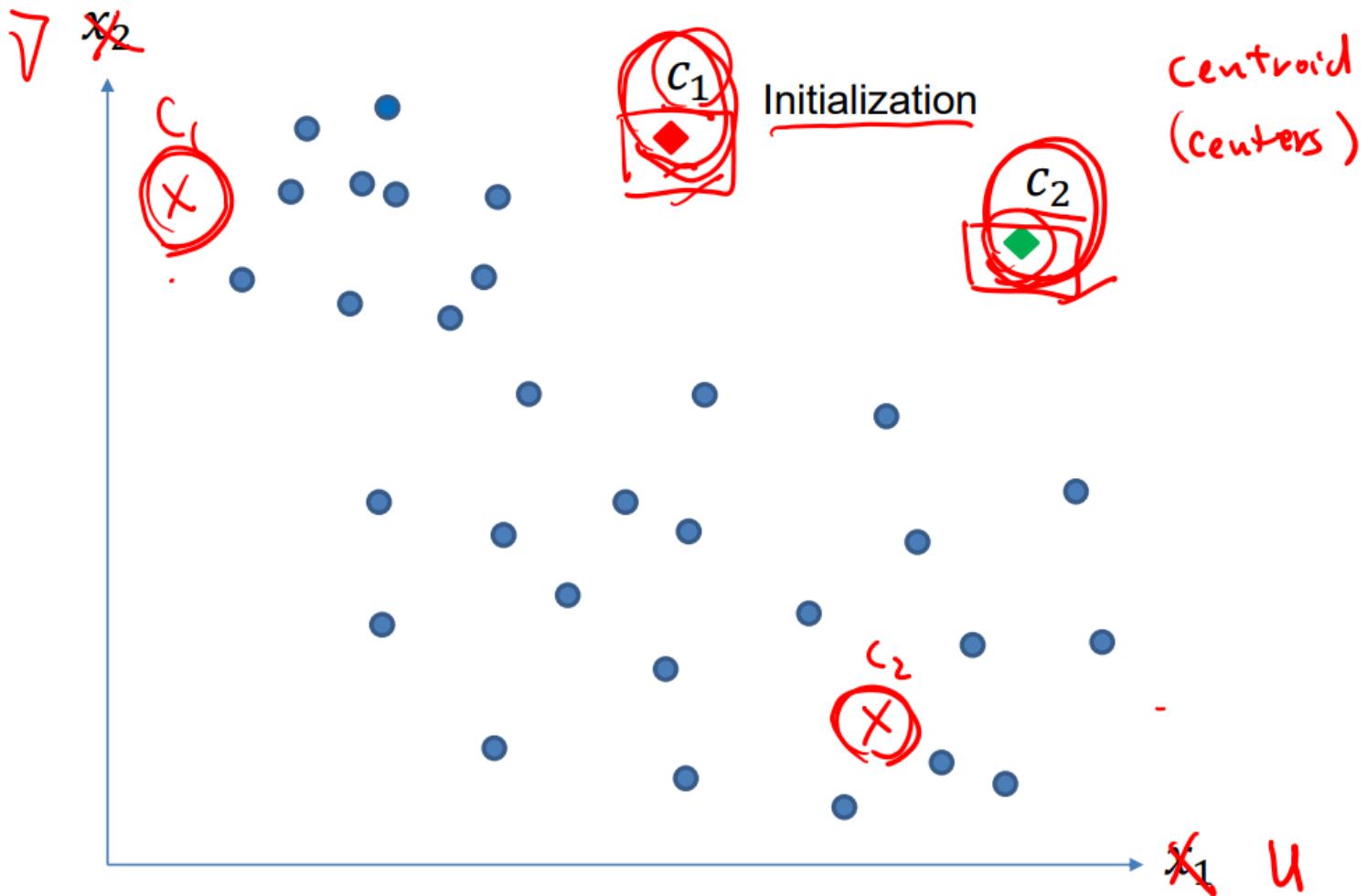
In the Final, no coding questions for Xinchao's part!

# K-means Clustering (2D)



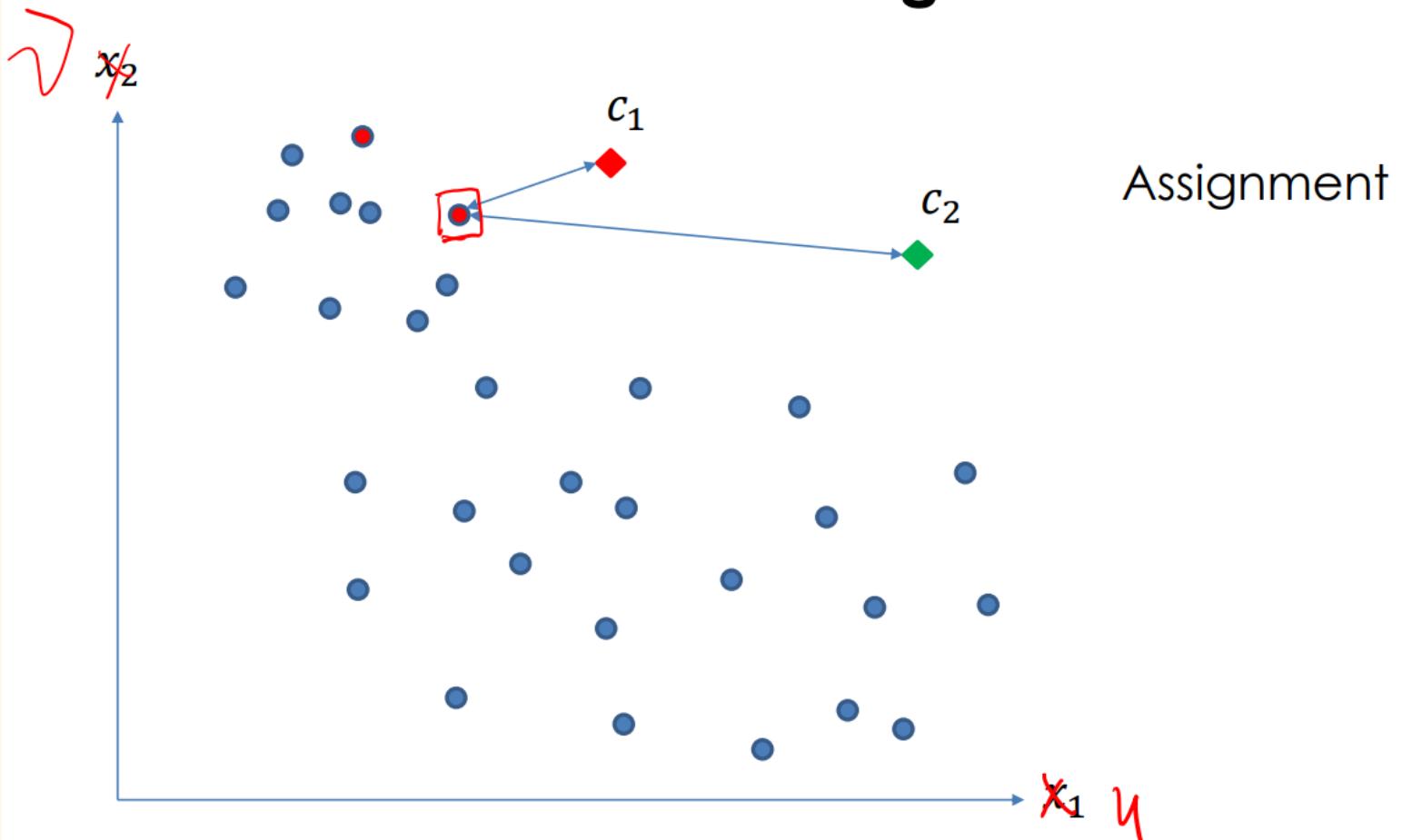
- $K$  is the number of cluster that the user wants, set by user
- $x_1$  and  $x_2$  are not input and output, they are the sample features

## Step 1: Initialisation

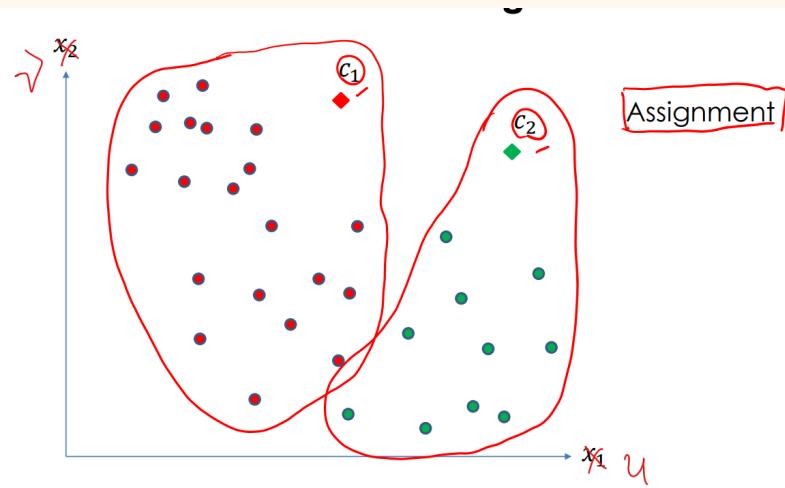


- Initialise centroid or centres, to be the centre of the first and second group
- Position is guessed

## Step 2: Assignment

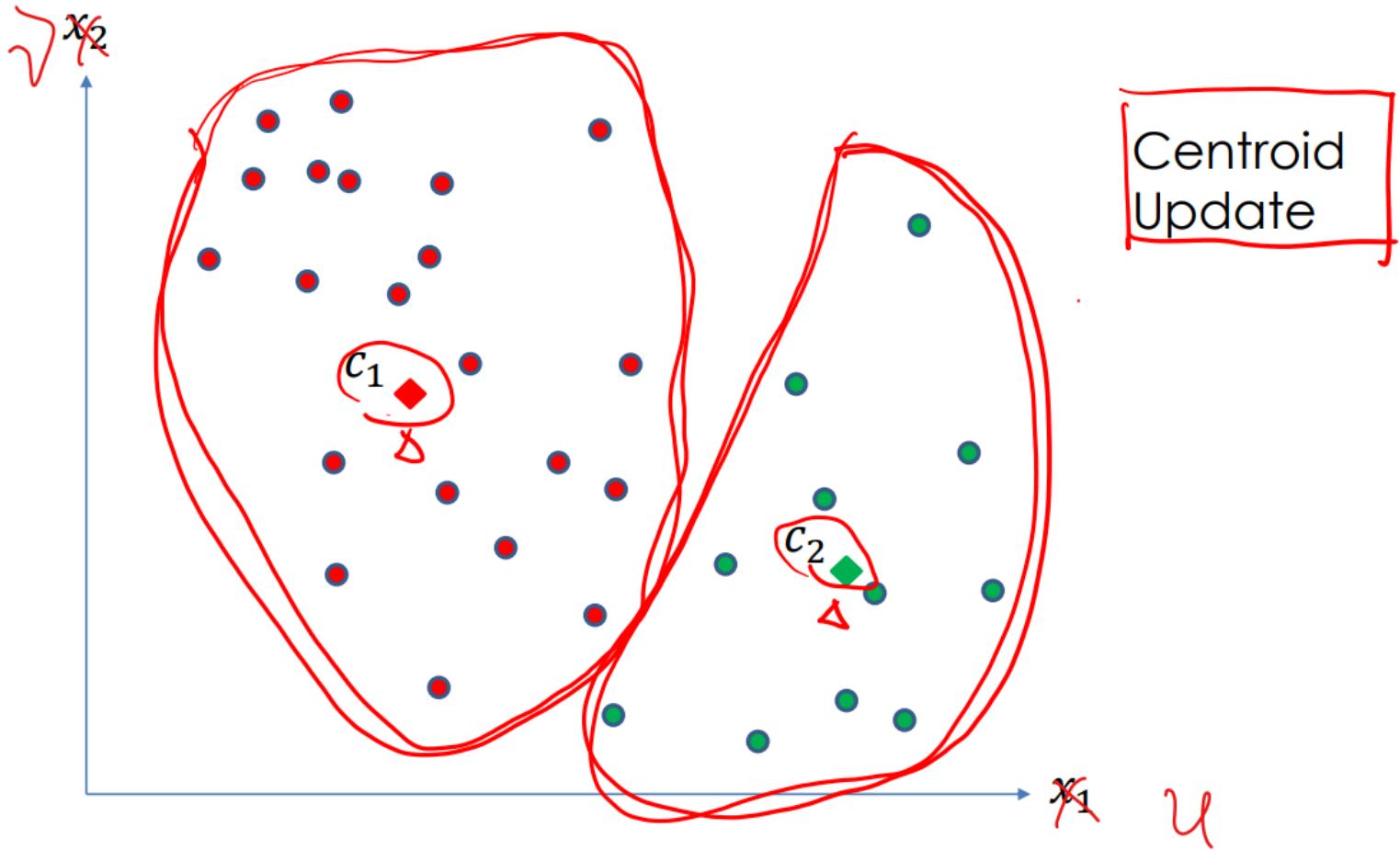


- For each sample in the feature space of  $x_1$  and  $x_2$ , assign them to  $C_1$  or  $C_2$  based on distance
- Assignment of sample to cluster, only one cluster per sample

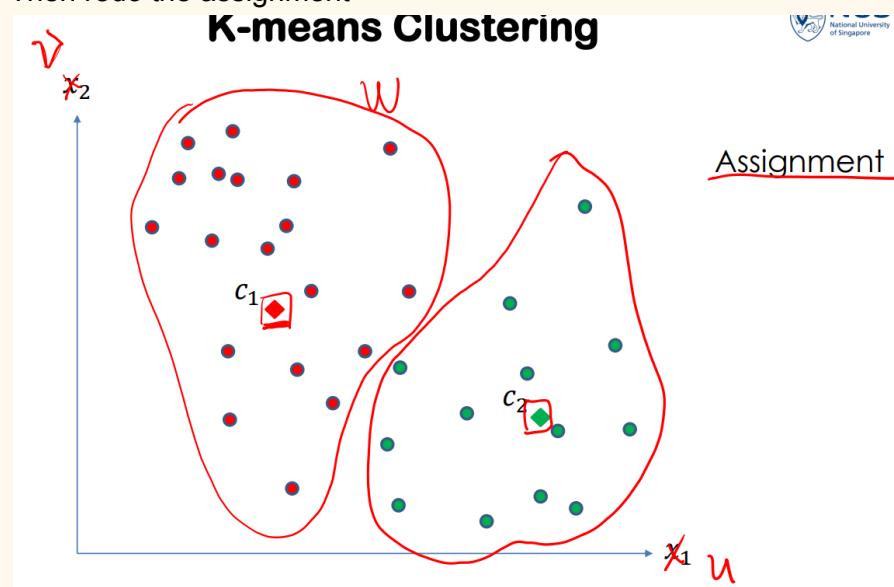


- - one iteration of assignment

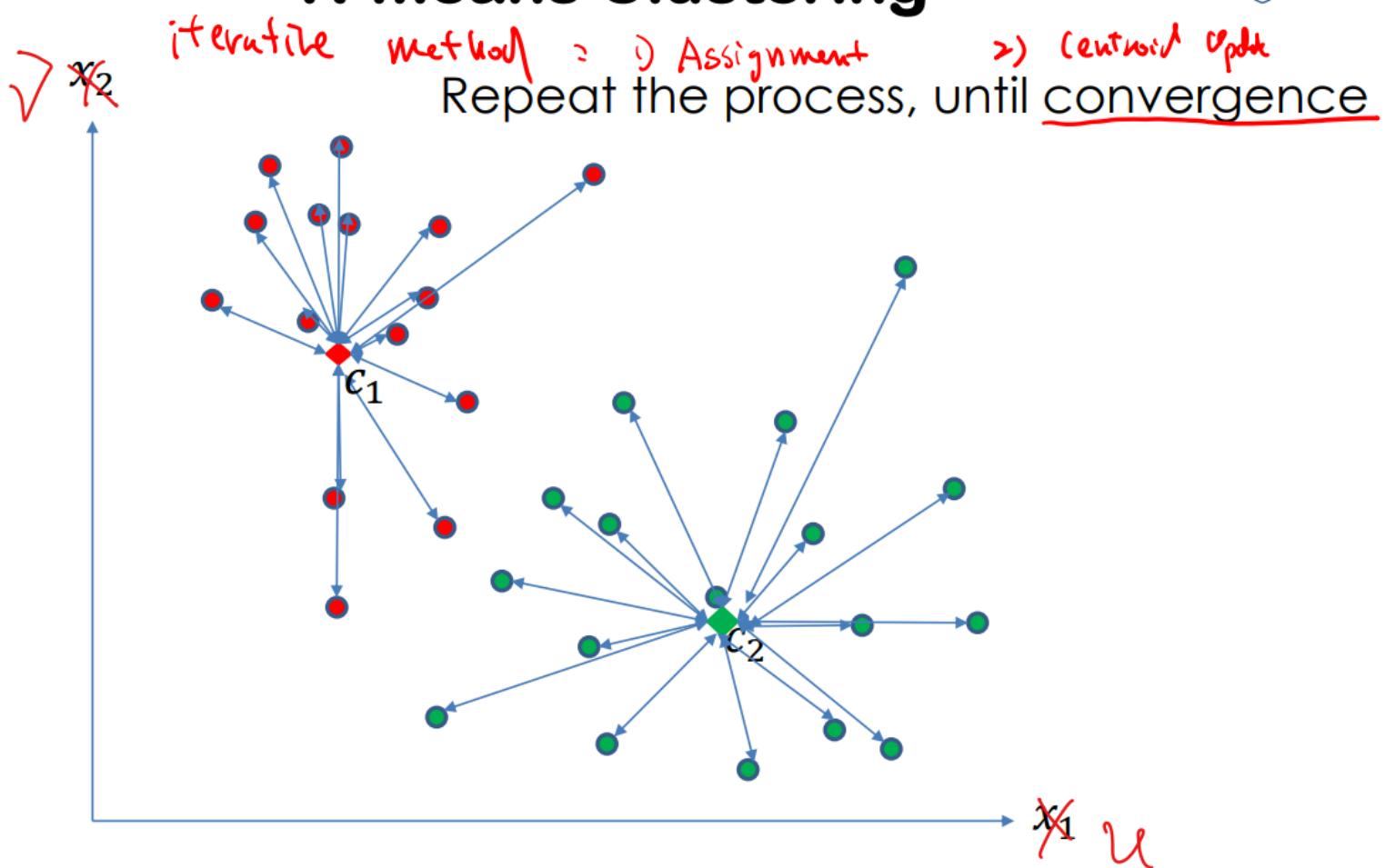
### Step 3: Centroid Update



- Update the new centroid by
- Need to update because the initial centroid may not be good
- Then redo the assignment



#### Step 4: Convergence



- Iterate and repeat assignment and centroid update until convergence (assignment or/and centroid doesn't change)

## Optimization Objective Function (within-cluster variance)

$m$ : # of samples;  $i$ : index of samples

$K$ : # of clusters;  $k$ : index of clusters

Minimize  $J$

$w_{ik} \in \{0, 1\}$   
 i-th sample  
 $\nwarrow$   
 k-th cluster

$$J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|x_i - c_k\|^2 \quad (1)$$

sample # $i$   
 centroid of cluster  $k$

$$w_{ik} = 1$$

compute  $\|x_i - c_k\|^2$

The term  $w_{ik}$  is equal to 1 for data point  $x_i$  if the data point belongs to cluster  $S_k$ , else  $w_{ik} = 0$ .

$C \rightarrow W$

Note: The optimization objective function was called  $C(w)$  in Lecture 8. Here, we use  $J$  (with parameters  $w_{ik}$  and  $c_k$ ) so that it is differentiated from the centroids  $c_k$ .

$W \rightarrow C$

Ref: <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>  
[https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)

- K means is solving the above minimisation problem
- $w_{ik} = 1$  if sample  $i$  is assigned to cluster  $k$ , else 0
  - it's used so that the distance is only counter for from the sample to the centroid its assign to (within cluster variance)
- The variable to change to optimise is  $C_k$ 
  - from  $C_k$  can find  $W_{ik}$
  - or  $W_{ik}$  which can be find  $C_k$

Minimize  $J$

$$J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|x_i - c_k\|^2 \quad (1)$$

The term  $w_{ik}$  is equal to 1 for data point  $x_i$  if the data point belongs to cluster  $S_k$ , else  $w_{ik} = 0$ .

Note: The optimization objective function was called  $C(\mathbf{w})$  in Lecture 8. Here, we use  $J$  (with parameters  $w_{ik}$  and  $c_k$ ) so that it is differentiated from the centroids  $\mathbf{c}_k$ .

Ref: <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>  
[https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)

## Naïve K-means Algorithm

1. Assignment Step (fix  $\mathbf{c}$  and update  $\mathbf{w}$ ): Computing distances to all centroids

$x_i \in S_k$  ( $w_{ik} = 1$ ) if  $\|x_i - c_k\|^2 < \|x_i - c_j\|^2$  (else  $w_{ik} = 0$ ),  $i = 1, \dots, m; j, k = 1, \dots, K$ .

2. Update Step (fix  $\mathbf{w}$  and update  $\mathbf{c}$ ):

$$\frac{\partial J}{\partial c_k} = -2 \sum_{i=1}^m w_{ik} (x_i - c_k) = 0 \Rightarrow c_k = \frac{\sum_{i=1}^m w_{ik} x_i}{\sum_{i=1}^m w_{ik}}$$

Solving an optimization, i.e., setting derivative to 0

$c$  turns out to be the avg. of all samples

Note:  $\|\mathbf{x} - \mathbf{c}\| = \sqrt{\sum_{d=1}^D (x_d - c_d)^2}$  is called the Euclidean distance.  
 where  $\mathbf{x} = (x_1, x_2, \dots, x_D)$ ,  $\mathbf{c} = (c_1, c_2, \dots, c_D)$

- Assign sample  $i$  to group  $k$  if it's the closest to group  $k$  than other group
- Update  $c_k$  is just finding the average of all sample in the cluster  $k$

### 1. Assignment Step (fix $c$ and update $w$ ):

$$\mathbf{x}_i \in S_k \quad (w_{ik} = 1) \text{ if } \|\mathbf{x}_i - \mathbf{c}_k\|^2 < \|\mathbf{x}_i - \mathbf{c}_j\|^2 \quad (\text{else } w_{ik} = 0), \\ i = 1, \dots, m; \quad j, k = 1, \dots, K.$$

### 2. Update Step (fix $w$ and update $c$ ):

$$\frac{\partial J}{\partial \mathbf{c}_k} = -2 \sum_{i=1}^m w_{ik} (\mathbf{x}_i - \mathbf{c}_k) = 0 \Rightarrow \mathbf{c}_k = \frac{\sum_{i=1}^m w_{ik} \mathbf{x}_i}{\sum_{i=1}^m w_{ik}}$$

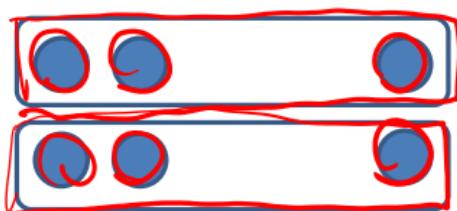
By repeating this two steps, the total loss  $J = \sum_{i=1}^m \sum_{k=1}^K w_{ik} \|\mathbf{x}_i - \mathbf{c}_k\|^2$ , is guaranteed to NOT increase (i.e., remain the same or decrease) until convergence.

Why? At Step 2: we compute the new mean, by solving an optimization, i.e., compute the derivative and set to zero, and solve  $\mathbf{c}_k$ .

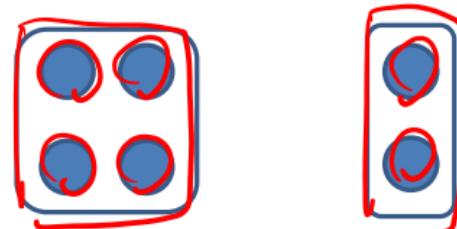
This means that, the new  $\mathbf{c}_k$  is guaranteed to give a smaller  $J$  value.

At Step 1: we only change the assignment, if the distance to the new centroid is smaller! In other words, we either remain in the old group, or change to a new group that is closer (i.e., gives a smaller  $J$ )

- Unfortunately, k-means is not guaranteed to find a global minimum, it finds only local minimum.
- Example:



K-means



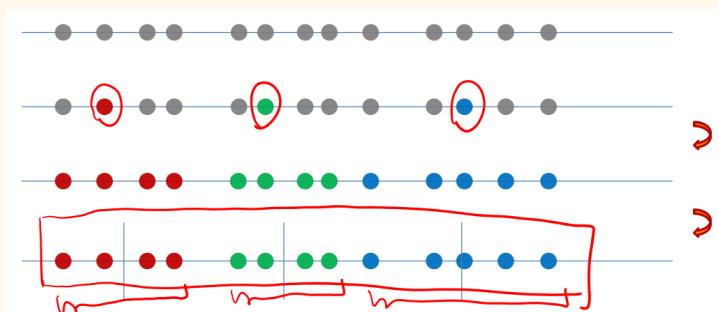
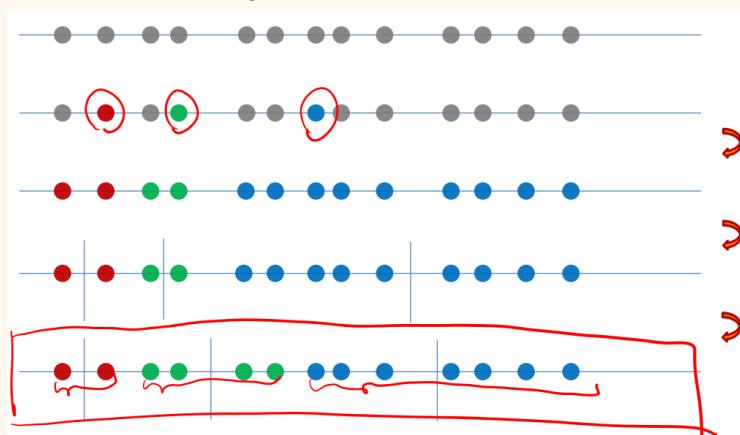
Optimal  $J$

- Finding the optimal  $J$  is NP-hard\*
- In practice, k-means clustering usually performs  $10^3$  well
- It can be very efficient, and its solution can be used as a starting point for other clustering algorithms

- To find the global minimum or optimal, need to do all possibility
  - for 10 samples and 3 clusters, there is  $10^3$  possibility, since for each sample it can be assigned to 3 clusters

## Initialization

### K-means Clustering Example (1D)



Different initializations give different clusters!

- **Initialization**

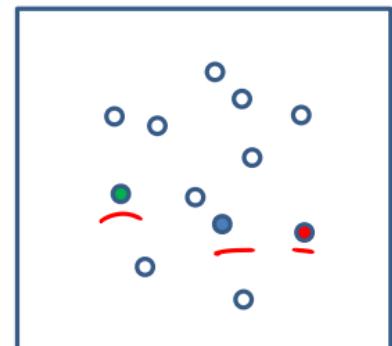
Forgy method:

- Randomly chooses  $k$  observations from the dataset and uses these as the initial means.

more popular

Initialization by centroid

$k=3$

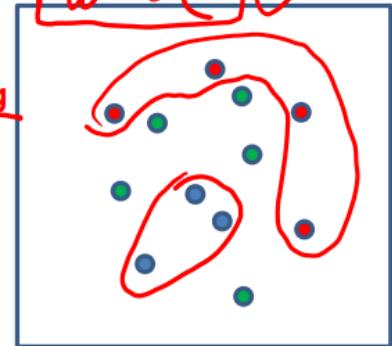


Random partition:

- First randomly assigns a cluster to each observation and then proceeds to the update step, thus computing the initial mean to be the centroid of the cluster's randomly assigned points

Initialization by grouping

$k=3$



Ref: [https://en.wikipedia.org/wiki/K-means\\_clustering#Standard\\_algorithm\\_\(naive\\_k-means\)](https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm_(naive_k-means))

- do clustering first also can, C find W or W find C

## Hard vs Soft Clustering

### **Hard clustering:**

Each data point can belong only one cluster, e.g. K-means

- For example, an apple can be red OR green (hard clustering)

### **Soft clustering (also known as Fuzzy clustering):**

Each data point can belong to more than one cluster.

- For example, an apple can be red AND green (fuzzy clustering)
- Here, the apple can be red to a certain degree as well as green to a certain degree.
- Instead of the apple belonging to green [green = 1] and not red [red = 0], the apple can belong to green [green = 0.5] and red [red = 0.5]. These value are normalized between 0 and 1; however, they do not represent probabilities, so the two values **do not need to add up to 1**.

## Fuzzy Clustering

- Soft clustering

## Objective Function for Fuzzy C-means

Minimize  $J$

$$J = \sum_{i=1}^m \sum_{k=1}^c (w_{ik})^r \|x_i - c_k\|^2$$

where

$$w_{ik} = \frac{1}{\sum_{j=1}^c \left( \frac{\|x_i - c_k\|}{\|x_i - c_j\|} \right)^{\frac{2}{r-1}}}$$

$w_{ik} = 0.8.$

No need to memorize the equation!

Each element,  $w_{ik} \in [0,1]$ , tells the degree to which element,  $x_i$ , belongs to cluster  $c_k$ .

The fuzzifier  $r > 1$  determines the level of cluster fuzziness; usually  $1.25 \leq r \leq 2$ . 1.5

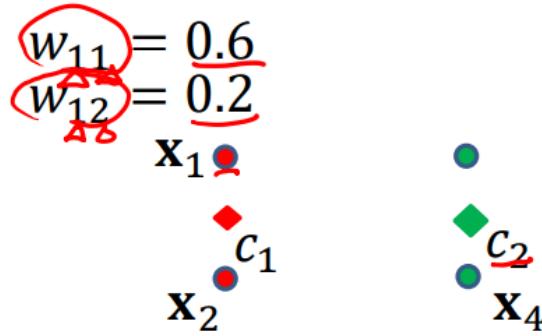
## Objective Function for Fuzzy C-means

Minimize  $J$

$$J = \sum_{i=1}^m \sum_{k=1}^c (w_{ik})^r \|x_i - c_k\|^2$$

where

$$w_{ik} = \frac{1}{\sum_{j=1}^c \left( \frac{\|x_i - c_k\|}{\|x_i - c_j\|} \right)^{\frac{2}{r-1}}}$$



$$w_{41} = 0.18$$
$$w_{42} = 0.75$$

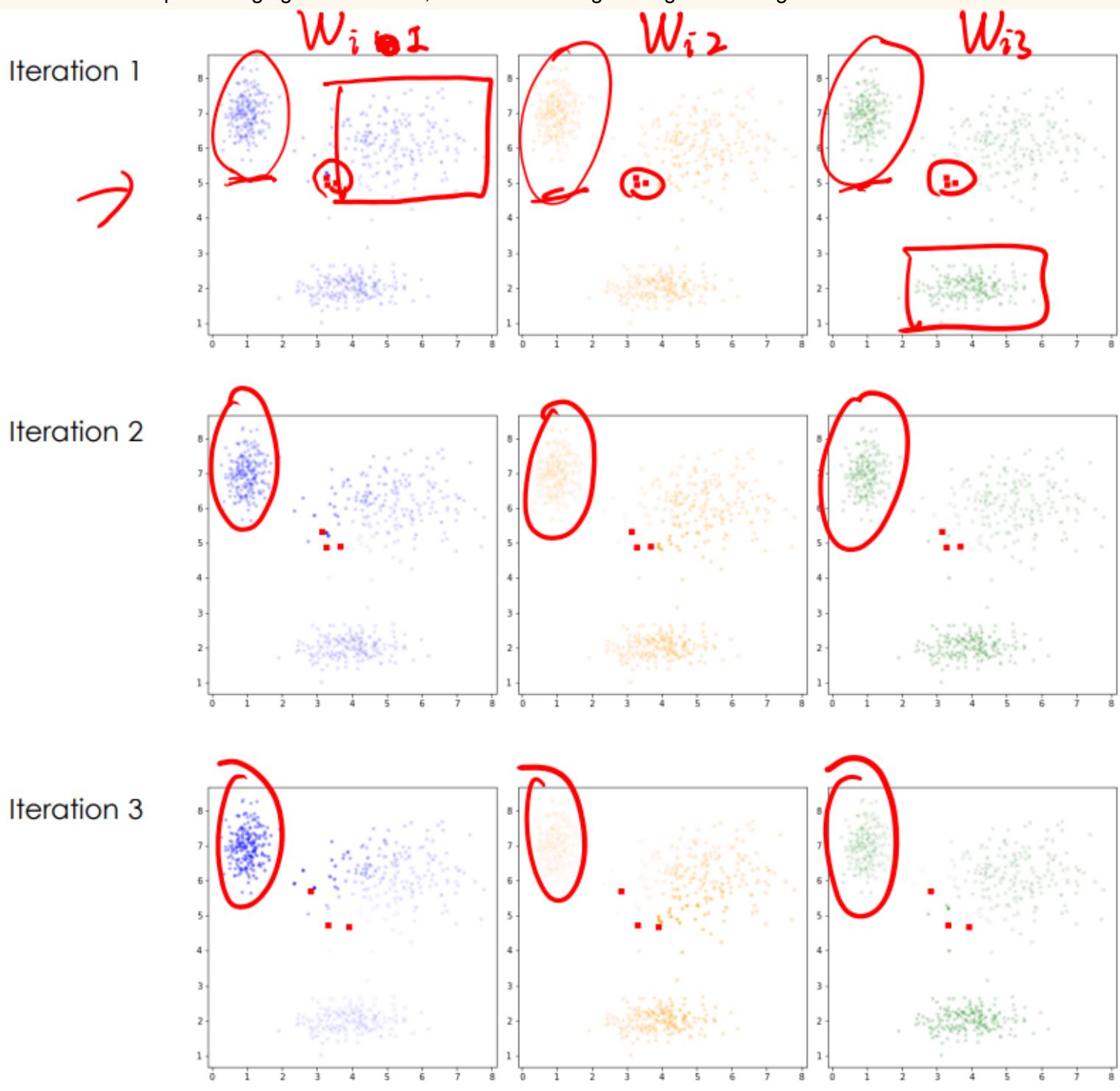
Each element,  $w_{ik} \in [0,1]$ , tells the degree to which element,  $x_i$ , belongs to cluster  $c_k$ .

The fuzzifier  $r > 1$  determines the level of cluster fuzziness; usually  $1.25 \leq r \leq 2$ .

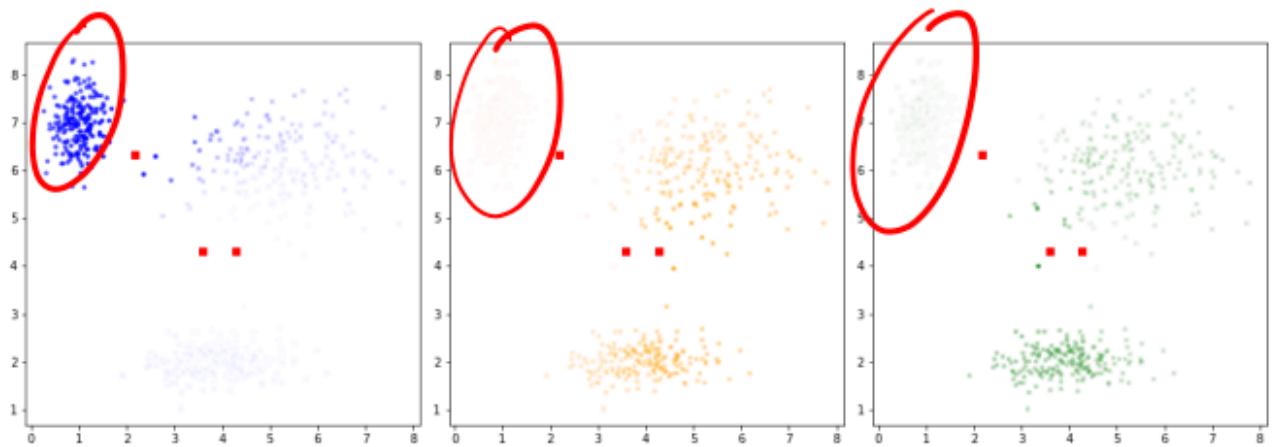
- for fuzzy,  $w_{ik}$  is no longer binary 0 or 1, its a range
- $w_{ik}$  is the degree of a sample belonging to a class

### Visualisation of Fuzzy C-means Iterations

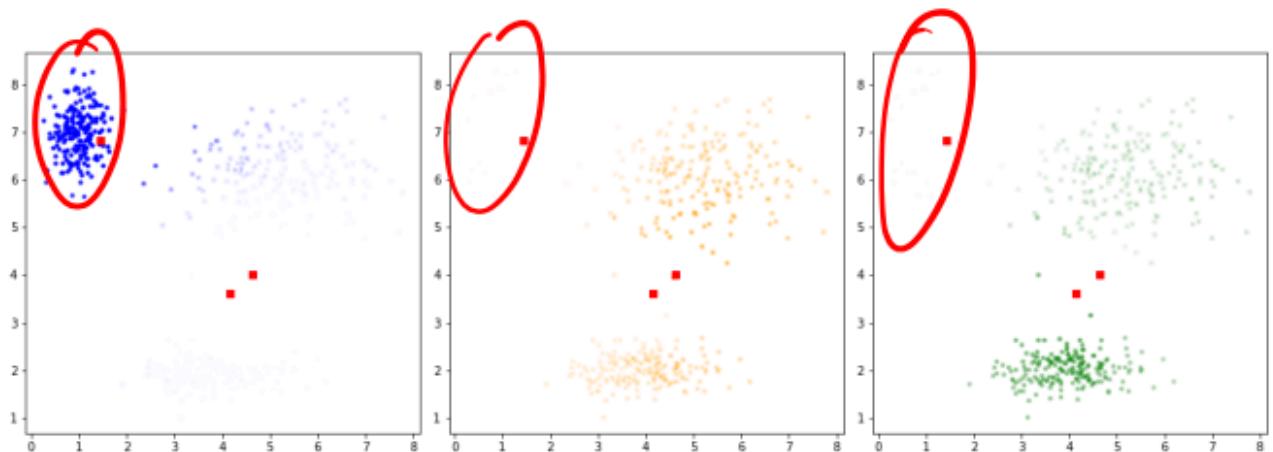
Below is each sample belonging to each class, darker means higher degree of assignment



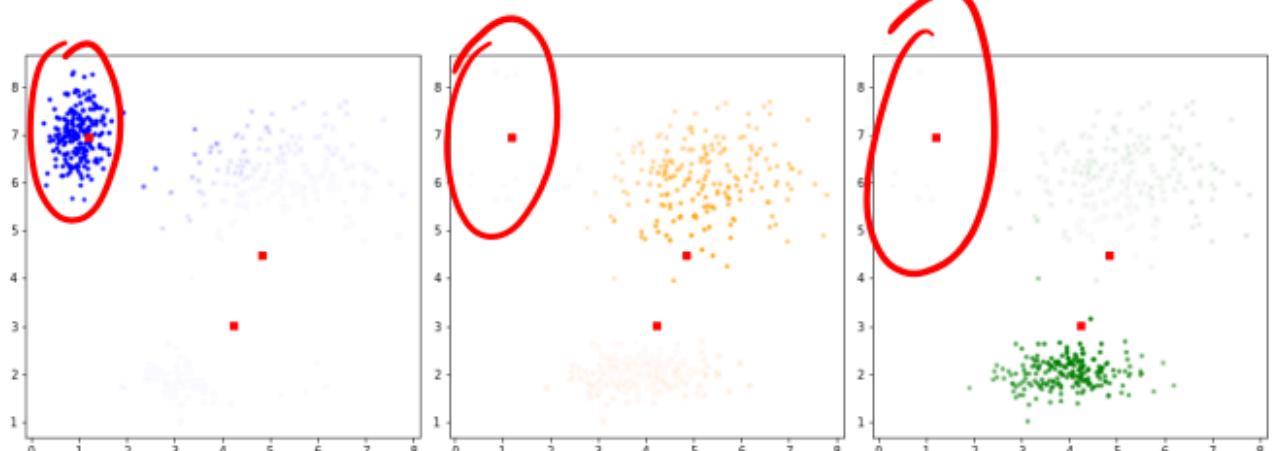
Iteration 4



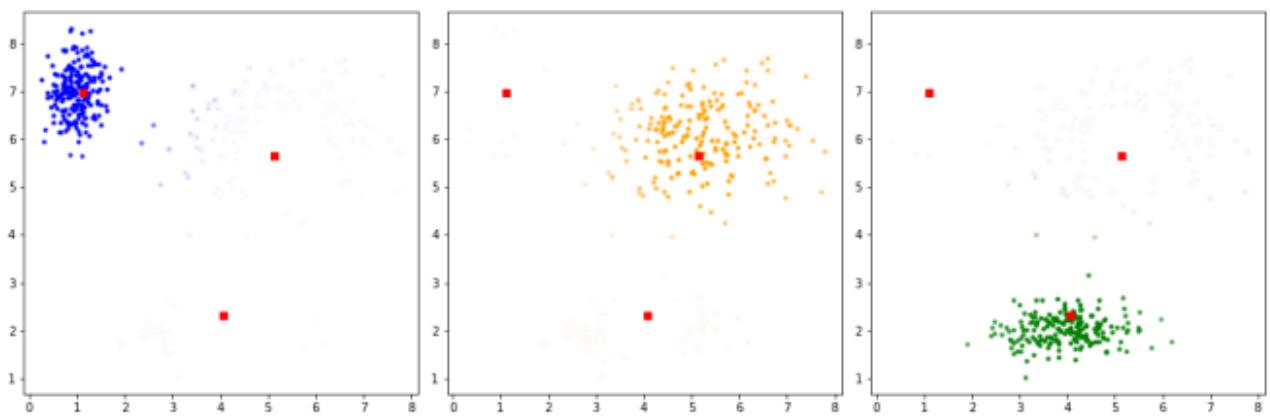
Iteration 5



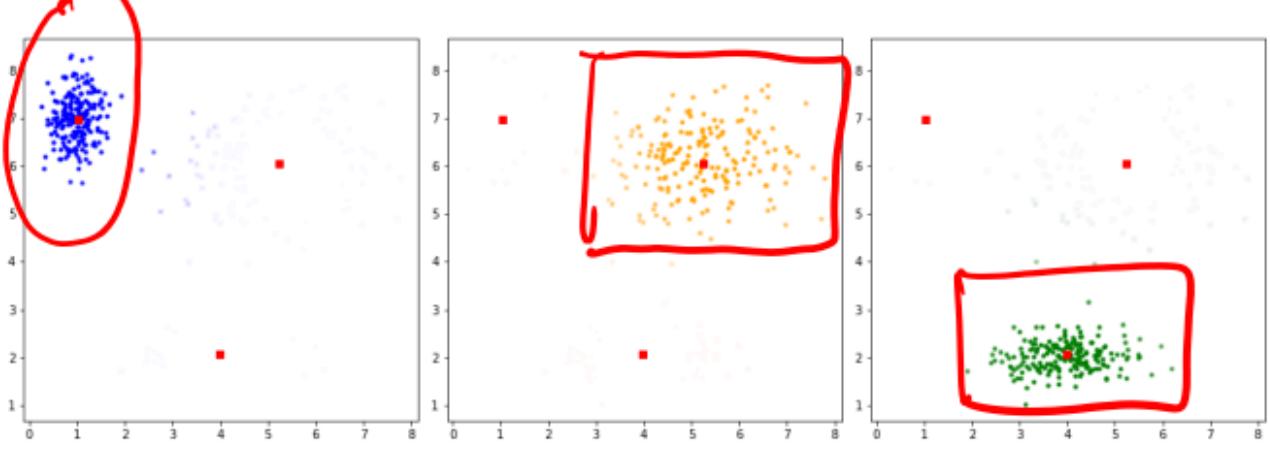
Iteration 6



Iteration 7

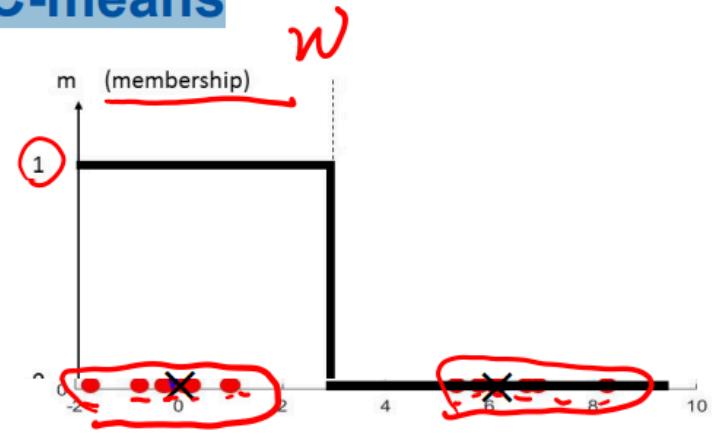


Iteration 8

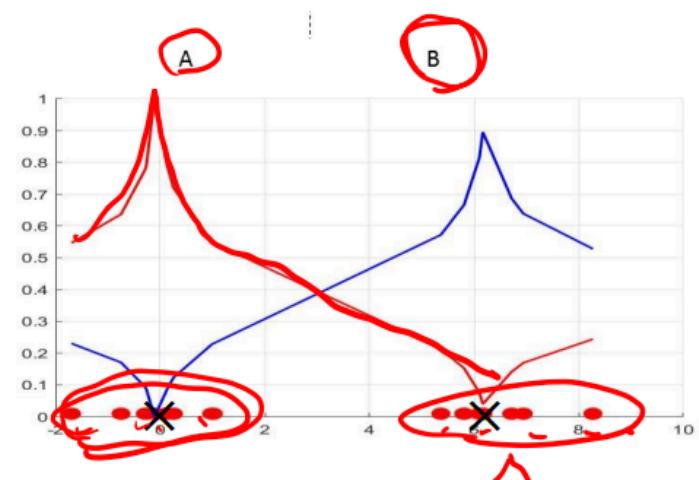


## Naïve K-means versus Fuzzy C-means

Naïve K-means:  $w_{ik} \in \{0,1\}$



Fuzzy C-means:  $w_{ik} \in [0,1]$



Ref: [https://en.wikipedia.org/wiki/Fuzzy\\_clustering](https://en.wikipedia.org/wiki/Fuzzy_clustering)

- w also called membership function

### Practice Question

We have a collection of 9 foreign coins. We measure their radius in millimeters and summarize them as follows.

Coin ID	01	02	03	04	05	06	07	08	09
Radius (mm)	10	11	12	15	16	17	20	21	22

We'd like to group the coins into three groups according to their radius.

Assume we pick coin 01 as the initial centroid for Group A, coin 04 for Group B, and coin 07 for Group C. We would like to assign the coins to the three groups using Euclidean distance. Before updating the new centroid, we will have BLANK1 coins in Group A (please enter an integer here).

3 coins {01, 02, 03} .

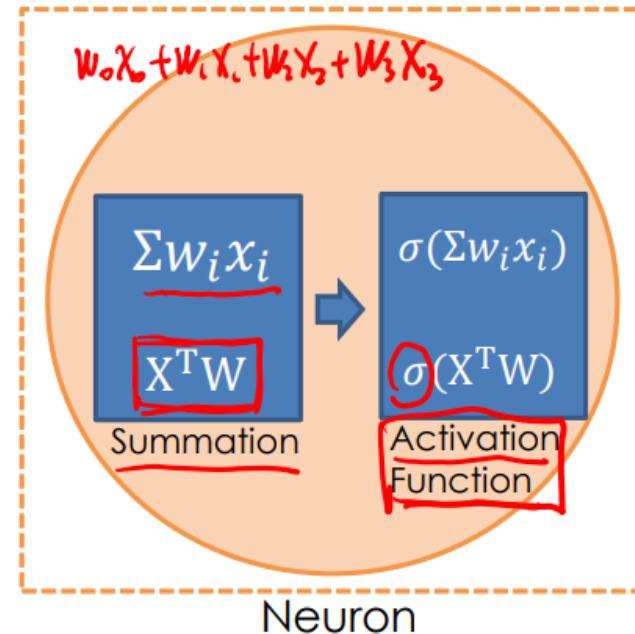
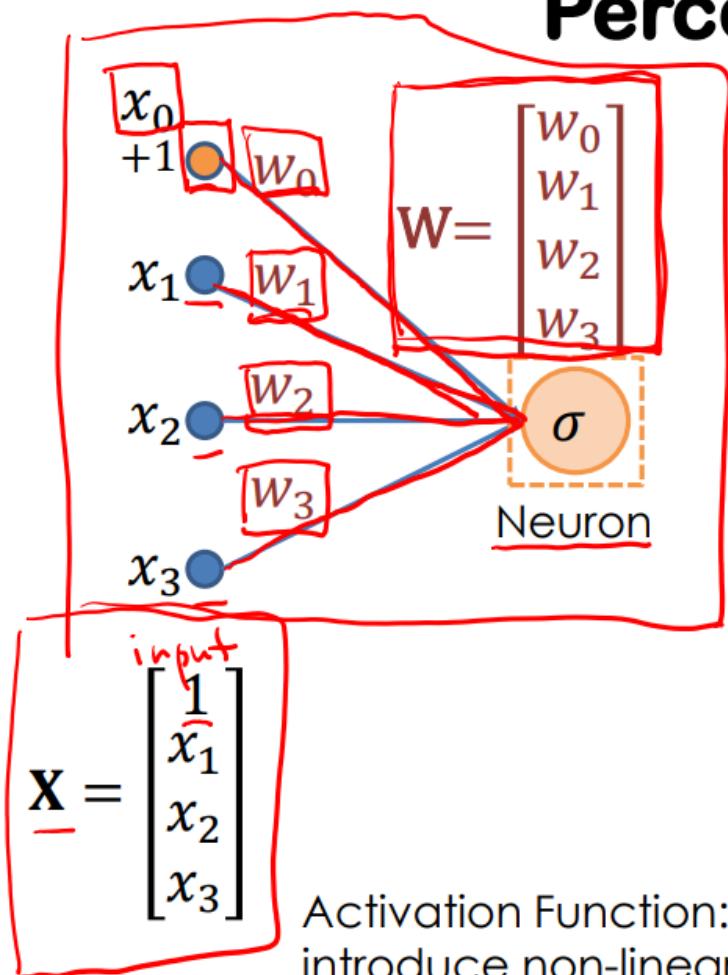
## Lec 12: Neural Networks

### Introduction to Neural Networks

#### Perceptron



# Perceptron



Output of Neuron:  $\sigma(\mathbf{X}^T \mathbf{W})$  or  $\sigma(\sum w_i x_i)$

Activation Function: non-linear function to introduce non-linearity into the neural networks!

Goal of training: to learn  $\mathbf{W}$

- Unit building block of NN
- Input is connected to the neuron through a link. Each link is associated with a weight
- Each neuron does two operations:
  - weighted sum of input and its weights, which is a dot product of the input and weight vectors
  - Then the weighted sum is pass into an activation function
- Activation function must be non linear to introduce non linearity into NN since weighted sum is linear
- Training a perceptron is to learn the weights  $\mathbf{W}$

## Activation Functions

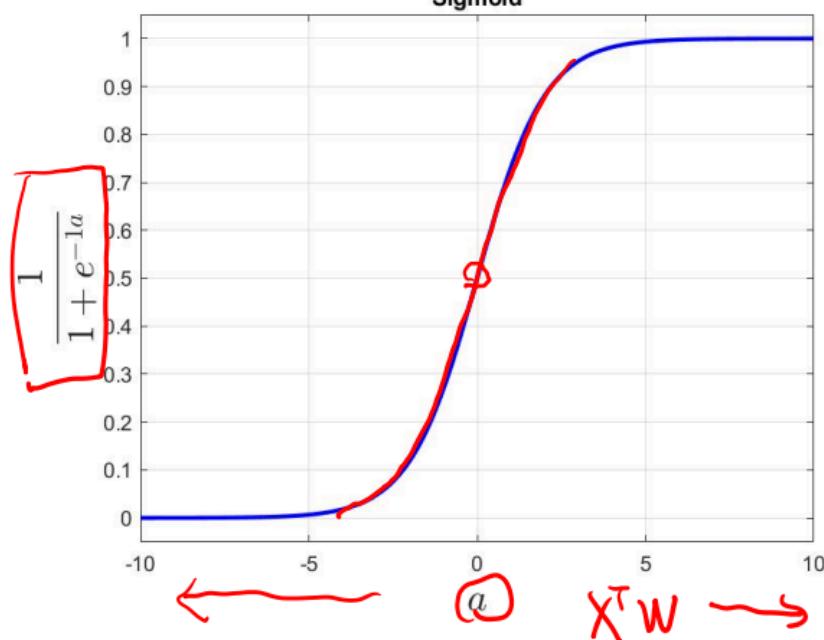
### Sigmoid

## Sigmoid Activation Function

$$\sigma(a) = \frac{1}{1 + e^{-\beta a}},$$

Sigmoid

$$\beta = 1$$



1)  
(0, 1)

2)  
 $\sigma(0=0)=0.5$

3) Derivative

↑,  
when  $a$  close to 0.

- Always between 0 and 1
- Slope is larger when  $a$  is close to 0

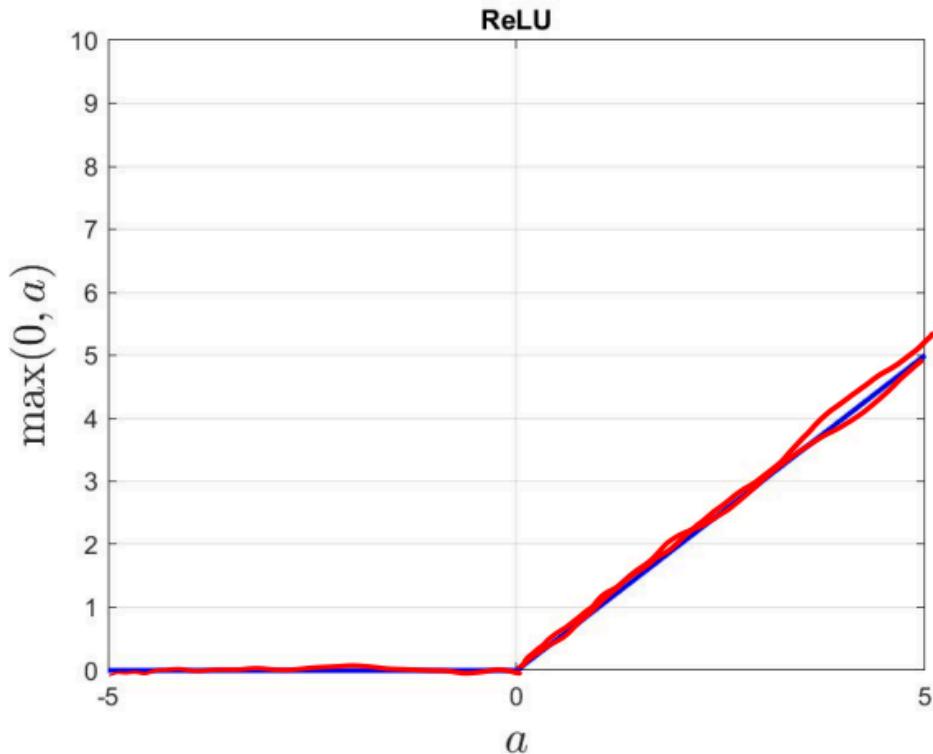
$$\sigma(a=0.5) = 0.5$$

## ReLU Activation Function

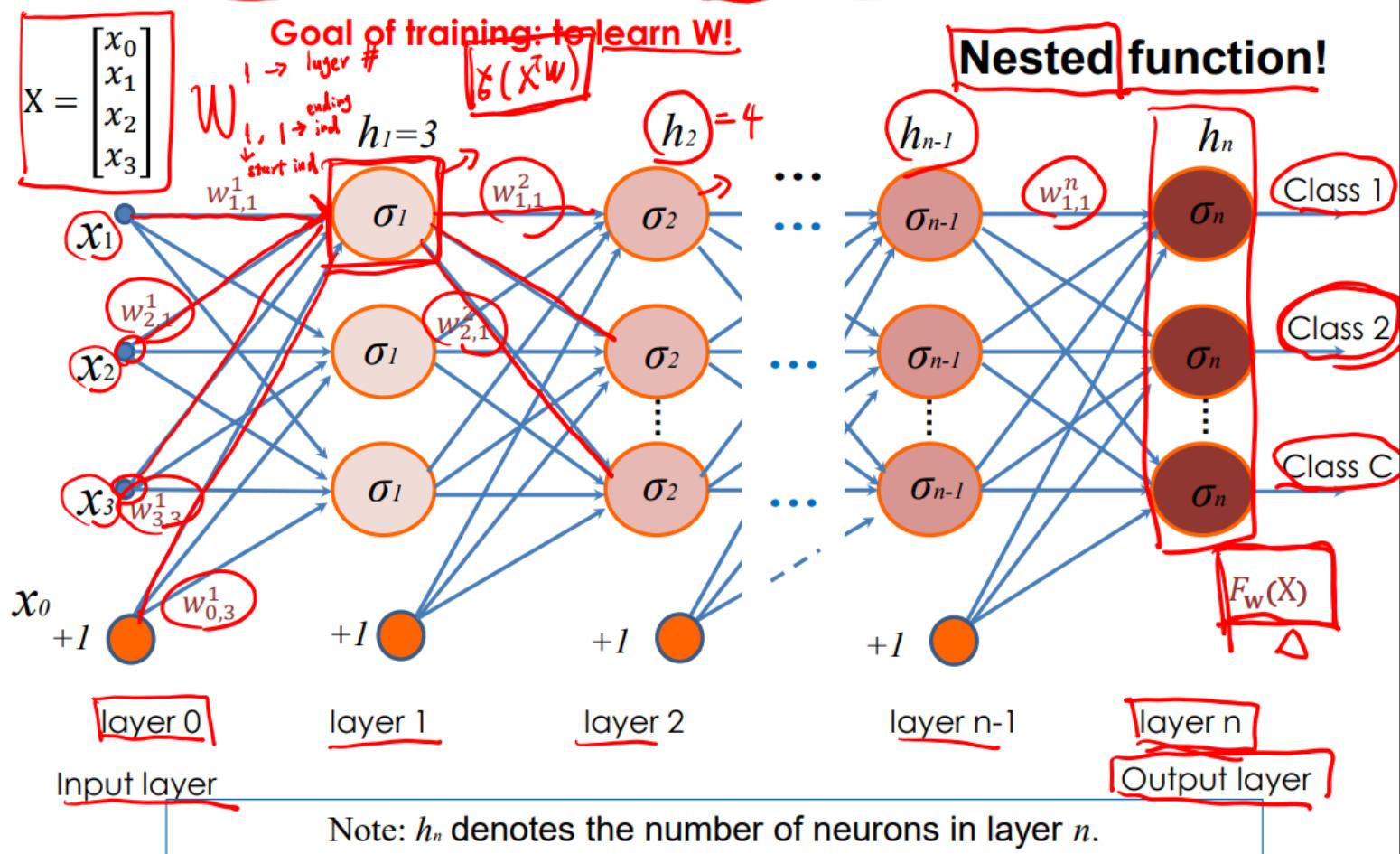
$$\sigma(a=-1) = 0$$

$$\sigma(a) = \max(0, a)$$

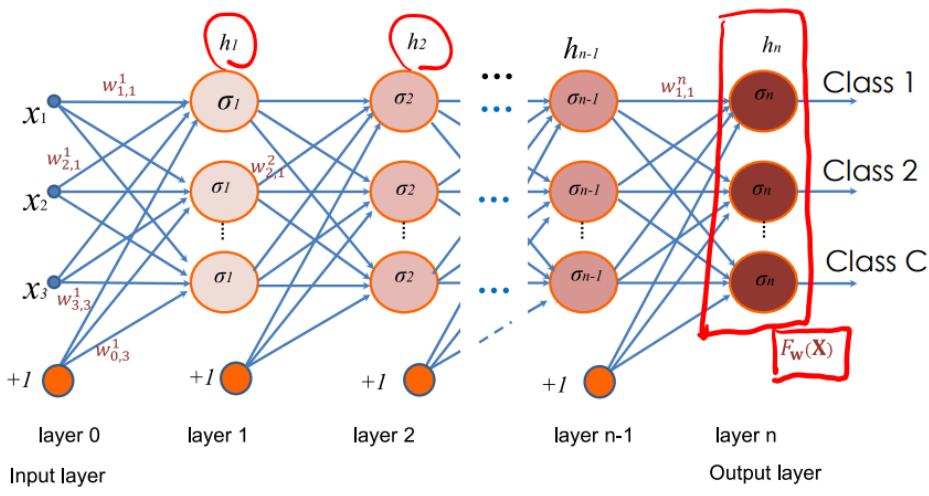
Rectified Linear Unit (ReLU)



# Multilayer Perceptron (Neural Network)

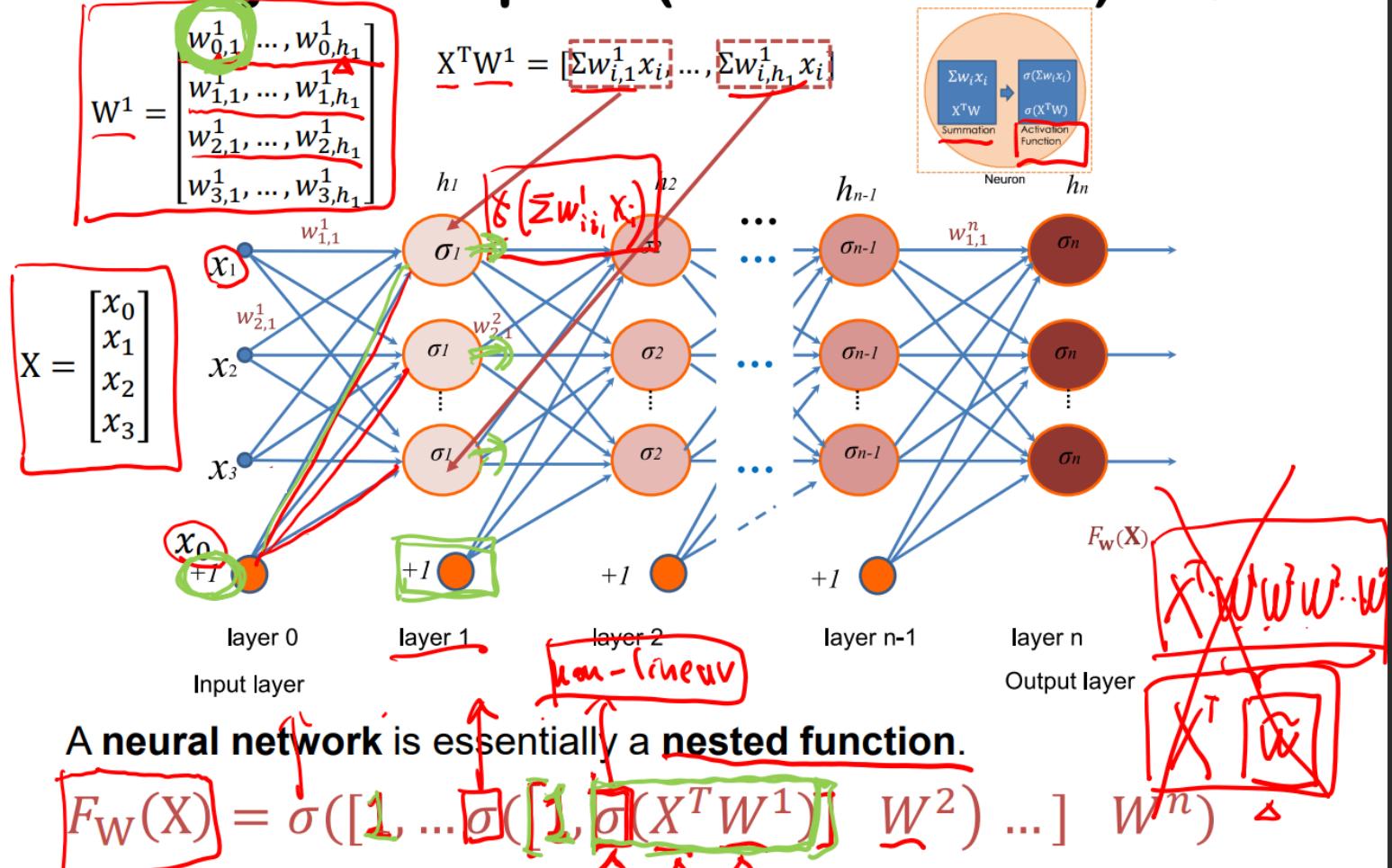


- Each neuron is supplied with +1 input for the bias
- $w_{a,b}^n$  = weight of the link at  $n$  ending layer, from neuron  $a$  in layer  $n - 1$  to neuron  $b$  in layer  $n$
- The input and output is all just numbers
  - output of layer  $n-1$  is the input of layer  $n$  after being weighed
- To train means to learn all the weights
- NN mostly used for classification, but regression is possible
  - for classification,  $n$  class means  $n$  output, with each of the output correspond to the probability of the input being of that class



1. The number of neurons in different layers may differ, i.e.,  $h_1$  don't have to be equal to  $h_2$ .
2. For **classification** task, the number of neurons in the last layer equals to the number of classes.
3. We can treat the whole network as a function  $F_W(X)$ , where  $w$  is to be learned.

## Multilayer Perceptron (Neural Network)



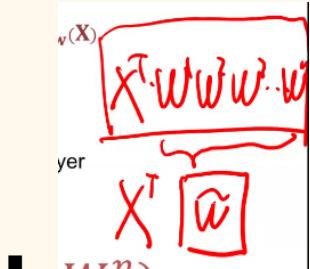
- Combine the weights in the first layer to one matrix

- $$W^1 = \begin{bmatrix} w_{0,1}^1, \dots, w_{0,h_1}^1 \\ w_{1,1}^1, \dots, w_{1,h_1}^1 \\ w_{2,1}^1, \dots, w_{2,h_1}^1 \\ w_{3,1}^1, \dots, w_{3,h_1}^1 \\ \dots \end{bmatrix}$$

A **neural network** is essentially a **nested function**.

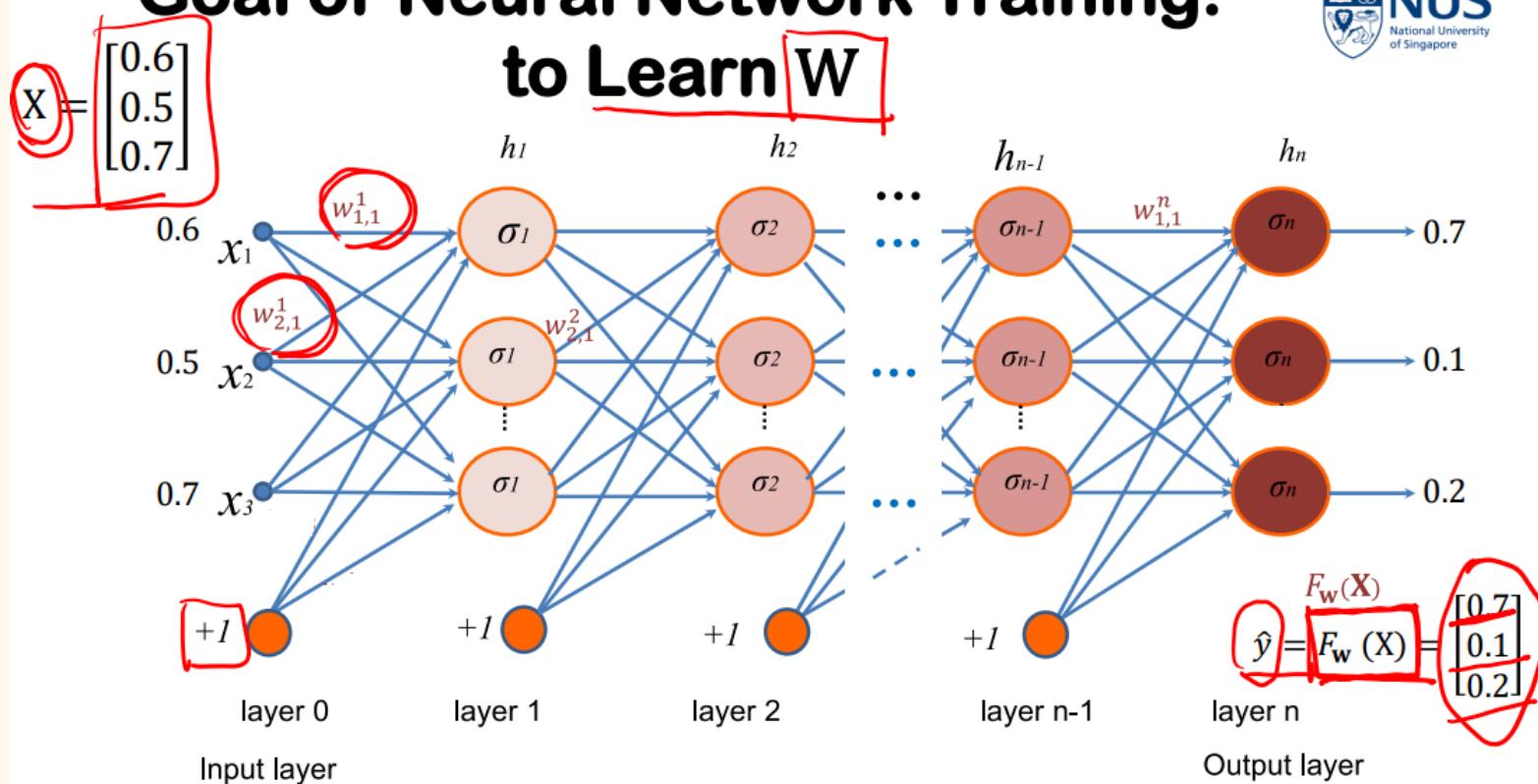
$$F_W(X) = \sigma([1, \dots \sigma([1, \sigma(X^T W^1)] \quad W^2) \dots] \quad W^n)$$

- sigma must be non linear,
    - if sigma is linear then the whole thing is just multiplication of input with one matrix



- all the layers collapse and equivalent to be a single matrix

# Goal of Neural Network Training: to Learn $\mathbf{W}$

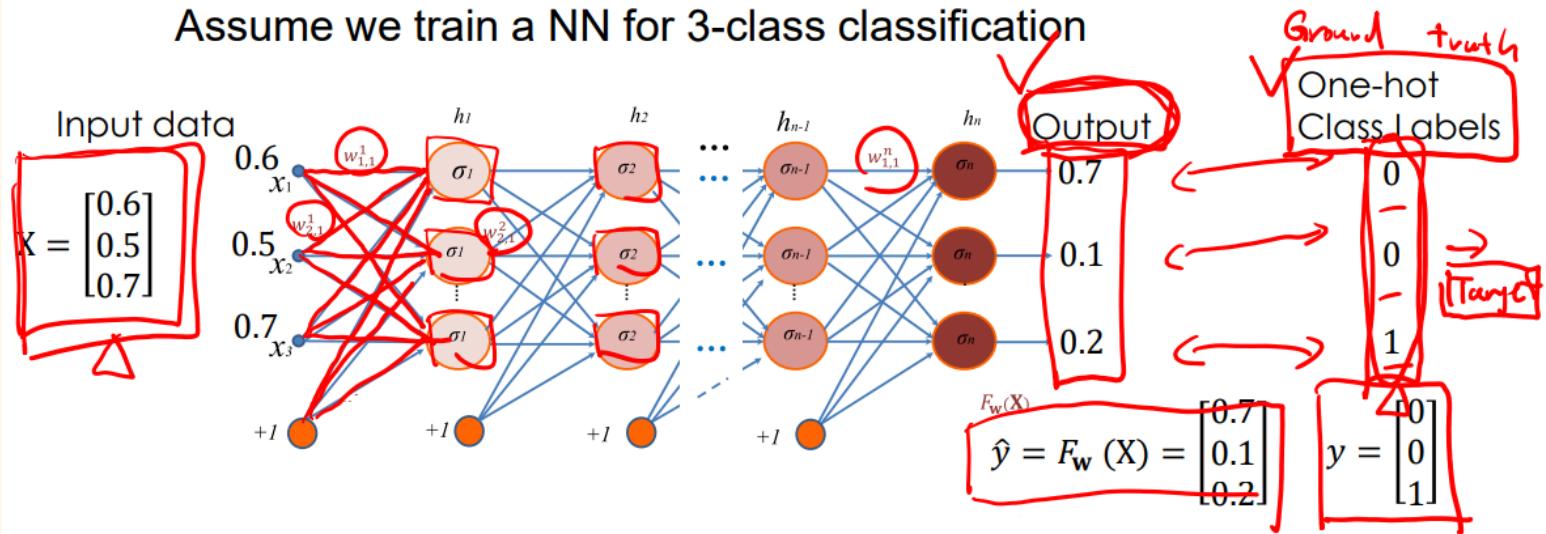


Specifically,  $\mathbf{W}$  is learned through

1. Random initialization
2. Backpropagation

# Neural Network Training: Backpropagation

Assume we train a NN for 3-class classification



**1. Forward:** (weights are fixed)  
To compute network responses  
To compute the errors at each output

**2. Backward:** (weights are updated)  
To pass back the error from the output to the hidden layers  
To update all weights to optimize the network

A loss function for a single sample:

$$\min_w \sum_{i=1}^C (\hat{y}_i - y_i)^2$$

$$\text{or } \min_w \|\hat{y} - y\|^2$$

Update ~~W!~~

- Randomly choose initial weight
- Use ground truth to update the weights to minimise the error
- Finish iteration when loss curve is at convergence
- Back propagation is both forward and backward

- Recall that the parameters  $W$  are randomly initialized.
- We use **Backpropagation** to update  $W$ .
- In essence, **Backpropagation** is gradient descent!
- Assume we have  **$N$  samples**, each sample denoted by  $X^j$  and the output of NN by  $\hat{y}^j$ , loss function is then

$$J = \sum_{j=1}^N \|\hat{y}^j - y^j\|^2, \quad \min_w J$$

Recall gradient descent in Lec 8:  $w \leftarrow w - \eta \nabla_w J$

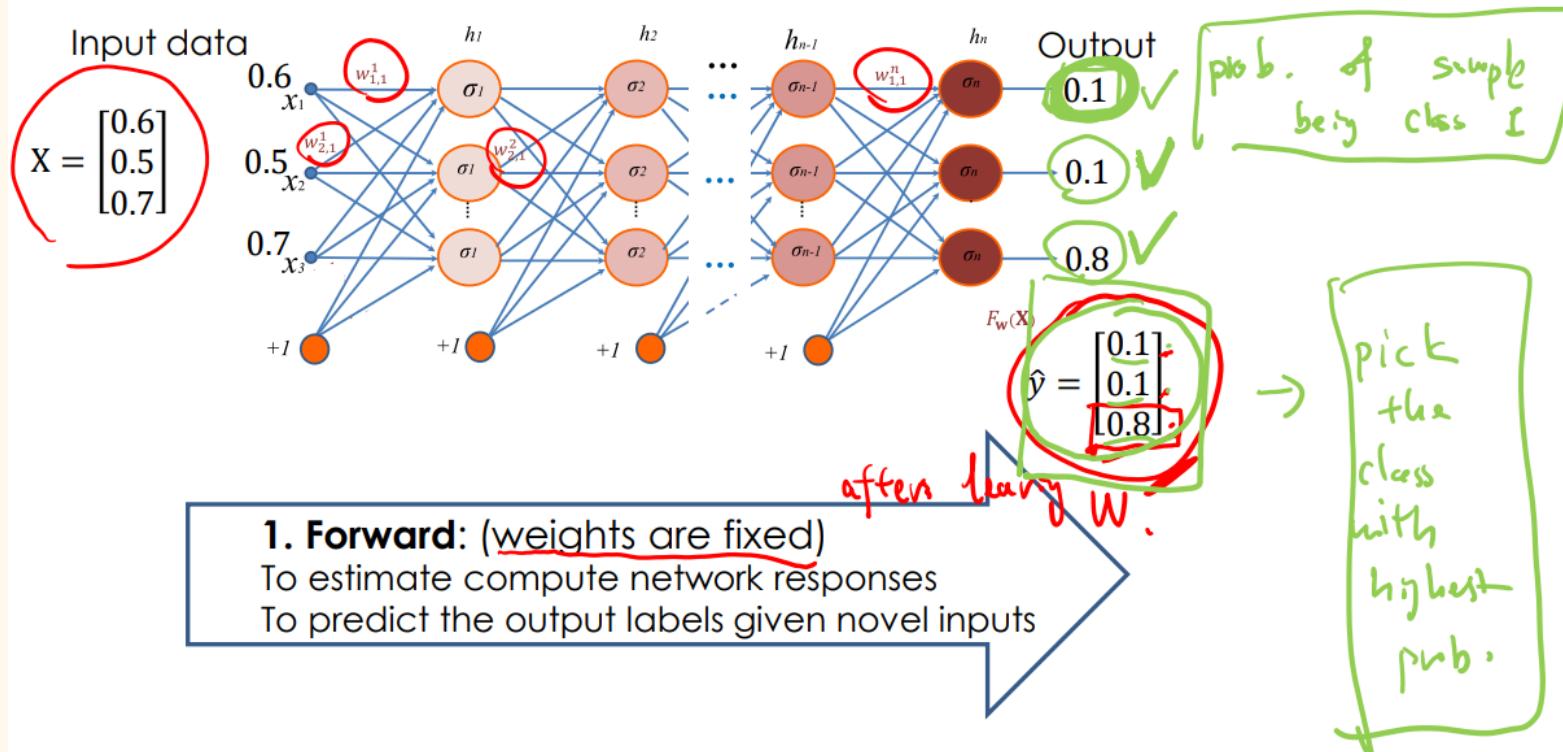
- We would therefore like to compute  $\nabla_w J$ !
  - $J$  is a function of  $\hat{y}$ , and  $\hat{y}$  is a function of  $w$ , i.e.,  $\hat{y} = F_w(X)$
  - Use gradient descent and chain rule!

Being aware of the basic concept is sufficient for exam. No calculation needed.

- Use gradient descent to tune the weights

### Testing: Forward

Once all network is trained and parameters are updated, we may conduct testing.



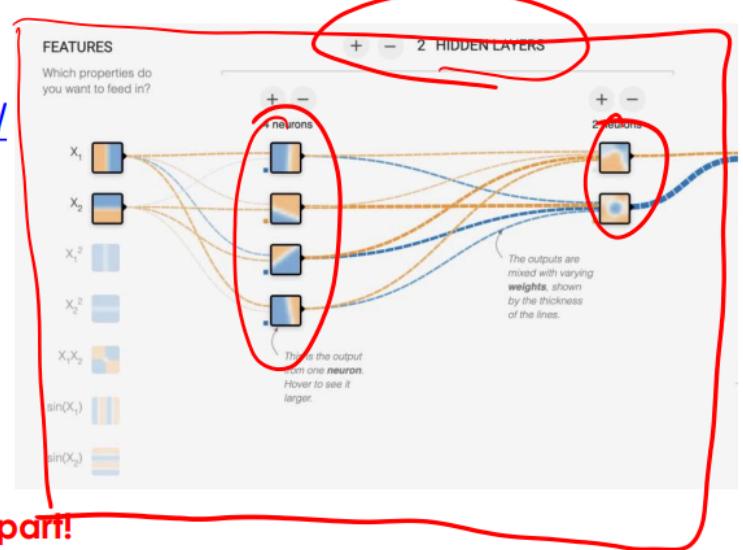
## Python Demo

```
[ ] from sklearn.neural_network import MLPClassifier  
# Train the neural network classifier with 3 hidden layers of 200 neurons each  
clf = MLPClassifier(hidden_layer_sizes=(200, 200, 200), activation="relu", learning_rate="invscaling", verbose=True)  
clf.fit(X, y)  
plot_clf(clf, X, y)
```

- Python Code (A simple multi-layer neural network classifier)
  - lec12.ipynb
- Demo
  - <https://playground.tensorflow.org/>

All available in Canvas  
Files\For Students\Lecture Notes

In the Final, no coding questions for Xinchao's part!



## Convolutional Neural Networks (CNN)

- Special case of NN
- CNN is sparse version of NN, NN is dense each neuron of last layer connects to every neuron of next layer
  - Deep NN is many layers
- CNN is sparse to reduce the amount of parameters/weights
  - good for image

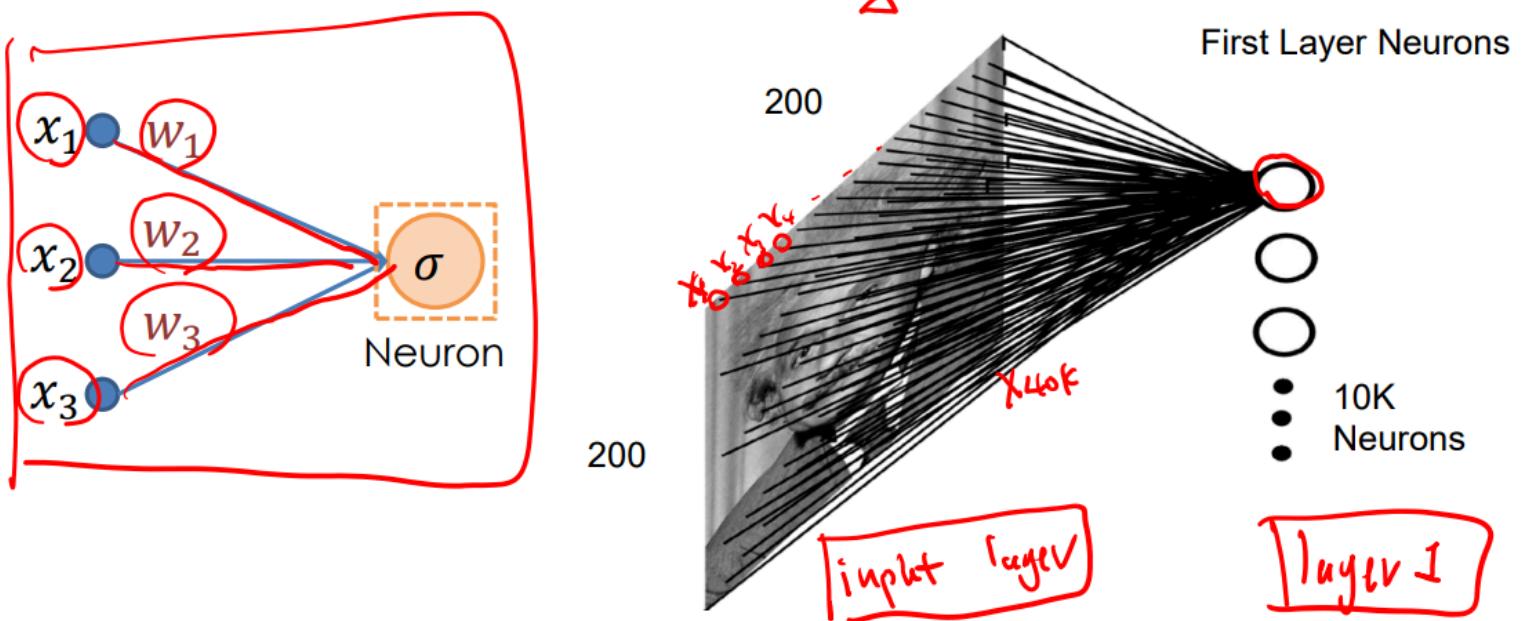
- A convolutional neural network (CNN) is a special type of neural network that significantly reduces the number of parameters in a deep neural network.
  - Very popular in image-related applications
  - Each image is stored as a matrix in a computer



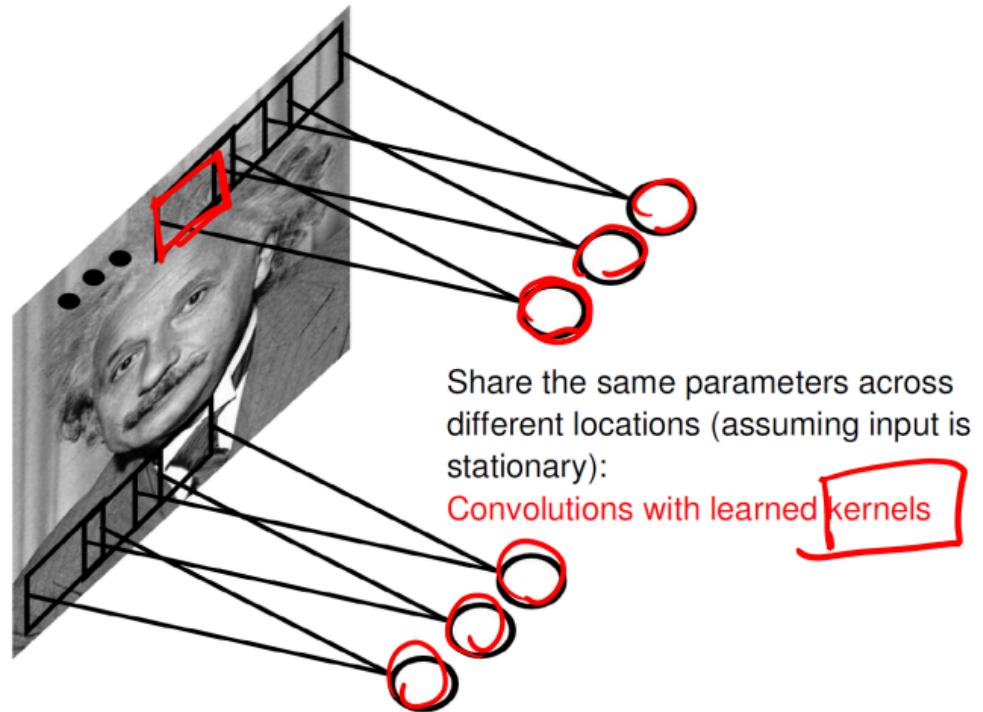
	0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29	
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0	
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1	
2	95	228	228	255	111	141	116	122	215	238	255	49	0	0	0	
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36	
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62	
6	141	245	255	215	25	11	9	3	0	115	236	243	255	137	0	
0	87	252	250	248	215	60	0	121	255	255	248	144	0	0	0	
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19	
1	0	5	117	251	255	241	255	247	245	241	162	17	0	7	0	
0	0	0	4	58	251	255	246	254	253	255	220	11	0	1	0	
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4	
0	22	206	252	246	251	241	100	24	113	255	255	245	194	9	0	
0	111	255	242	242	255	158	24	0	0	6	39	255	232	230	56	
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3	
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0	
0	101	251	241	255	230	98	55	19	118	247	248	253	52	4	0	
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5	
0	0	23	113	215	255	250	248	255	255	248	249	115	14	12	0	
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1	
0	0	5	5	0	0	0	0	14	0	6	6	0	0	6	0	

    <https://medium.com/lifeandtech/convert-csv-file-to-images-309b6fdb8c49>

- If we model all matrix entries as inputs all at once
    - Assume we have an image/matrix size of  $200 \times 200$   $40K$
    - Assume we have  $10K$  neurons in the first layer
    - We already have  $200 \times 200 \times 10K = 400$  Million parameters to learn!



- Hence, we introduce CNN to reduce the number of parameters.
- Works in a sliding-window manner!



- Each neuron look at a local neighbourhood (kernel) of input
- Same kernel is used for all the neuron, each neuron is based on the location of the input image
- Location of image being convolve with the kernel is called patch

# Convolutional Neural Network (CNN)

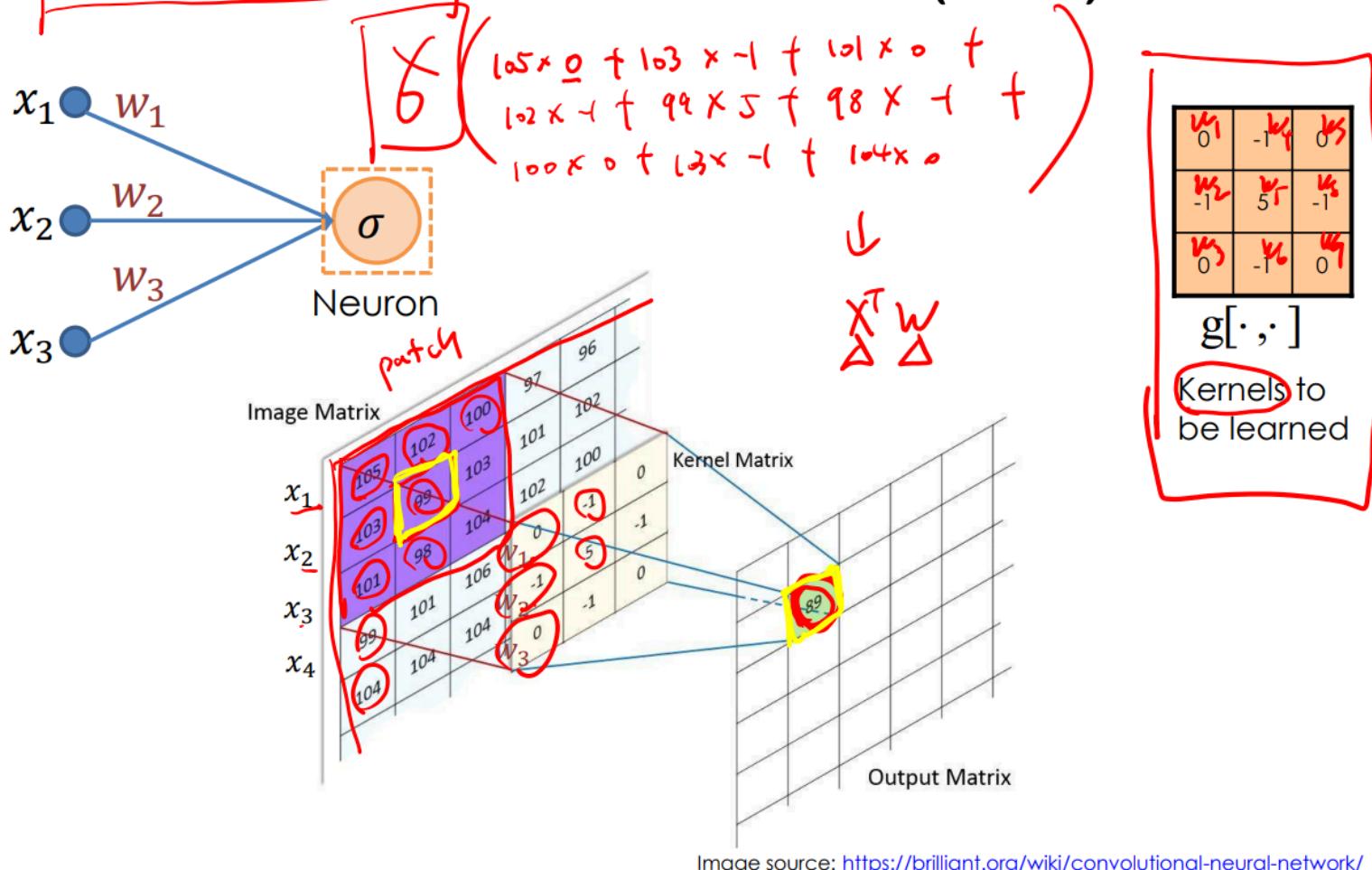
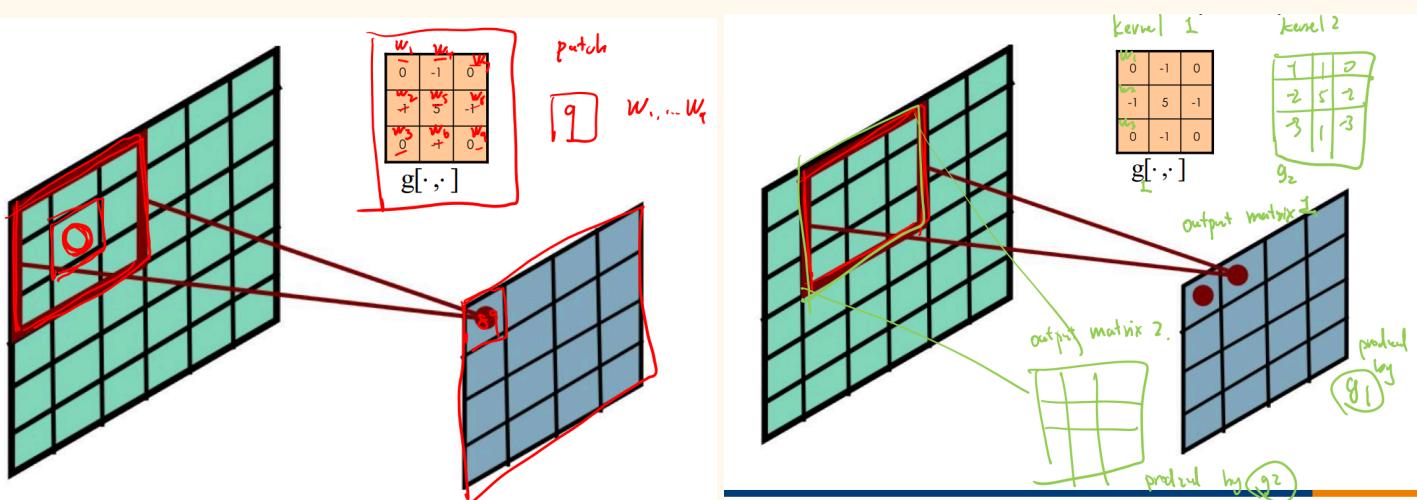
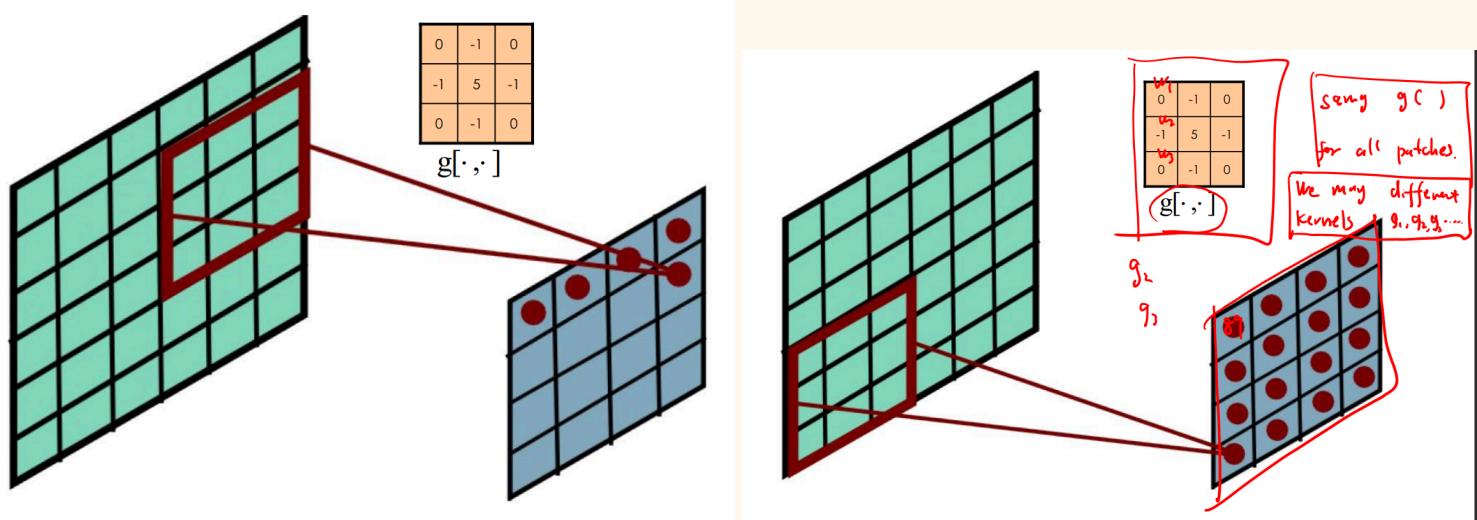


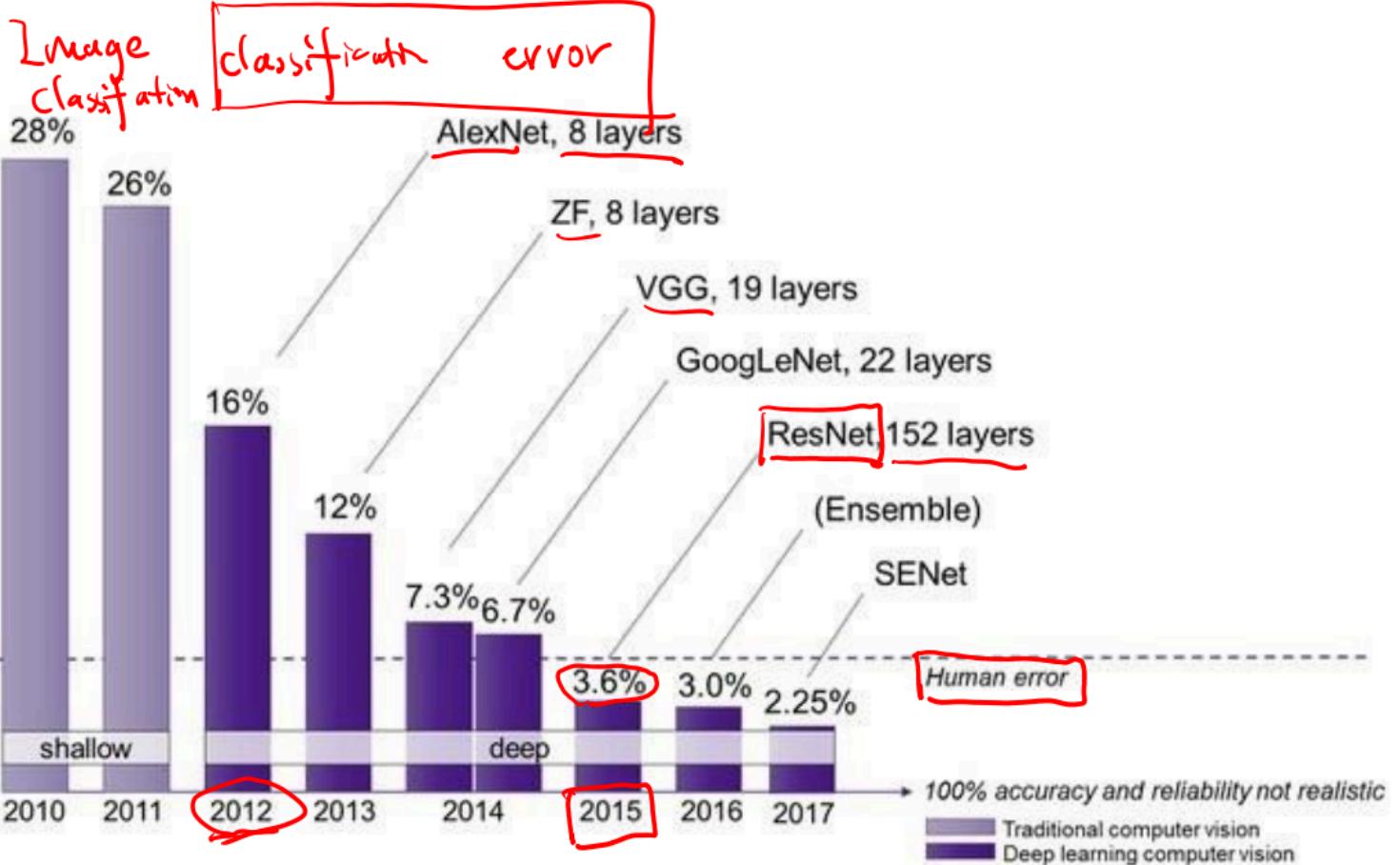
Image source: <https://brilliant.org/wiki/convolutional-neural-network/>

- Output put at the centre location of the patch, this is called convolution
- So the output matrix will be smaller than the input
- Same kernel used for all patches, this reduce the number of weights





## Neural Networks are Effective



- NN requires a lot of data to train

## Practice Question

- Which of the following statements is correct?

- A) Sigmoid serves as a ~~linear~~ activation function in neural networks.  
*Non-linear*
- B) Compared to dense connection (connecting each node in previous layer to the current layer), ~~convolutional neural networks (CNNs)~~ yield more computational complexity.  
*NN dense.*  
*less*  
*sparse*
- C) When training neural networks, its network weights W are fixed.
- D) None of the others is correct.

training changes the W, when going forward W is fixed

## Python and Conda Stuff

need to run the conda commands in the conda prompt

### 2.1 Installation

After installing Anaconda<sup>[3]</sup>, you will probably find python=3.8.8 in your base(root) environment as shown in the Anaconda Navigator under the environment tab. In this module we will use

```
python=3.8 numpy=1.24
```

So let's create a new environment for EE2211 and we will call it ee2211. To do this we enter the following command in the terminal (for Windows user search Anaconda Powershell Prompt, for MacOS user just search for terminal)

```
conda create --name ee2211 python=3.8 numpy=1.24 matplotlib scikit-learn pandas
```

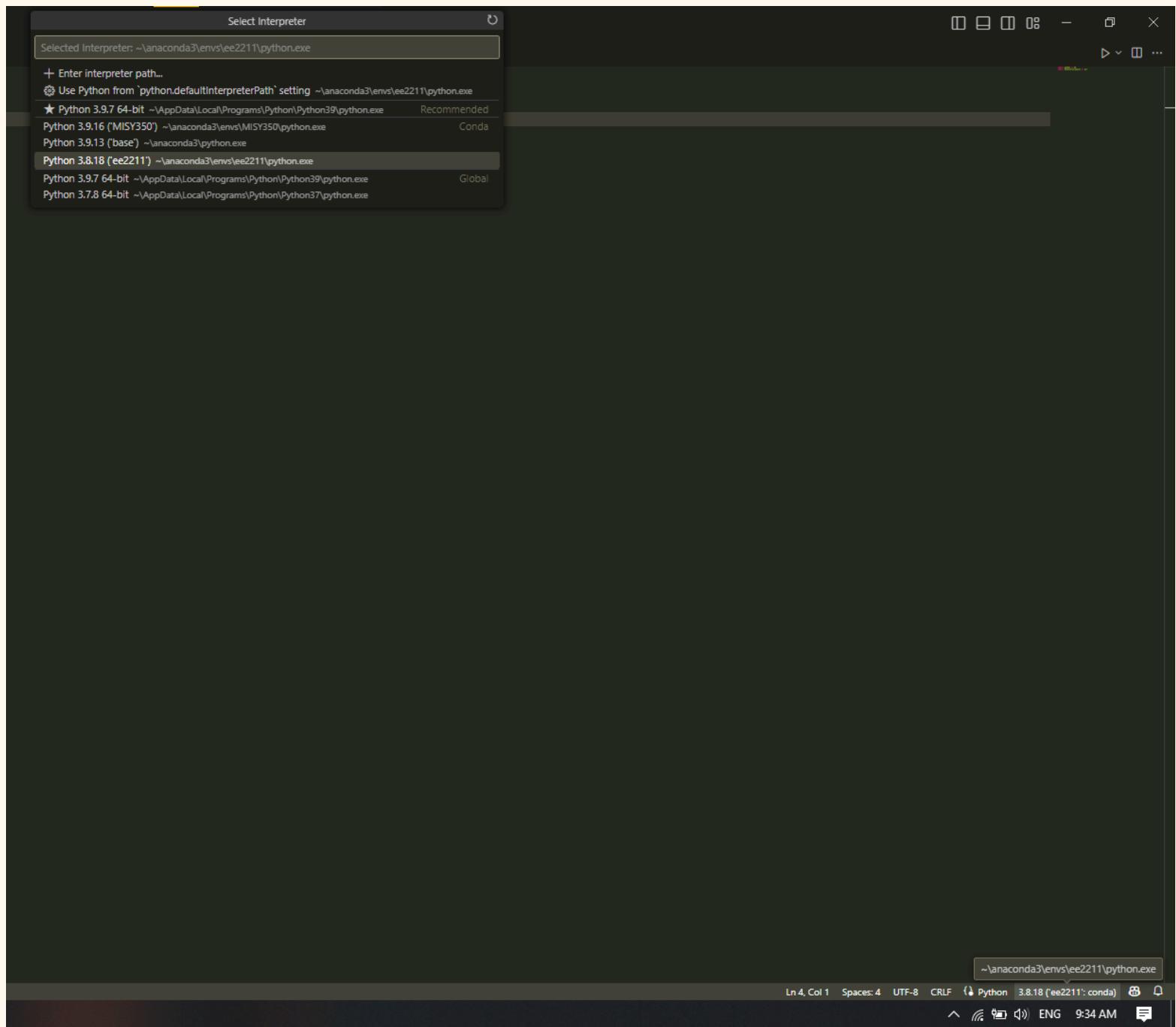
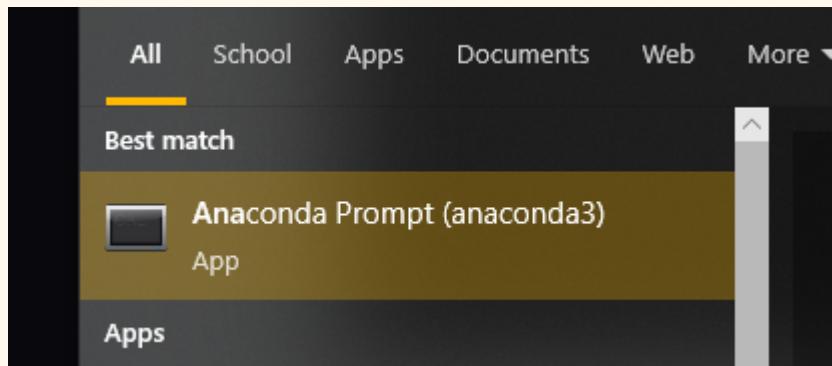
Now in the conda navigator you should find a new environment we just created ee2211, with the version we want :-)

Go ahead and activate the ee2211 environment, by the following command

```
conda activate ee2211
```

In order to view the Jupyter notebook, let's install it in ee2211

```
conda install -c conda-forge jupyterlab
```



## MATLAB

MULTIPLYING A\*B .

INVERSE Y = inv( X ) .  
TRANSPOSE B = A.'

## Tutorial 1

## Tutorial 2

pandas mean python data analysis

matplotlib to plot

sklearn for machine learning

csv easy to read and create, used for data processing and manipulation

pandas reading csv will add index to the data

```
csv_file = pd.read_csv('GovernmentExpenditureonEducation.csv')
print(csv_file)
```

```
year  total_expenditure_on_education
0    1981          942517
1    1982         1358430
2    1983         1611647
3    1984         1769728
4    1985         1812376
5    1986         1641893
6    1987         1654115
7    1988         1604473
8    1989         1765250
9    1990         2056374
10   1991         2816371
11   1992         2597894
12   1993         2902886
13   1994         3318956
14   1995         3443857
15   1996         3771955
16   1997         4449754
```

[https://pandas.pydata.org/Pandas\\_Cheat\\_Sheet.pdf](https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf)

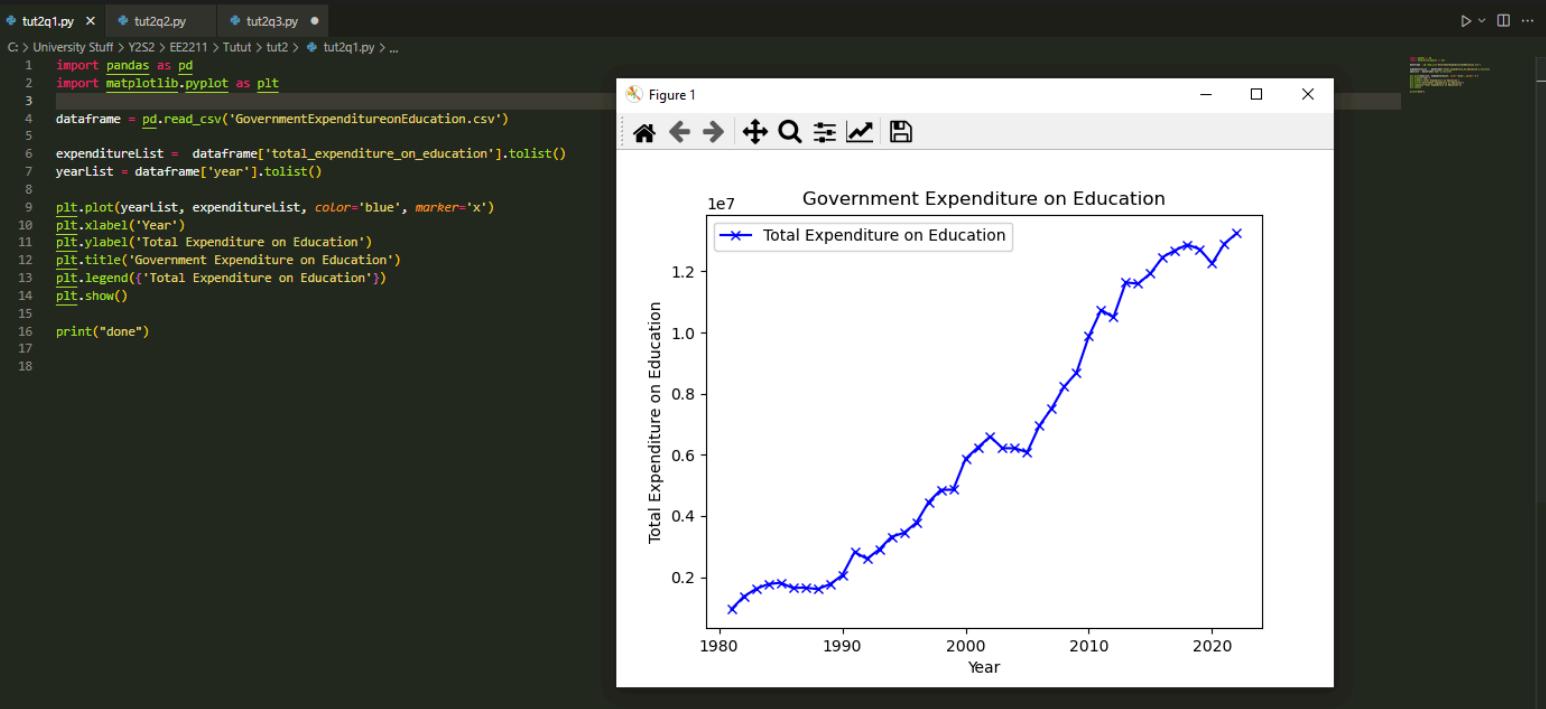
[https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.pyplot.plot.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html)

## Q1

(Data Reading and Visualization, simple data structure)

### **Question 1:**

A Comma Separated Values (CSV) file is a plain text file that contains a list of data. These files are often used for exchanging data between different applications. Download the file “government-expenditure-on-education.csv” from <https://data.gov.sg/dataset/government-expenditure-on-education>. Plot the educational expenditure over the years. (Hint: you might need “import pandas as pd” and “import matplotlib.pyplot as plt”.)



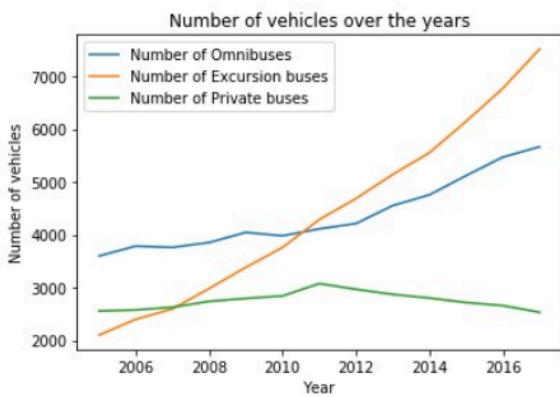
- if absolute file path not used, need to run in the same directory
- use `.tolist()` to make column into list in python

## Q2

(Data Reading and Visualization, slightly more complicated data structure)

### Question 2:

Download the CSV file from <https://data.gov.sg/dataset/annual-motor-vehicle-population-by-vehicle-type>. Extract and plot the number of Omnibuses, Excursion buses and Private buses over the years as shown below. (Hint: you might need “import pandas as pd” and “import matplotlib.pyplot as plt”).



```

# Display the plot
plt.show()

Replace 'column1' and 'column2' with the names of the columns you want to plot. If you're not sure what the column names are, you can print them out with print(csv_file.columns).

Here's how your code would look:

import pandas as pd
import matplotlib.pyplot as plt

csv_file = pd.read_csv('GovernmentExpenditureonEducation.csv')

# Replace 'column1' and 'column2' with your column names
plt.plot(csv_file['column1'], csv_file['column2'])

plt.show()
print("done")

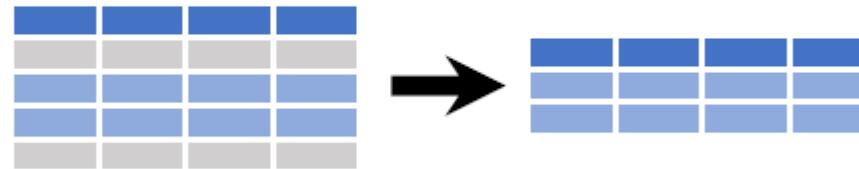
```

YTTRIUM398890585  
what does marker do  
GitHub Copilot  
Used 1 reference >  
The marker parameter in the plt.plot() function is used to emphasize each point with a specified marker.  
In your code:  
plt.plot(csv\_file['year'], csv\_file['total\_expenditure\_on\_education'],  
The marker='o' part means that each point on the plot will be marked with a circle.  
There are many other markers you can use, such as:  
• '+' point marker  
• 'x' pixel marker  
• 'x' x marker  
• 'x' plus marker  
• 's' square marker  
• 'd' diamond marker  
• and many more.  
You can find the full list of available markers in the matplotlib documentation.

How can I customize the line style in a matplotlib plot?  
Ask Copilot or type / for commands

- instead of using for loop to extract rows

## Subset Observations - rows



`df[df.Length > 7]`

Extract rows that meet logical criteria.

- extract the index out instead of getting the whole list

```

import pandas as pd
import matplotlib.pyplot as plt

dataframe = pd.read_csv('AnnualMotorVehiclePopulationbyVehicleType.csv')

privateBusIndexList = dataframe[dataframe['type'] == "Private buses"].index.tolist()

print(privateBusIndexList)

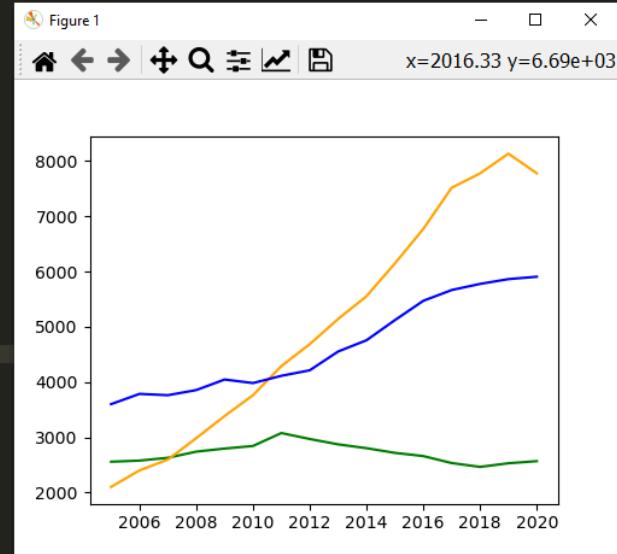
privateBusIndexList = dataframe.loc[dataframe['type'] == "Private buses"].index

print(privateBusIndexList)

```

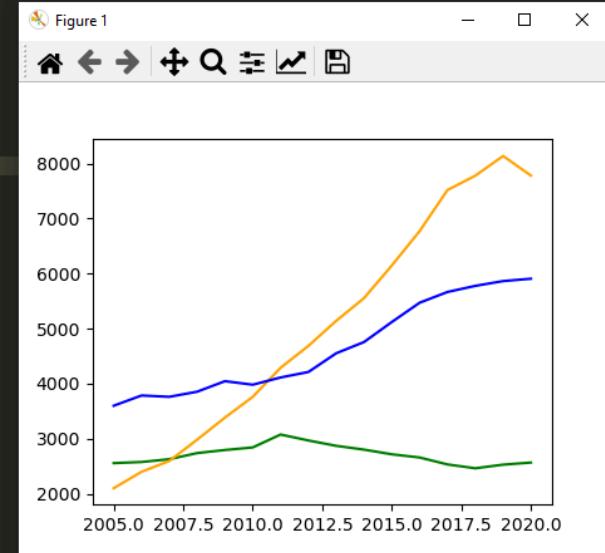
- [132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 216, 237, 258, 279, 300, 321]
   
Index([132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 216, 237, 258, 279,
 300, 321],
 dtype='int64')
- PS C:\University Stuff\Y2S2\EE2211\Tutut\tut2> []
- with .index, the data type is in a diff form

```
tut2q1.py tut2q2.py tut2q3.py
C: > University Stuff > Y2S2 > EE2211 > Tutut > tut2 > tut2.py ...
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4
5 dataframe = pd.read_csv('AnnualMotorVehiclePopulationbyVehicleType.csv')
6
7 # privateBusIndexList = dataframe[dataframe['type'] == "Private buses"].index.tolist()
8 # print(privateBusIndexList)
9
10 # privateBusIndexList = dataframe.loc[dataframe['type'] == "Private buses"].index
11 # excursionBusIndexList = dataframe.loc[dataframe['type'] == "Excursion buses"].index
12 # omniBusIndexList = dataframe.loc[dataframe['type'] == "Omnibuses"].index
13
14 # privateBusDataframe = dataframe.loc[privateBusIndexList]
15 # excursionBusDataframe = dataframe.loc[excursionBusIndexList]
16 # omniBusDataframe = dataframe.loc[omniBusIndexList]
17
18 # or without extracting the index first
19 privateBusDataframe = dataframe.loc[dataframe['type'] == "Private buses"]
20 excursionBusDataframe = dataframe.loc[dataframe['type'] == "Excursion buses"]
21 omniBusDataframe = dataframe.loc[dataframe['type'] == "Omnibuses"]
22
23 plt.plot(privateBusDataframe['year'], privateBusDataframe['number'], color='green')
24 plt.plot(excursionBusDataframe['year'], excursionBusDataframe['number'], color='orange')
25 plt.plot(omniBusDataframe['year'], omniBusDataframe['number'], color='blue')
26
27 plt.show()
28
29
30
31 # privateBusNumber = []
32 # privateBusYear = []
33
34 # excursionBusNumber = []
35 # excursionBusYear = []
36
37 # omniBusNumber = []
```



- without extracting index first

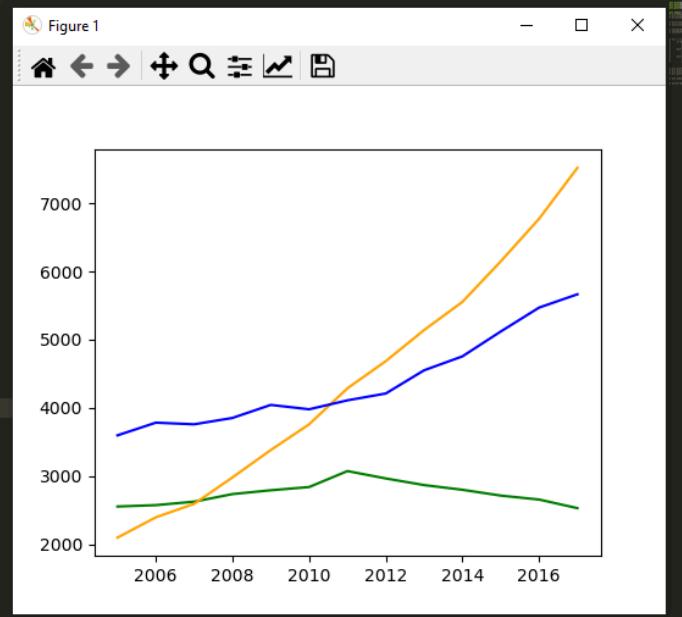
```
tut2q1.py tut2q2.py tut2q3.py
C: > University Stuff > Y2S2 > EE2211 > Tutut > tut2 > tut2.py ...
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4
5 dataframe = pd.read_csv('AnnualMotorVehiclePopulationbyVehicleType.csv')
6
7 # privateBusIndexList = dataframe[dataframe['type'] == "Private buses"].index.tolist()
8 # print(privateBusIndexList)
9
10 privateBusIndexList = dataframe.loc[dataframe['type'] == "Private buses"].index
11 excursionBusIndexList = dataframe.loc[dataframe['type'] == "Excursion buses"].index
12 omniBusIndexList = dataframe.loc[dataframe['type'] == "Omnibuses"].index
13
14 privateBusDataframe = dataframe.loc[privateBusIndexList]
15 excursionBusDataframe = dataframe.loc[excursionBusIndexList]
16 omniBusDataframe = dataframe.loc[omniBusIndexList]
17
18 # or without extracting the index first
19 # privateBusDataframe = dataframe.loc[dataframe['type'] == "Private buses"]
20 # excursionBusDataframe = dataframe.loc[dataframe['type'] == "Excursion buses"]
21 # omniBusDataframe = dataframe.loc[dataframe['type'] == "Omnibuses"]
22
23 plt.plot(privateBusDataframe['year'], privateBusDataframe['number'], color='green')
24 plt.plot(excursionBusDataframe['year'], excursionBusDataframe['number'], color='orange')
25 plt.plot(omniBusDataframe['year'], omniBusDataframe['number'], color='blue')
26
27 plt.show()
28
29
30
31 # privateBusNumber = []
32 # privateBusYear = []
33
34 # excursionBusNumber = []
35 # excursionBusYear = []
36
37 # omniBusNumber = []
```



- extracting index first

## Cut away the year after 2017

```
tut2q1.py tut2q2.py x tut2q3.py
C:\Users\Stuff\Y2S2\EE2211>Tutut>tut2> tut2q2.py > ...
10 privateBusIndexList = dataframe.loc[dataframe['type'] == "Private buses"].index
11 excursionBusIndexList = dataframe.loc[dataframe['type'] == "Excursion buses"].index
12 omniBusIndexList = dataframe.loc[dataframe['type'] == "Omnibuses"].index
13
14 privateBusDataframe = dataframe.loc[privateBusIndexList]
15 excursionBusDataframe = dataframe.loc[excursionBusIndexList]
16 omniBusDataframe = dataframe.loc[omniBusIndexList]
17
18 # or without extracting the index first
19 # privateBusDataframe = dataframe.loc[dataframe['type'] == "Private buses"]
20 # excursionBusDataframe = dataframe.loc[dataframe['type'] == "Excursion buses"]
21 # omniBusDataframe = dataframe.loc[dataframe['type'] == "Omnibuses"]
22
23 plt.plot(privateBusDataframe['year'], privateBusDataframe['number'], color='green')
24 plt.plot(excursionBusDataframe['year'], excursionBusDataframe['number'], color='orange')
25 plt.plot(omniBusDataframe['year'], omniBusDataframe['number'], color='blue')
26
27 plt.show()
28
29 yearCut = 2017
30
31 print("cutting away years after {}".format(yearCut))
32 privateBusDataframe = privateBusDataframe.loc[privateBusDataframe['year'] <= yearCut]
33 excursionBusDataframe = excursionBusDataframe.loc[excursionBusDataframe['year'] <= yearCut]
34 omniBusDataframe = omniBusDataframe.loc[omniBusDataframe['year'] <= yearCut]
35
36 plt.plot(privateBusDataframe['year'], privateBusDataframe['number'], color='green')
37 plt.plot(excursionBusDataframe['year'], excursionBusDataframe['number'], color='orange')
38 plt.plot(omniBusDataframe['year'], omniBusDataframe['number'], color='blue')
39
40 plt.show()
41 # privateBusNumber = []
42 # privateBusYear = []
43
44 # excursionBusNumber = []
45 # excursionBusYear = []
46
```

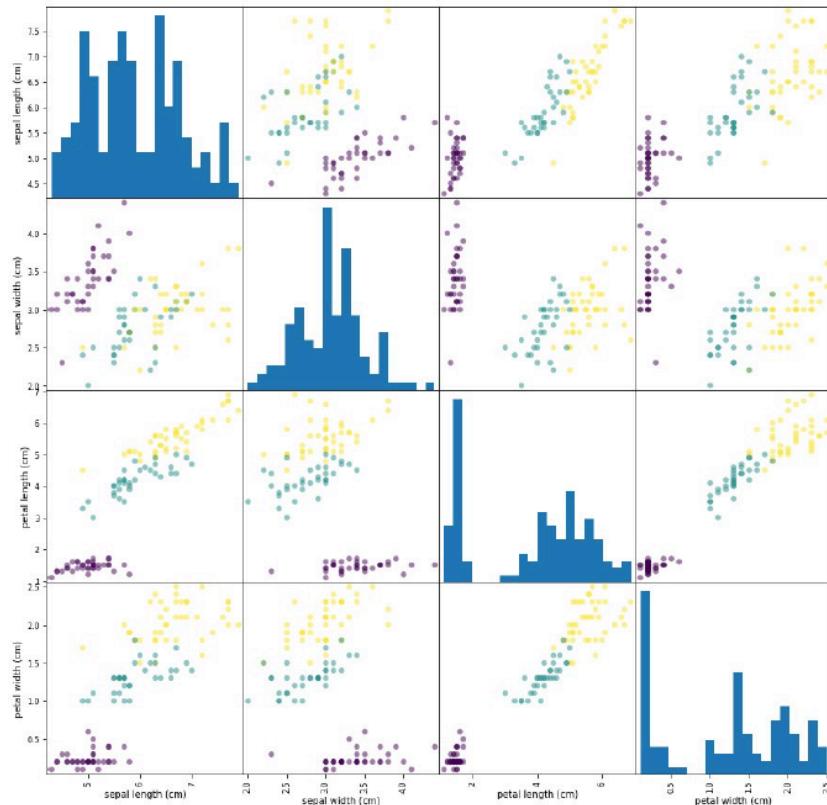


### Q3

(Data Reading and Visualization, distribution)

#### **Question 3:**

The “iris” flower data set consists of measurements such as the length, width of the petals, and the length, width of the sepals, all measured in centimeters, associated with each iris flower. Get the data set “from sklearn.datasets import load\_iris” and do a scatter plot as shown below. (Hint: you might need “from pandas.plotting import scatter\_matrix”)



C: > University Stuff > Y2S2 > EE2211 > Tutut > tut2 > tut2q3.py > ...

```
1  from sklearn.datasets import load_iris
2
3  import pandas as pd
4
5  from pandas.plotting import scatter_matrix
6
7  import matplotlib.pyplot as plt
8
9  # Load the iris dataset
10 iris_dataset = load_iris()
11
12 print(iris_dataset)
13
```

sklearn build in dataset for trying shit out

```
2, 2, 2, 2,
), 'frame': None, 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='|<U10'), 'DESCR': '...
es: 150 (50 in each of three classes)\n      :Number of Attributes: 4 numeric, predictive attributes and the
```

- target name is specified here,
  - target name/species name should be the output of learning algo

```
• {'data': array([[5.1, 3.5, 1.4, 0.2],  
•                 [4.9, 3. , 1.4, 0.2],  
•                 [4.7, 3.2, 1.3, 0.2],  
•                 [4.6, 3.1, 1.5, 0.2],  
•                 [5. , 3.6, 1.4, 0.2],  
•                 [5.4, 3.9, 1.7, 0.4],
```

```
ansactions\n      on Information Theory, May 1972, 431-433.\n      - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II\n', 'feature_names': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename': 'iris.csv', 'data_modul
```

- feature is here
  - feature should be the input to learning algo

## sklearn.model\_selection.train\_test\_split

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
```

[source]

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation, `next(ShuffleSplit().split(X, y))`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner.

Read more in the [User Guide](#).

<b>Parameters:</b>	<b>*arrays : sequence of indexables with same length / shape[0]</b> Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.
	<b>test_size : float or int, default=None</b> If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If <code>train_size</code> is also None, it will be set to 0.25.
	<b>train_size : float or int, default=None</b> If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.
	<b>random_state : int, RandomState instance or None, default=None</b> Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See <a href="#">Glossary</a> .
	<b>shuffle : bool, default=True</b> Whether or not to shuffle the data before splitting. If <code>shuffle=False</code> then <code>stratify</code> must be <code>None</code> .
	<b>stratify : array-like, default=None</b> If not <code>None</code> , data is split in a stratified fashion, using this as the class labels. Read more in the <a href="#">User Guide</a> .
<b>Returns:</b>	<b>splitting : list, length=2 * len(arrays)</b> List containing train-test split of inputs.  <i>New in version 0.16:</i> If the input is sparse, the output will be a <code>scipy.sparse.csr_matrix</code> . Else, output type is the same as the input type.

- random state is the seed
- [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.train\\_test\\_split.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

[https://pandas.pydata.org/docs/reference/api/pandas.plotting.scatter\\_matrix.html](https://pandas.pydata.org/docs/reference/api/pandas.plotting.scatter_matrix.html)

## Q5

(Missing Data)

### Question 5:

The Pima Indians Diabetes Dataset involves predicting the onset of diabetes within 5 years in Pima Indians given medical details. Download the Pima-Indians-Diabetes data from

<https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv>.

It is a binary (2-class) classification problem. The number of observations for each class is not balanced. There are 768 observations with 8 input variables and 1 output variable. The variable names are as follows:

0. Number of times pregnant.
1. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
2. Diastolic blood pressure (mm Hg).
3. Triceps skinfold thickness (mm).
4. 2-Hour serum insulin (mu U/ml).
5. Body mass index (weight in kg/(height in m)^2).
6. Diabetes pedigree function.
7. Age (years).
8. Class variable (0 or 1).

- (a) Print the summary statistics of this data set.
- (b) Count the number of “0” entries in columns [1,2,3,4,5].
- (c) Replace these “0” values by “NaN”.

(Hint: you might need the “.describe ()” and “.replace (0, numpy.NaN)” functions “from pandas import read\_csv”).

- {dataset}.describe() gives summary states, print out to read before processing data

## Tutorial 3

## Q5

`pdf(x, loc=0, scale=1)`

Probability density function.

`logpdf(x, loc=0, scale=1)`

Log of the probability density function.

`cdf(x, loc=0, scale=1)`

Cumulative distribution function.

The probability density function for `norm` is:

$$f(x) = \frac{\exp(-x^2/2)}{\sqrt{2\pi}}$$

for a real number  $x$ .

The probability density above is defined in the “standardized” form. To shift and/or scale the distribution use the `loc` and `scale` parameters. Specifically, `norm.pdf(x, loc, scale)` is identically equivalent to `norm.pdf(y) / scale` with  $y = (x - \text{loc}) / \text{scale}$ . Note that shifting the location of a distribution does not make it a “noncentral” distribution; noncentral generalizations of some distributions are available in separate classes.

- loc and scale is mean and standard deviation

## Q10-Q11

### Question 10: (Fill-in-blank)

The rank of the matrix  $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$  is \_BLANK\_.

2

### Question 11: (Fill-in-blank)

The rank of the matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$  is \_BLANK\_.

3

2, must do ero to ref

- find rank must do ref, for  $> 2 \times 2$ , cannot eye ball

## Tutorial 4

- $X^T X$  is always symmetrical if  $\det(X^T X) \neq 0$
- $XX^T$  is always symmetrical if  $\det(XX^T) \neq 0$

## Q1

(Systems of Linear Equations)

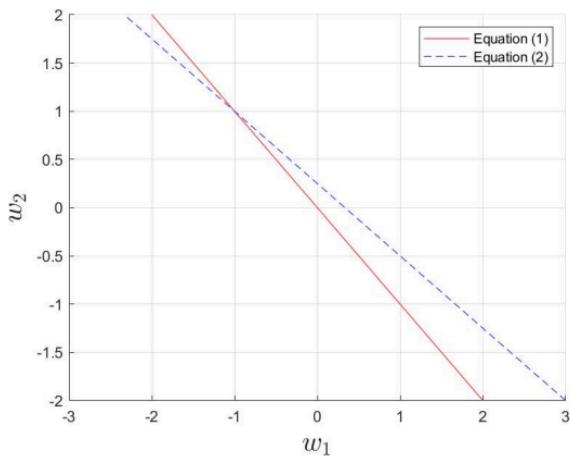
### Question 1:

Given  $\mathbf{X}\mathbf{w} = \mathbf{y}$  where  $\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 3 & 4 \end{bmatrix}$ ,  $\mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ .

even  
 $\det \mathbf{X} = 1 \neq 0$ , invertible

- What kind of system is this? (even-, over- or under-determined?)
- Is  $\mathbf{X}$  invertible? Why?
- Solve for  $\mathbf{w}$  if it is solvable.

$\mathbf{w} = (-1, 1)$



## Q2

(Systems of Linear Equations)

**Question 2:**

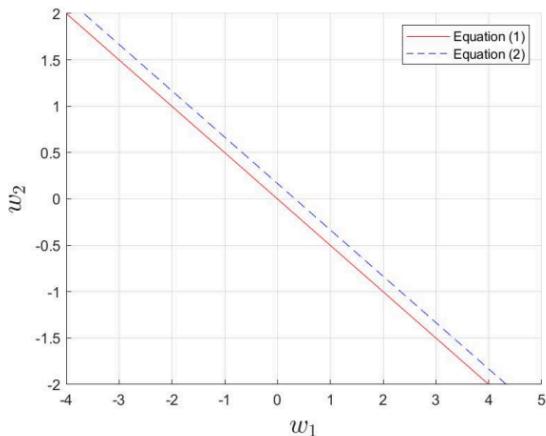
Given  $\mathbf{X}\mathbf{w} = \mathbf{y}$  where  $\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix}$ ,  $\mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ .

even  
 $\det \mathbf{X} = 0$ , non invertible

- (a) What kind of system is this? (even-, over- or under-determined?)
- (b) Is  $\mathbf{X}$  invertible? Why?
- (c) Solve for  $\mathbf{w}$  if it is solvable.

not solvable? can use least square?

- Two parallel lines confirm no solution, there is no least square solution also



• Let  $V \subseteq \mathbb{R}^n$  and  $S = \{u_1, u_2, \dots, u_k\}$  be a basis for  $V$ . Then the orthogonal projection of any vector  $w \in \mathbb{R}^n$  onto  $V = \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T w$ , where  $\mathbf{A} = (u_1 \ u_2 \ \dots \ u_k) \quad A_{n \times k} \quad n \geq k$

↳ since  $S$  is basis, columns of  $A$  are linearly independent ( $\text{Rank}(A) = k$ ,  $A$  is full ranked) so  $A^T A$  is invertible

$\therefore \underline{u} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T w$  is the unique solution to  $\mathbf{A}^T \mathbf{A} \underline{u} = \mathbf{A}^T w$

$\therefore$  project of  $w$  onto  $\text{Col}(A) = V = \mathbf{A} \underline{u} = \mathbf{A} (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T w \#$

- - column must be independent so cannot use this
  - if take out one column then can apply

**Question 3:**

Given  $\mathbf{X}\mathbf{w} = \mathbf{y}$  where  $\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 1 & -1 \end{bmatrix}$ ,  $\mathbf{y} = \begin{bmatrix} 0 \\ 0.1 \\ 1 \end{bmatrix}$ .

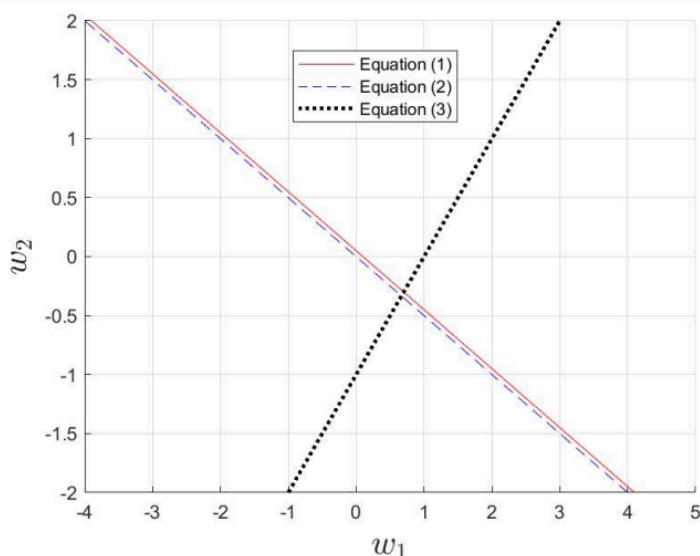
over

X is full rank, rank = 2, left inverse exist

$$\mathbf{w} = [0.68 \ 0.32]$$

- (a) What kind of system is this? (even-, over- or under-determined?)
- (b) Is  $\mathbf{X}$  invertible? Why?
- (c) Solve for  $\mathbf{w}$  if it is solvable.

- Two parallel line and one non parallel line intersecting at two points, the  $\mathbf{w}$  which is least square, gives the  $\mathbf{w}$  between ish between the two intersection points

**Q6****Question 6:**

Given  $\mathbf{w}^T \mathbf{X} = \mathbf{y}^T$  where

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 1 & -1 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

$$\mathbf{X}^T \mathbf{w} = \mathbf{y}$$

after transpose its under determined  
over

rank 2 = full rank, left inverse exist

$$\mathbf{w} = [0.06666667 \ 0.13333333 \ -0.33333333]$$

$$= [1/15, 2/15, -1/3]$$

- (a) What kind of system is this? (even-, over- or under-determined?)
- (b) Is  $\mathbf{X}$  invertible? Why?
- (c) Solve for  $\mathbf{w}$  if it is solvable.

- (a) This is an under-determined system (there are 3 unknowns with 2 equations).
- (b)  $\mathbf{X}$  is NOT invertible but  $\mathbf{X}^T \mathbf{X}$  is. The determinant of  $\mathbf{X}^T \mathbf{X} = 6 \times 21 - 9 \times 9 = 45$ .
- (c) A constrained solution (exact) is given by

$\hat{\mathbf{w}}^T = (\mathbf{X}\mathbf{a})^T$  (The 3-dimensional vector  $\mathbf{w}$  can be constrained by projecting  $\mathbf{X}$  onto a 2-dimensional vector  $\mathbf{a}$ )

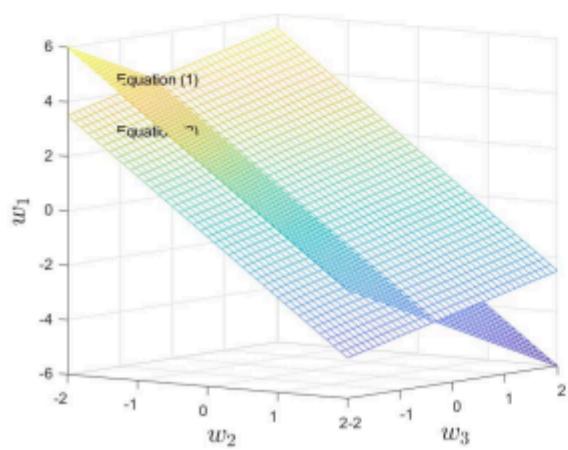
$$= \mathbf{a}^T \mathbf{X}^T$$

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

$$= [0 \quad 1] \begin{bmatrix} 0.4667 & -0.2 \\ -0.2 & 0.1333 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & -1 \end{bmatrix}$$

$$= [0.0667 \quad 0.1333 \quad -0.3333].$$

Note:  $\dim(\mathbf{X})$  is  $3 \times 2$ ,  $\dim(\mathbf{a})$  is  $2 \times 1$ , estimation is done/constrained on/to the lower dimension of  $(3 \times 2)$  and then projected back to the higher dimension 3.



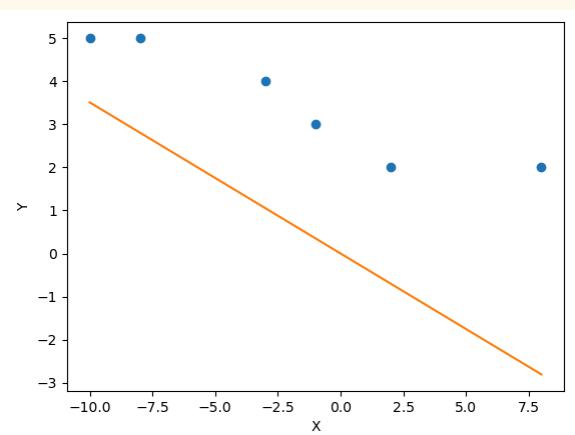
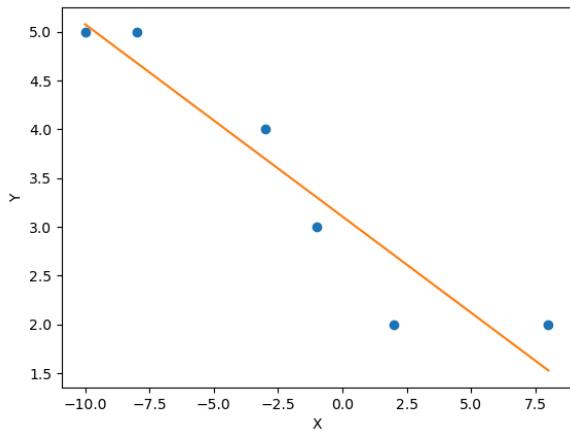
## Tutorial 5

### Q1

```
# Question 1
print("Question 1")

X = np.array([[-10], [-8], [-3], [-1], [2], [8]])
Y = np.array([[5], [5], [4], [3], [2], [2]])
w = helper.linearRegressionWithBias(X, Y, printResult=False, printFeature=1)
print(w)

X = np.array([[-10], [-8], [-3], [-1], [2], [8]])
Y = np.array([[5], [5], [4], [3], [2], [2]])
w = helper.linearRegressionWithoutBias(X, Y, printResult=False, printFeature=0)
print(w)
```



```
W =
[[ 3.10550459]
[-0.19724771]]
```

```
W =
[[ -0.35123967]]
```

bias term allows the regression to move away from the origin and reduce error

## Q2

*even-determined*

$$X = \begin{bmatrix} 1 & 0 & 1 \\ 2 & -1 & 1 \\ 1 & 1 & 5 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- even determined with full rank can get answer by finding the inverse

```
Question 2
rank(X) = 3
W =
[[ 0.33333333]
[-0.66666667]
[ 0.66666667]]
MSE = 1.472540356412555e-29
Ytest1_nobias =
[[3.66666667]]
Ytest2_nobias =
[[-3.66666667]]
rank(X) = 3
W =
[[-1.5]
[-1. ]
[ 3. ]
[-0.25]]
MSE = 34.895833333333336
Ytest1_bias =
[[3.5]]
Ytest2_bias =
[[12.75]]
```

$$\hat{\omega} = X^\top (X X^\top)^{-1} y$$

$$= \begin{bmatrix} -0.1429 \\ 0.5238 \\ -0.4762 \\ 0.6190 \end{bmatrix}$$

- should use right inverse

$$y = X \hat{\omega} = \begin{bmatrix} 1 & 0 & 1 \\ 2 & -1 & 1 \\ 1 & 1 & 5 \end{bmatrix} \begin{bmatrix} -0.1429 \\ 0.5238 \\ -0.4762 \\ 0.6190 \end{bmatrix}$$

$$= \begin{bmatrix} 3.3333 \\ -2.6190 \end{bmatrix}$$

```

# this is wrong, should use right inverse
W_bias = helper.linearRegressionWithBias(
    X, Y, printResult=True, printFeature=None)

# add in bias
Xtest1 = np.array([[1, -1, 2, 8]])
Xtest2 = np.array([[1, 1, 5, -1]])

Ytest1_bias = helper.testData(Xtest1, W_bias, printResult=False)
print("Ytest1_bias =\n", Ytest1_bias)
Ytest2_bias = helper.testData(Xtest2, W_bias, printResult=False)
print("Ytest2_bias =\n", Ytest2_bias)

# after adding bias, its a wider matrix so should solve with right inverse
W_bias_right = helper.rightInverse(helper.paddingOfOnes(X)) @ Y
print("W_bias_right =\n", W_bias_right)

```

You, 1 second ago • Uncommitted changes

```

Ytest1_bias = helper.testData(Xtest1, W_bias_right, printResult=False)
print("Ytest1_bias =\n", Ytest1_bias)
Ytest2_bias = helper.testData(Xtest2, W_bias_right, printResult=False)
print("Ytest2_bias =\n", Ytest2_bias)

```

```

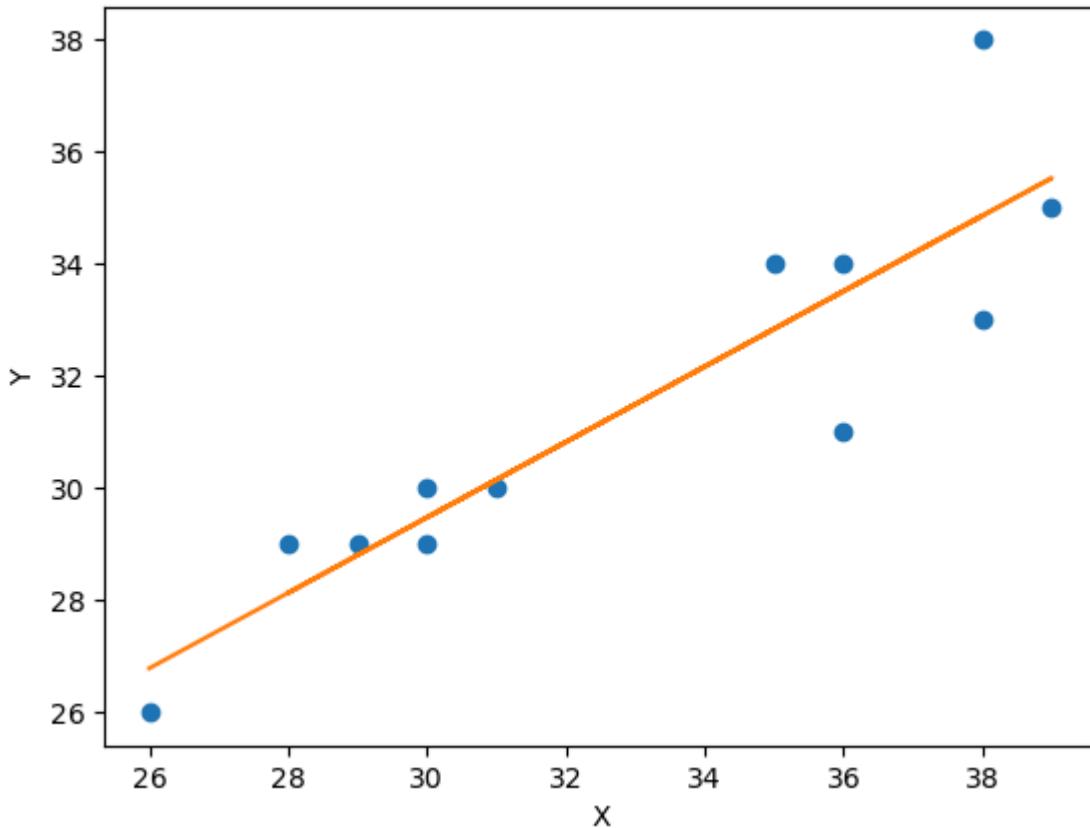
W_bias_right =
[[ -0.14285714]
 [ 0.52380952]
 [ -0.47619048]
 [ 0.61904762]]
Ytest1_bias =
[[3.333333333333333]]
Ytest2_bias =
[[-2.61904762]]

```

The wide matrix has a left inverse as the  $3 \times 3$  is already full rank, so the dim of row is 3, after padding ones, dimension remains

- hence can find left inverse, but it's not the most accurate regression as there is no interpretation of least squares error in this case

### Q3



```

rank(X) = 2
W =
[[9.3
[0.67272727]]
MSE = 1.9530303030303038
Ytest1 =
[[29.48181818]]
Ytest2 =
[[12.66363636]]

```

always include bias if not specified

- predicted that 30 student buy around 30 books, one per student
- predicted that 5 student buy around 12 books, 2 per student
- 5 student is out of the range of regression, not appropriate to use to same regression
  - not sure whether the regression still fits, cannot extrapolate
  - like gae1000

## Q4

Semester	Students	Books
1	36	31
2	26	20
3	35	34
4	39	35
5	26	20
6	30	30
7	31	30
8	38	38
9	36	34
10	38	33
11	26	20
12	26	20

Without removing duplicates VS removing the duplicates

```

X = np.array([[36], [26], [35], [39], [26], [30],
|           | [31], [38], [36], [38], [26], [26]])
Y = np.array([[31], [20], [34], [35], [20], [30],
|           | [30], [38], [34], [33], [20], [20]])

# xw = Y regression
w = helper.linearRegressionWithBias(X, Y, printResult=True, printFeature=1)

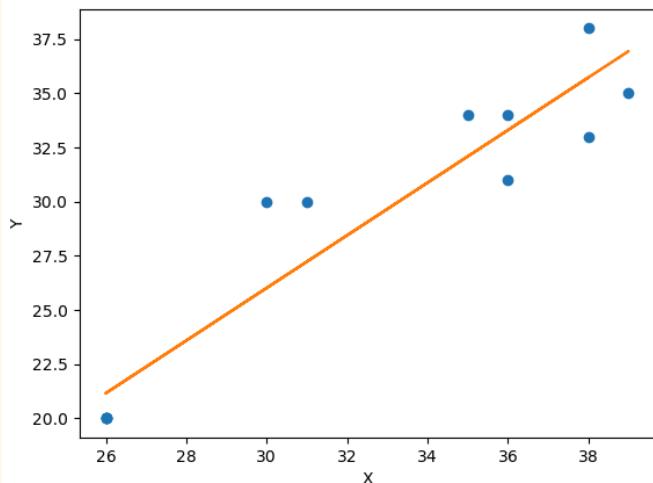
# predict
Xtest1 = np.array([[1, 30]])
Ytest1 = helper.testData(Xtest1, w, printResult=False)
print("Ytest1 =\n", Ytest1)

```

```

rank(X) = 2
W =
[-10.41257051]
[ 1.21434327]
MSE = 4.562181036798282
Ytest1 =
[[26.01772764]]

```



```

# remove duplicates
print("X =\n", X)
print("Y =\n", Y)
combined = np.hstack((X, Y))
print("combined =\n", combined)
new_array = [tuple(row) for row in combined]
print("new_array =\n", new_array)
uniques = np.unique(new_array, axis=0)
print("uniques =\n", uniques)
X_unique = uniques[:, 0].reshape(-1, 1)
Y_unique = uniques[:, 1].reshape(-1, 1)
# In this context, -1 is used as a placeholder for "figure out what the dimension should be". It means that the length in that dimension is inferred from the length of the array and remaining dimensions.
# So, .reshape(-1, 1) means to reshape the array to have one column and as many rows as necessary to accommodate the original data.
print("X_unique =\n", X_unique)
print("Y_unique =\n", Y_unique)

# xw = Y regression
w_unique = helper.linearRegressionWithBias(X_unique, Y_unique, printResult=True, printFeature=1)

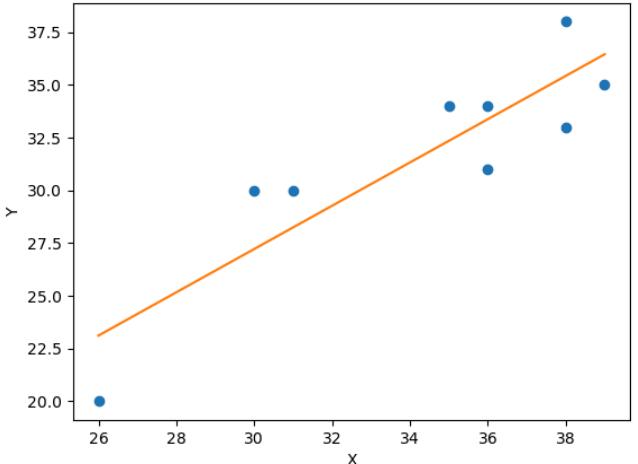
# predict
Ytest1_unique = helper.testData(Xtest1, w_unique, printResult=False)
print("Ytest1_unique =\n", Ytest1_unique)

```

```

rank(X) = 2
W =
[[ -3.55844156]
 [ 1.02597403]]
MSE = 4.8773448773448775
Ytest1_unique =
[[27.22077922]]

```



- can see that the effect (26, 20) has on the regression is less

Combine so it's easier to compare

```

# Combining the plots for easier comparison
plt.plot(X, Y, 'o', color='blue')

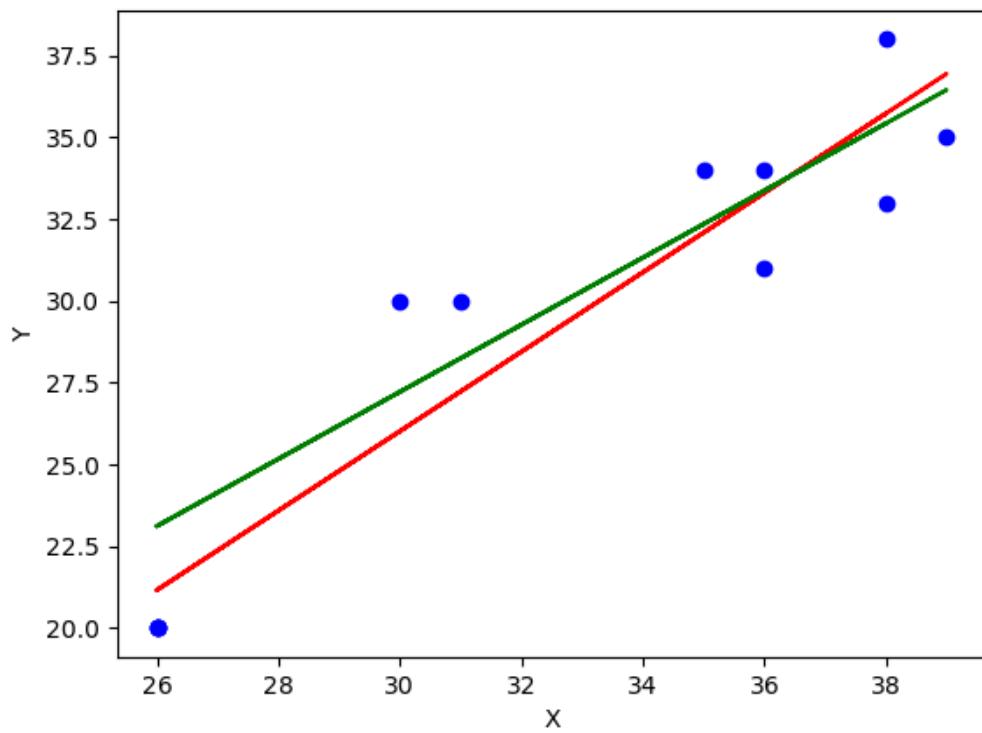
# plot the graph for the unique data, X needs to be padded for the offset
plt.plot(X, np.hstack((np.ones((X.shape[0], 1)), X)) @ w, '-', color='red')

# plot the graph for the unique data
plt.plot(X, np.hstack((np.ones((X.shape[0], 1)), X)) @ w_unique, '--', color='green')

plt.xlabel('X')
plt.ylabel('Y')

plt.show()

```



- more optimistic after purging before 37

## Q5

```
# Question 5 #####
print("Question 5 #####")
exp_df = pd.read_csv("GovernmentExpenditureonEducation.csv")
exp_df.info()

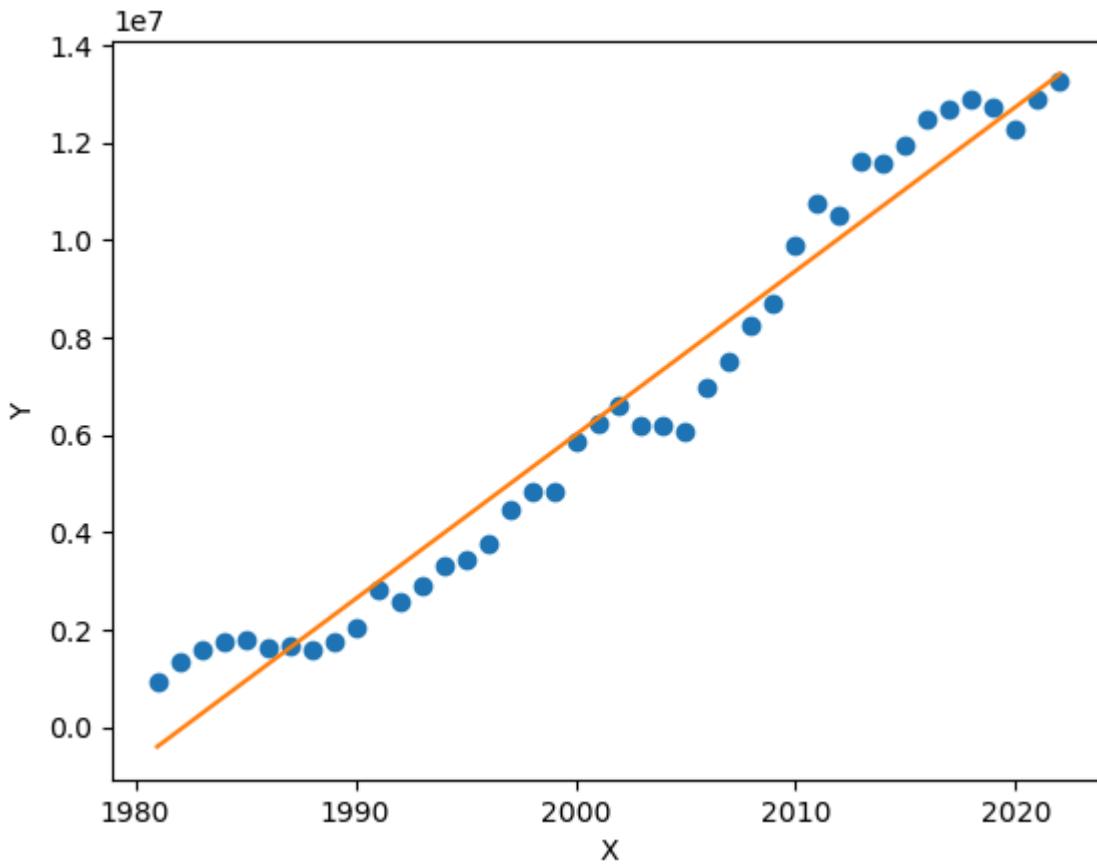
X = np.array(exp_df['year']).reshape(-1, 1)
Y = np.array(exp_df['total_expenditure_on_education']).reshape(-1, 1)
print("X =\n", X)
print("Y =\n", Y)

# xw = Y regression
W = helper.linearRegressionWithBias(X, Y, printResult=True, printFeature=1)

x_test = helper.paddingOfOnes(np.array([[2021]]))

y_predict = helper.testData(x_test, W, printResult=False)

print("y_predict =\n", y_predict)      You, 1 second ago • Uncommitted changes
```



```
y_predict =
[[13075210.44420362]]
```

## Answer:

The predicted educational expenditure in year 2021 is 12102904.270643068.

- diff from answer probably cuz diff dataset

## Q6

```
+ CategoryInfo          : InvalidArgument: (:) [Invoke-WebRequest], ParameterBindingException
+ FullyQualifiedErrorId : MissingArgument,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\University Stuff\Y2S2\EE2211\Tutut\tut5> curl -o ./wine.csv https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv
PS C:\University Stuff\Y2S2\EE2211\Tutut\tut5> ls
Directory: C:\University Stuff\Y2S2\EE2211\Tutut\tut5

Mode                LastWriteTime         Length Name
----                -----        ----
-a----   2/20/2024 12:43 PM           4325 tut5.py
-a----   2/19/2024 10:11 PM      193462 Tutorial15-Questions.pdf
-a----   2/20/2024 12:54 PM          84199 wine.csv

Ask Copilot or type / for commands >
```

- downloading files using curl

```

# Question 6 #####
print("Question 6 ######")
# wine_df = pd.read_csv("wine.csv", sep=';')
# print(wine_df)

wine_df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv", sep=";")
wine_df.info()

```

- get data without downloading

```

wine_df = pd.read_csv("wine.csv", sep=';')
print(wine_df)

X = np.array(wine_df.drop('quality', axis=1)) # drop the quality column, the other columns are the features
Y = np.array(wine_df['quality'].tolist()).reshape(-1, 1) # the quality column is the output

print("X\n", X)
print("Y\n", Y)

Xtrain = X[:1500]
Ytrain = Y[:1500]
print("Xtrain\n", Xtrain)
print("Ytrain\n", Ytrain)

Xtest = X[1500:]
Ytest = Y[1500:]
print("Xtest\n", Xtest)
print("Ytest\n", Ytest)

# Training
w = helper.linearRegressionWithBias(Xtrain, Ytrain, printResult=True, printFeature=None)

# Prediction Test
Ytest_pred = helper.testData(helper.paddingOfOnes(Xtest), w, printResult=False)
print("Ytest_pred\n", Ytest_pred)

MSE = mean_squared_error(Ytest_pred, Ytest)
print("MSE = ", MSE)

```

```

W =
[[ 2.22330327e+01]
 [ 2.68702621e-02]
 [-1.12838019e+00]
 [-2.06141685e-01]
 [ 1.22000584e-02]
 [-1.77718503e+00]
 [ 4.29357454e-03]
 [-3.18953315e-03]
 [-1.81795124e+01]
 [-3.98142390e-01]
 [ 8.92474793e-01]
 [ 2.77147239e-01]]
MSE =  0.4216923253120808

```

MSE on training data

```

MSE =  0.3435263811803818

```

MSE on testing data, lower than 0.2 better regression

## Q8

### Question 8:

MCQ: There could be more than one answer.

Suppose  $f(\mathbf{x})$  is a scalar function of  $d$  variables where  $\mathbf{x}$  is a  $d \times 1$  vector. Then, without taking data points into consideration, the outcome of differentiation of  $f(\mathbf{x})$  w.r.t.  $\mathbf{x}$  is

- a) a scalar
- b) a  $d \times 1$  vector
- c) a  $d \times d$  matrix
- d) a  $d \times d \times d$  tensor
- e) None of the above

Answer: b)  $\frac{df}{dx} = \begin{bmatrix} a \\ a \\ \vdots \\ a \end{bmatrix} \left\{ \text{row} = d \right. \right)$

$$f(x) = \frac{ax}{\downarrow \text{constant}} + \frac{b}{\downarrow \text{independent of } x} ?$$

$$f(x) = \left[ \begin{array}{c} ax_1 + b \\ ax_2 + b \\ \vdots \\ ax_d + b \end{array} \right] \left\{ \text{row} = d \right. \right)$$

## Q9

```
X = np.array([[3, -1, 0], [5, 1, 2], [9, -1, 3], [-6, 7, 2], [3, -2, 0]])
Y = np.array([[1, -1], [-1, 0], [1, 2], [0, 3], [1, -2]])

# Training
w = helper.linearRegressionWithBias(X, Y, printResult=True, printFeature=None)

Xtest = helper.paddingOfOnes(np.array([[8, 0, 2]]))
Ytest = helper.testData(Xtest, w, printResult=False)
print("Ytest\n", Ytest)
```

```
rank(X) = 4
W =
[[ 1.14668974 -0.95997404]
 [-0.630463 -0.33427088]
 [-1.10601471 -0.24426655]
 [ 1.3595846  1.77953267]]
MSE = 0.254110774556469
Ytest
 [[-1.17784509 -0.07507572]]
```

## Tutorial 6

### Q1

Derivation as homework (see tutorial 6).

Hint: start off with  $(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})\mathbf{w} = \mathbf{X}^T \mathbf{y}$  and make use of  $\mathbf{w} = \mathbf{X}^T \mathbf{a}$  and  $\mathbf{a} = \lambda^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w})$ ,  $\lambda > 0$

Given  $(X^T X + \lambda I) w = X^T y$

derive  $\hat{w} = X^T (X X^T + \lambda I)^{-1} y$

Hint:  $w = X^T a$  where

$$a = \lambda^{-1} (y - Xw), \lambda > 0$$

$$\begin{aligned} : X^T X w + \lambda I w &= X^T y \\ \lambda w &= X^T y - X^T X w \\ &= X^T (y - Xw) \end{aligned}$$

$$\begin{aligned} w &= \frac{\lambda^{-1}}{\lambda} X^T (y - Xw) \\ &= X^T a \quad \text{--- (1)} \end{aligned}$$

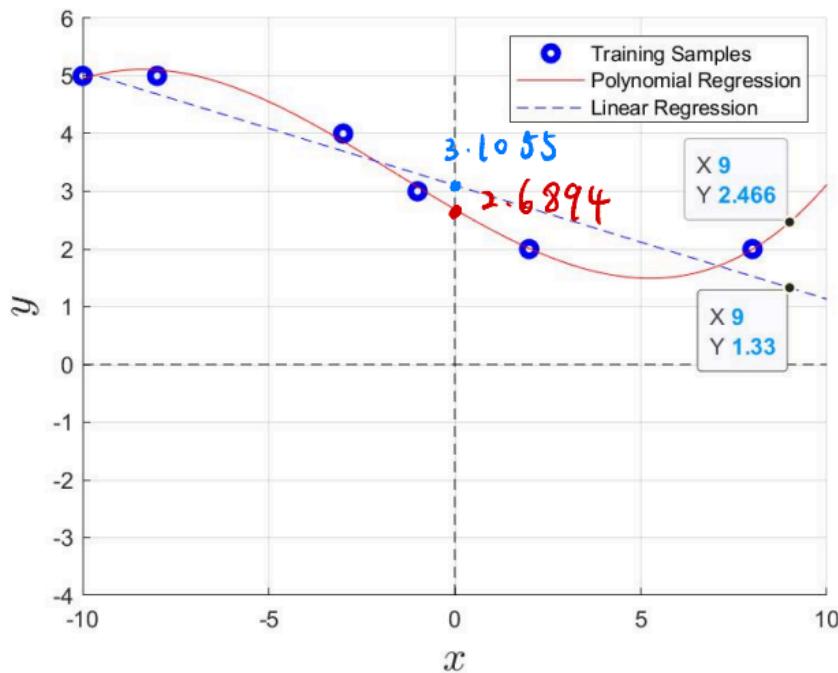
$$\begin{aligned} : a &= \lambda^{-1} (y - Xw) \\ \lambda a &= y - Xw \\ &= y - X X^T a \end{aligned}$$

$$X X^T a + \lambda a = y$$

$$(X X^T + \lambda) a = y$$

$$a = (X X^T + \lambda I)^{-1} y \quad \text{--- (2)}$$

## Q2



- Can see that when model fits the data better

### Q3

(a) Write down the expression for a 3<sup>rd</sup> order polynomial model having a 3-dimensional input.

- let d be the number of feature and k be the order of polynomial to be generated from the features

$$\text{number of terms in polynomial} = {}^{k+d}C_d = {}^{k+d}C_k = {}^6C_3 = 20$$

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$\begin{aligned}
 &+ w_{12} x_1 x_2 + w_{23} x_2 x_3 + w_{13} x_1 x_3 + w_{11} x_1^2 + w_{22} x_2^2 + w_{33} x_3^2 \\
 &+ w_{211} x_2 x_1^2 + w_{311} x_3 x_1^2 + w_{122} x_1 x_2^2 + w_{322} x_3 x_2^2 + w_{133} x_1 x_3^2 + w_{233} x_2 x_3^2 \\
 &+ w_{123} x_1 x_2 x_3 + w_{111} x_1^3 + w_{222} x_2^3 + w_{333} x_3^3 \quad (1)
 \end{aligned}$$

(c) Given  $\mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ , can a unique solution be obtained in dual form? If so, proceed to solve it.

(d) Given  $\mathbf{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ , can the primal ridge regression be applied to obtain a unique solution? If so, proceed to solve it.

- C is possible as a wide matrix can be found in dual form with no ridge = right inverse
  - very likely to have linearly independent columns
- D is not possible without ridge as cannot find inverse

Here, at  $\lambda = 0.0001$ , we observe a very close solution to that in (c) even though (d) constitutes an approximation whereas (c) is exact.

```
[polynomialRegression]: m < d, method used = dual ridge regression
[ridgeRegressionDual]: W =
[[ 0. ]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0.1]
 [-0.1]
 [ 0. ]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0.1]
 [-0.1]
 [ 0. ]
 [-0.1]
 [ 0. ]
 [ 0.1]
 [-0.1]
 [ 0. ]
 [-0.1]
 [ 0.1]
 [-0.1]
 [ 0. ]]
[ridgeRegressionDual]: MSE = 0.0
[ridgeRegressionPrimal]: W =
[[ 9.99966915e-07]
 [ 9.99971917e-07]
 [-9.99980001e-02]
 [ 9.99970780e-07]
 [ 9.99969984e-07]
 [-9.99980001e-02]
 [ 9.99968506e-07]
 [ 9.99980001e-02]
 [-9.99980001e-02]
 [ 9.99973622e-07]
 [ 9.99967710e-07]
 [-9.99980000e-02]
 [ 9.99965664e-07]
 [ 9.99980001e-02]
 [-9.99980001e-02]
 [ 9.99967369e-07]
 [-9.99980001e-02]
 [ 9.99980001e-02]
 [-9.99980000e-02]
 [ 9.99967597e-07]]
[ridgeRegressionPrimal]: MSE = 9.999490719753084e-11
```

```

print("Question 3 #####")
X = np.array([[1, 0, 1], [1, -1, 1]])
Y = np.array([[0], [1]])

order = 3
poly = PolynomialFeatures(order)
print(poly)
P = poly.fit_transform(X)
print("matrix P =\n", P)

# order 3 with 3 features means 6C3 monomials
# P is 2x20
# m < d use dual with no ridge which is just the right inverse
w_poly = helper.rightInverse(P) @ Y
print("w_poly =\n", w_poly)

# or just use linear regression in helper, without bias since we have bias in P already
w_poly = helper.linearRegressionWithoutBias(P, Y, printResult=True, printFeature=None)

# or use dual form with ridge = 0
w_poly = helper.ridgeRegressionDual(P, Y, 0, True)

# if want to use primal form, need to do with ridge
w_poly = helper.ridgeRegressionPrimal(P, Y, 0.0001, True)

w_poly = helper.polynomialRegression(X, Y, order, 0, True)

```

## Q4

(Binary Classification, Python)

### Question 4:

Given the training data:

$$\begin{aligned} \{x = -1\} &\rightarrow \{y = \text{class1}\} \\ \{x = 0\} &\rightarrow \{y = \text{class1}\} \\ \{x = 0.5\} &\rightarrow \{y = \text{class2}\} \\ \{x = 0.3\} &\rightarrow \{y = \text{class1}\} \\ \{x = 0.8\} &\rightarrow \{y = \text{class2}\} \end{aligned}$$

Predict the class label for  $\{x = -0.1\}$  and  $\{x = 0.4\}$  using linear regression with signum discrimination.

```

91 |     print("Question 4 #####")
92 |     X = np.array([[-1.0], [0.0], [0.5], [0.3], [0.8]])
93 |     Y = np.array([[1], [1], [2], [1], [2]])
94 |
95 |     # encode class1 to 1 and class2 to -1
96 |     Yclass = np.where(Y == 1, 1, -1)
97 |
98 |     w = helper.linearRegressionWithBias(X, Yclass, True)
99 |
100 |    Xtest = np.array([-0.1, [0.4]])
101 |    Ypred = helper.testData(helper.paddingOfOnes(Xtest), w, True)
102 |
103 |    YpredClass = np.where(Ypred > 0, 1, 2)
104 |    print("YpredClass =\n", YpredClass) You, 1 second ago • Uncom

```

```

import numpy as np
from numpy.linalg import inv
from sklearn.preprocessing import PolynomialFeatures
X = np.array([[1,-1], [1,0], [1,0.5], [1,0.3], [1,0.8]])
y = np.array([1, 1, -1, 1, -1])
## Linear regression for classification
w = inv(X.T @ X) @ X.T @ y
print(w)
Xt = np.array([[1,-0.1], [1,0.4]])
y_predict = Xt @ w
print(y_predict)
y_class_predict = np.sign(y_predict)
print(y_class_predict)

```

- Use np.where or np.sign to get from prediction to classification

## Q5

```

# Question 5 #####
print("Question 5 #####")
X = np.array([-1.0, 0.0, 0.5, 0.3, 0.8])
Y = np.array([1, 1, 2, 3, 2])

# one hot encoding
oneHotEncoder = OneHotEncoder(sparse=False)
print("oneHotEncoder", oneHotEncoder)
Yencoded = oneHotEncoder.fit_transform(Y)
print("Yencoded =\n", Yencoded)

# linear model
w_lin = helper.linearRegressionWithBias(X, Yencoded, True)
Xtest = np.array([-0.1, 0.4])
Ypred = helper.testData(helper.paddingOfOnes(Xtest), w_lin, True)

YpredClass = oneHotEncoder.inverse_transform(Ypred)
print("YpredClass =\n", YpredClass)

# polynomial model with order 5
w_poly = helper.polynomialRegression(X, Yencoded, 5, 0, True)
Xtest = np.array([-0.1, 0.4])
Ypred = helper.testPolyReg(Xtest, w_poly, 5, True)

YpredClass = oneHotEncoder.inverse_transform(Ypred)
print("YpredClass =\n", YpredClass)

```

```
YpredClass = oneHotEncoder.inverse_transform(Ypred)
```

- can use to this get back class from encoded values

Note that polynomial regression dimension to check to use primal or dual is the polynomial matrix, not original X

```
# if m >= d, use primal ridge regression, if lam == 0, its just left i
if P.shape[0] >= P.shape[1]:      You, 2 minutes ago • Uncommitted ch
    print("[polynomialRegression]: m >= d, method used = primal ridge")
    W = ridgeRegressionPrimal(P, Y, Lam, printResult=printResult)
# else m < d, use dual ridge regression, if lam == 0, its just right i
else:
    print("[polynomialRegression]: m < d, method used = dual ridge reg")
    W = ridgeRegressionDual(P, Y, Lam, printResult=printResult)

return W
```

## Q6

(Multi-Category Classification, Python)

**Question 6 (continued from Q3 of Tutorial 2):**

Get the data set “from sklearn.datasets import load\_iris”. Use Python to perform the following tasks.

- Split the database into two sets: 74% of samples for training, and 26% of samples for testing. Hint: you might want to utilize from sklearn.model\_selection import train\_test\_split for the splitting.
- Construct the target output using one-hot encoding.
- Perform a linear regression for classification (without inclusion of ridge, utilizing one-hot encoding for the learning target) and compute the number of test samples that are classified correctly.
- Using the same training and test sets as in above, perform a 2<sup>nd</sup> order polynomial regression for classification (again, without inclusion of ridge, utilizing one-hot encoding for the learning target) and compute the number of test samples that are classified correctly. Hint: you might want to use from sklearn.preprocessing import PolynomialFeatures for generation of the polynomial matrix.

- answer used linear regression with no bias for some reason and get 28/39, using bias get 30/39

## Q7

### Question 7

MCQ: there could be more than one answer. Given three samples of two-dimensional data points  $\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 3 & 3 \end{bmatrix}$

with corresponding target vector  $\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ . Suppose you want to use a full third-order polynomial model to fit these data. Which of the following is/are true?

- a) The polynomials model has 10 parameters to learn
- b) The polynomial learning system is an under-determined one
- c) The learning of the polynomial model has infinite number of solutions
- d) The input matrix  $\mathbf{X}$  has linearly dependent samples
- e) None of the above

**Answer:** a, b, c, d

## Q8

### Question 8

MCQ: there could be more than one answer. Which of the following is/are true?

- a) The polynomial model can be used to solve problems with nonlinear decision boundary.
- b) The ridge regression cannot be applied to multi-target regression.
- c) The solution for learning feature  $\mathbf{X}$  with target  $\mathbf{y}$  based on linear ridge regression can be written as  $\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$  for  $\lambda > 0$ . As  $\lambda$  increases,  $\hat{\mathbf{w}}^T \hat{\mathbf{w}}$  decreases.
- d) If there are four data samples with two input features each, the full second-order polynomial model is an over-determined system.

**Answer:** a, c

## Tutorial 7

### Q1

This question explores the use of Pearson's correlation as a feature selection metric. We are given the following training dataset.

	Datapoint 1	Datapoint 2	Datapoint 3	Datapoint 4	Datapoint 5
Feature 1	0.3510	2.1812	0.2415	-0.1096	0.1544
Feature 2	1.1796	2.1068	1.7753	1.2747	2.0851
Feature 3	-0.9852	1.3766	-1.3244	-0.6316	-0.8320
Target y	0.2758	1.4392	-0.4611	0.6154	1.0006

What are the top two features we should select if we use Pearson's correlation as a feature selection metric? Here's the definition of Pearson's correlation. Given  $N$  pairs of datapoints

$\{(a_1, b_1), (a_2, b_2), \dots, (a_N, b_N)\}$ , the Pearson's correlation  $r$  is defined as  $r =$

$$\frac{\frac{1}{N} \sum_{n=1}^N (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\frac{1}{N} \sum_{n=1}^N (a_i - \bar{a})^2} \sqrt{\frac{1}{N} \sum_{n=1}^N (b_i - \bar{b})^2}}$$
, where  $\bar{a} = \frac{1}{N} \sum_{n=1}^N a_n$  and  $\bar{b} = \frac{1}{N} \sum_{n=1}^N b_n$  are the empirical means of  $a$  and  $b$  respectively.  $\sigma_a = \sqrt{\frac{1}{N} \sum_{n=1}^N (a_i - \bar{a})^2}$  and  $\sigma_b = \sqrt{\frac{1}{N} \sum_{n=1}^N (b_i - \bar{b})^2}$  are referred to as the empirical standard deviation of  $a$  and  $b$ .  $Cov(a, b) = \frac{1}{N} \sum_{n=1}^N (a_i - \bar{a})(b_i - \bar{b})$  is known as the empirical covariance between  $a$  and  $b$

```
28 def correlation(Feature, Target_y):
29     mean_Feature = sum(Feature)/len(Feature)
30     mean_Target_y = sum(Target_y)/len(Target_y)
31
32     sd_Feature = (sum([(i - mean_Feature)**2 for i in Feature])/len(Feature))**0.5
33     sd_Target_y = (sum([(i - mean_Target_y)**2 for i in Target_y])/len(Target_y))**0.5
34
35     cov_Feature_Target_y = sum([(Feature[i] - mean_Feature)*(Target_y[i] - mean_Target_y) for i in range(len(Feature))])/len(Feature)
36
37     return cov_Feature_Target_y/(sd_Feature*sd_Target_y)
38
39 print("Correlation between Feature_1 and Target_y: ", correlation(Feature_1, Target_y))
40 print("Correlation between Feature_2 and Target_y: ", correlation(Feature_2, Target_y))
41 print("Correlation between Feature_3 and Target_y: ", correlation(Feature_3, Target_y))
```

- Correlation between Feature\_1 and Target\_y: 0.5987722845227617
- Correlation between Feature\_2 and Target\_y: 0.4536921850899784
- Correlation between Feature\_3 and Target\_y: 0.8008916458435772

- Choose feature 1 and 3 as the correlation is the highest

## Q2

- (a) Use the polynomial model from orders 1 to 6 to train and test the data without regularization. Plot the Mean Squared Errors (MSE) over orders from 1 to 6 for both the training and the test sets. Which model order provides the best MSE in the training and test sets? Why? [Hint: the underlying data was generated using a quadratic function + noise]
- (b) Use regularization (ridge regression)  $\lambda=1$  for all orders and repeat the same analyses. Compare the plots of (a) and (b). What do you see? [Hint: the underlying data was generated using a quadratic function + noise]

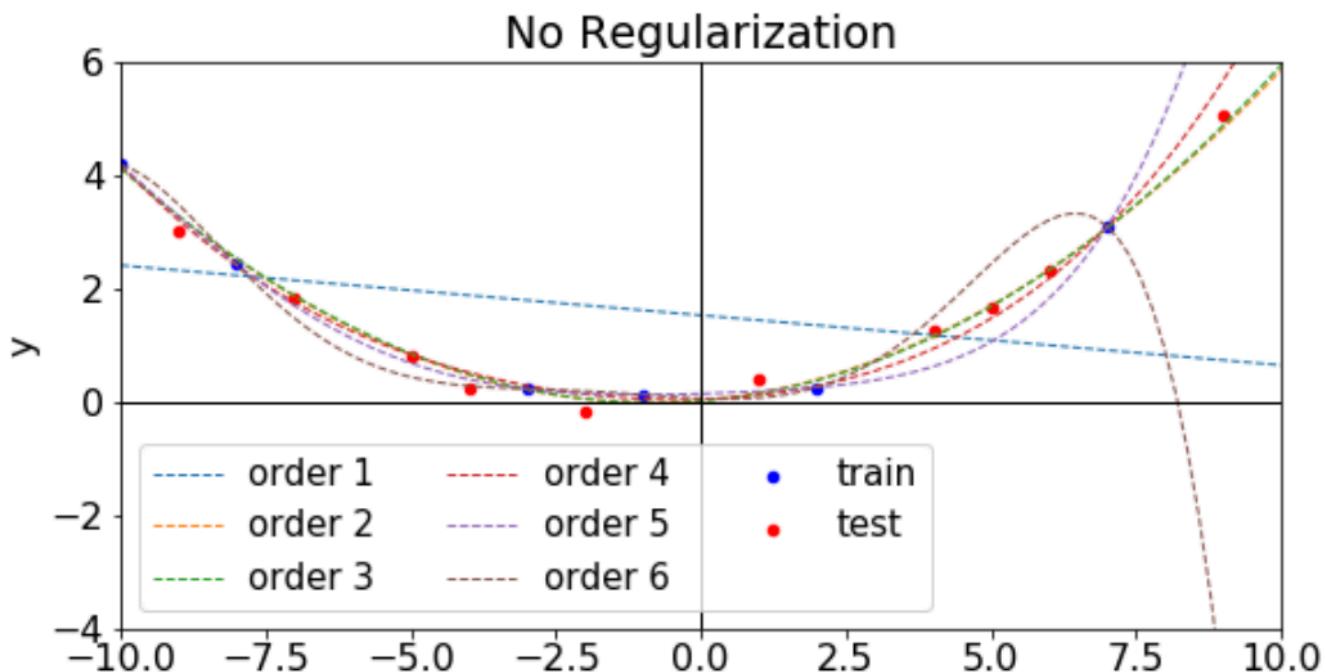
### **Answer:**

Please see code: Tut7\_Q2\_yeo.py

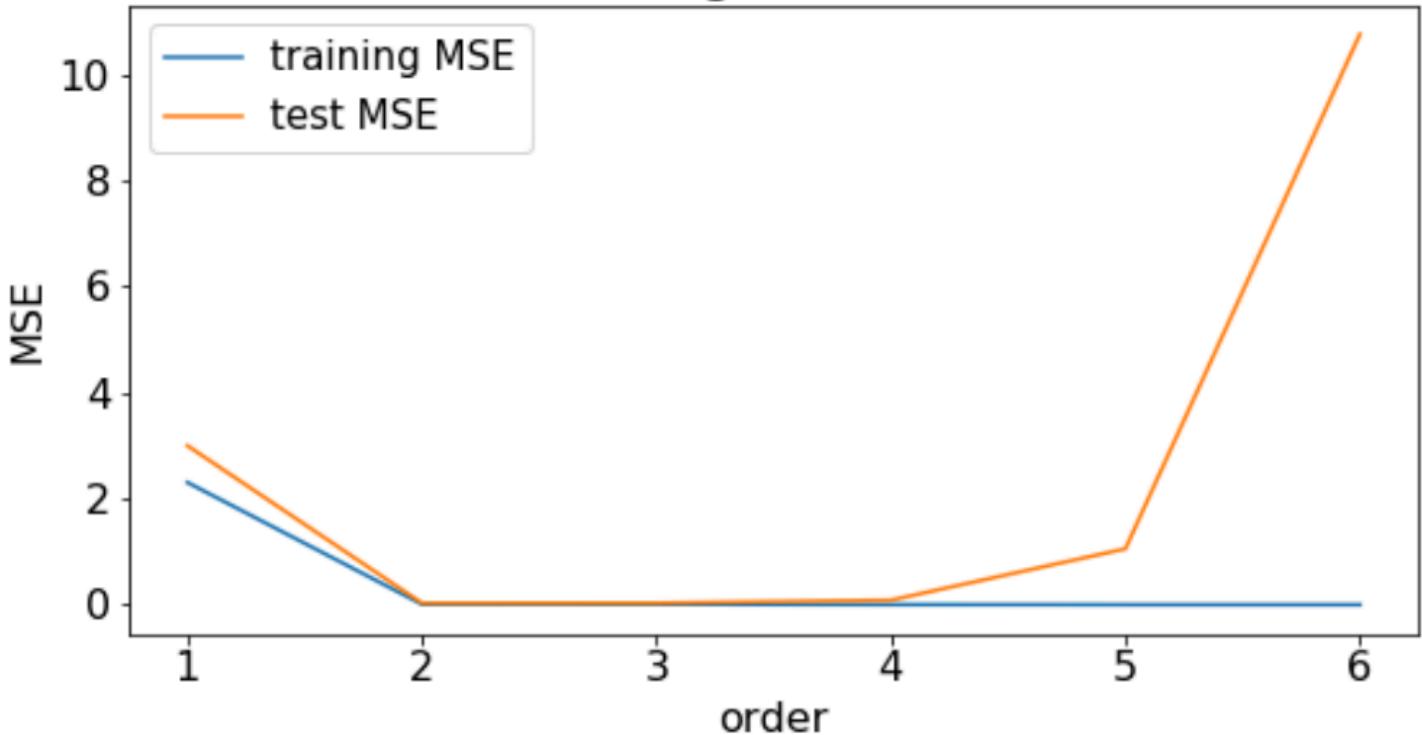
### Q1(a)

- There are 6 training data points. For polynomial orders 1 to 5, we can use the **Primal** solution:  $\hat{w} = (P^T P)^{-1} P^T y$ . For order 6, there are 7 unknowns, so the system is under-determined, so we use the **Dual** solution:  $\hat{w} = P^T (P P^T)^{-1} y$
- See plots for estimated polynomial curves and MSE below

- ===== No Regularization =====
- Training MSE: [2.3071. 8.4408e-03 8.3026e-03 1.7348e-03 3.8606e-25 2.3656e-17]
- Test MSE: [ 3.0006 0.0296 0.0301 0.0854 1.0548 10.7674]
- Observe that the estimated polynomial curves for orders 5 and 6 pass through the training samples exactly. This results in training MSE of virtually 0, but high test MSE => overfitting
- Note that even though the true underlying data came from a quadratic model (order = 2), estimated polynomial curves for orders 2, 3 and 4 have relatively low training and test MSE.
- Polynomial curve of order 1 (linear curves) have high training and test MSE => underfitting



## No Regularization



```
[-2.62/00563]]  
('MSE_train_noRidge', 'MSE_test_noRidge', 'MSE_train_Ridge', 'MSE_test_Ridge')  
(2.3070729253981557, 3.0006478167488604, 2.3585531011197105, 3.2755677280832316)  
(0.008440835471198293, 0.029594585750404377, 0.008456477077018444, 0.030169873887064692)  
(0.0083026159070491, 0.03011128182353708, 0.008356013483798833, 0.031417148304253864)  
(0.0017347778137363516, 0.08535502039490317, 0.001807976222390347, 0.09393142698402805)  
(3.860607659026733e-25, 1.054823988048789, 0.0007264963730617796, 0.4368968008239905)  
(1.4781923101220727e-17, 10.76736645005596, 0.00019347846088121716, 6.02023594611855)
```

- for train MSE, for some reason increases

```
[ 5.54142550e+21]
('MSE_train_noRidge', 'MSE_test_noRidge', 'MSE_train_Ridge', 'MSE_test_Ridge')
(2.3070729253981557, 3.0006478167488604, 2.3585531011197105, 3.2755677280832316)
(0.008440835471198293, 0.029594585750404377, 0.008456477077018444, 0.030169873887064692)
(0.0083026159070491, 0.03011128182353708, 0.008356013483798833, 0.031417148304253864)
(0.0017347778137363516, 0.08535502039490317, 0.001807976222390347, 0.09393142698402805)
(3.860067659026733e-25, 1.054823988048789, 0.0007264963730617796, 0.4368968008239905)
(1.4781923101220727e-17, 10.76736645005596, 0.00019347846088121716, 6.02023594611855)
(2.322752682468194e-16, 2.1505668981811428, 0.00021006219280030516, 16.33645034466042)
(1.635082563175943e-13, 27960.903818582236, 0.00010800976550158301, 20417.500086622582)
(7.113868601377851e-13, 219535.84955085, 8.41256691087616e-05, 97265.57730060458)
(4.767473480811858e-11, 54514281.52764622, 5.6910199103238124e-05, 41637067.03682511)
(1.666257829328616e-11, 1542545843.257259, 4.4119837134425876e-05, 1087658517.1758678)
(1.5277977719499835e-06, 151326613254.225, 3.3773647129502766e-05, 119956137029.13225)
(2.7761470910044558e-05, 7393807698458.597, 5.7422315456067714e-05, 5830875412081.731)
(0.0032152303381050348, 556561130549815.6, 0.00014439714233174114, 457329457536198.3)
(0.6189767329154813, 3.39450986456064e+16, 0.43446959729681445, 2.8169729344701828e+16)
(0.5976784745583653, 2.39692444703143e+18, 24.29692452140203, 2.0291043877662339e+18)
(1813.9405527831304, 1.600681833024551e+20, 5.3656611862921855, 1.3698126252774977e+20)
(4416.478105427893, 1.1257028780433011e+22, 247593.18732521168, 9.752294534310506e+21)
(35169180.139082186, 7.8384679322898e+23, 863665.4299649261, 6.852808165237522e+23)
(437806573.3362065, 5.573909189344849e+25, 1437590165.4157057, 4.915679704880598e+25)
(54363115032.24775, 3.9728969536740905e+27, 93451945.5148596, 3.529417380588829e+27)
(20228698671828.74, 2.8621954027467253e+29, 8376469656186.126, 2.559739677501799e+29)
(796258879459632.9, 2.0720982611261944e+31, 2673922426524157.0, 1.8640551589162345e+31)
(1.6995423612561018e+17, 1.5103811368606187e+33, 122489315722576.45, 1.366001984660548e+33)
(8.766783869007027e+18, 1.1062701867092796e+35, 5.388930576659721e+19, 1.0053333890018944e+35)
(2.478245581488931e+19, 8.143939992360202e+36, 1.92136663956865e+20, 7.433314585780862e+36)
(7.719741875774136e+22, 6.0206983375663976e+38, 9.304635630396475e+22, 5.517283981243925e+38)
(3.770567479611282e+23, 4.469111116366611e+40, 6.066402635734564e+24, 4.1104242367166677e+40)
(6.539490325654769e+26, 3.329400414693486e+42, 1.1736285997980705e+27, 3.072489457251329e+42)
```

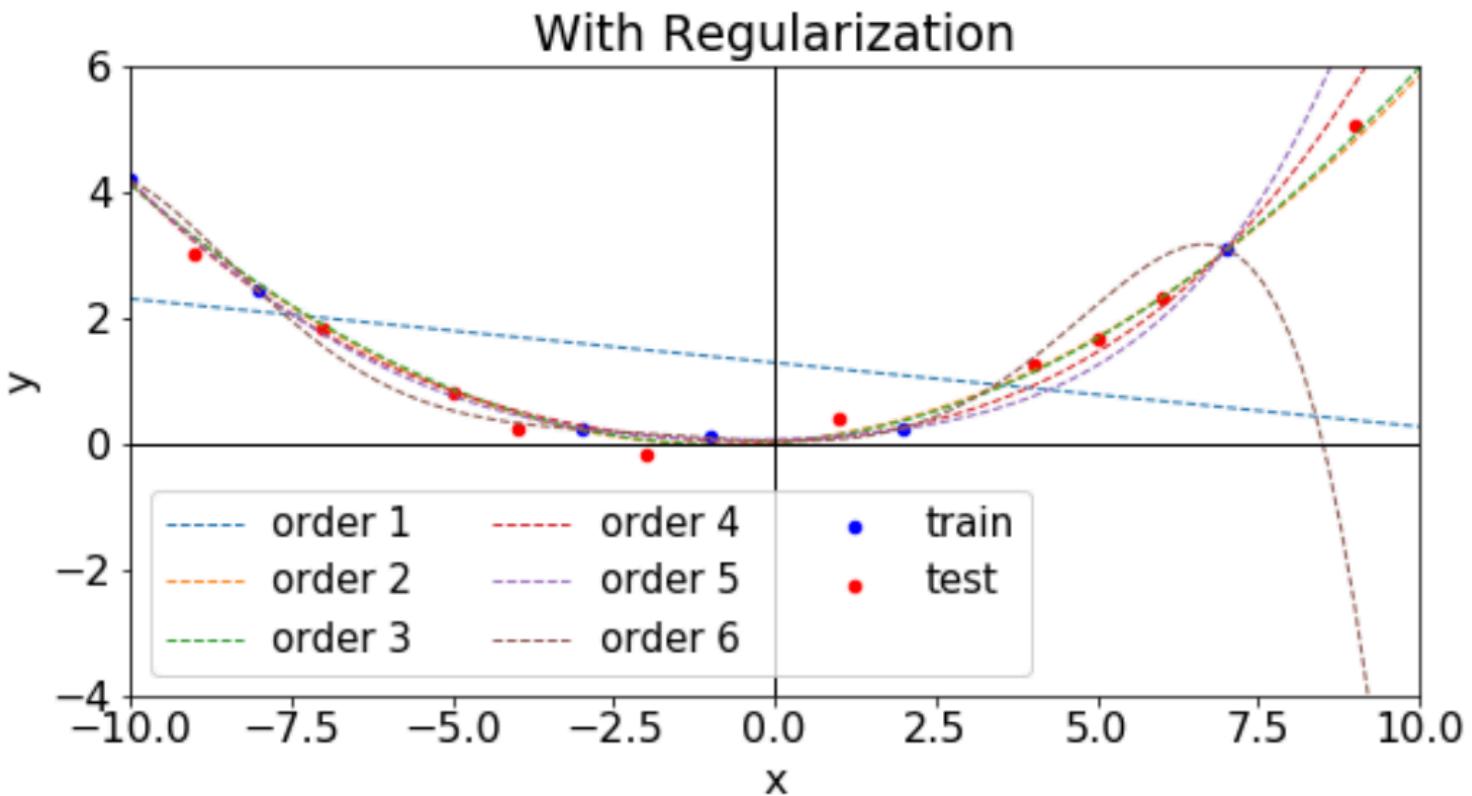
- PS C:\University Stuff\Y252\EE2211\Tutut\tut7> []

- same shit with model ans

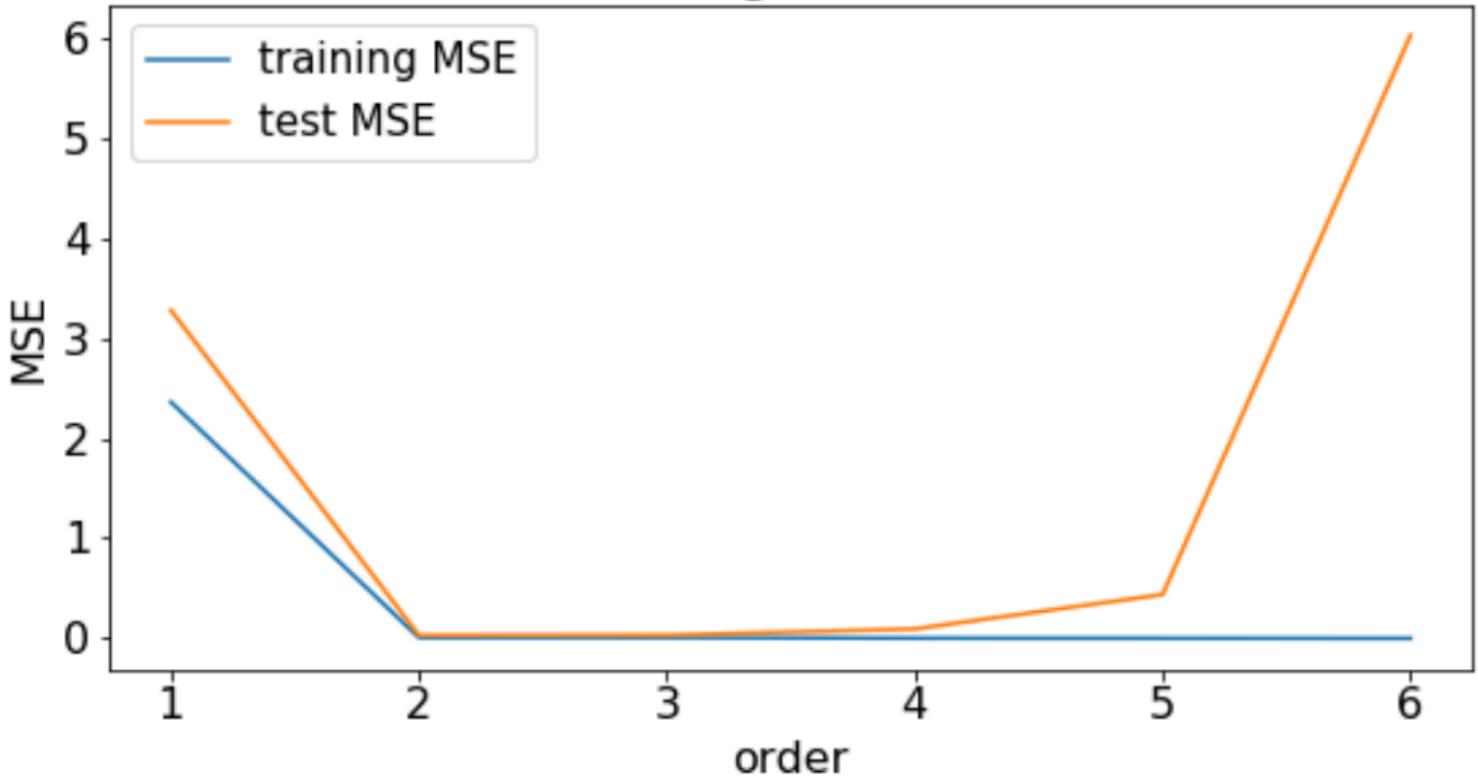
### Q1(b)

- With regularization, we can simply use the primal solution even for order 6:  $\hat{w} = (P^T P + \lambda I)^{-1} P^T y$ .
- See plots for estimated polynomial curves and MSE below
- Note that without regularisation, must use dual or primal carefully, but with regularisation can use primal or dual will get the same thing

- ===== Regularization =====
- Training MSE: [2.3586 8.4565e-03 8.3560e-03 1.8080e-03 7.2650e-04 1.9348e-04]
- Test MSE: [3.2756 0.0302 0.0314 0.0939 0.4369 6.0202]
- With the regularization, none of the polynomial curves passes through the training samples exactly. In the case of orders 5 and 6, test MSE dropped from 1.0548 (order 5) and 10.7674 (order 6) to 0.4369 (order 5) and 6.0202 (order 6) after regularization was added. Thus, the regularization reduces the overfitting
- On the other hand, the regularization did not help orders 1 to 4. Observe that the test MSE actually went up. In this case, the regularization was overly strong, which ended up hurting these orders.
- Note that the addition of the regularization does not necessarily favor the lower order (compared with the higher order). For example, the loss for order 2 is 0.018476, but the loss for order 6 is 0.0036308, which is lower. Thus, adding regularization does not help us choose the best polynomial order for best prediction.
- In fact, selecting the best regularization parameter or model complexity (e.g., polynomial order) is typically done through inner-loop (nested) cross-validation or training-validation-test scheme (**which will be taught in a future lecture**).



# With Regularization



## Tutorial 8

derivative is a linear map, a linear approximation of small change, it maps small movement to how much the output changes

- temperature of 3d space, derivative must map  $3 \times 1$  to  $1 \times 1$ , a  $1 \times 3$  row vector, so derivative is always a row vector
- the gradient is the direction of maximum derivative, so must be a  $3 \times 1$  column position vector, and from dot product it must be the derivative transposed

## Q1

Suppose we are minimizing  $f(x) = x^4$  with respect to  $x$ . We initialize  $x$  to be 2. We perform gradient descent with learning rate 0.1. What is the value of  $x$  after the first iteration?

```
37 print(helper.gradientDescentApprox(Lambda a:a**4, 2.5, learning_rate, num_iters)[0])
38 print(helper.gradientDescentApprox(Lambda a:a**4, 2.5, learning_rate, num_iters)[1])
39
40 print(helper.gradientDescentApprox(Lambda b:np.sin(b)**2, 0.6, learning_rate, num_iters)[0])
41 print(helper.gradientDescentApprox(Lambda b:np.sin(b)**2, 0.6, learning_rate, num_iters)[1])
42
43 print(helper.gradientDescent(Lambda cd:cd[0]**5 + (cd[1]**2) * np.sin(cd[1]), Lambda cd:(5*cd[0]**4, (2*cd[1]) * np.sin(cd[1]) + (cd[1]**2) * np.cos(cd[1])), (2, 3), learning_rate, num_iters))
44 print(helper.gradientDescent(Lambda cd:cd[0]**5 + (cd[1]**2) * np.sin(cd[1]), Lambda cd:(5*cd[0]**4, (2*cd[1]) * np.sin(cd[1]) + (cd[1]**2) * np.cos(cd[1])), (2, 3), learning_rate, num_iters))
45
46 # by experiment, the best smallStep is 1e-1
47 # its best if can input fprime manually, then check with approximation, scalar function seems to be fine
48 print(helper.gradientDescentApprox(Lambda cd:cd[0]**5 + (cd[1]**2) * np.sin(cd[1]), (2, 3), learning_rate, num_iters, smallStep=1e-1)[0])
49 print(helper.gradientDescentApprox(Lambda cd:cd[0]**5 + (cd[1]**2) * np.sin(cd[1]), (2, 3), learning_rate, num_iters, smallStep=1e-1)[1])
```

- can use helper function, but better check with manual derivative and not rely on approximated one

```
... Question 1 ...
print(helper.gradientDescentApprox(Lambda x:x**4, 2, 0.1, 1)[0])
print(helper.gradientDescent(Lambda x:x**4, Lambda x:4*x**3, 2, 0.1, 1)[0])
```

```
[[ 2. ]
 [-1.2]]
 [[ 2. ]
 [-1.2]]
 PC-GW-MacBook-Pro:~/Desktop
```

## Q2

Please consider the csv file (government-expenditure-on-education.csv), which depicts the government's educational expenditure over the years. We would like to predict expenditure as a function of year. To do this, fit an exponential model  $f(\mathbf{x}, \mathbf{w}) = \exp(-\mathbf{x}^T \mathbf{w})$  with squared error loss to estimate  $\mathbf{w}$  based on the csv file and gradient descent. In other words,  $C(\mathbf{w}) = \sum_{i=1}^m (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$ .

Note that even though year is one dimensional, we should add the bias term, so  $\mathbf{x} = [1 \text{ year}]^T$ . Furthermore, optimizing the exponential function is tricky (because a small change in  $\mathbf{w}$  can lead to large change in  $f$ ). Therefore for the purpose of optimization, divide the "year" variable by the largest year (2018) and divide the "expenditure" by the largest expenditure, so that the resulting normalized year and normalized expenditure variables have maximum values of 1. Use a learning rate of 0.03 and run gradient descent for 2000000 iterations.

- Plot the cost function  $C(\mathbf{w})$  as a function of the number of iterations.
- Use the fitted parameters to plot the predicted educational expenditure from year 1981 to year 2023.
- Repeat (a) using a learning rate of 0.1 and learning rate of 0.001. What do you observe relative to (a)?

The goal of this question is for you to code up gradient descent, so I will provide you with the gradient derivation. First, please note that in general,  $\nabla_{\mathbf{w}}(\mathbf{x}^T \mathbf{w}) = \mathbf{x}$ . To see this:

$$\nabla_{\mathbf{w}}(\mathbf{x}^T \mathbf{w}) = \begin{bmatrix} \frac{\partial(\mathbf{x}^T \mathbf{w})}{\partial w_1} \\ \frac{\partial(\mathbf{x}^T \mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial(\mathbf{x}^T \mathbf{w})}{\partial w_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial(w_1 x_1 + w_2 x_2 + \dots + w_d x_d)}{\partial w_1} \\ \frac{\partial(w_1 x_1 + w_2 x_2 + \dots + w_d x_d)}{\partial w_2} \\ \vdots \\ \frac{\partial(w_1 x_1 + w_2 x_2 + \dots + w_d x_d)}{\partial w_d} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \mathbf{x} \quad (1)$$

The above equality will be very useful for the other questions as well. Now, going back to our question,

$$\nabla_{\mathbf{w}} C(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^m (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 \quad (2)$$

$$= \sum_{i=1}^m \nabla_{\mathbf{w}} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 \quad (3)$$

$$= \sum_{i=1}^m 2(f(\mathbf{x}_i, \mathbf{w}) - y_i) \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}) \quad \text{chain rule} \quad (4)$$

$$= \sum_{i=1}^m 2(f(\mathbf{x}_i, \mathbf{w}) - y_i) \nabla_{\mathbf{w}} \exp(-\mathbf{x}_i^T \mathbf{w}) \quad (5)$$

$$= - \sum_{i=1}^m 2(f(\mathbf{x}_i, \mathbf{w}) - y_i) \exp(-\mathbf{x}_i^T \mathbf{w}) \nabla_{\mathbf{w}} (\mathbf{x}_i^T \mathbf{w}) \quad \text{chain rule} \quad (6)$$

$$= - \sum_{i=1}^m 2(f(\mathbf{x}_i, \mathbf{w}) - y_i) \exp(-\mathbf{x}_i^T \mathbf{w}) \mathbf{x}_i \quad (7)$$

$$= - \sum_{i=1}^m 2(f(\mathbf{x}_i, \mathbf{w}) - y_i) f(\mathbf{x}_i, \mathbf{w}) \mathbf{x}_i \quad (8)$$

- From the defined model and square error loss, we defined a cost function
- To perform gradient descent, the derivative of the cost function with respect to  $\mathbf{w}$  is needed

Please see code Tut8\_Q2.py.

- See Figure 1 below. The cost function decreases rapidly at first and then converges to a final value.
- See Figure 2 below.
- See Figures 3 and 4 below. A learning rate of 0.1 is too big, so the cost function does not decrease monotonically with increasing iterations, but instead fluctuates a lot without convergence. The final cost function value is much worse than (a). On the other hand, a learning rate of 0.001 is too small. So even though the cost function decreases monotonically with increasing iterations, gradient descent has not converged even after 2000000 iterations. The final cost function value is much worse than (a).

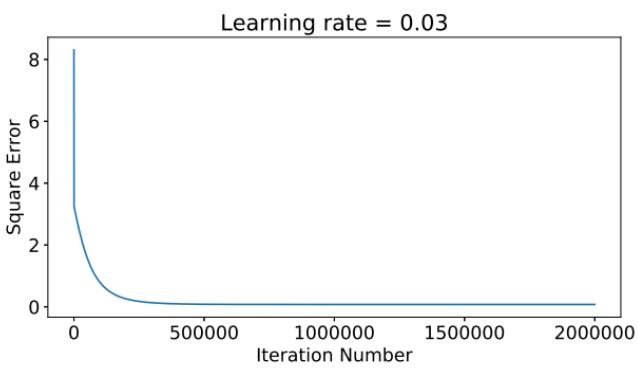


Figure 1: Cost function value as a function of iterations.

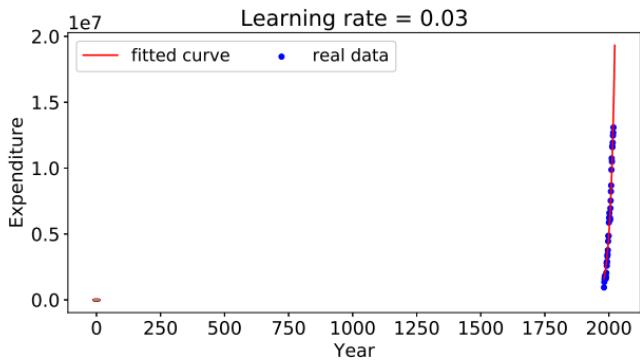


Figure 2: Fitted curve from 1981 to 2023

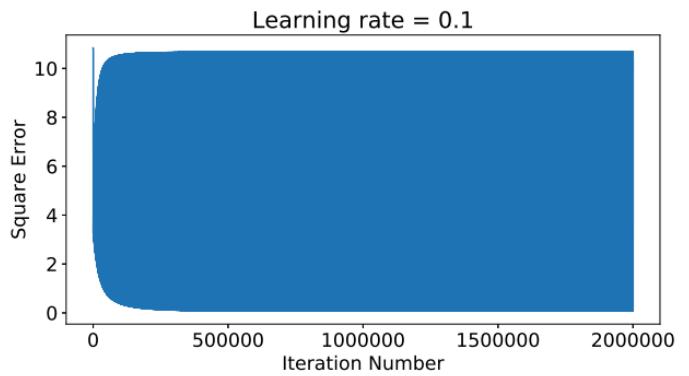


Figure 3: Cost function value as a function of iterations.

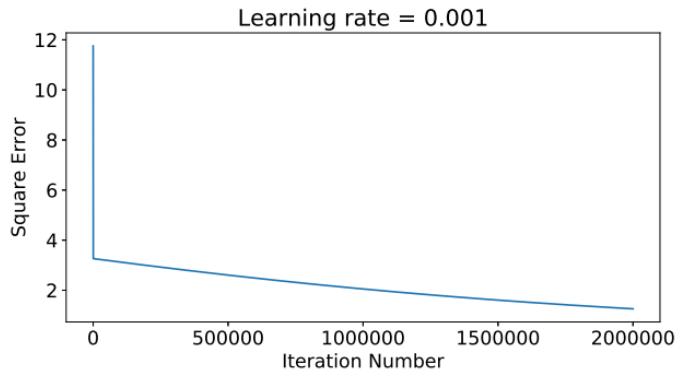


Figure 4: Cost function value as a function of iterations.

### Q3

Given the linear learning model  $f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T \mathbf{w}$ , where  $\mathbf{x} \in \mathbb{R}^d$ . Consider the loss function  $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4$ , where  $i$  indexes the  $i$ -th training sample. The final cost function is  $C(\mathbf{w}) = \sum_{i=1}^m L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ , where  $m$  is the total number of training samples. Derive the gradient of the cost function with respect to  $\mathbf{w}$ .

$$\nabla_{\mathbf{w}} C(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^m (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4 \quad (9)$$

$$= \sum_{i=1}^m \nabla_{\mathbf{w}} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4 \quad (10)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}) \quad \text{chain rule} \quad (11)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \nabla_{\mathbf{w}} (\mathbf{x}_i^T \mathbf{w}) \quad (12)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \mathbf{x}_i \quad (13)$$

- Use general formula and sub it values

#### Q4

Repeat Question 3 using  $f(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{x}^T \mathbf{w})$ , where  $\sigma(a) = \frac{1}{1+\exp(-\beta a)}$

**Answer:**

$$\nabla_{\mathbf{w}} C(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^m (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4 \quad (14)$$

$$= \sum_{i=1}^m \nabla_{\mathbf{w}} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4 \quad (15)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}) \quad \text{chain rule} \quad (16)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \nabla_{\mathbf{w}} \sigma(\mathbf{x}_i^T \mathbf{w}) \quad (17)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \frac{\partial \sigma(a)}{\partial a} \nabla_{\mathbf{w}} (\mathbf{x}_i^T \mathbf{w}) \quad \text{chain rule} \quad (18)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \frac{\partial \sigma(a)}{\partial a} \mathbf{x}_i \quad (19)$$

So we just have to evaluate  $\frac{\partial \sigma(a)}{\partial a}$  and plug it into the above equation. Note that  $\frac{\partial \sigma(a)}{\partial a}$  is evaluated at  $a = \mathbf{x}_i^T \mathbf{w}$ , so

$$\frac{\partial \sigma(a)}{\partial a} = \frac{\partial}{\partial a} \left( \frac{1}{1 + \exp(-\beta a)} \right) \quad (20)$$

$$= -\frac{1}{(1 + e^{-\beta a})^2} \frac{\partial(1 + e^{-\beta a})}{\partial a} \quad (21)$$

$$= \frac{\beta}{(1 + e^{-\beta a})^2} e^{-\beta a} \quad (22)$$

$$= \frac{\beta}{(1 + e^{-\beta a})^2} (1 + e^{-\beta a} - 1) \quad (23)$$

$$= \beta \left( \frac{1}{1 + e^{-\beta a}} - \frac{1}{(1 + e^{-\beta a})^2} \right) \quad (24)$$

$$= \beta (\sigma(a) - \sigma^2(a)) \quad (25)$$

$$= \beta \sigma(a)(1 - \sigma(a)) \quad (26)$$

$$= \beta \sigma(\mathbf{x}_i^T \mathbf{w})(1 - \sigma(\mathbf{x}_i^T \mathbf{w})) \quad (27)$$

Therefore,

$$\nabla_{\mathbf{w}} C(\mathbf{w}) = \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \beta \sigma(\mathbf{x}_i^T \mathbf{w})(1 - \sigma(\mathbf{x}_i^T \mathbf{w})) \mathbf{x}_i \quad (28)$$

## Q5

Repeat Question 3 using  $f(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{x}^T \mathbf{w})$ , where  $\sigma(a) = \max(0, a)$

**Answer:**

$$\nabla_{\mathbf{w}} C(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^m (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4 \quad (29)$$

$$= \sum_{i=1}^m \nabla_{\mathbf{w}} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^4 \quad (30)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \nabla_{\mathbf{w}} f(\mathbf{x}_i, \mathbf{w}) \quad \text{chain rule} \quad (31)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \nabla_{\mathbf{w}} \sigma(\mathbf{x}_i^T \mathbf{w}) \quad (32)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \frac{\partial \sigma(a)}{\partial a} \nabla_{\mathbf{w}} (\mathbf{x}_i^T \mathbf{w}) \quad \text{chain rule} \quad (33)$$

$$= \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \frac{\partial \sigma(a)}{\partial a} \mathbf{x}_i \quad (34)$$

So we just have to evaluate  $\frac{\partial \sigma(a)}{\partial a}$  and plug it into the above equation. Note that  $\frac{\partial \sigma(a)}{\partial a}$  is evaluated at  $a = \mathbf{x}_i^T \mathbf{w}$ . When  $a < 0$ ,  $\sigma(a) = 0$ , so  $\frac{\partial \sigma(a)}{\partial a} = 0$ . When  $a > 0$ ,  $\sigma(a) = a$ , so  $\frac{\partial \sigma(a)}{\partial a} = 1$ . Let us define  $\delta(\mathbf{x}_i^T \mathbf{w} > 0) = \begin{cases} 1 & \text{if } \mathbf{x}_i^T \mathbf{w} > 0 \\ 0 & \text{if } \mathbf{x}_i^T \mathbf{w} \leq 0 \end{cases}$ , so we get

$$\frac{\partial \sigma(a)}{\partial a} = \delta(\mathbf{x}_i^T \mathbf{w} > 0) \quad (35)$$

Therefore, we get

$$\nabla_{\mathbf{w}} C(\mathbf{w}) = \sum_{i=1}^m 4(f(\mathbf{x}_i, \mathbf{w}) - y_i)^3 \mathbf{x}_i \delta(\mathbf{x}_i^T \mathbf{w} > 0), \quad (36)$$

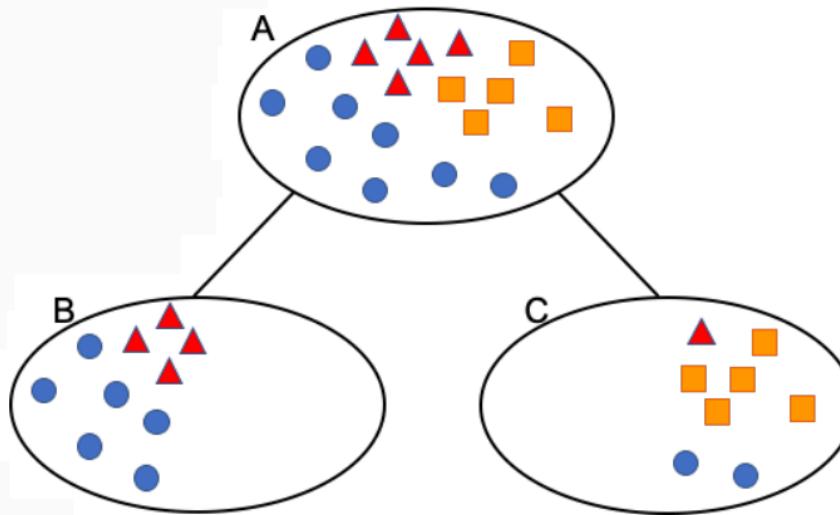
## Tutorial 9

### Q1

(Gini impurity, entropy and misclassification rate)

#### Question 1:

Compute the Gini impurity, entropy, misclassification rate for nodes A, B and C, as well as the overall metrics (Gini impurity, entropy misclassification error) at depth 1 of the decision tree shown below.



RYB total  
A 5 5 8 18  
GINI A =  $1 - (5/18)^2 - (5/18)^2 - (8/18)^2 = 0.64814814814$

Entropy A =  $-(5/18)\log_2(5/18) - (5/18)\log_2(5/18) - \log_2(8/18) = 1.54663161539$

MissClass A =  $1 - 8/18 = 0.555555555556$

B 4 0 6 10  
GINI B =  $1 - (4/10)^2 - (0/10)^2 - (6/10)^2 = 0.48$

Entropy B =  $-(4/10)\log_2(4/10) - (0/10)\log_2(0/10) - \log_2(6/10) = 0.97095059445$

MissClass B =  $1 - 6/10 = 0.4$

C 1 5 2 8  
GINI B =  $1 - (1/8)^2 - (5/8)^2 - (2/8)^2 = 0.53125$

Entropy C =  $-(1/8)\log_2(1/8) - (5/8)\log_2(5/8) - (2/8)\log_2(2/8) = 1.2987949407$

MissClass C =  $1 - 5/8 = 0.375$

- log 0 ignore since multiple by 0

Let's assume class 1, class 2 and class 3 correspond to red triangles, orange squares and blue circles respectively.

- For node A,  $p_1 = \frac{5}{18}$ ,  $p_2 = \frac{5}{18}$ ,  $p_3 = \frac{8}{18} = \frac{4}{9}$
- For node B,  $p_1 = \frac{4}{10} = \frac{2}{5}$ ,  $p_2 = \frac{0}{10} = 0$ ,  $p_3 = \frac{6}{10} = \frac{3}{5}$
- For node C,  $p_1 = \frac{1}{8}$ ,  $p_2 = \frac{5}{8}$ ,  $p_3 = \frac{2}{8} = \frac{1}{4}$

For **Gini impurity**, recall formula is  $1 - \sum_{i=1}^K p_i^2$

- Node A:  $1 - \left(\frac{5}{18}\right)^2 - \left(\frac{5}{18}\right)^2 - \left(\frac{4}{9}\right)^2 = 0.6481$
- Node B:  $1 - \left(\frac{2}{5}\right)^2 - (0)^2 - \left(\frac{3}{5}\right)^2 = 0.48$
- Node C:  $1 - \left(\frac{1}{8}\right)^2 - \left(\frac{5}{8}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.5312$
- Overall Gini at depth 1:  $\left(\frac{10}{18}\right) 0.48 + \left(\frac{8}{18}\right) 0.5312 = 0.5028$

Observe the decrease in Gini impurity from root (0.6481) to depth 1 (0.5028)

For **entropy**, recall formula is  $-\sum_i p_i \log_2 p_i$

- Node A:  $-\left(\frac{5}{18}\right) \log_2 \left(\frac{5}{18}\right) - \left(\frac{5}{18}\right) \log_2 \left(\frac{5}{18}\right) - \left(\frac{4}{9}\right) \log_2 \left(\frac{4}{9}\right) = 1.5466$
- Node B:  $-\left(\frac{2}{5}\right) \log_2 \left(\frac{2}{5}\right) - (0) \log_2 (0) - \left(\frac{3}{5}\right) \log_2 \left(\frac{3}{5}\right) = 0.9710$
- Node C:  $-\left(\frac{1}{8}\right) \log_2 \left(\frac{1}{8}\right) - \left(\frac{5}{8}\right) \log_2 \left(\frac{5}{8}\right) - \left(\frac{1}{4}\right) \log_2 \left(\frac{1}{4}\right) = 1.2988$
- Overall entropy at depth 1:  $\left(\frac{10}{18}\right) 0.9710 + \left(\frac{8}{18}\right) 1.2988 = 1.1167$

Observe the decrease in entropy from root (1.5466) to depth 1 (1.1167)

For **misclassification rate**, recall formula is  $1 - \max_i p_i$

- Node A:  $1 - \max\left(\frac{5}{18}, \frac{5}{18}, \frac{4}{9}\right) = 1 - \left(\frac{4}{9}\right) = \frac{5}{9} = 0.5556$
- Node B:  $1 - \max\left(\frac{2}{5}, 0, \frac{3}{5}\right) = 1 - \left(\frac{3}{5}\right) = \frac{2}{5}$
- Node C:  $1 - \max\left(\frac{1}{8}, \frac{5}{8}, \frac{1}{4}\right) = 1 - \left(\frac{5}{8}\right) = \frac{3}{8}$
- Overall misclassification error rate at depth 1:  $\left(\frac{10}{18}\right) \left(\frac{2}{5}\right) + \left(\frac{8}{18}\right) \left(\frac{3}{8}\right) = 0.3889$
- We can also double check that at depth 1, the 4 red triangles will be classified wrongly for node B and the 1 red triangle + 2 blue circles will be classified wrongly for node C. So in total, there will be 7 wrong classifications out of 18 datapoints, which corresponds to  $\left(\frac{7}{18}\right) = 0.3889$
- Observe the decrease in misclassification rate from root (0.5556) to depth 1 (0.3889)

## Q2

### Question 2:

Calculate the overall MSE for the following data at depth 1 of a regression tree assuming a decision threshold is taken at  $x = 5.0$ . How does it compare with the MSE at the root?

$\{x, y\}: \{1, 2\}, \{0.8, 3\}, \{2, 2.5\}, \{2.5, 1\}, \{3, 2.3\}, \{4, 2.8\}, \{4.2, 1.5\}, \{6, 2.6\}, \{6.3, 3.5\}, \{7, 4\}, \{8, 3.5\}, \{8.2, 5\}, \{9, 4.5\}$

### Answer:

At depth 1, when  $x > 5$

- $y = \{2.6, 3.5, 4, 3.5, 5, 4.5\} \Rightarrow \bar{y} = 3.85$
- $MSE = \frac{1}{6}((2.6 - \bar{y})^2 + (3.5 - \bar{y})^2 + (4 - \bar{y})^2 + (3.5 - \bar{y})^2 + (5 - \bar{y})^2 + (4.5 - \bar{y})^2) = 0.5958$

At depth 1, when  $x \leq 5$

- $y = \{2, 3, 2.5, 1, 2.3, 2.8, 1.5\} \Rightarrow \bar{y} = 2.1571$
- $MSE = \frac{1}{7}((2 - \bar{y})^2 + (3 - \bar{y})^2 + (2.5 - \bar{y})^2 + (1 - \bar{y})^2 + (2.3 - \bar{y})^2 + (2.8 - \bar{y})^2 + (1.5 - \bar{y})^2) = 0.4367$

Overall MSE at depth 1:  $\frac{6}{13} \times 0.5958 + \frac{7}{13} \times 0.4367 = 0.5102$

At the root:

- $y = \{2, 3, 2.5, 1, 2.3, 2.8, 1.5, 2.6, 3.5, 4, 3.5, 5, 4.5\} \Rightarrow \bar{y} = 2.9385$
- $MSE = \frac{1}{13}((2.6 - \bar{y})^2 + (3.5 - \bar{y})^2 + (4 - \bar{y})^2 + (3.5 - \bar{y})^2 + (5 - \bar{y})^2 + (4.5 - \bar{y})^2 + (2 - \bar{y})^2 + (3 - \bar{y})^2 + (2.5 - \bar{y})^2 + (1 - \bar{y})^2 + (2.3 - \bar{y})^2 + (2.8 - \bar{y})^2 + (1.5 - \bar{y})^2) = 1.2224$

Therefore, MSE has decreased from 1.2224 at the root to 0.5102 at depth 1

### Q3

(Regression tree, Python)

#### Question 3:

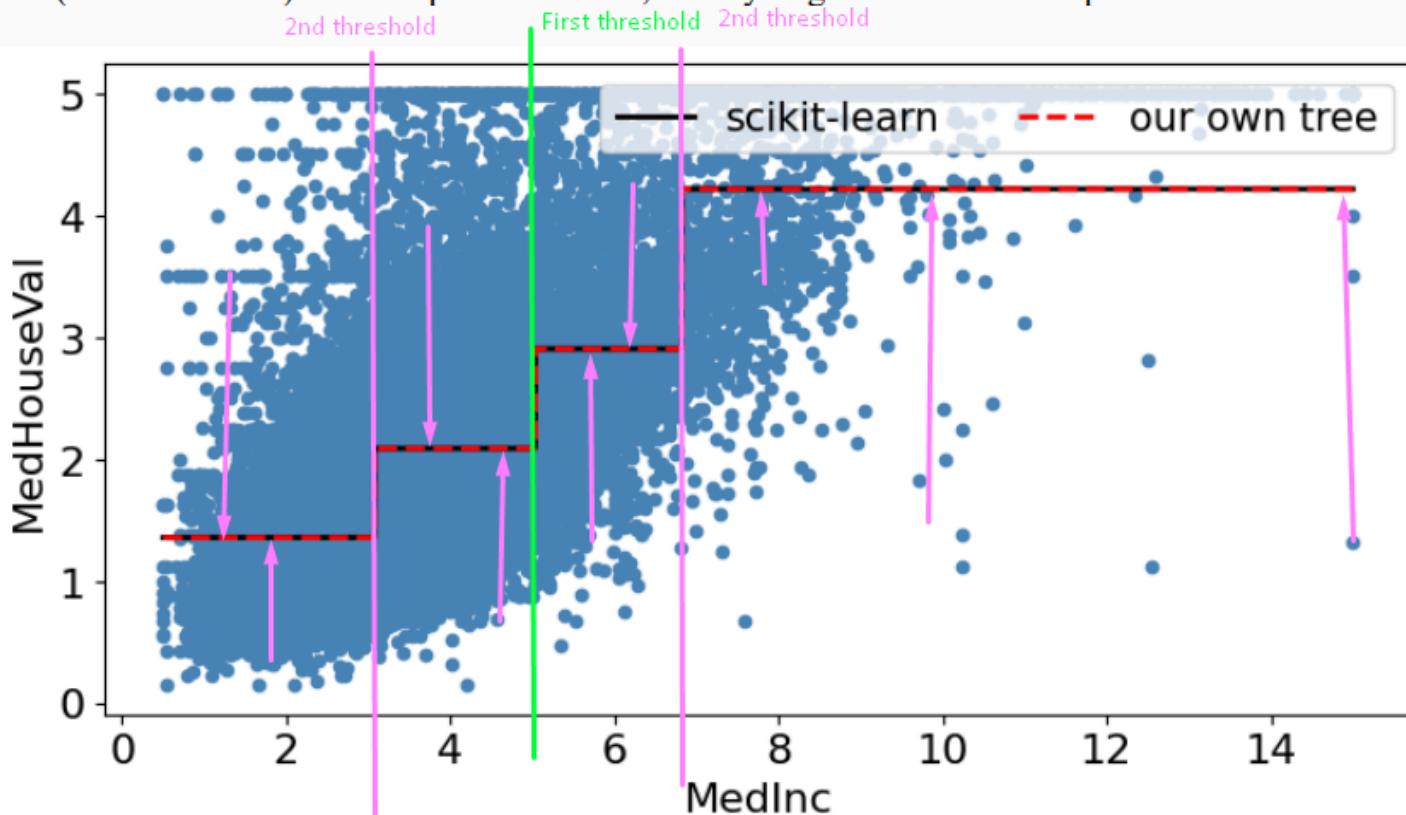
Import the California Housing dataset “`from sklearn.datasets import fetch_california_housing`” and “`housing = fetch_california_housing()`”. This data set contains 8 features and 1 target variable listed below. Use “`MedInc`” as the input feature and “`MedHouseVal`” as the target output. Fit a regression tree to depth 2 and compare your results with results generated by “`from sklearn.tree import DecisionTreeRegressor`” using the “squared error” criterion.

Target: `['MedHouseVal']`

Features: `['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']`

#### Answer:

Please refer to `Tut9_Q3_zhou.py`. We can exactly replicate the results from scikit-learn. Note that in the plot below, the blue dots are the training datapoints. The curves from scikit-learn (black line) and our own tree (red dashed line) are on top of each other, so they might be hard to tell apart.



- The first threshold is gonna be the middle one
- if something is below the left 2nd threshold, its gonna predict every point in that to 1.3

```

# Compute total square error
# Note that MSE = total square error divided by number of data points
below_sq_err = np.sum(np.square(data_below_threshold - mean_below_threshold))
above_sq_err = np.sum(np.square(data_above_threshold - mean_above_threshold))
sq_err_vec[index] = below_sq_err + above_sq_err
meansq_err_vec[index] = sq_err_vec[index]/len(y)

```

- can add without taking the mean and weighted average since it cancels out, with divide the total number of sample,
- then comparing MSE / total is still valid since its scaled
- but the actual MSE need to divide

## **Q4**

(Classification tree, Python)

### **Question 4:**

Get the data set “from sklearn.datasets import load\_iris”. Perform the following tasks.

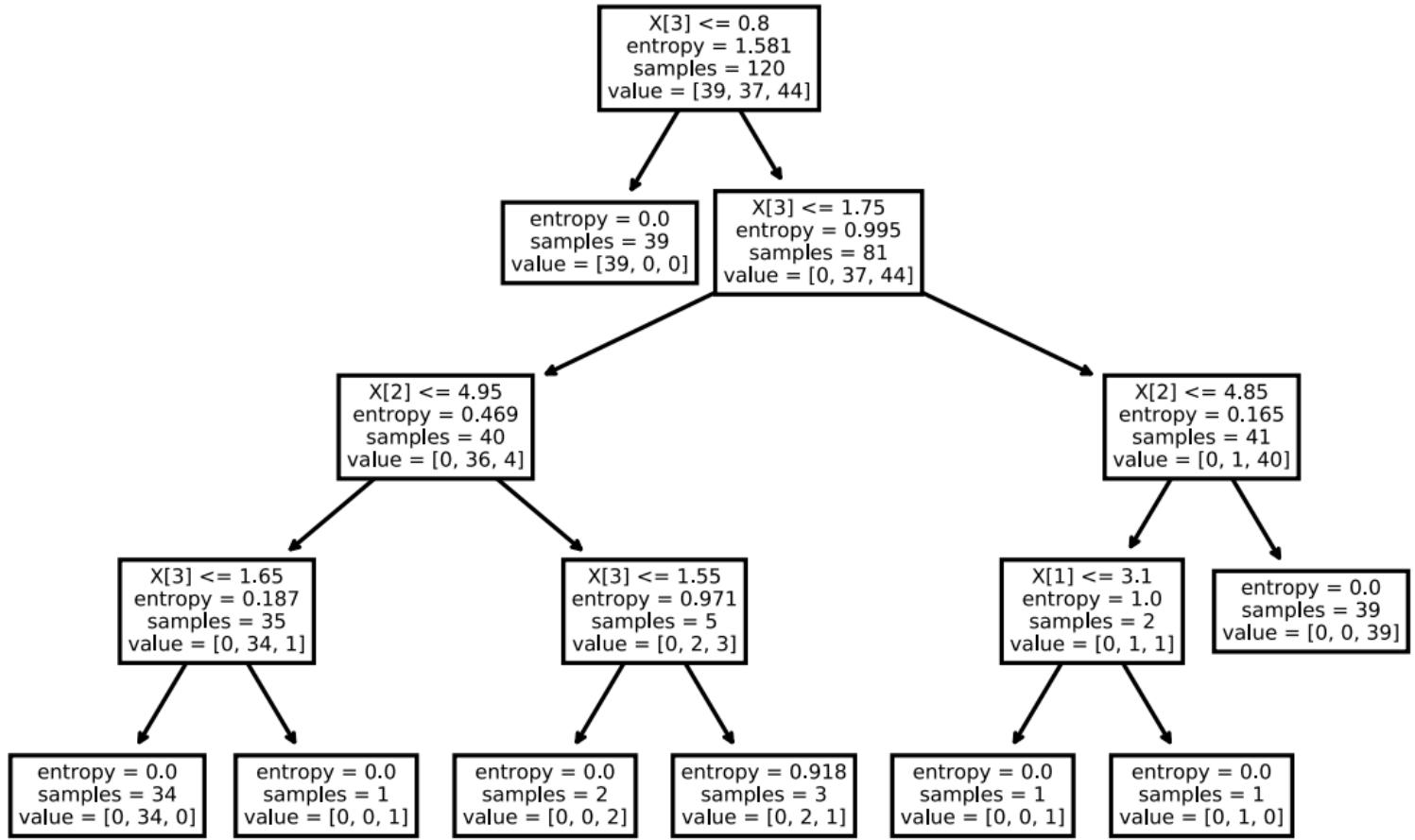
- Split the database into two sets: 80% of samples for training, and 20% of samples for testing using `random_state=0`
- Train a decision tree classifier (i.e., “`tree.DecisionTreeClassifier`” from `sklearn`) using the training set with a maximum depth of 4 based on the “entropy” criterion.
- Compute the training and test accuracies. You can use `accuracy_score` from `sklearn.metrics` for accuracy computation
- Plot the tree using “`tree.plot_tree`”.

Please refer to Tut9\_Q4\_yeo.py.

Training accuracy: 0.9917

Test accuracy: 1.0

The resulting tree looks like this:



## Tutorial 10

### Question 1:

We have two classifiers showing the same accuracy with the same cross-validation. The more complex model (such as a 9<sup>th</sup>-order polynomial model) is preferred over the simpler one (such as a 2<sup>nd</sup>-order polynomial model).

- a) True
- b) False

B, complex lead to overfitting

### Question 2:

We have 3 parameter candidates for a classification model, and we would like to choose the optimal one for deployment. As such, we run 5-fold cross-validation.

Once we have completed the 5-fold cross-validation, in total, we have trained \_\_\_\_\_ classifiers. Note that, we treat models with different parameters as different classifiers.

- A) 10
- B) 20
- C) 25
- D) 15

15, just multiply

### Question 3:

Suppose the binary classification problem, which you are dealing with, has highly imbalanced classes. The majority class has 99 hundred samples and the minority class has 1 hundred samples. Which of the following metric(s) would you choose for assessing the classification performance? (Select all relevant metric(s) to get full credit)

- a) Classification Accuracy
- b) Cost sensitive accuracy
- c) Precision and recall
- d) None of these

b, c

### Question 4:

Given below is a scenario for Training error rate Tr, and Validation error rate Va for a machine learning algorithm. You want to choose a hyperparameter (P) based on Tr and Va.

P	Tr	Va
10	0.10	0.25
9	0.30	0.35
8	0.22	0.15
7	0.15	0.25
6	0.18	0.15

Which value of P will you choose based on the above table?

- a) 10
- b) 9
- c) 8
- d) 7
- e) 6

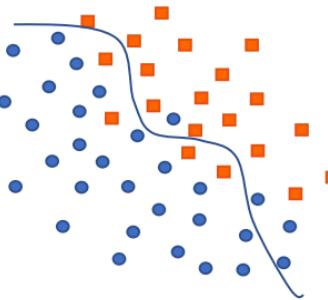
6, both lowest, but validation accuracy comes first, then training accuracy

(Binary and Multicategory Confusion Matrices)

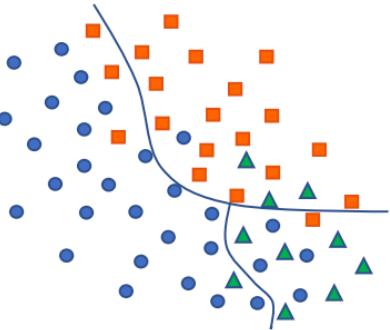
**Question 5:**

Tabulate the confusion matrices for the following classification problems.

(a) Binary problem (the class-1 and class-2 data points are respectively indicated by squares and circles)



(b) Three-category problem (the class-1, class-2 and class-3 data points are respectively indicated by squares, circles and triangles).



**Answer:**

(a)

	$P_{\hat{1}}$	$P_{\hat{2}}$
$P_1$	16	4
$P_2$	4	26

(b)

	$P_{\hat{1}}$	$P_{\hat{2}}$	$P_{\hat{3}}$
$P_1$	16	3	1
$P_2$	1	25	4
$P_3$	3	1	6

**(5-fold Cross-validation)**

**Question 6:**

Get the data set “from sklearn.datasets import load\_iris”. Perform a 5-fold Cross-validation to observe the best polynomial order (among orders 1 to 10 and without regularization) for validation prediction. Note that, you will have to partition the whole dataset for training/validation/test parts, where the size of validation set is the same as that of test. Provide a plot of the average 5-fold training and validation error rates over the polynomial orders. The randomly partitioned data sets of the 5-fold shall be maintained for reuse in evaluation of future algorithms.

**Tutorial 11**

sigmoid with no given beta, assume beta = 1

### Q3

#### Question 3 *improve*

How can you prevent a clustering algorithm from getting stuck in bad local optima?

k-means → local optima  
global optima ✗

- a) Set the same seed value for each run ✗ ↑ productivity only
- b) Use the bottom ranked samples for initialization ✗
- c) Use the top ranked samples for initialization ✗ ↑ bias
- d) All of the above
- e) None of the above

Ans: e).

① multiple runs with random initialisation  
② diverse initialisation

#### Question 4

### Q4

#### Question 4

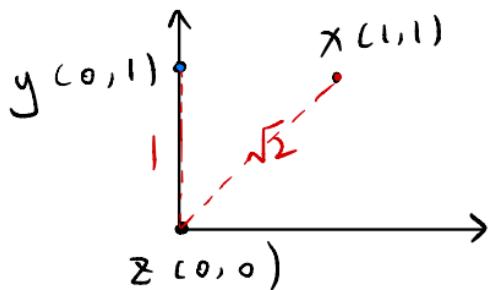
② diverse initialisation

Consider the following data points:  $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ ,  $y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ , and  $z = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ . The k-means algorithm is initialized with centers at  $x$  and  $y$ . Upon convergence, the two centres will be at

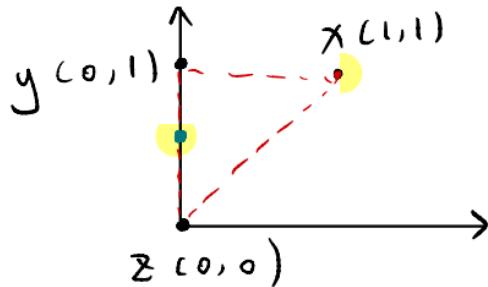
$$\textcircled{1} : x$$

- a)  $x$  and  $z$
- b)  $x$  and  $y$

$$\textcircled{2} : y \& z$$



1



$$\textcircled{1} : x$$

$$\textcircled{2} : y \& z$$

- c)  $y$  and the midpoint of  $y$  and  $z$
- d)  $z$  and the midpoint of  $x$  and  $y$
- e) None of the above

Ans: e). The converged centers should be  $x$  and the midpoint of  $y$  and  $z$ .

$x$  and midpoint of  $y$  and  $z$

Three balls are drawn from three urns sequentially, one ball from each urn. The first urn contains 1 blue and 7 red balls, the second urn contains 2 blue and 6 red balls, and the third urn contains 3 red and 5 green balls. Find the probability that 2 red balls are chosen. (3 marks)

Currently Selected: B

A 270/1024

B 270/512

C 226/64

D 226/512

E None of the rest.

This question is related to the understanding of linear systems and partial derivatives. Which of the following statements below is correct?

Currently Selected: A

A  $\begin{bmatrix} 1 & 4 \\ 2 & 7 \\ -3 & 11 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.5 \\ 4 \end{bmatrix}$   
The system has no exact solution but an approximated solution is available using the left inverse.

B If  $f(\mathbf{x})$  is a vector-valued function of size  $p \times 1$  and  $\mathbf{x}$  is an  $m \times 1$  vector, then differentiation of  $f(\mathbf{x})$  with respect to  $\mathbf{x}$  is an  $m \times p$  matrix.

C A linear function needs to satisfy the properties of homogeneity only.

D In over-determined linear systems, the number of parameters is greater than the number of unknown equations.

E None of the other options.

Which of the following task is likely to be achieved via supervised learning?

Currently Selected: A

A Using historical data for weather forecast.

B Grouping together users with similar viewing patterns in order to recommend similar content.

C Grouping a number of oranges by their size.

D None of the rest.

Causality is a deterministic relationship; suppose we know A and B have causal relation, if A occurs, B is for sure to take place.

Currently Selected: A

A True

B False

- False
- [Causality is a Statistical Relationship](#)

A set of linear equations is written as  $\mathbf{w}^T \mathbf{X} = \mathbf{y}^T$  where  $\mathbf{X} \in \mathcal{R}^{3 \times 2}$  and  $\mathbf{y} \in \mathcal{R}^{2 \times 1}$ . How many simultaneous equations are there in this set of equations?

Currently Selected: B

A 1

B 2

C 3

D 5

E 4

A discrete random variable takes a finite number of values, while a continuous random variable can only take infinite number of values. (2 marks)

Currently Selected: A

A True

B False

- False: Discrete can be infinite also

This question is related to the understanding of modelling assumptions.  $f(x) = 5x - 3$  is a linear function. (2 marks)

Currently Selected: B

A True

B False

A machine learning algorithm takes the temperature as one of its input features. The temperature is measured in Celsius. Please select the correct option.

Currently Selected: E

A The temperature in Celsius is considered as interval data.

B We can calculate the mean and standard deviation of temperature.

C The temperature in Celsius is considered as ratio data.

D None of the rest.

E (a), and (b)

F (a), and (c)

A machine learning algorithm takes the letter grade of students as one of its input features. The letter grade can take any element from {A+, A, A-, B+, B, B-, C+, C, D+, D, F}, subject to some distribution curving. For example, A+ corresponds to 95%, A corresponds to 85%, and A- corresponds to 80%. Which of the following statements is/are true?

Currently Selected: F

A The letter grade is an example of nominal variable.

B The letter grade is an example of ordinal variable.

C The letter grade is an example of interval variable.

D The letter grade is an example of discrete variable.

E (a), and (c)

F (b), and (d)

G (b), and (c)

A person draws 2 cards from a deck of 52 cards, one after another without replacing the previous card back. What is the probability of drawing two Queens in a row?

Currently Selected: B

A 4/52

B 1/221

C 3/51

D 2/52

One key step in Data Cleaning is to check the missing features of data samples. When we have insufficient number of training samples in our dataset, we may consider removing the examples with missing features.

Currently Selected: B

A True

B False

Suppose the random variable X has a probability mass function (pmf) given in the table below.

X	1	2	3	4	5
Pr[X]	0.1	(BLANK1)	0.2	0.4	(BLANK2)

We also know that the expected value of X is 3.5.

1) What is the probability of  $\text{Pr}[X=5]?$  ① 0.2 (2 Marks)

2) What is the probability of  $\text{Pr}[X \leq 2]?$  ② 0.2 (2 Marks)

Answers 1 - 2

1. 0.2

2. 0.2

1. Range - Min:0.19999 Max:0.20001
2. Range - Min:0.19999 Max:0.20001

We have a collection of 1,000 images from three classes: cat, bird, and dog. With these images, we would like to train an image classifier that categorizes an input image into one of the three classes.

To ensure the 1,000 images are of good quality for training the classifier, we ask a well-trained human inspector to go through all the images, to label the images and remove noisy ones. Eventually, we removed 200 images of low quality suggested by the inspector, and use the remaining 800 images to train the classifier; the 800 images comprise 200 cat images, 300 bird images, and 300 dog images.

Please select the correct option.

Currently Selected: A

- A The human inspection process can be considered as a data cleaning step.

If we are to use one-hot encoding for the labels of the three classes, we can set:

- B Cat = [1 1 1]  
Dog = [0 1 0]  
Bird = [0 0 1]

- C The image classification conducted here is an unsupervised-learning task.

- D If we keep the 200 noisy images (suggested by the human inspector), we will end up having more training images and hence a better-performed image classifier.

- E (a) and (b)

- F (a), (b), and (d)

- G None of others is correct.

You are given a collection of 5 training data points of two features  $(x_1, x_2)$  and their target output ( $y$ ) which are packed as follows:

$$\text{Feature matrix: } X = \begin{bmatrix} 1 & 2 \\ 0 & 6 \\ 1 & 0 \\ 0 & 5 \\ 1 & 7 \end{bmatrix}, \text{ Target output: } y = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

Predict the output (up to 4 decimal places) of  $(x_1, x_2) = (1, 3)$  using the linear regression model. (4 marks)

1) What is the mean of squared error of the estimated model? 1. 1.3887 (up to 4 decimal places, 2 marks)

2) The prediction for  $y$  is 2. 2.6603 (up to 4 decimal places, 2 marks).

Answers 1 - 2

1. 1.3887

2. 2.6603

1. Range - Min:1.3886 Max:1.3888
2. Range - Min:2.9999 Max:3.0001

- type in wrongly lmao, get 3

```

X = np.array([[1, 2], [0, 6], [1, 0], [0, 5], [1, 7]])
Y = np.array([[1], [2], [3], [4], [5]])

w = helper.linearRegressionWithBias(X, Y, printResult=True, printFeature=None)

xtest = np.array([[1, 3]])
xtest = helper.paddingOfOnes(xtest)
ytest = helper.testData(xtest, w, True)

```

```

rank(X) = 3
W =
[[1.13207547]
[0.8490566 ]
[0.33962264]]
MSE = 1.388679245283019
X @ W = Y =
[[3.]]

```

The values of feature x and their corresponding values of target y are shown in the table below.

x	3	4	5	6	7
y	5	4	3	2	1

Find the least square regression line  $y = a + bx$  and then estimate the value of y when  $x = 8$ .

Currently Selected: A

A   $y = 0$

B   $y = -1$

C   $y = +1$

D   $y = 8$

E  None of the above

```

X = np.array([[3], [4], [5], [6], [7]])
Y = np.array([[5], [4], [3], [2], [1]])

w = helper.linearRegressionWithBias(X, Y, printResult=True, printFeature=None)

xtest = np.array([[8]])
xtest = helper.paddingOfOnes(xtest)
ytest = helper.testData(xtest, w, True)

```

```

rank(X) = 2
W =
[[ 8.]
[-1.]]
MSE = 4.733165431326071e-30
X @ W = Y =
[[4.4408921e-15]]

```

## final PY

7. **TF7.1** Suppose the model parameters are given by  $w = [w_1, \dots, w_d]^T$ . Then an inclusion of the L2-regularization during training reduces the value of  $\sum_{i=1}^d w_i^6$ .
- a) True
  - b) False

**Ans:** False

- technically true but bad wording and L2 is to minimise  $w^2$

11. **TF10.2** A higher training accuracy not necessarily leads to a higher test accuracy; however, in most cases, a higher training accuracy indeed leads to a higher validation accuracy.

- a) True
- b) False

**Ans:** False. There is no connection between higher training accuracy and higher validation accuracy.

- no connection because validation to training is as if test to training, it won't see the validation set until fully trained, doesn't mean will perform well for validation if test is good

8. **MCQ8.1** Suppose we are minimizing a cost function  $C(w)$  with respect to  $w$  using a gradient descent algorithm. We observe the following: At iteration 1,  $C(w)$  is 11. At iteration 2,  $C(w)$  becomes 10.9. At iteration 3,  $C(w)$  becomes 10.7. At iteration 4,  $C(w)$  becomes 10.6. At iteration 5,  $C(w)$  becomes 10.55. Which of the following is true? Note that there might be more than one true option. If so, you should select all the correct options in order to get all the marks.

- (1) Decreasing the learning rate will speed up the optimization further
- (2) Increasing the learning rate will speed up the optimization further
- (3) Adding a regularization to the cost function will speed up the optimization further
- (4) Insufficient information in the question to tell which option is true

**Ans:** (4) Even though the cost function is decreasing slowly, it's entirely possible that we were lucky and initialized near the local minimum, so hard to say without further experimentation.

9. **MCQ8.2** Consider the cost function  $C(w) = \text{DataLoss}(w) + \lambda \text{Regularization}(w)$ .

Assume that the global minimum of  $C(w)$  when  $\lambda = 2$  is 12. Now we change  $\lambda$  to be equal to 20 and again minimize  $C(w)$ . Assuming we attain the global minimum, the new optimal cost function value  $C$

- (1) will be higher than before
- (2) will be lower than before
- (3) will stay the same
- (4) Insufficient information in the question to tell which option is true.

**Ans:** (4) Insufficient information in the question to tell which option is true.

10. **MCQ8.3** Consider the cost function  $C(w) = \text{DataLoss}(w) + \lambda \text{Regularization}(w)$ .

When  $\lambda = 3$ , the minimum of  $C(w)$  is achieved at  $w = w^*$  and  $C(w^*) = 25$ . Now we change  $\lambda$  to be equal to 30 and minimize  $C(w)$ . The minimum now is achieved at  $w = w^{**}$ . Assume that we always achieve the global minimum. Which of the following is true? Note that there might be more than one true option. If so, you should select all the correct options.

- (1) In general,  $\text{Regularization}(w^*) > \text{Regularization}(w^{**})$
- (2) In general,  $\text{Regularization}(w^*) < \text{Regularization}(w^{**})$
- (3) In general,  $\text{DataLoss}(w^*) > \text{DataLoss}(w^{**})$
- (4) In general,  $\text{DataLoss}(w^*) < \text{DataLoss}(w^{**})$
- (5) Insufficient information in the question to tell which option is true.

**Ans:** By increasing regularization hyperparameter, we are putting more weight on the regularization term, so (1)  $\text{Regularization}(w^*) > \text{Regularization}(w^{**})$  and less weight on the data-loss term, so (4)  $\text{DataLoss}(w^*) < \text{DataLoss}(w^{**})$ . Note that one special case is that the Regularization and DataLoss have the same global minimum, which is why I added "in general" qualifier in the statements.

**11. MCQ9.1** Suppose in a 2-class classification problem, our decision tree algorithm achieves 52% accuracy on our training set, and also 52% accuracy on our test set. Which of the following modification might potentially improve our algorithm's test accuracy? Note that there might be more than one true option. If so, you should select all the correct options in order to get all the marks.

- (1) Increase maximum depth of tree
- (2) Decrease maximum depth of tree
- (3) Increase minimum number of samples for splitting a leaf node
- (4) Decrease minimum number of samples for splitting a leaf node
- (5) Try random forest instead of decision tree

**Ans:** Underfitting regime, so (1) Increase maximum tree depth and (4) Decrease minimum number of samples for splitting a leaf node. Although we are in the underfitting regime, but since we are increasing model complexity, we can use try random forest to decrease variance, so (5) is true as well.

- increase model complexity and reduce variance by using random forest

**17.** Which of the following statement(s) is/are true about deep neural networks? Note that there might be more than one true option. If so, you should select all the correct options in order to get all the marks.

- (a) If we build a multilayer perceptron network, we do not need non-linear activation functions.
- (b) Backpropagation involves a backward stage but not a forward stage.
- (c) Parameters of the neural networks are chosen manually in a smart way, in order to produce the outputs as similar as the labels.
- (d) **None of the others.**

Back propagation is both forward and backward

4. **FIB8.2** We would like to minimize the cost function  $C(x, y) = x^2 + xy^2$  using gradient descent. Suppose we initialize our algorithm with  $x = 3$ ,  $y = 2$ . Suppose the step size is 0.2. After the first round of gradient descent, the updated value of  $x$  is BLANK1 (your answer should be up to 2 decimal places) and the updated value of  $y$  is BLANK2 (your answer should be up to 2 decimal places).

**[2 Blanks]**

Gradient wrt  $x$  is given by  $2x + y^2$ .

At  $x = 3$ ,  $y = 2$ , gradient =  $2(3) + 2^2 = 10$ .

After first round of gradient descent  $x = 3 - 0.2 \cdot 10 = 1.00$

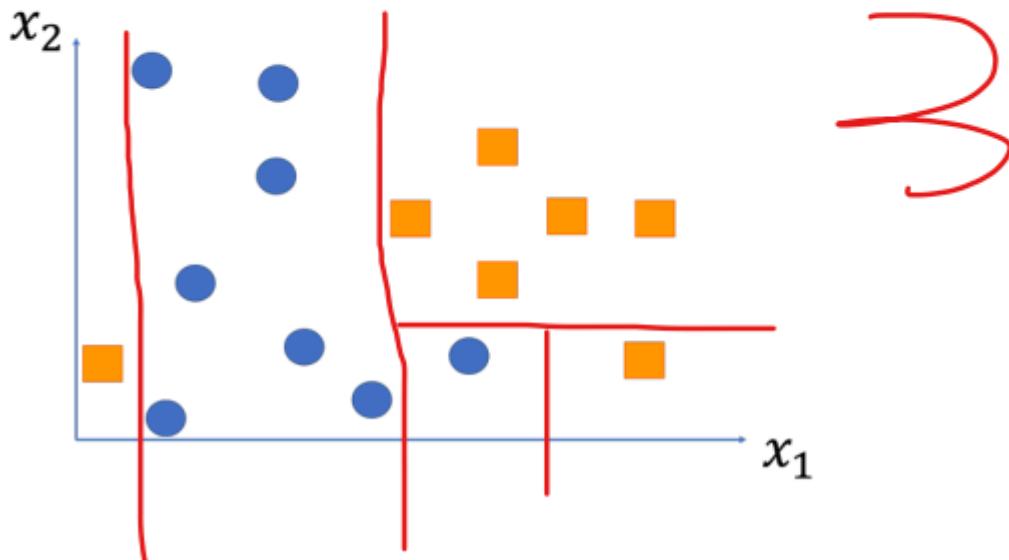
Gradient wrt  $y$  is given by  $2xy$ .

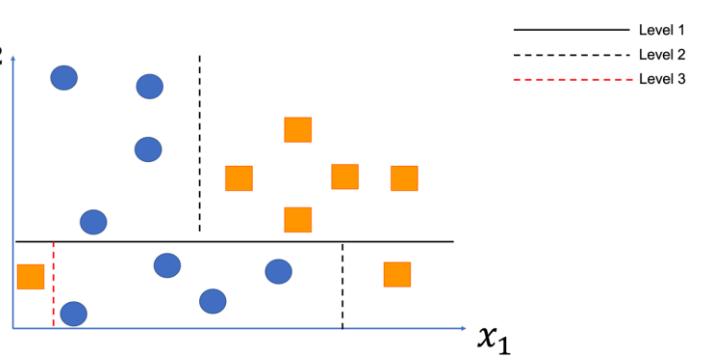
At  $x = 3$ ,  $y = 2$ , gradient =  $2(3)(2) = 12$

After first round of gradient descent,  $y = 2 - 0.2 \cdot 12 = -0.40$

5. To classify the data points in the plot below with 100% accuracy, a decision tree needs to have a minimum depth of BLANK (your answer should be a positive integer)

**[1 Blank]**





## 7. FIB 10.1

In a binary classification task, we use a dataset that contains in total 1000 samples, where we take out 200 samples as the test set. We then conduct a standard 4-fold cross validation. In each fold, we set the number of validation samples to be the same as that of test samples. As such, in each fold, the binary classifier is trained on BLANK1 samples.

Now we have three classifier candidates, where their average training, validation, and number of parameters are given as follows. We should adopt classifier #BLANK2 for the test set.

Classifiers	Average Training Accuracy	Average Validation Accuracy	Number of parameters
Classifier #1	90.5%	92.1%	5000
Classifier #2	91.5%	92.1%	3000
Classifier #3	93.5%	90.5%	3000

### [2 Blanks]

Ans:

BLANK1: 600

BLANK2: 2

For BLANK1, Number of training samples =  $1000 * (1 - 2 * 20\%) = 600$

For BLANK1, Classifier #2 has the highest validation accuracy and fewer number of parameters, and a higher training accuracy as compared to Classifier #1.

- number of parameters relate to the model complexity, if all else equal, choose the lower complexity one

8.

Jack developed a classifier to detect spam emails. The classifier conducts binary classification, and identifies an email to be spam (positive class) or not spam (negative class). The detected spam emails go into the 'Spam' folder and non-spam emails go into the 'Inbox' folder.

In an unseen dataset of 100 emails, the classifier achieves the following performances

	Spam (Predicted)	Not Spam (Predicted)
Spam (Actual)	40	x
Not Spam (Actual)	15	35

The number x is \_BLANK1\_. 10

We can also derive that, among the 100 emails, \_BLANK2\_ emails have the actual label of being Spam, and in total \_BLANK3\_ emails are correctly classified.

55, 75      actual label of spam = 50

Assume we have the following cost matrix

	Spam (Predicted)	Not Spam (Predicted)
Spam (Actual)	$C_{s,s} * 40$	$C_{s,n} * x$
Not Spam (Actual)	$C_{n,s} * 15$	$C_{n,n} * 35$

We assume that, if an email is correctly classified, we impose no penalty, i.e.,  $C_{s,s} = C_{n,n} = 0$ . On the other hand, we know that customers will be very unhappy if the spam folder contains non-spam emails. But customers find it acceptable that some spam emails go into the inbox folder. In this case, when comparing two classifiers, we should ensure that

$C_{s,n} \text{ } _BLANK4\text{ } C_{n,s}$  (choose '=', '>' or '<' and fill in the blank).

<

With the same dataset, David also developed a classifier that achieves the following performance

	Spam (Predicted)	Not Spam (Predicted)
Spam (Actual)	50	y
Not Spam (Actual)	10	25

With the same set of  $\{C_{n,n}, C_{s,s}, C_{n,s}, C_{s,n}\}$  as above, David's classifier is considered to be \_BLANK5\_ (1:Better or 2:Worse; choose '1' or '2' and fill in the blank) than the one of Jack's.

1

**Blank1**

- use the cost to find that second is better since not spam being predicted as spam has higher cost and is of lower number

9.

We have a collect of 10 books. We measure their thickness in millimetres and summarize them as follows

Book ID	01	02	03	04	05	06	07	08	09	10
Thickness (mm)	50	60	66	68	71	72	75	82	90	99

71.25

We'd like to group the books into two groups according to their thickness.

- 1) Assume we pick book 03 as the initial centroid for Group A, book 07 for Group B, and assign the books to the two groups using Euclidean distance. Before updating the new centroid, we will have BLANK1 books in Group A (please enter an integer here)

4

EE2211 - Introduction to Machine Learning/ Page 14

- 2) With the above group assignments, we re-estimate the new centroids for the two groups. The new centroid of Group A is BLANK2, and the new centroid of Group B is BLANK3. ( please specify your answer with two decimal places)

61.00, 81.50

- 3) With updated centroid, we run group assignment again. In this second round, the number of books in the Group A (the group with lower centroid) is BLANK4.

5

- for 1D kmeans, the centroid parting line is the average of the two centroids

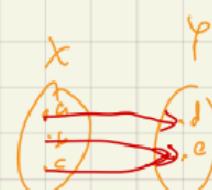
## Functions

Functions : (roughly speaking) data of  $(X, Y, x \mapsto f(x))$   
 where  $X, Y$  are sets and  $x \mapsto f(x)$  assigns  $x \in X$  to some  $f(x) \in Y$ .

$X$ : domain  
 $Y$ : target / codomain

Notation  $f: X \rightarrow Y$

image definition:  
 e.g.  $f: \mathbb{R} \rightarrow \mathbb{R}$  } e.g.  $f(x) = x + 1$



- function is defined with three stuff
  - input space  $X$
  - output space  $Y$
  - and the transformation

Right-Inverse:  $L F =$

Left-Inverse:  $F L =$

### DEFINITIONS

Let  $f: X \rightarrow Y$  be a function.

- $f$  is injective (one-one) if

for all  $x_1, x_2 \in X$ ,  $f$  has the property that

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

implies

- $f$  is surjective (onto) if

for all  $y \in Y$ , there

exist some  $x \in X$  s.t.

$$f(x) = y$$

- Injective means the function

- $x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$ , if input different, output must be different
- $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$ , if output same, input must be same
- Contrapositive example:

- if rain  $\Rightarrow$  cloud
- not contrapositive, converse: no rain  $\Rightarrow$  no cloud
  - this is not the same statement, no rain could be cloud or no cloud
  - if apply contrapositive to (no rain  $\Rightarrow$  no cloud): cloud  $\Rightarrow$  rain which is confirm not the same statement
- contrapositive: no cloud  $\Rightarrow$  no rain
  - this is the same statement, if rain must have cloud, if no cloud must be no rain
- contrapositive is done by negation and flipping  $\Rightarrow$
- for linear function:
  - injective  $\Leftrightarrow (f(x) = 0 \Rightarrow x = 0)$ 
    - showing  $\Leftarrow$ 
      - assume  $(f(x) = 0 \Rightarrow x = 0)$  is true
      - want to show:  $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$ 
        - assume  $f(x_1) = f(x_2)$
        - $f(x_1) - f(x_2) = 0$
        - by linearity of  $f$ ,  $f(x_1 - x_2) = 0$
        - by hypothesis,  $x_1 - x_2 = 0$
        - $x_1 = x_2$
      - have shown  $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$
    - showing  $\Rightarrow$ 
      - assume injective:  $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$
      - want to show:  $f(x) = 0 \Rightarrow x = 0$ 
        - assume  $f(x) = 0$
        -
      - have shown  $f(x) = 0 \Rightarrow x = 0$
  - Example:  $f(x) = mx$  for  $m \neq 0$ 
    - $f(x_1) = f(x_2)$
    - $mx_1 = mx_2$
    - $x_1 = x_2$
    - $\Rightarrow$  injective
- Surjective means the function can output all  $y$  in  $Y$  as define in the set
  - surjectivity of a function depends on the  $Y$  that is defined
- Example:  $f(x) = x^2$ ,  $X = \mathbb{R}$ ,  $Y = \mathbb{R}$ 
  - $f$  is not injective as  $x^2$  maps many to one
  - $f$  is not surjective as  $f$  cannot map to negative numbers
  - $f$  does not have inverse, nor right nor left inverse
- Example:  $f(x) = x^2$ ,  $X = \mathbb{R}$ ,  $Y = \text{non negative } \mathbb{R}$ 
  - $f$  is not injective as  $x^2$  maps many to one
  - $f$  is surjective as  $f$  can map to all values in  $Y$
  - $f$  has a right inverse (means apply right inverse to input first)
    - $\sqrt()$  then square
  -

## A2 Dump

<https://stackoverflow.com/questions/50132322/how-does-multiple-target-ridge-regression-work-in-scikit-learn>

## A3 Dump

Write a single python file to perform the following tasks:

- Perform gradient descent to minimize the cost function  $f_1(a) = a^4$  with an initialization of  $a = 2.5$
- Perform gradient descent to minimize the cost function  $f_2(b) = \sin^2(b)$  with an initialization of  $b = 0.6$  (where  $b$  is assumed to be in radians)
- Perform gradient descent to minimize the cost function  $f_3(c, d) = c^5 + d^2\sin(d)$  with an initialization of  $(c, d) = (2, 3)$ . (where  $c, d$  are assumed to be in radians)

Submit a single python file with filename “[A3\\_StudentMatriculationNumber.py](#)”. It should contain a function [A3\\_MatricNumber](#) that takes the following inputs and returns the following outputs in the following order:

```
# f1(a) = a^4
# f1'(a) = 4a^3
# final a should be 0, f1(a) should be 0
a0 = 2.5
a_out = []
a_out.append(a0)

# f2(b) = sin^2(b)
# f2'(b) = 2sin(b)cos(b)
# final b should be 0, f2(b) should be 0
b0 = 0.6
b_out = []
b_out.append(b0)
```



```

23
24 # f3(c, d) = c^5 + d^2 sin(d)
25 # f3'(c, d) = [5c^4, d^2 cos(d) + 2d sin(d)]
26 # take del of f3
27 # c = c - learning_rate * c component of del(f3(c, d))
28 # d = d - learning_rate * d component of del(f3(c, d))
29 # final c should be 0, final d should be 5.087, f3(c, d) should be -24.083
30 c0 = 2
31 d0 = 3
32 c_out = []
33 d_out = []
34 c_out.append(c0)
35 d_out.append(d0)
36
37 # gradient descent
38 for iteration in range(num_iters):

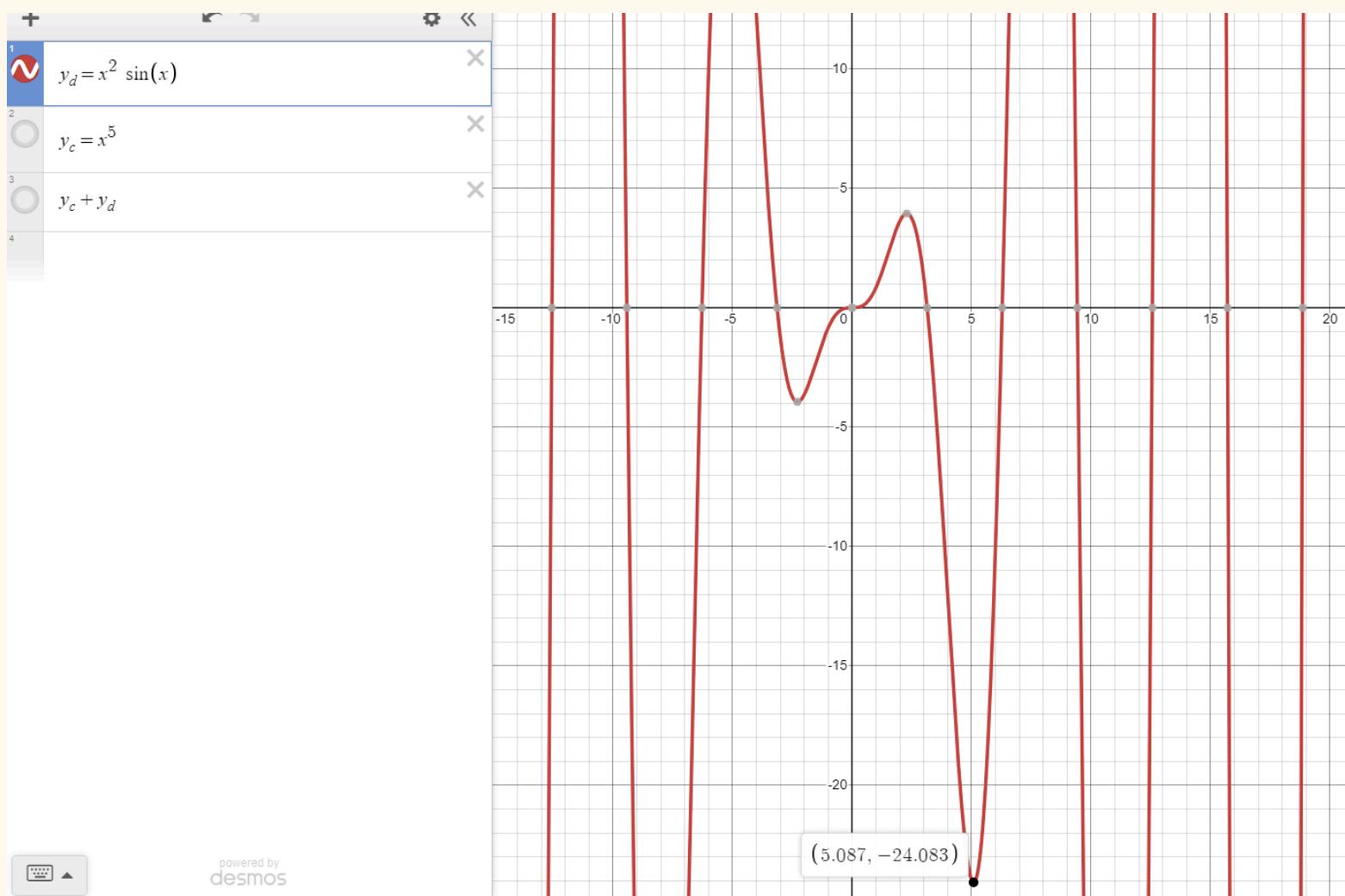
```

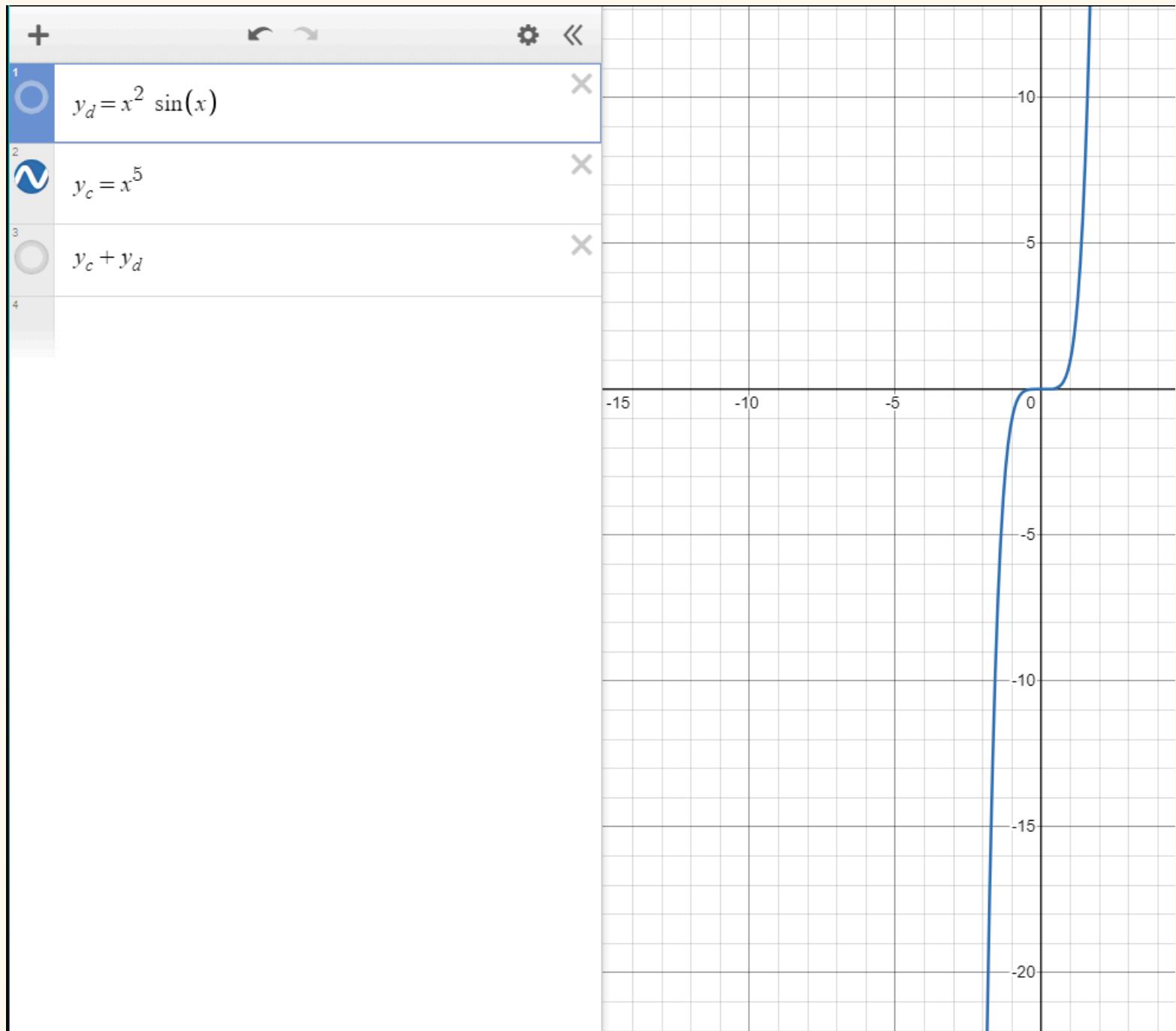
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```

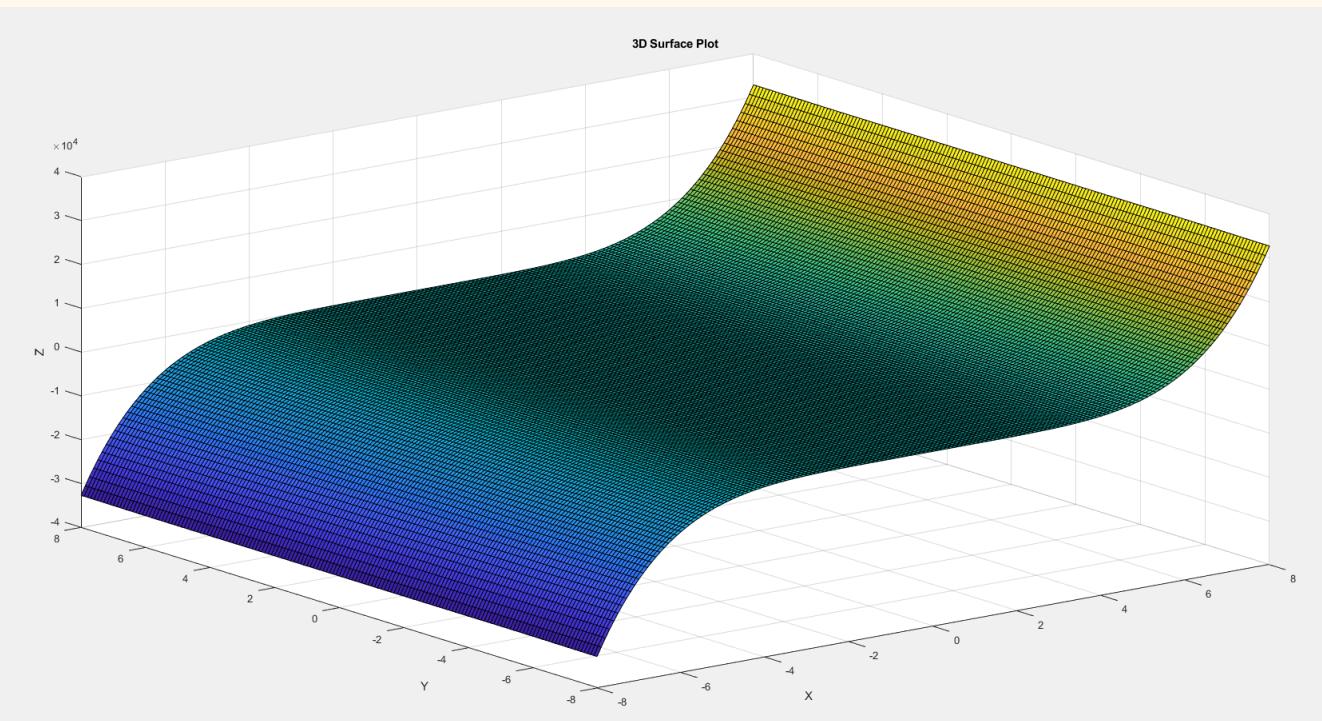
[3.16126425 3.36358473 3.61392905 ... 5.08698509 5.08698509 5.08698509]
f3_out =
[-0.18633721 -2.48105817 -5.93249414 ... -24.08295522 -24.08295522
-24.08295523]
PS C:\University Stuff\Y2S2\EE2211> & C:/Users/YTTRIUM/anaconda3/envs/ee2211/python.exe "c:/University Stuff/Y2S2/EE22
c_out =
● [0.4       0.39744    0.39494491 ... 0.08705649 0.08705075 0.08704501]
d_out =
[3.16126425 3.36358473 3.61392905 ... 5.08698509 5.08698509 5.08698509]
f3_out =
[-0.18633721 -2.48105817 -5.93249414 ... -24.08295522 -24.08295522
-24.08295523]
PS C:\University Stuff\Y2S2\EE2211> & C:/Users/YTTRIUM/anaconda3/envs/ee2211/python.exe "c:/University Stuff/Y2S2/EE22
● c_out =
[0.4       0.39744    0.39494491 ... 0.06921499 0.06921269 0.0692104 ]
d_out =
[3.16126425 3.36358473 3.61392905 ... 5.08698509 5.08698509 5.08698509]
f3_out =
[-0.18633721 -2.48105817 -5.93249414 ... -24.08295863 -24.08295863
-24.08295864]
PS C:\University Stuff\Y2S2\EE2211>

```





- plotting surface in matlab cant see shit since  $x^5$  term is dominate



```
>> [x, y] = meshgrid(-8:0.1:8);
z = x.^5 + (y.^2) .* sin(y);
surf(x, y, z)
title('3D Surface Plot')
xlabel('X')
ylabel('Y')
zlabel('Z')
fx >>
```

- note the `.*` and `.^` for element wise multiplication and exponentiation

```
PS C:\University Stuff\Y2S2\EE2211> & C:/Users/YTTRIUM/anaconda3/envs/ee2211/python.exe
● a_out =
[1.8750000e+00 1.61132812e+00 1.44398343e+00 ... 1.11803333e-03
 1.11803327e-03 1.11803322e-03]
f1_out =
[1.23596191e+01 6.74118041e+00 4.34759256e+00 ... 1.56249631e-12
 1.56249600e-12 1.56249569e-12]
b_out =
[5.90679609e-001 5.81428380e-001 5.72248978e-001 ... 1.18575755e-322
 1.18575755e-322 1.18575755e-322]
f2_out =
[0.31016613 0.30164059 0.29324835 ... 0. 0. 0. ]
c_out =
[1.2 1.09632 1.02408971 ... 0.0087358 0.0087358 0.0087358 ]
d_out =
[3.08063212 3.17160517 3.27405413 ... 5.08698509 5.08698509 5.08698509]
f3_out =
[-3.06649511 1.28189725 -0.28936978 ... -24.08296022 -24.08296022
 -24.08296022]
PS C:\University Stuff\Y2S2\EE2211> []
```

## Admin

- Introduction and Preliminaries (Xinchao)
  - Introduction
  - Data Engineering
  - Introduction to Probability and Statistics
- Fundamental Machine Learning Algorithms I (Yueming)
  - Systems of linear equations
  - Least squares, Linear regression
  - Ridge regression, Polynomial regression
- Fundamental Machine Learning Algorithms II (Yueming)
  - Over-fitting, bias/variance trade-off
  - Optimization, Gradient descent
  - Decision Trees, Random Forest
- Performance and More Algorithms (Xinchao)
  - Performance Issues
  - K-means Clustering
  - Neural Networks

- Schedule
  - **12 Weeks Lectures**, starting from Week 1
  - **12 Weeks Tutorials**, starting from Week 2
  - **2 Programming Tutorials** (optional and highly recommended)
    - Week 1 – 2, Friday
    - Right after the lecture (i.e., 2 to 3 PM)
  - **1 Mid-term Quiz** (using ExamSoft)
    - Tentatively held offline on 9 March 2024 (Recess Week)
    - Content up to Week 5 (inclusive)
  - **1 Briefing Session on ExamSoft**
    - Tentative on Week 3, exact time to be confirmed with CIT Staff
  - **1 Final Exam** (using ExamSoft)
    - Held on 3-May-2024
  - **3 Assignments**
    - Assignment 1: released on Week 4, due on Week 6 (tentatively)
    - Assignment 2: released on Week 6, due on Week 9 (tentatively)
    - Assignment 3: released on Week 9, due on Week 13 (tentatively)

- midterm is 16 march

- 3 Assignments (36%) + Tutorial Attendance (4%)
  - 1 Mid-term (30%)
  - 1 Final Exam (30%)
- 
- Held online:
    - Lectures
  - Held offline (in classrooms):
    - Tutorials
- 
- Videos of lectures are made available after lectures.
  - midterm is open book