

# **CS5228: Knowledge Discovery and Data Mining**

## Lecture 7 — Classification & Regression III

# Course Logistics — Update

- Assignment 3
  - Available on Canvas (since Oct 13)
  - Submission deadline: Thu Oct 24, 11.59 pm
- Project
  - Progress reports completed + feedback provided to almost most team  
(there will be an announcement with a short summary)
  - 1st TEAMMATES session live (deadline: Oct 17, 11.59 pm)

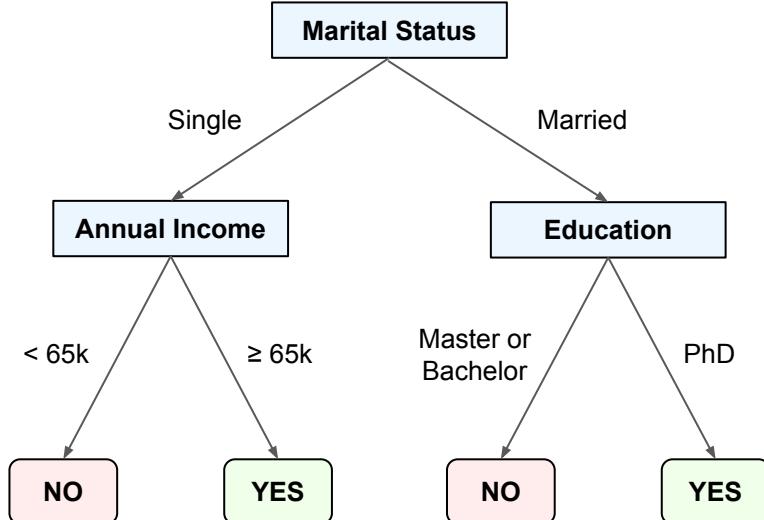
# Quick Recap — Decision Trees

- **Decision Trees**

- Flowchart-like structure mapping input features to output labels or values
- Applicable to classification & regression tasks
- Support for categorical & numerical features
- Typically quite interpretable

- **Building Decision Trees**

- Greedy algorithm iteratively finding the best splits
- Best split = split that minimizes impurity

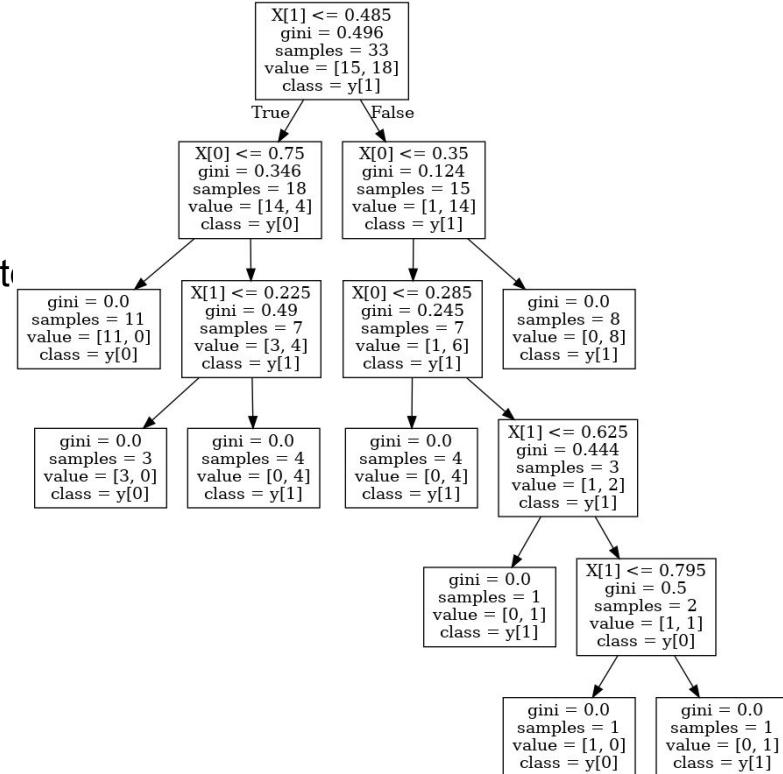


# Quick Recap — Decision Trees

- Challenges

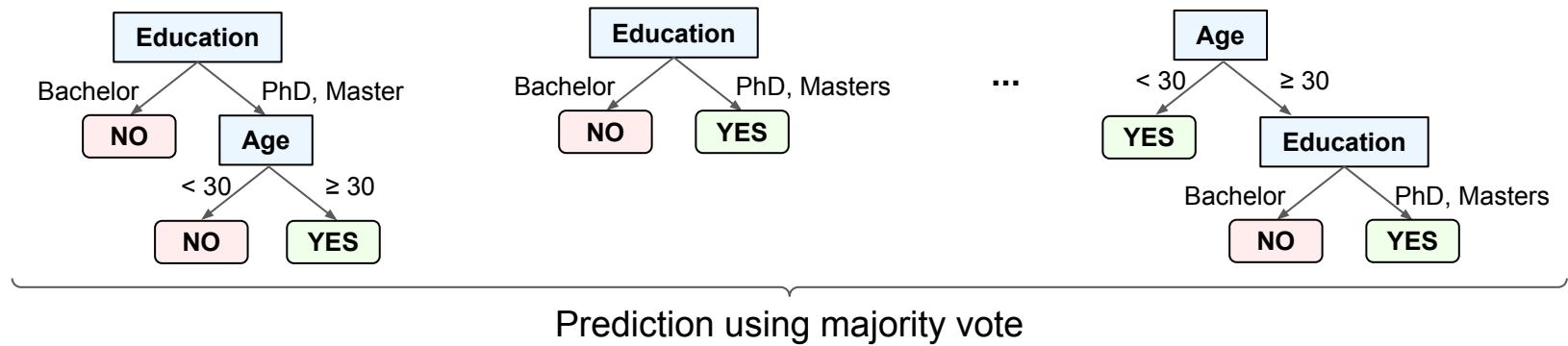
- Sensitive to small changes in training data  
→ High variance
- High chance of overfitting in case of maximum/completely pure tree  
→ Pruning of tree to avoid overfitting
- In practice, Decision Trees do not show state-of-the-art performance

→ Tree Ensemble methods



# Quick Recap — Tree Ensembles

- **Tree ensembles:** Construct many decision trees and combine their predictions
  - Pros: Higher accuracy, lower variance
  - Cons: Lower interpretability, longer training time
- **Ensembles of independent models:**
  - Bagging — Train decision trees over re-sampled training data (bootstrap sampling)
  - Random Forests — bootstrap sampling + feature sampling



# Quick Recap — Tree Ensembles

- Ensembles of dependent models (boosting methods)
  - Sequential training of decision trees
  - Next tree tries to improve errors of previous trees
  - Trees have different amount of say in predictions
- Two introduced boosting methods:
  - AdaBoost — resample training data to favour previously misclassified samples
  - Gradient Boosted Trees — start with initial prediction, improve based on predicted errors

# Outline

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Linear Models

- Basic setup

- Dataset of  $n$  samples  $\{(x_i, y_i)\}_{i=1}^n$
- Input data with  $d$  features  $x_i = (x_{i1}, x_{i2}, \dots, x_{id})$

- Assumption

- There exists linear relationship between  $x_i$  and dependent variable  $y_i$

$$\hat{y}_i = h_\theta(x_i) = f(\theta_0 x_{i0} + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots + \theta_d x_{id})$$

Predicted value which  
is hopefully close to  $y_i$

$$\theta = \{\theta_0, \theta_1, \theta_2, \dots, \theta_d\}, \theta_i \in \mathbb{R}$$

| Age | Edu-<br>cation | Marital<br>Status | Income<br>Level | Credit<br>Approval | Credit<br>Limit |
|-----|----------------|-------------------|-----------------|--------------------|-----------------|
| 23  | Masters        | Single            | Mid             | No                 | \$S5,000        |
| 35  | College        | Married           | High            | Yes                | \$S7,000        |
| 26  | Masters        | Single            | High            | No                 | \$S9,000        |
| ... | ...            | ...               | ...             | ...                | ...             |

# Linear Models

- Vector representation

- Introduce constant feature  $x_{i0}$

$$h_\theta(x_i) = f(\underbrace{\theta_0 x_{i0} + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots + \theta_d x_{id}}_{= 1})$$

- Represent  $x_i$  with new constant feature

$$x_i = (1, x_{i1}, x_{i2}, \dots, x_{id})$$

- Rewrite linear relationship using vectors representing  $x_i$  and  $\theta$

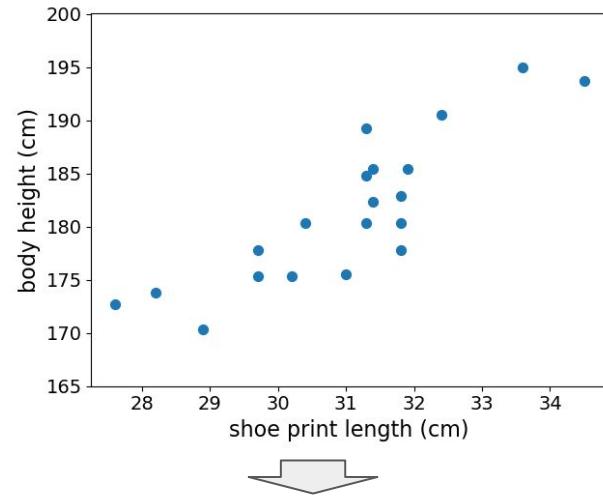
$$h_\theta(x_i) = f(\theta^T x_i)$$

# Outline

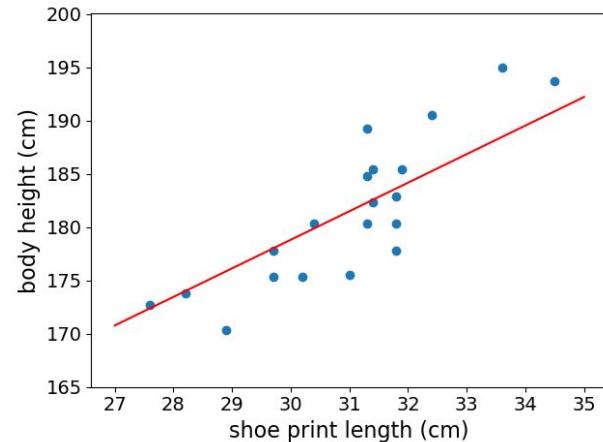
- **Linear Models**
  - Basic setup
- **Linear Regression**
  - **Problem formulation**
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Linear Regression — Simple Example

- Crime scene investigation (CSI)
  - Found a shoe print of size 32.2cm
  - What is the estimated height of the suspect?



- Approach: Linear Regression
  - Collect a dataset of (size, height)-pairs
  - Quantify linear relationship between shoe print size and body height  $\rightarrow h_{\theta}(size) = \theta_0 + \theta_1 size$
  - Predict suspect's height via  $\hat{y} = h_{\theta}(32.2)$



# Linear Regression

- Regression → Real-valued predictions

- Function  $f$  is the identity function  $f(x) = x$

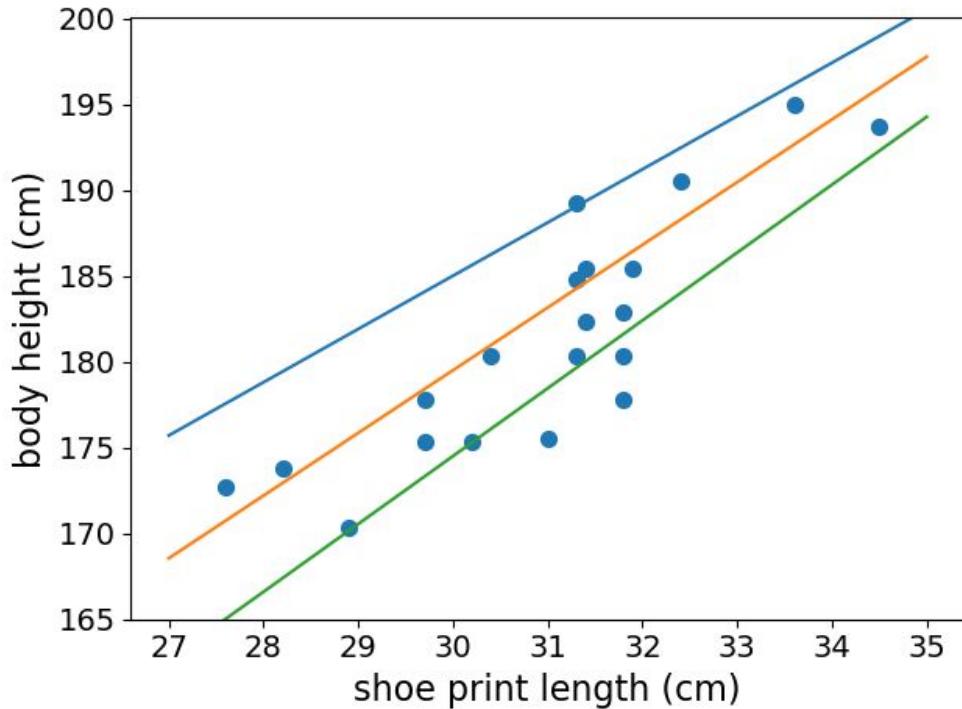
$$\hat{y}_i = h_{\theta}(x_i) = f(\theta^T x_i) = \theta^T x_i$$

$$\hat{y}_i = \theta^T x_i$$

$$\hat{y} = X\theta$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$$

# Quick Quiz



Which is the best line?

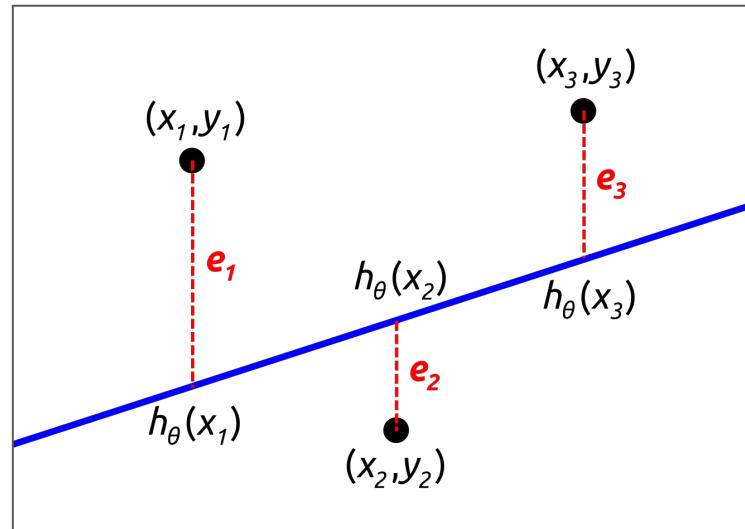
Why?

How to find it?

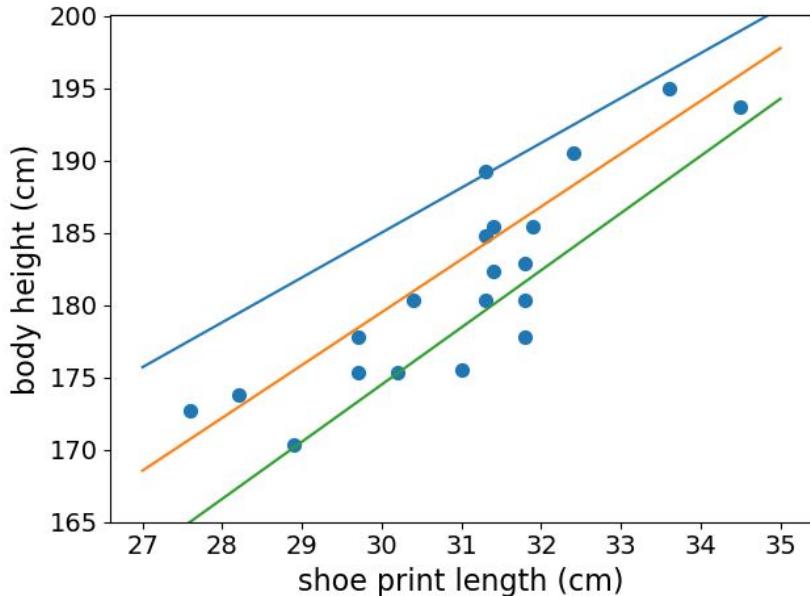
# Linear Regression — Loss Function

- **Loss function** (also: cost function, error function)
  - Quantifies how good or bad a given set of values for  $\theta$  is?
  - Measures the difference between predictions  $\hat{y}$  and true values  $y$
- Loss function for Linear Regression:  
**Mean Squared Error (MSE)**

$$L = \frac{1}{n} \sum_{i=1}^n e_i^2 = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$



# Losses for CSI Example



$$h_{blue} = 92 + 3.10 \cdot x$$
$$h_{orange} = 69 + 3.61 \cdot x$$
$$h_{green} = 56 + 3.95 \cdot x$$
$$h_{random} = 100 - 5.00 \cdot x$$

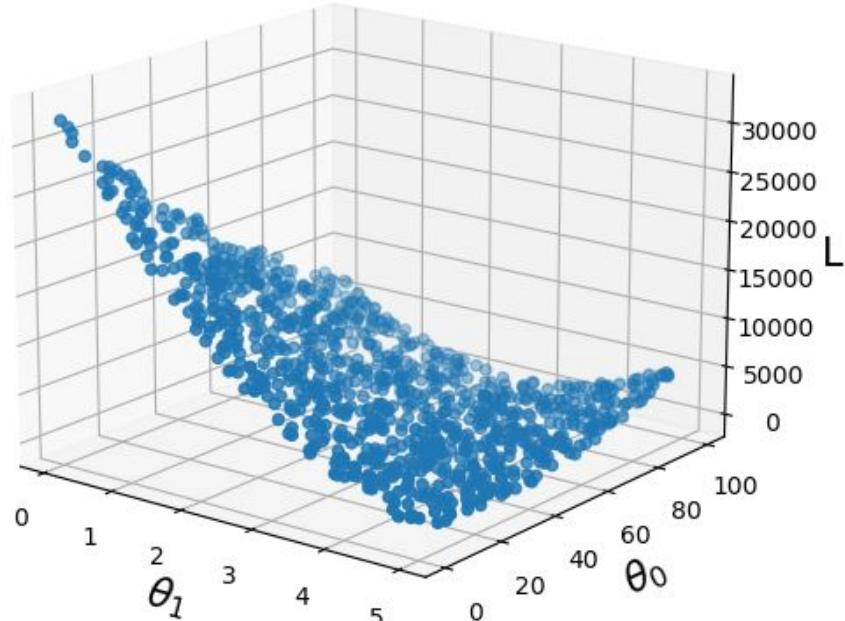
$$L_{blue} = 57.47$$
$$L_{orange} = 12.12$$
$$L_{green} = 20.83$$
$$L_{random} = 56, 129.23$$

→ How to find the best values for  $\theta$ ?

# Method 1: Random Search (the "stupid" way)

- Repeat "enough" times
  - Select random values for  $\theta = \{\theta_0, \theta_1, \dots, \theta_d\}$
  - Calculate loss L for current  $\theta$
- Return  $\theta$  with smallest loss
- Limitation:
  - Not practical beyond toy examples
- Don't do that! :)

Plot of 1,000 losses



# Outline

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - **Normal Equation** (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

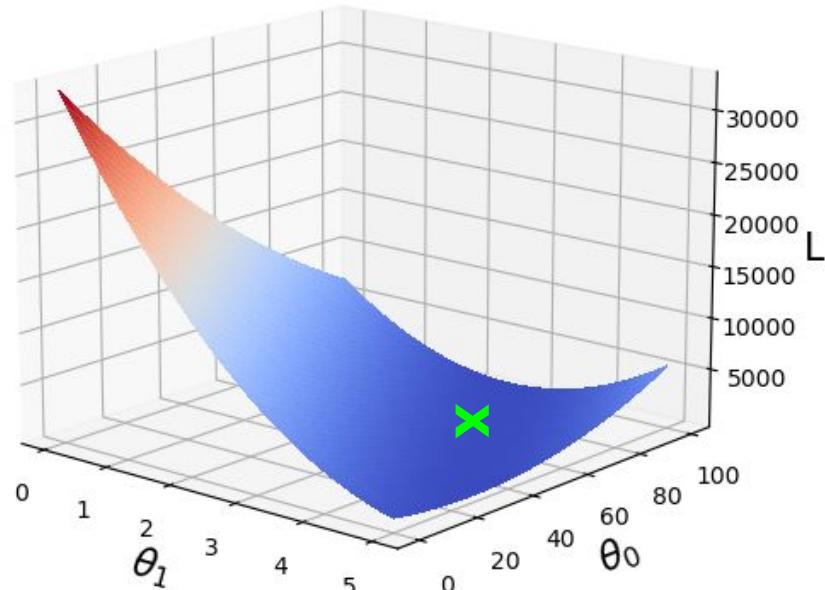
# Method 2: Find Minimum of $L$ Analytically (the proper way)

- Minimum of loss function  $L \rightarrow$  Calculus to the rescue!

- Partial derivatives w.r.t. to all  $\theta_i$  are 0

$$\frac{\partial L}{\partial \theta_0} = 0, \frac{\partial L}{\partial \theta_1} = 0, \dots, \frac{\partial L}{\partial \theta_d} = 0$$

- $d+1$  equations with  $d+1$  unknowns
- (no need to check if minimum or maximum)



# Method 2: Find Minimum of $L$ Analytically (the proper way)

- Rewrite loss function  $L$ 
  - Vector representation mathematically more convenient to handle
  - Avoids dealing with  $d$  equations

$$\begin{aligned} L &= \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\theta^T x_i - y_i)^2 \\ &= \frac{1}{n} \|X\theta - y\|^2 \end{aligned}$$

- Derive  $L$  w.r.t. to  $\theta$ 
  - Using chain rule
- Set  $\partial L / \partial \theta$  to 0

$$\frac{\partial L}{\partial \theta} = \frac{2}{n} X^T (X\theta - y)$$

$$\frac{2}{n} X^T (X\theta - y) \stackrel{!}{=} \overrightarrow{0}$$

# Linear Regression — Normal Equation

- Solve for  $\theta$

$$\frac{2}{n} X^T (X\theta - y) = \vec{0}$$

$$X^T X\theta = X^T y$$

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

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

$$\theta = X^\dagger y, \text{ with } X^\dagger = \underbrace{(X^T X)^{-1} X^T}_{\text{"pseudo inverse" of } X}$$

# Pseudo Inverse $X^\dagger$

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

$$\begin{matrix} X \\ X^T \end{matrix} \begin{matrix} X \\ (d+1) \times n \end{matrix} \left[ \begin{matrix} X^T X \\ n \times (d+1) \end{matrix} \right] = \left[ \begin{matrix} X^T X \\ (d+1) \times (d+1) \end{matrix} \right] \longrightarrow \underbrace{\left[ \begin{matrix} (X^T X)^{-1} \\ (d+1) \times (d+1) \end{matrix} \right]^{-1}}_{(d+1) \times n} \left[ \begin{matrix} X^T \\ (d+1) \times n \end{matrix} \right]$$

## • Performance analysis

- Most expensive operation: calculating the inverse of  $(X^T X)^{-1}$
- Calculation of inverse depends on number of features  $d$ , not on number of data samples  $n$
- Complexity of calculating inverse of a  $d \times d$  matrix:  $O(d^3)$

# Quick Quiz

What condition is **not required** for a matrix A to be **invertible**?

**A**

The determinant of A is non-zero

**B**

A is a square matrix

**C**

A has full rank

**D**

All diagonal values are non-zero

# Quick Quiz

When will  $X^T X$  **not** be invertible?

A

There are more features  $d$  than data samples  $n$

B

The data is not normalized

C

In practice,  $X^T X$  will always be invertible

D

$X$  has no determinant

# Outline

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - **Gradient Descent** (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Method 2: Find Minimum of $L$ Analytically (the proper way)

- **Algorithm**

- Construct matrix  $X$  and vector  $y$  from data
- Calculate pseudo inverse  $X^\dagger = (X^T X)^{-1} X^T$
- Return  $\theta = X^\dagger y$

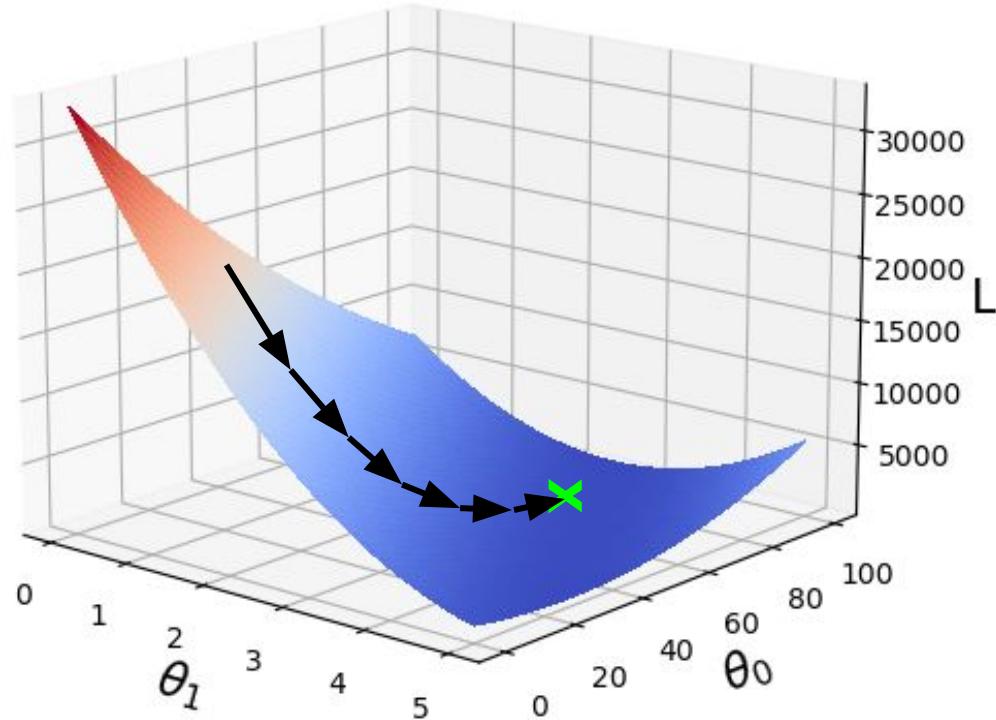
- For the CSI example:

$$\theta = \begin{bmatrix} 20.0 & 620.2 \\ 620.0 & 19284.9 \end{bmatrix}^{-1} X^T y = \begin{bmatrix} 18.4 & -0.6 \\ -0.6 & 0.02 \end{bmatrix} X^T y = \begin{bmatrix} 69.5 \\ 3.6 \end{bmatrix}$$

$$h_\theta(\text{size}) = 69.5 + 3.6\text{size} \quad h_\theta(32.2) = 185.4$$

# Method 3: Gradient Descent

- Core idea
  - Start with a random setting of  $\theta$
  - Adjust  $\theta$  iteratively to minimize  $L$



# Gradient — Quick Refresher

- Gradient

- Vector of partial derivatives of a multivariable function (e.g.,  $\theta_0, \theta_1, \dots, \theta_d$ )
- Partial derivative: slope w.r.t. to a single variable given a current set of values for all  $\theta_0, \theta_1, \dots, \theta_d$
- Points in the direction of the steepest ascent

$$\nabla_{\theta} L = \frac{\partial L}{\partial \theta} = \begin{bmatrix} \frac{\partial L}{\partial \theta_0} \\ \frac{\partial L}{\partial \theta_1} \\ \frac{\partial L}{\partial \theta_2} \\ \vdots \\ \frac{\partial L}{\partial \theta_d} \end{bmatrix}$$

# Gradient for CSI Example

Best value:  $\theta_0 = 69.5, \theta_1 = 3.6$

- Assume  $\theta_0 = 60$  and  $\theta_1 = 4$

$$\nabla_{\theta}L = \frac{2}{n}X^T(X\theta - y) = \frac{2}{n}X^T(X \cdot \begin{bmatrix} 60 \\ 4 \end{bmatrix} - y) = \begin{bmatrix} 5.2 \\ 163.9 \end{bmatrix}$$

- Interpretation of  $\nabla_{\theta}L = \begin{bmatrix} 5.2 \\ 163.9 \end{bmatrix}$ 
  - Both values positive: a small increase of  $\theta_0$  or  $\theta_1$  will increase the loss
  - A small change in  $\theta_1$  affects the loss more than the same change in  $\theta_0$
  - Absolute values of gradient not a direct indicator of how to update  $\theta$

# Gradient Descent Algorithm

- Important concept: learning rate
  - Scaling factor for gradient (typical range: 0.01 - 0.0001)

**Input** : data  $(X, y)$ , loss function  $L$ , learning rate  $\eta$

**Initialization** : Set  $\theta$  to random values

**while true :**

    Calculate gradient  $\nabla_{\theta} L$

$\theta \leftarrow \theta - (\eta \cdot \nabla_{\theta} L)$

In practice: stop loop  
when  $\theta$  converges

# Gradient Descent for CSI Example

- Input
  - $\eta = 0.0001$
- Initialization
  - $\theta_0 = 100$
  - $\theta_1 = 50$

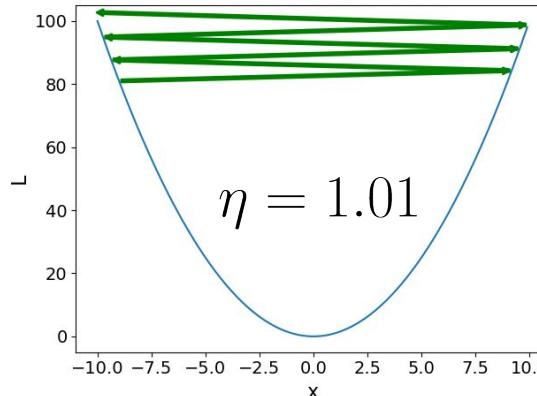
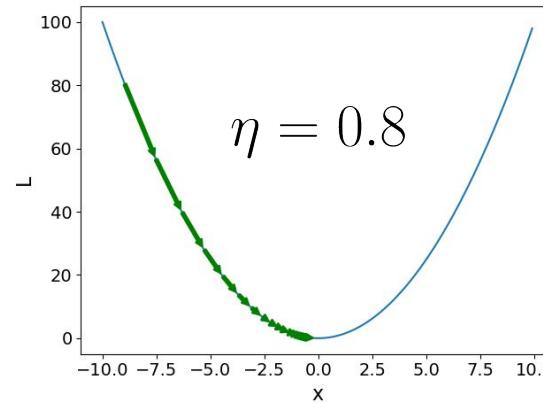
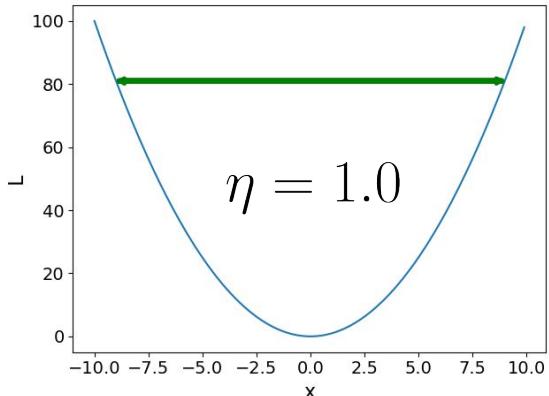
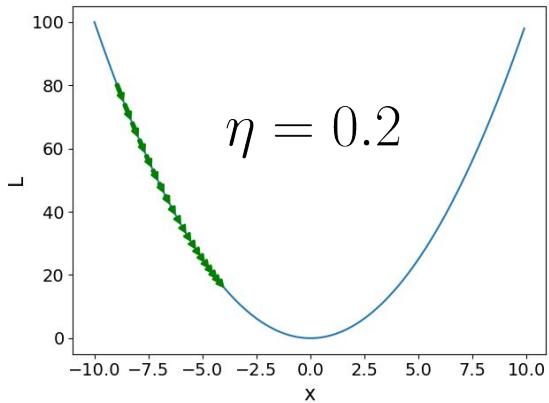
```
[001] theta0=99.70619, theta1=40.86448, loss=2163825.04200
[002] theta0=99.46909, theta1=33.49256, loss=1409025.31406
[003] theta0=99.27776, theta1=27.54378, loss=917521.53239
[004] theta0=99.12336, theta1=22.74340, loss=597468.46605
[005] theta0=98.99877, theta1=18.86972, loss=389059.15347
[006] theta0=98.89823, theta1=15.74385, loss=253349.02868
[007] theta0=98.81709, theta1=13.22143, loss=164978.51518
[008] theta0=98.75161, theta1=11.18595, loss=107434.18922
[009] theta0=98.69877, theta1=9.54342, loss=69962.98624
[010] theta0=98.65613, theta1=8.21798, loss=45562.82115
[011] theta0=98.62172, theta1=7.14841, loss=29674.13840
[012] theta0=98.59395, theta1=6.28532, loss=19327.88711
[013] theta0=98.57153, theta1=5.58885, loss=12590.70710
[014] theta0=98.55344, theta1=5.02683, loss=8203.65008
[015] theta0=98.53884, theta1=4.57330, loss=5346.92525
[016] theta0=98.52706, theta1=4.20733, loss=3486.70856
[017] theta0=98.51754, theta1=3.91201, loss=2275.38917
[018] theta0=98.50986, theta1=3.67370, loss=1486.61298
[019] theta0=98.50366, theta1=3.48140, loss=972.98470
[020] theta0=98.49866, theta1=3.32622, loss=638.52480

...
[098] theta0=98.47656, theta1=2.67760, loss=14.17658
[099] theta0=98.47655, theta1=2.67760, loss=14.17658
[100] theta0=98.47653, theta1=2.67760, loss=14.17658
```

**Quick Quiz:** Why not just increase the learning rate to speed things up?

# Effects of Learning Rate for

$$L = x^2, \frac{\partial L}{\partial x} = 2x, \text{ 20 steps}$$

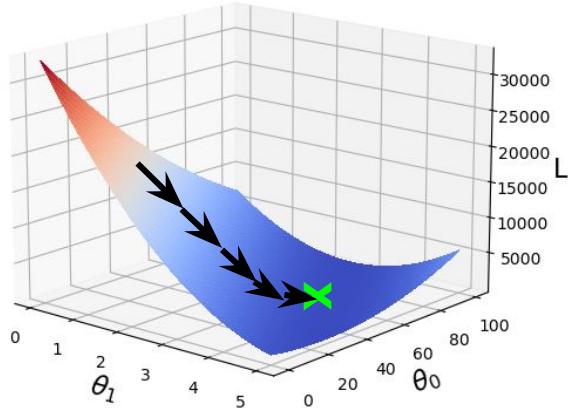


# Gradient Descent — Variations

- (Basic) Gradient Descent
  - Calculate gradient und update  $\theta$  for whole dataset
- Stochastic Gradient Descent (SGD)
  - Calculate gradient und update  $\theta$  for each data sample
- Mini-batch Gradient Descent
  - Calculate gradient und update  $\theta$  for batches of sample
  - e.g., batch = 64 data samples
  - In practice often referred to as SGD

# Gradient Descent — Variations

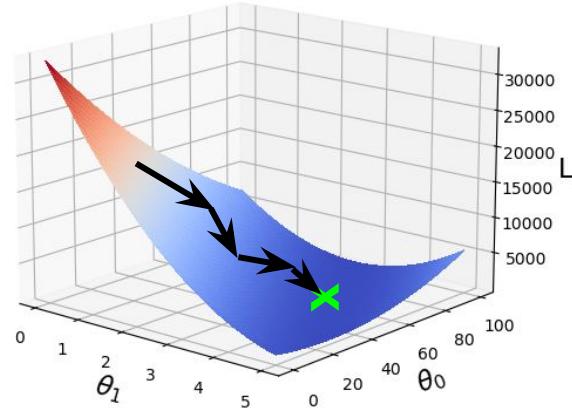
Gradient Descent



Gradient averaged over all data items

- Smooth descent
- Small(er) gradients
- Small(er) update steps

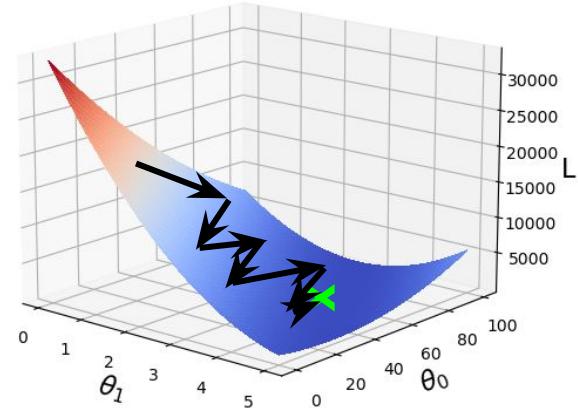
Mini-Batch Gradient Descent



Gradient averaged over some data items

- Well, "somewhere in-between" :)

Stochastic Gradient Descent



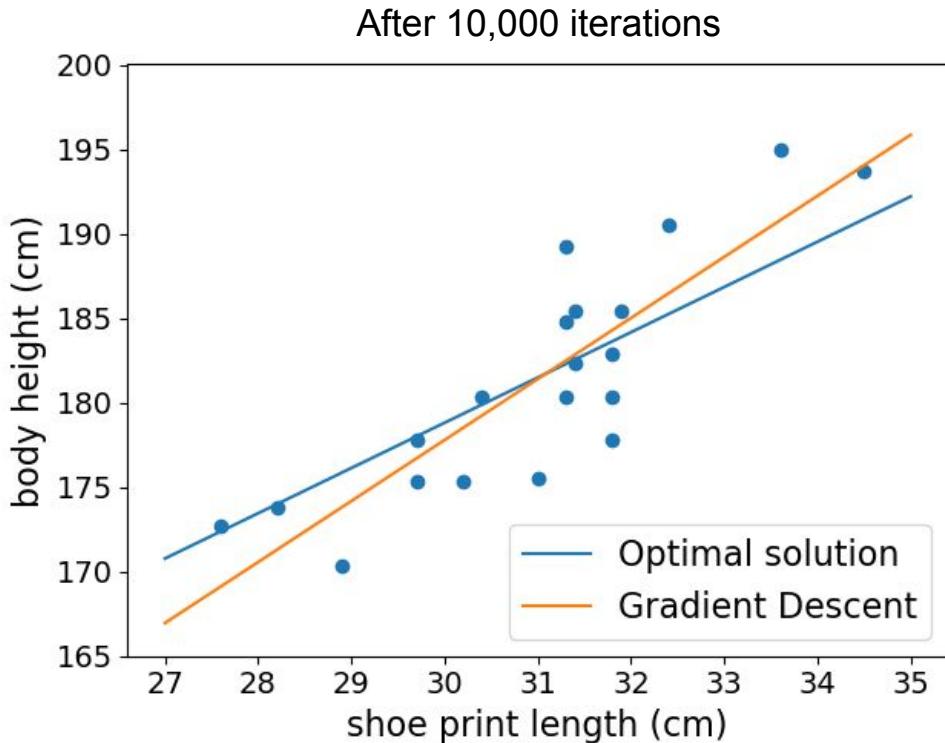
Gradient for each data item considered

- Choppy descent
- Large(r) gradients
- Large(r) steps

# Normal Equation vs. Gradient Descent

- Gradient Descent
  - Works well even if  $d$  is large
  - Works even if  $X^T X$  is non-invertible
  - Iterative process; may not find optimal solution in practice
  - Learning rate a critical hyperparameter
  
- Normal Equation
  - Finds optimal solutions
  - Non-iterative; no need of learning rate
  - Calculation of  $(X^T X)^{-1}$  in  $O(d^3)$

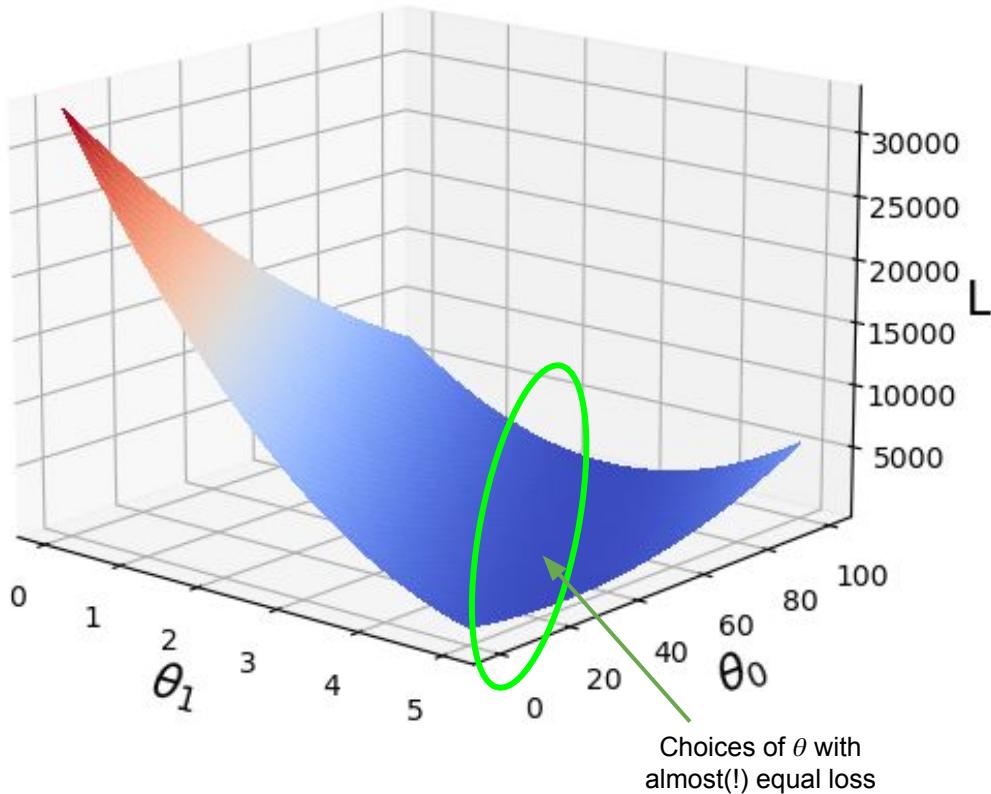
# Quick Quiz



Gradient Descent not reaching optimal solution after 10k iterations.

Why?

# Quick Quiz



Region of "near-plateau":

- Gradient  $\nabla_{\theta} L$  very small
- Step  $\eta \nabla_{\theta} L$  extremely small
- Very slow convergence

# Outline

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - **Polynomial Linear Regression** (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Polynomial Linear Regression

- Linear Regression  $\not\rightarrow$  line / plane / hyperplane
- Polynomial Linear Regression
  - Allows to capture nonlinear relationships between  $X$  and  $y$
  - Polynomial regression model for 1 input feature

Still linear in  $\theta$ !

$$\hat{y}_i = \theta_0 1 + \theta_1 x_i + \theta_2 x_i^2 + \dots + \theta_p x_i^p$$

- Matrix representation (again, 1 input feature!)

$$X^{(1)} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \quad X^{(2)} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \quad X^{(3)} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} \quad X^{(p)} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & x_2^3 & \dots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \dots & x_n^p \end{bmatrix}$$

# Polynomial Linear Regression

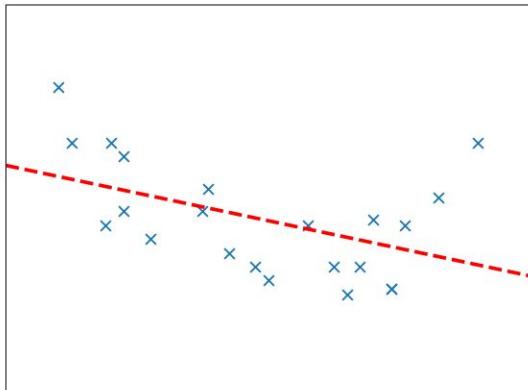
- Example: 1 input feature, 3 data samples

$$X^{(1)} = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} \quad X^{(2)} = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 1 & 1 \end{bmatrix} \quad X^{(3)} = \begin{bmatrix} 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Polynomial terms "look the same" as additional features
- Finding best  $\theta$  using the Normal Equation or Gradient Descent

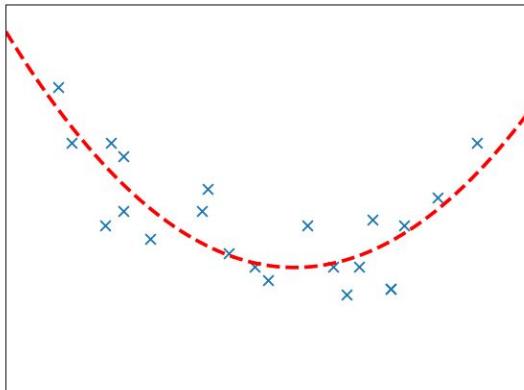
# Polynomial Linear Regression — Example

$p = 1$



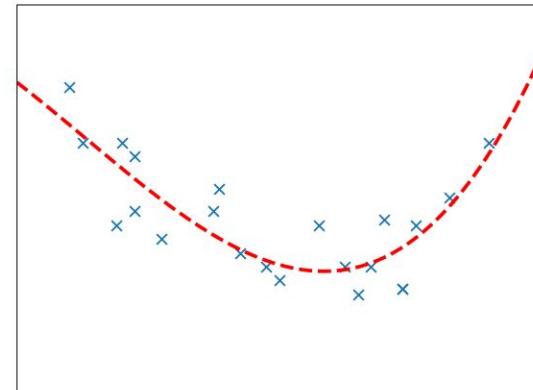
Underfitting

$p = 2$



Good fit

$p = 3$



Overfitting?

# Polynomial Linear Regression — Overfitting

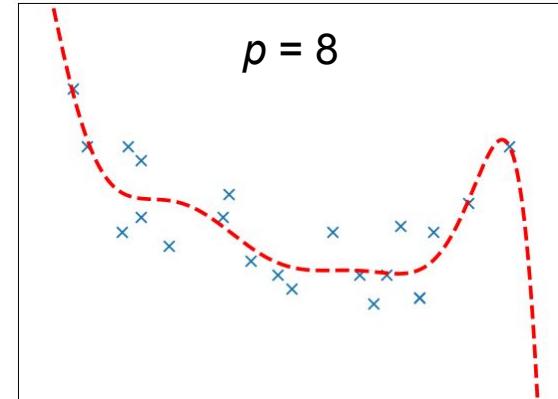
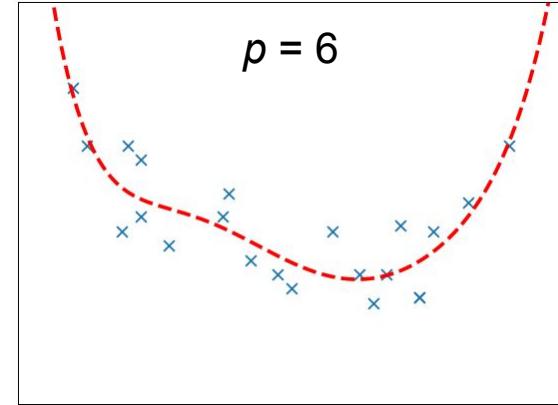
- Increasing degree of polynomial  $p$ 
  - More capacity to capture nonlinear relationships
  - Much higher sensitivity to noise and outliers
- Countermeasure: **Regularization**
  - Extend loss function to "punish" large values of  $\theta$

$$L = \frac{1}{n} \|X\theta - y\|^2 + \lambda \frac{1}{n} \|\theta\|_2^2$$

Regularization parameter

$$\|\theta\|_2^2 = \sum_{i=1}^d \theta_i^2$$

Note: excludes  $\theta_0$ !



# Polynomial Linear Regression — Minimizing Loss $L$

- Normal Equation

$$\theta = (X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix})^{-1} X^T y$$

- Gradient Descent

$$\nabla_{\theta} L = \frac{2}{n} X^T (X\theta - y) + \lambda \frac{2}{n} \theta$$

# Polynomial Linear Regression — More than 1 Feature

- Polynomial of degree  $p=2$  and two input features ( $d=2$ )

$$\hat{y}_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \theta_3 x_{i1}^2 + \theta_4 x_{i2}^2 + \underbrace{\theta_5 x_{i1} x_{i2}}_{\text{interaction terms (cross terms)}}$$

- Polynomial of degree  $p=3$  and two input features ( $d=2$ )

$$\begin{aligned}\hat{y}_i = & \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \theta_3 x_{i1}^2 + \theta_4 x_{i2}^2 + \theta_5 x_{i1}^3 + \theta_6 x_{i2}^3 + \\ & \theta_7 x_{i2}^2 x_{i1} + \theta_8 x_{i2} x_{i1}^2 + \theta_9 x_{i2} x_{i1}\end{aligned}$$

- Polynomial of degree  $p=2$  and three input features ( $d=3$ )

$$\begin{aligned}\hat{y}_i = & \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \theta_3 x_{i3} + \theta_4 x_{i1}^2 + \theta_5 x_{i2}^2 + \theta_6 x_{i3}^2 \\ & \theta_7 x_{i2} x_{i1} + \theta_8 x_{i3} x_{i1} + \theta_9 x_{i3} x_{i2}\end{aligned}$$

# Polynomial Linear Regression — More than 1 Feature

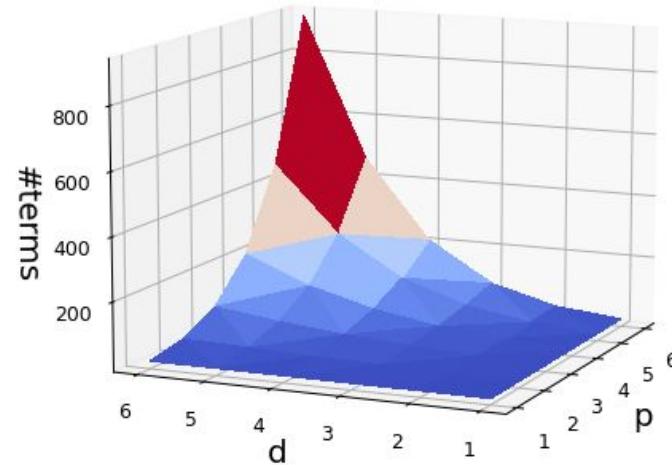
- Number of terms in multivariate polynomial given given  $p, d$

$$\theta_i, 0 \leq i \leq M, \text{ with } M = \binom{p+d}{p}$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Practical considerations

- Limited to small number of features and small polynomial degrees
- In principle, terms can be dropped (e.g., all interaction terms)



} risk of overfitting + low interpretability

} justification typically not obvious

# Outline

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - **Interpretation of Coefficients**
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Linear Regression — Interpretation of $\theta_i$

- Example: prediction of house prices

$$\left. \begin{array}{l} \text{price} = \theta_0 + \theta_1(\#\text{rooms}) + \theta_2(\text{area}) + \theta_3(\text{floor}) + \dots \\ \text{price}' = \theta_0 + \theta_1(\#\text{rooms} + 1) + \theta_2(\text{area}) + \theta_3(\text{floor}) + \dots \end{array} \right\} \Delta(\text{price}) = \text{price}' - \text{price} = \theta_1$$

- Interpretation
  - Change of value of feature  $i$  by 1 unit → change of output value by  $\theta_i$
  - Assumption: all other features values remain the same

→ What about normalizing the data?

# Data Normalization — Yes or No? (standardization, min-max scaling)

- Data normalization does not affect model performance  
(assuming basic Linear Regression without regularization)
- In favor of "No"
  - Preserves unit of feature  $i \rightarrow$  direct interpretation of  $\theta_i$
  - Better for comparing  $\theta_i$  for the same features across different datasets
- In favour of "Yes"
  - When using regularization
  - When using Polynomial Linear Regression
  - Better for comparing  $\theta_i$  within a model (e.g.,  $\theta_i > \theta_j \rightarrow$  feature  $i$  more important than feature  $j$ )

# Quick Quiz

Which of the statements regarding Linear Regression is **True**?

**A**

It's impossible to overfit given a dataset with only 1 feature

**B**

Scaling the data will change the coefficients

**C**

Gradient Descent can get stuck in local minimum

**D**

Regularization can improve the training loss/error

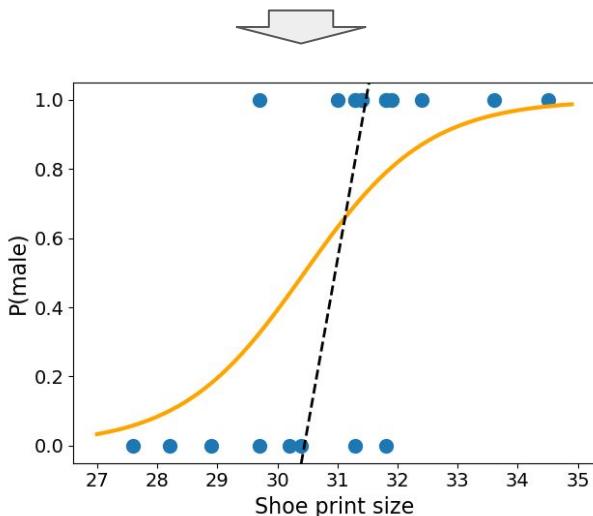
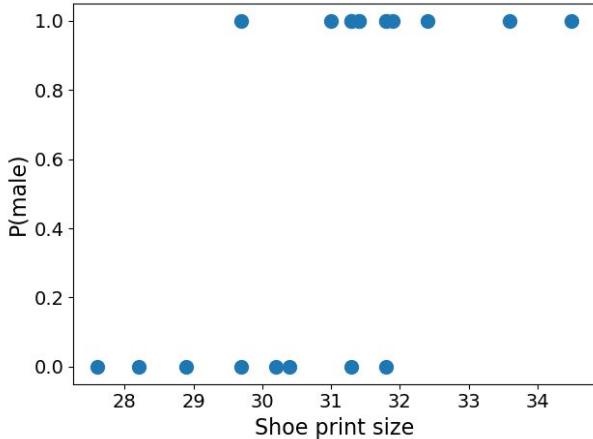
# Overview

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Logistic Regression — Simple Example

- Crime scene investigation (CSI)
  - Found a shoe print of size 32.2cm
  - Is the suspect a male or not?
- Approach: Logistic Regression for binary classification  $y_i \in \{0, 1\}$ 
  - Collect a dataset of (size, sex)-pairs
  - Train linear classifier such that

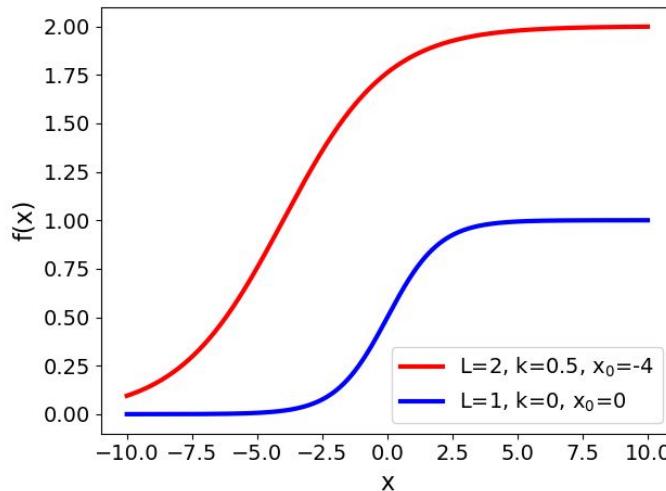
$$0 \leq h_{\theta}(x_i) \leq 1$$



# Logistic Regression

- Logistic Regression → Real-valued predictions interpreted as probability
  - Function  $f$  is the standard **Logistic Function** (Sigmoid function)

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \xrightarrow{L=1, k=1, x_0=0} f(x) = \frac{1}{1 + e^{-x}}$$



# Logistic Regression — Probabilistic Interpretation

- $\hat{y}$  interpreted as a probability

$$\hat{y} = h_{\theta}(x) = f(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad \text{with } \hat{y} \in [0, 1]$$

→  $\hat{y} = h_{\theta}(x)$  is the estimated probability that  $y_i = 1$  (male) given  $x$  and  $\theta$

$$\hat{y} = P(y = 1|x, \theta)$$

→ Given only discrete 2 outcomes:  $P(y = 1|x, \theta) + P(y = 0|x, \theta) = 1$

$$\hat{y} = 1 - P(y = 0|x, \theta)$$

# Logistic Regression — Probabilistic Interpretation

$$\hat{y} = P(y = 1|x, \theta) = 1 - P(y = 0|x, \theta)$$

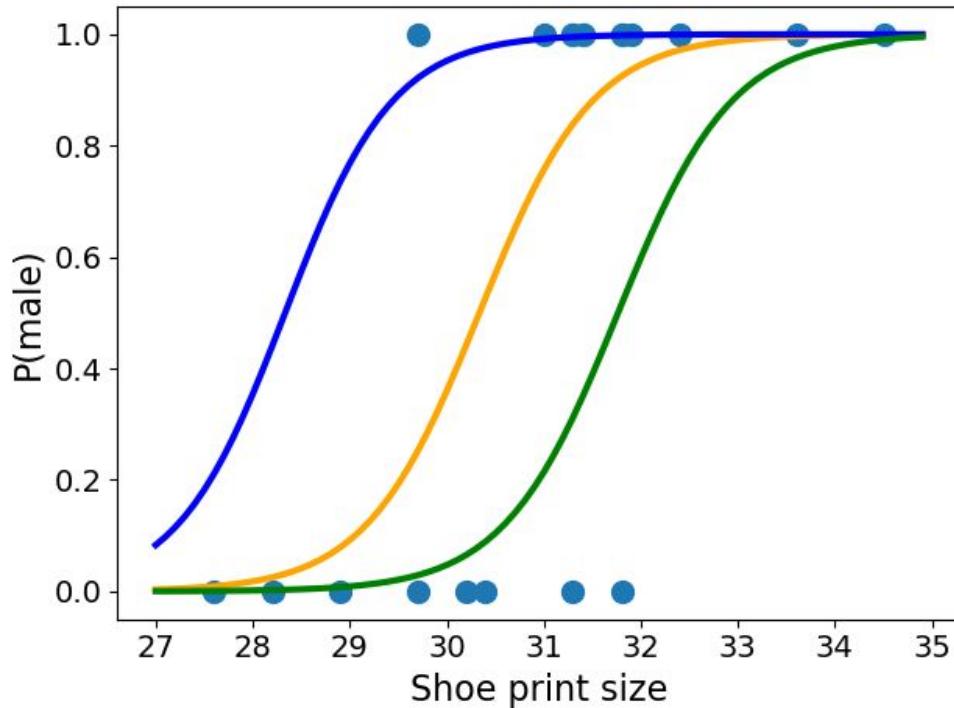
→  $P(y|x)$  is a Bernoulli distribution

$$P(y|x) = \begin{cases} \hat{y} & , y = 1 \\ 1 - \hat{y} & , y = 0 \end{cases}$$

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

# Logistic Regression



Which  $h_{\theta}(x)$  is the best?

How to find it  $\theta$ ?

# Logistic Regression — Loss Function

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

- Goal: Maximize probability of true  $y$  label given training sample  $x$

- Find  $\theta$  that **maximizes**

$$P(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

$$\begin{aligned}\log P(y|x) &= \log [\hat{y}^y (1 - \hat{y})^{1-y}] \\ &= y \log \hat{y} + (1 - y) \log (1 - \hat{y})\end{aligned}$$

- Find  $\theta$  that **minimizes**

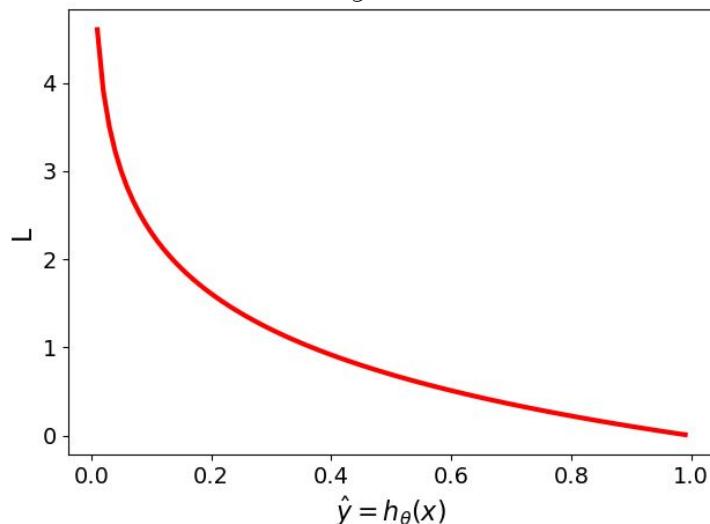
$$L = -P(y|x) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

Cross-Entropy Loss

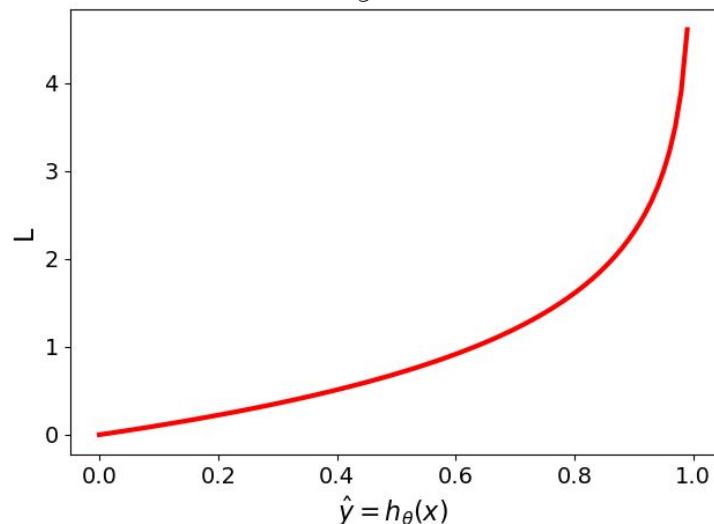
# Cross-Entropy Loss — Visualization

$$L = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

if  $y = 1$



if  $y = 0$



# Quick Quiz

What happens if  
Logistic Regression gets a  
training sample **correct**?

A

No loss will be calculated

B

The loss will be 0

C

The loss will be small

D

The loss will be large

# Overview

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Logistic Regression — Loss Function

$$\hat{y} = \frac{1}{1 + e^{-\theta^T x}}$$

- Loss for all training samples

$$\begin{aligned} L &= -\frac{1}{n} \sum_{i=1}^n [y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)] \\ &= -\frac{1}{n} \sum_{i=1}^n [y_i \log h_\theta(x_i) + (1 - y_i) \log (1 - h_\theta(x_i))] \\ &= -\frac{1}{n} \sum_{i=1}^n \left[ y_i \log \frac{1}{1 + e^{\theta^T x_i}} + (1 - y_i) \log \left(1 - \frac{1}{1 + e^{\theta^T x_i}}\right) \right] \end{aligned}$$

- Required for minimizing the loss:  $\nabla_\theta L = \frac{\partial L}{\partial \theta} = ???$

# Logistic Regression — Loss Function

- After lots of tedious math...

$$\frac{\partial L}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^n [h_\theta(x_i) - y_i] x_{ij}$$

$$\nabla_\theta L = \frac{1}{n} X^T (h_\theta(X) - y)$$

- Problem:  $\frac{1}{n} X^T (h_\theta(X) - y) \stackrel{!}{=} 0$  has no closed-form solution for  $\theta$

→ Gradient Descent!

# Logistic Regression in Practice (CSI Example)

Vanilla implementation

```
Start training for 1,000,000 iterations...
Loss: 0.6931471805599453      0.0%
Loss: 0.765069661957756      10.0%
Loss: 0.6093119537276577     20.0%
Loss: 0.5066673325457598     30.0%
Loss: 0.4551164039897514     40.0%
Loss: 0.4280635319707213     50.0%
Loss: 0.4154389042684111     60.0%
Loss: 0.4152887492109594     70.0%
Loss: 0.4151849953246913     80.0%
Loss: 0.41511286000667447    90.0%
loss: 0.41506245481850296   100.0%
Training finished in 0:00:09.617970
```

`sklearn.linear_model.LogisticRegression`  
(with default L-BFGS-B solver)

```
Start training...
Training finished in 0:00:00.000035 (#iterations: 21)
```

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -43.45 \\ 1.42 \end{bmatrix}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -44.91 \\ 1.47 \end{bmatrix}$$

# Logistic Regression in Practice (CSI Example)

Vanilla implementation

```
Start training for 78,000 iterations...
Loss: 0.6931471805599453      0.0%
Loss: 0.6565544350291496      10.0%
Loss: 0.6475478825116517      20.0%
Loss: 0.6389819932273285      30.0%
Loss: 0.6308342158505089      40.0%
Loss: 0.6230827400881452      50.0%
Loss: 0.6157065622831859      60.0%
Loss: 0.6086855316895313      70.0%
Loss: 0.6020003798552925      80.0%
Loss: 0.5956327355164103      90.0%
loss: 0.5895658866765062     100.0%
Training finished in 0:00:00.784118
```

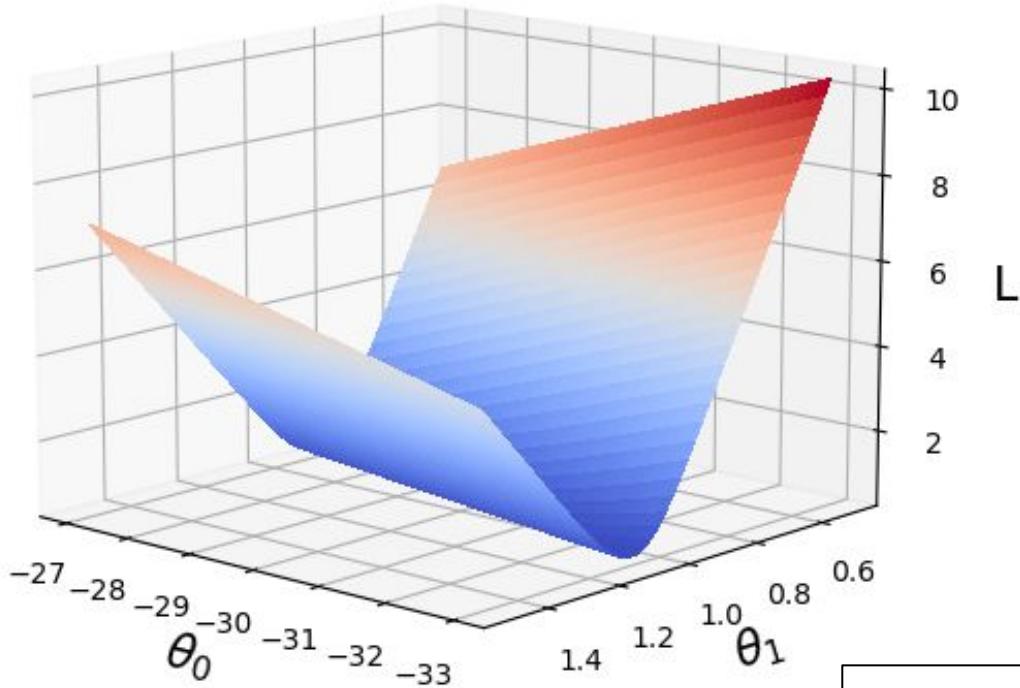
sklearn.linear\_model.LogisticRegression  
(with SAG solver)

```
Start training...
Training finished in 0:00:00.007022 (#iterations: 5820)
```

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -5.44 \\ 0.19 \end{bmatrix}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -5.45 \\ 0.19 \end{bmatrix}$$

# Logistic Regression in Practice (CSI Example)



Region of "near-plateau":

- Gradient  $\nabla_{\theta}L$  very small
- Step  $\eta \nabla_{\theta}L$  extremely small
- Very slow convergence

**Note:** The Cross-Entropy loss of Logistic Regression is convex  
→ There always exists exactly one minimum (global minimum)!

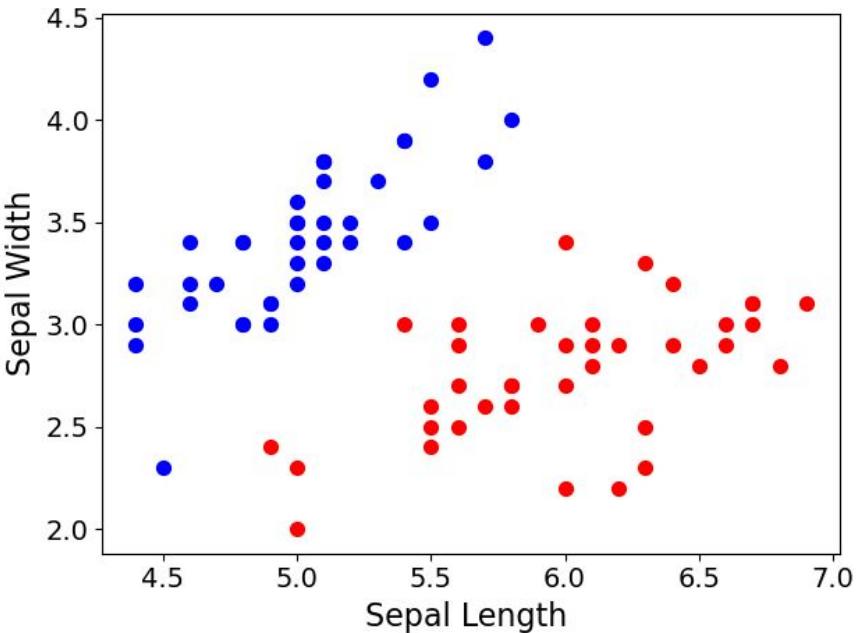
# Logistic Regression in Practice

- Logistic Regression with vanilla Gradient Descent (also Linear Regression)
  - Works in principle — the math is sound!
  - Often poor performance compared to more sophisticated implementations
- Many techniques to boost performance
  - Smart(er) initialization of  $\theta$
  - Adaptive learning rates
  - Extensions to Gradient Descent
  - Regularization
  - ...and others

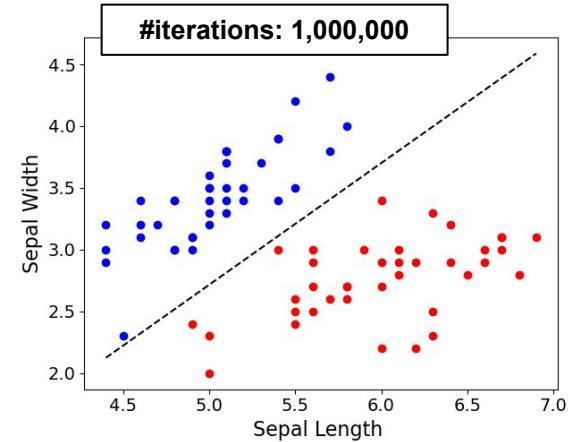
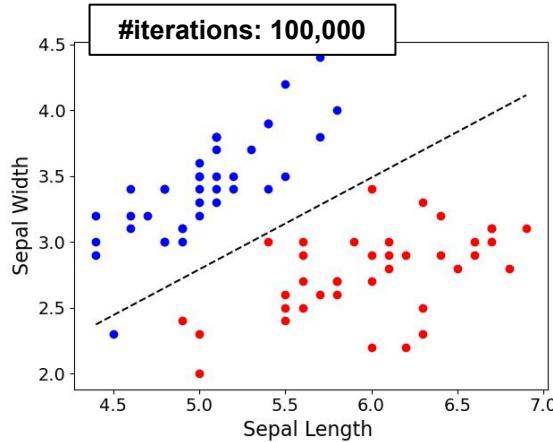
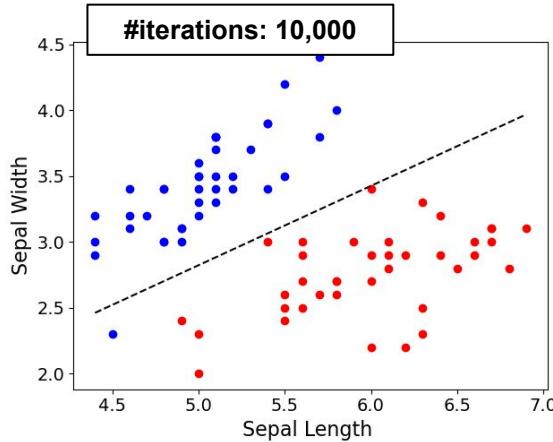
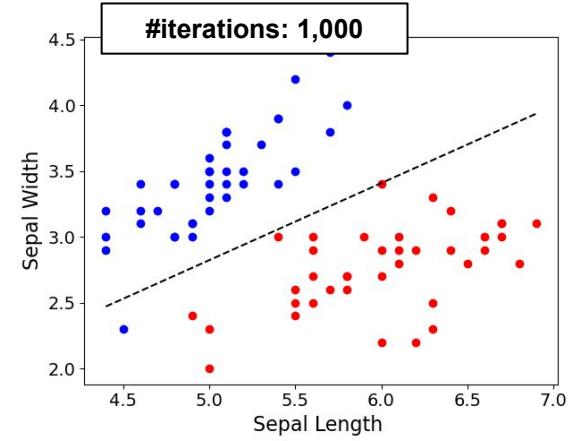
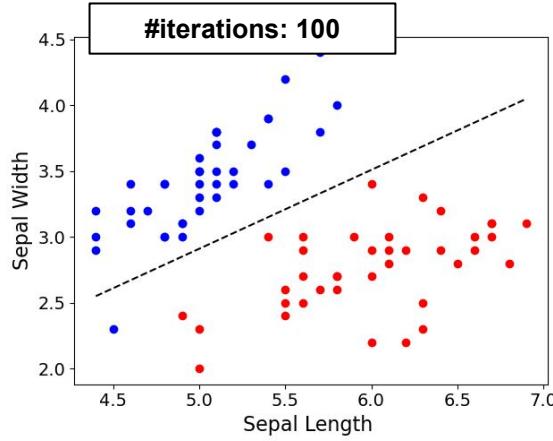
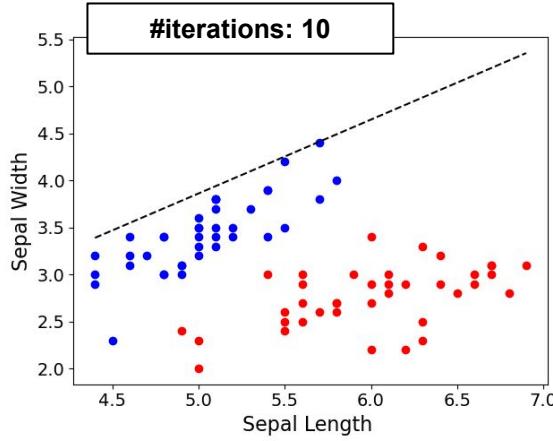
# Logistic Regression — 2D Example

- IRIS dataset

- 3 classes of Iris plants  
(only 2 considered)
- 50 samples per class
- 4 continuous features  
(only sepal length/width considered)



# Logistic Regression — 2D Example (Vanilla Gradient Descent)



# Polynomial Logistic Regression

- Analogous to Polynomial Linear Regression
  - Allows to capture nonlinear relationships between  $X$  and  $y$
  - Polynomial Logistic Regression model for 1 input feature

$$\hat{y}_i = \frac{1}{1 + e^{-\theta^T x_i}}, \quad \text{with } \theta^T x_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \dots + \theta_p x_i^p$$

- Identical practical considerations

- Solve using Gradient Descent as usual  
(optionally with regularization to avoid overfitting)
- Limited to small number of features  
and small polynomial degrees

$$\nabla_{\theta} L = \frac{1}{n} X^T (h_{\theta}(X) - y) + \lambda \frac{2}{n} \theta$$

# Overview

- **Linear Models**
  - Basic setup
- **Linear Regression**
  - Problem formulation
  - Normal Equation (analytical solution)
  - Gradient Descent (iterative optimization)
  - Polynomial Linear Regression (overfitting, regularization)
  - Interpretation of Coefficients
- **Logistic Regression**
  - Problem formulation
  - Gradient Descent

# Summary

- **Linear Models**
  - Assume linear relationship between input features and output  
(i.e., output = sum of weighted feature values)
  - However, regression line / decision boundary not always a line/plane/hyperplane  
(data transformation to add polynomial terms based on input features)
  - Generally good interpretability
- **Linear Regression & Logistic Regression**
  - Very fundamental and popular regression and classification methods
  - Gradient Descent to solve both tasks + Normal Equation to solve Linear Regression
  - Effects of data normalization mainly(!) on explainability / interpretability of results
  - Straightforward extension of Logistic Regression beyond 2 classes (not covered here)

# Solutions to Quick Quizzes

- Slide 13: Yellow (smallest average error)
- Slide 22: D
- Slide 23: A
- Slide 35: "near plateau" → very small gradients and updates
- Slide 48: B
- Slide 57: C

# CS5228: Knowledge Discovery and Data Mining

## Lecture 8 — Recommender Systems

# Course Logistics

- **Deadline Reminders**
  - Submission of A3: Thu Oct 24, 11.59 pm – answer please in English only :)
  - Submission of project report: Nov 14, 11.59 pm
  - Extended TEAMMATES deadline: Sat, Oct 19, 11.59pm
- **Marks upload**
  - A2 + midterm result soon ready

# Quick Recap — Linear Models

- Basic Assumption

- Linear relationship between  $x_i$  and dependent variable  $y_i$

$$\hat{y}_i = h_{\theta}(x_i) = f(\underbrace{\theta_0 x_{i0} + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots + \theta_d x_{id}}_{= 1})$$

Predicted value which is hopefully close to  $y_i$

1, 2, ...,  $d$  input features

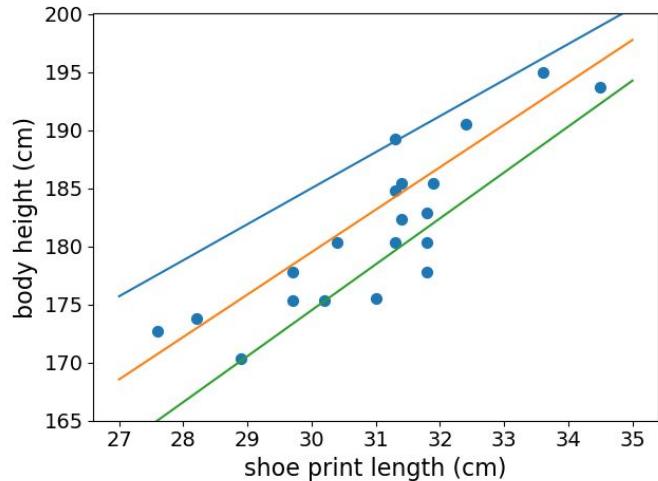
The diagram illustrates the linear model equation  $\hat{y}_i = h_{\theta}(x_i) = f(\theta_0 x_{i0} + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots + \theta_d x_{id})$ . A box labeled "Predicted value which is hopefully close to  $y_i$ " has an upward-pointing arrow from its center to the equation. Another box labeled "1, 2, ...,  $d$  input features" has three arrows pointing to the terms  $\theta_0 x_{i0}$ ,  $\theta_d x_{id}$ , and the constant term 1.

- Learned parameters of model:  $\theta = \{\theta_0, \theta_1, \theta_2, \dots, \theta_d\}$ ,  $\theta_i \in \mathbb{R}$

# Quick Recap — Linear Regression

- Find  $\theta$  that minimizes MSE loss  $L$

$$\begin{aligned} L &= \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \\ &= \frac{1}{n} \|X\theta - y\|^2 \end{aligned}$$



- Solve using Normal Equation

$$\frac{\partial L}{\partial \theta} = \frac{2}{n} X^T (X\theta - y) \rightarrow \frac{2}{n} X^T (X\theta - y) \stackrel{!}{=} \vec{0} \rightarrow \theta = (X^T X)^{-1} X^T y$$

- Solve using Gradient Descent

$$\nabla_{\theta} L = \frac{2}{n} X^T (X\theta - y) \rightarrow \text{repeat: } \theta \leftarrow \theta - (\eta \cdot \nabla_{\theta} L)$$

# Quick Recap — Logistic Regression

- Regression model for classification
  - Interpret  $\hat{y}$  as probability that  $x$  belongs to Class 1

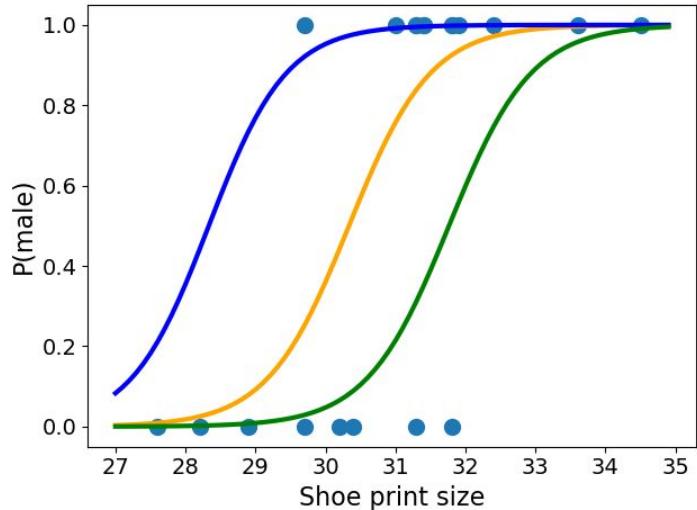
$$\hat{y} = h_{\theta}(x) = f(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

- Minimize **Cross-Entropy Loss L**

$$L = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})]$$

- Solve using **Gradient Descent**

$$\nabla_{\theta} L = \frac{1}{n} X^T (h_{\theta}(X) - y) \quad \rightarrow \quad \text{repeat: } \theta \leftarrow \theta - (\eta \cdot \nabla_{\theta} L)$$



# Quick Recap — Linear Models

- **Polynomial Linear/Logistic Regression**

- Data transformation to include polynomial terms of features
- No change of algorithms needed

$$X^{(1)} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \quad X^{(2)} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \quad X^{(3)} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

- **Regularization to avoid overfitting**

- Extend loss function to "punish" large values of  $\theta$
- Only minor changes to Normal Equation and calculation of Gradient

$$L = \frac{1}{n} \|X\theta - y\|^2 + \lambda \frac{1}{n} \|\theta\|_2^2$$

# Outline

- **Overview**
  - Motivation
  - Naive / alternative approaches
- **Content-based recommendation systems**
  - Pairwise item-item similarity
  - User-item similarity
- **Collaborative filtering (CF)**
  - Memory-based CF
  - Model-based CF

# Recommender Systems — Motivation

- In the online world: information and item overload
  - Too many items: products, songs, movies, news articles, restaurants, etc.
  - More choices require better filters → recommendation engines

| User perspective  | Provider perspective   |
|---|--|
| <ul style="list-style-type: none"><li>• Identify relevant items</li><li>• Minimize effort<br/>(to find relevant items)</li><li>• Maximize satisfaction</li><li>• Optimize spending of<br/>money and attention</li></ul> | <ul style="list-style-type: none"><li>• Maximize sales / transactions</li><li>• Maximize user engagement<br/>(e.g., to maximize ad revenue)</li><li>• Gain competitive advantage</li></ul> |

# Editorial Recommendations

- **Recommendations by "experts"**
  - Expert = person with expertise about item(s)  
(e.g., movie or restaurant critic, staff writers, journalists)
  - Objective, elaborate, trustworthy, credible  
(at least in an ideal world...)
  - Writing editorial recommendations is generally paid work

A Foodie's Guide to the Best Burgers in Singapore

**The best burgers in Singapore**

15 Best Burgers In Singapore So Good, You Won't Stop At Just Bun

Burgers, burgers, burgers: The best to sink your teeth into whether dine-in or delivered

Best Burger Joints You Must Try in Singapore

**Snackdown review: The best burgers in Singapore**

**The best restaurants with burger delivery options in Singapore**

# Peer Recommendations

- **Recommendations by normal users**

- Online word-of-mouth recommendations  
(however, users are typically strangers)
- Common feature on shopping/booking sites
- Typically subjective, short, biased
- Many reviews per item create average view  
but represents again information overload

● ● ● ○○ Reviewed 11 November 2016 □ via mobile

## One of the best burgers in town

Food n drink above average. Service has much room for improvement. Ordered the beef burger which is huge n delicious. However the server failed to check for the preference on the done-ness of the patty.

### VALUE FOR MONEY

Created on 08/22/2020



“ basic bike, not that suitable for actual MTB but can't beat the value for easy bike riding ”



by Anand ✓ Verified Purchase

28 May 2020

The product looks good and very difficult to tilt vertically otherwise it's a worth the money. Due to Circuit Breaker, the shipping took a Long time.

### 5/5 Excellent

#### Verified traveller

Travelled with family, Travelled with group

25 Oct 2019

☺ Liked: Cleanliness, staff & service, amenities, property conditions & facilities

It was too crowded and busy hotel. Otherwise everything was good

### 5/5 ★★★★★ A dream come true.

Reviewed in the United States on August 1, 2019

#### Verified Purchase

A lot of us thought we would never see all these characters in one movie, but these guys did it. And they gave us some of the best movies we know now. What a spectacular journey it was, starting with Iron Man and now here. Very well done everyone who was apart of it, and thank you.

# Manual Recommendations — Pros & Cons

- Pros
  - Semantically rich (ratings, plain text, images, videos, etc.)
  - Explainability / Interpretability
- Cons
  - Manual effort — What is the incentive for writing a review?
  - Lack of personalization

# Recommendation Fraud

Online reviews 'used as blackmail'

'Why I write fake online reviews'

Spotify tests sponsorship of full-screen album recommendations

A new study analyses the murky world of fake Amazon reviews

How merchants use Facebook to flood Amazon with fake reviews

Army of fake reviewers being built to dupe buyers, drive online sales

Influencer Marketing Fraud: The Shady Side of Social Media

Can We Trust Social Media Influencers?

## Buy Lazada Review - Votes - Sells

### How to Get Paid to Write Reviews

Amazon's Fake Review Problem Is Getting Worse

Threatening a business with a bad review is ugly bullying

Amazon is filled with fake reviews and it's getting harder to spot them

# Quick Quiz

How many Amazon reviews for electronic products are "**fake**"?

(according to a study from 2018)

**A**

~10%

**B**

~30%

**C**

~60%

**D**

~90%

# Outline

- **Overview**
  - Motivation
  - **Naive / alternative approaches**
- **Content-based recommendation systems**
  - Pairwise item-item similarity
  - User-item similarity
- **Collaborative filtering (CF)**
  - Memory-based CF
  - Model-based CF

# Recommendations Simple Aggregations

- Rank items based on aggregated scores

**Rotten Tomatoes Top-100 Movies**

**BEST OF RT**

Movies with 40 or more critic reviews vie for their place in history at Rotten Tomatoes. Eligible movies are ranked based on their Adjusted Scores.

| Rank | Rating | Title                                | No. of Reviews |
|------|--------|--------------------------------------|----------------|
| 1.   | 96%    | Black Panther (2018)                 | 516            |
| 2.   | 94%    | Avengers: Endgame (2019)             | 531            |
| 3.   | 93%    | Us (2019)                            | 536            |
| 4.   | 97%    | Toy Story 4 (2019)                   | 445            |
| 5.   | 99%    | Lady Bird (2017)                     | 394            |
| 6.   | 100%   | Citizen Kane (1941)                  | 94             |
| 7.   | 97%    | Mission: Impossible - Fallout (2018) | 430            |
| 8.   | 98%    | The Wizard of Oz (1939)              | 120            |
| 9.   | 96%    | The Irishman (2019)                  | 441            |
| 10.  | 96%    | BlacKkKlansman (2018)                | 438            |

**IMDB Top-250 Movies**

Top Rated Movies  
Top 250 as rated by IMDb Users

SHARE

| Rank & Title   | IMDb Rating | Your Rating | Add |
|--|-------------|-------------|-----|
| 1. The Shawshank Redemption (1994)                           | ★ 9.2       | ☆           | +   |
| 2. The Godfather (1972)                                      | ★ 9.1       | ☆           | +   |
| 3. The Godfather: Part II (1974)                             | ★ 9.0       | ☆           | +   |
| 4. The Dark Knight (2008)                                    | ★ 9.0       | ☆           | +   |
| 5. 12 Angry Men (1957)                                       | ★ 8.9       | ☆           | +   |
| 6. Schindler's List (1993)                                   | ★ 8.9       | ☆           | +   |
| 7. The Lord of the Rings: The Return of the King (2003)      | ★ 8.9       | ☆           | +   |
| 8. Pulp Fiction (1994)                                       | ★ 8.8       | ☆           | +   |
| 9. The Good, the Bad and the Ugly (1966)                     | ★ 8.8       | ☆           | +   |
| 10. The Lord of the Rings: The Fellowship of the Ring (2001) | ★ 8.8       | ☆           | +   |

# Simple Aggregations — Pros & Cons

- Pros

- Relatively easy to compute (typically weighted aggregated based on different factors)
- Typically good/safe recommendations (particularly for new/unknown users)

- Cons

- Requires sufficient number of ratings per items
- High risk of popularity bias; lack of diversity  
("rich get richer" effects, "few get richer" effects)
- Lack of personalization

# Personalized Recommendations

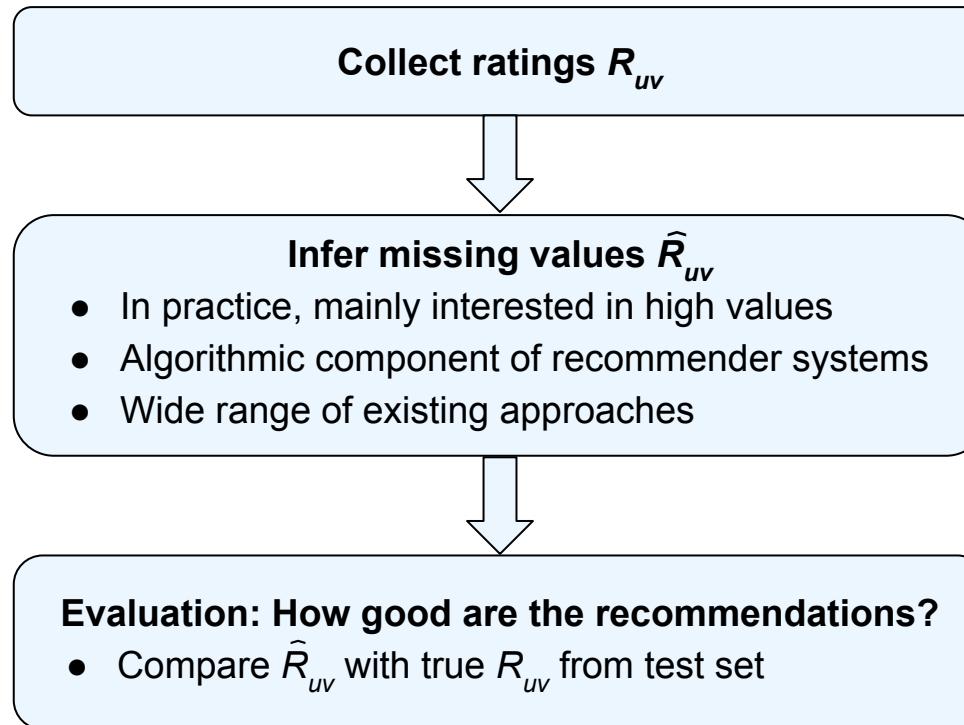
- Users have different preferences that define the relevance of items
  - Preferences = interests, tastes, likings, needs, wants, desires, etc.
  - Relevant items = items that match users' preferences best

- Basic setup

- Set of users  $U = \{u_1, u_2, \dots, u_n\}$
- Set of items  $V = \{v_1, v_2, \dots, v_m\}$
- Rating matrix  $R$  with  
 $|U|$  rows and  $|V|$  columns
- Matrix element  $R_{uv}$ :  $u$ 's rating of  $v$   
(e.g., 1-5 stars, binary 0/1)

|       | movies |   |   |   |   |   |   |
|-------|--------|---|---|---|---|---|---|
| users |        | 2 | 5 |   | 1 |   |   |
|       |        |   | 4 |   | 5 | 5 |   |
|       |        | 4 | 2 |   |   |   |   |
|       | 2      |   | 3 |   | 2 |   | 3 |
|       |        |   |   |   | 4 |   |   |
|       | 5      |   | 5 |   |   | 2 |   |
|       |        |   |   |   |   | 4 | 3 |
|       | 3      |   |   | 1 |   |   |   |
|       |        |   | 2 |   | 2 |   |   |
|       |        | 5 |   | 1 |   |   | 4 |

# Personalized Recommendations — Core Tasks



# Collecting Ratings

- **Explicit**
  - Ask/invite/encourage users to rate items
  - Pay users to rate items (e.g., crowdsourcing)

- **Implicit — derive ratings from users' behavior, e.g.:**

- Product bought
- Video watched
- Article read
- Link clicked
- ...



→ High ratings

(But how to get low ratings?)

**Key challenge: Rating matrix R  
is in practice very sparse!**

# Evaluation

- Split R into training and test set
- Performance metrics
  - Root Mean Squared Error (for numerical ratings)

$$\sqrt{\frac{1}{|S|} \sum_{(u,v) \in S} (\hat{R}_{uv} - R_{uv})^2}$$

Set of  $(u, v)$  pairs in test set

- Precision, Recall, F1 score, etc.  
(TP, TN, FP, FN for binary ratings or binary recommendation after converting numerical ratings)
- Precision@k, Recall@k  
(precision and recall w.r.t. to the top-k highest predicted ratings)
- Compare rankings induced by  $\hat{R}_{uv}$  and  $R_{uv}$  with  $(u, v) \in S$   
(also consider the order of the top-k highest ratings)

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 2 | 5 |   | 1 |   |   |
|   |   | 4 |   | 5 | 5 |   |
|   |   | 4 | 2 |   |   |   |
| 2 |   | 3 |   |   | 2 | 3 |
|   |   |   |   | 4 |   |   |
| 5 |   | 5 |   |   |   | 2 |
|   |   |   |   | 4 | 3 |   |
| 3 |   |   |   | 1 |   |   |
|   |   | 2 |   | 2 |   |   |
|   |   | 5 |   | 1 |   | 4 |

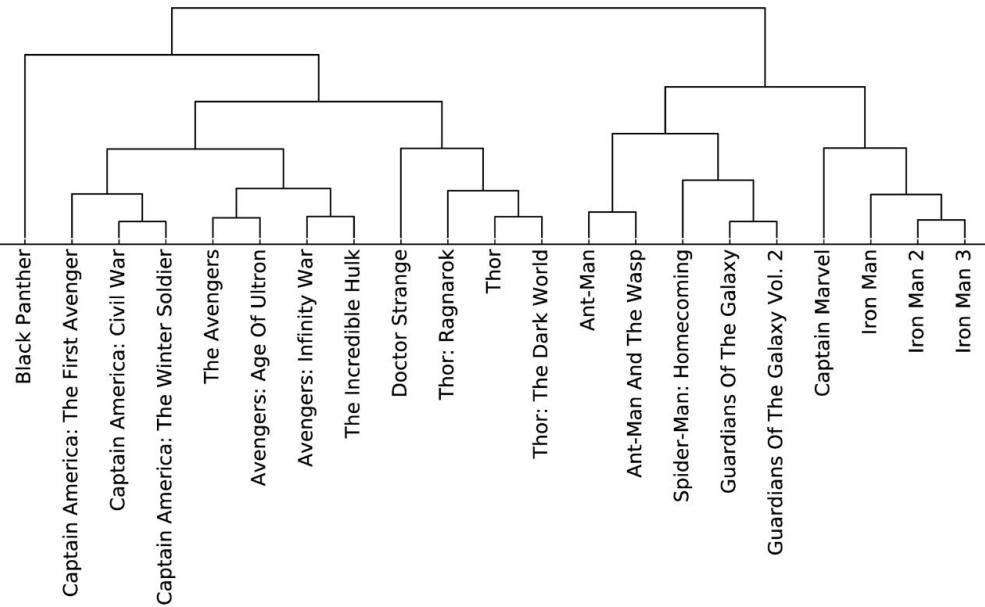
# Recommendations Using Association Rules

- User preferences & likings
  - Items: movies, songs, books, etc.
  - Transaction: viewing/listening/reading history
- Interesting rules (movies):
  - Viewer who watched movies {a, b} also watched movies {x, y}
  - Example: {Jaws} → {It}
- Limitations
  - Basic AR algorithm ignores ratings
  - Popularity bias: user with very unique tastes likely to get subpar recommendations

| TID | Items                          |
|-----|--------------------------------|
| 1   | Jaws, Halloween, Scream, It    |
| 2   | Alien, Jaws, Scream, It        |
| 3   | Tenet, Inception, Interstellar |
| 4   | Jaws, Halloween, It            |
| 5   | Alien, Tenet, Jaws, It         |
| ... |                                |

# Recommendations Using Clustering

Example: Hierarchical clustering of MCU movies



## ● Approach

- Cluster movies based on "useful" features (genre, director, writer, length, ...)
- Recommend movies from clusters with movies a user has rated highly

## ● Limitations

- Find good feature in practice very difficult (we come back to that)
- Unsystematic: no well-defined process to pick recommendations

# Recommendations Using Regression (or Classification)

- Example approach: Linear Regression

- Independent variable: movie features
- Dependent variable: user rating

→ Build a linear regression model for each users

- Limitations

- Requires good features for each item
- Cold-start problem: requires a lot of user ratings to build a good model

Rated movies of an individual user

| original | powerful | absorbing | comical | romantic | ... | Rating |
|----------|----------|-----------|---------|----------|-----|--------|
| 0.20     | 0.95     | 0.80      | 0.00    | 0.10     | ... | 4.5    |
| 0.80     | 0.25     | 0.50      | 0.50    | 0.75     | ... | 4.0    |
| 0.05     | 0.2      | 0.20      | 0.95    | 0.90     | ... | 2.0    |
| 0.60     | 0.20     | 0.80      | 0.00    | 0.85     | ..  | 3.5    |
| 0.90     | 0.95     | 0.90      | 0.10    | 0.40     | ..  | 5.0    |
| 0.45     | 0.50     | 0.20      | 0.60    | 0.30     | ..  | 2.0    |
| 0.10     | 0.40     | 0.55      | 0.90    | 0.30     | ..  | 2.5    |
| 0.75     | 0.50     | 0.50      | 0.40    | 0.40     | ..  | 3.5    |
| 0.80     | 0.80     | 0.85      | 0.10    | 0.10     | ... | 4.5    |
| ...      | ...      | ...       | ...     | ...      | ... | ...    |

# Outline

- Overview
  - Motivation
  - Naive / alternative approaches
- Content-based recommendation systems
  - Pairwise item-item similarity
  - User-item similarity
- Collaborative filtering (CF)
  - Memory-based CF
  - Model-based CF

# Content-Based Recommender System

- Intuition
  - Recommend item  $v$  to user  $u$  that are similar to  $v$  and  $u$  has rated highly
  - Examples: movies of the same genre, songs from the same artist, articles about the same topic, products with similar features, etc.
- Basic requirement: **item profiles** = feature vector for each item, e.g.:
  - Movie: genre, director, writer, cast, length, year, ...
  - Product: type, brand, price, weight, color, ...
  - Article: set of (important) words / tf-idf vector / ...

# Running Example

- MovieLens dataset
  - Items: Movies of different genres
  - Features: 20 genres (incl. "uncategorized")
  - Ratings: 1-5 stars (incl. half stars)
- Numbers for "Small" dataset
  - 610 users, 9,742 movies
  - ~100k ratings → sparsity: ~1.7%

|                 | comedy | action | romance | drama | fantasy | thriller | ... |
|-----------------|--------|--------|---------|-------|---------|----------|-----|
| <i>Clueless</i> | 1      | 0      | 1       | 0     | 0       | 0        | ... |
| <i>Heat</i>     | 0      | 1      | 0       | 0     | 0       | 1        | ... |
| <i>Bad Boys</i> | 1      | 1      | 0       | 1     | 0       | 1        | ... |
| <i>Leon</i>     | 0      | 1      | 0       | 1     | 0       | 1        | ... |
| <i>Alice</i>    | 1      | 0      | 1       | 1     | 1       | 0        | ... |
| <i>Jarhead</i>  | 0      | 1      | 0       | 1     | 0       | 0        | ... |
| <i>Rocky</i>    | 0      | 0      | 0       | 1     | 0       | 0        | ... |
| <i>Big</i>      | 1      | 0      | 1       | 1     | 1       | 0        | ... |
| <i>Krull</i>    | 0      | 1      | 0       | 0     | 1       | 0        | ... |
| ...             | ...    | ...    | ...     | ...   | ...     | ...      | ... |

|       | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ | ... |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-----|
| $u_1$ | 3.0   |       |       |       |       | 4.5   |       |       | ... |
| $u_2$ | 3.5   | 5.0   |       |       |       |       |       | 2.0   | ... |
| $u_3$ |       |       |       | 3.0   |       | 5.0   |       |       | ... |
| $u_4$ |       | 4.5   | 2.0   |       |       |       | 2.0   |       | ... |
| $u_5$ | 3.5   |       | 2.5   |       |       |       |       |       | ... |
| $u_6$ |       |       |       |       | 3.0   |       |       | 3.0   | ... |
| $u_7$ |       |       |       |       | 3.5   | 3.0   |       |       | ... |
| $u_8$ |       |       | 2.0   |       |       |       | 3.0   | 3.0   | ... |
| $u_9$ | 5.0   |       |       |       | 4.5   |       |       |       | ... |
| ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ...   | ... |

Source: [MovieLens Dataset](#)

# Simple Approach — Pairwise Item Similarity

- **Pairwise item similarity  $sim(x,y)$**

- $x, y$  — feature vectors of movies
- Common metric: cosine similarity

$$sim(x, y) = \cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|}$$

- **Limitation: Requires reference item, e.g.:**

- Movie(s) the user was most recently watching
- Movie(s) the user has rated the highest
- Movie(s) the user is currently browsing

$sim(Heat, Heat) = 1.0$

$sim(Heat, Clueless) = 0.0$

$sim(Heat, Bad Boys) = 0.77$

$sim(Heat, Jarhead) = 0.33$

$sim(Heat, Alice) = 0.0$

Similar titles you might also like [What is this?](#)

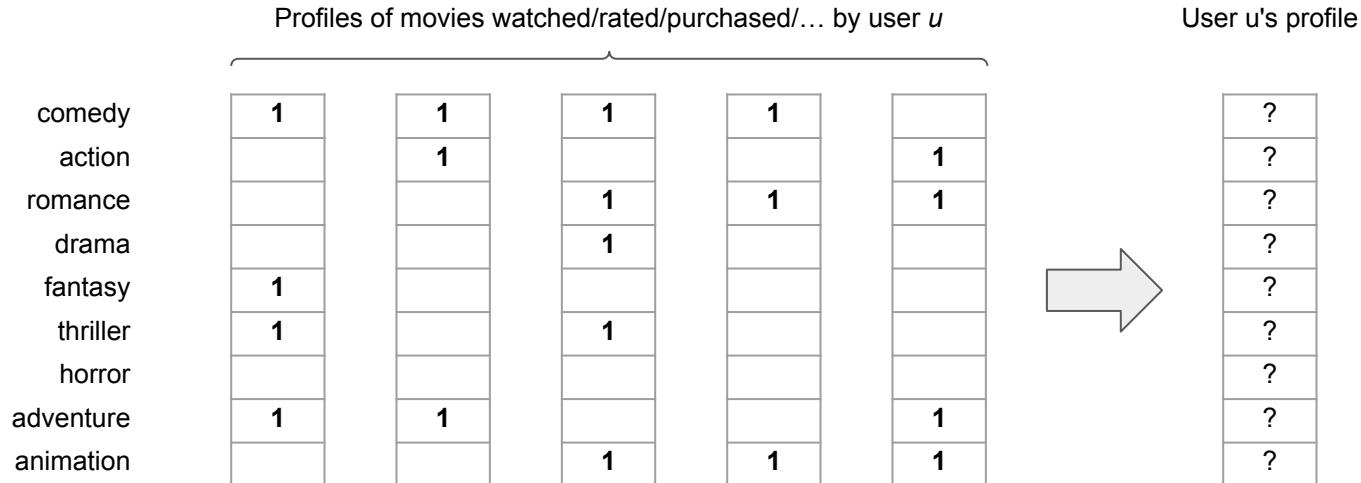


# Outline

- Overview
  - Motivation
  - Naive / alternative approaches
- Content-based recommendation systems
  - Pairwise item-item similarity
  - User-item similarity
- Collaborative filtering (CF)
  - Memory-based CF
  - Model-based CF

# User-Item Similarities

- Needed: **user profiles** = feature vector for each user
  - Requirement: same shape as item profiles to calculate similarities
  - Approach: user profile = "some aggregate" over item profiles rated by the user



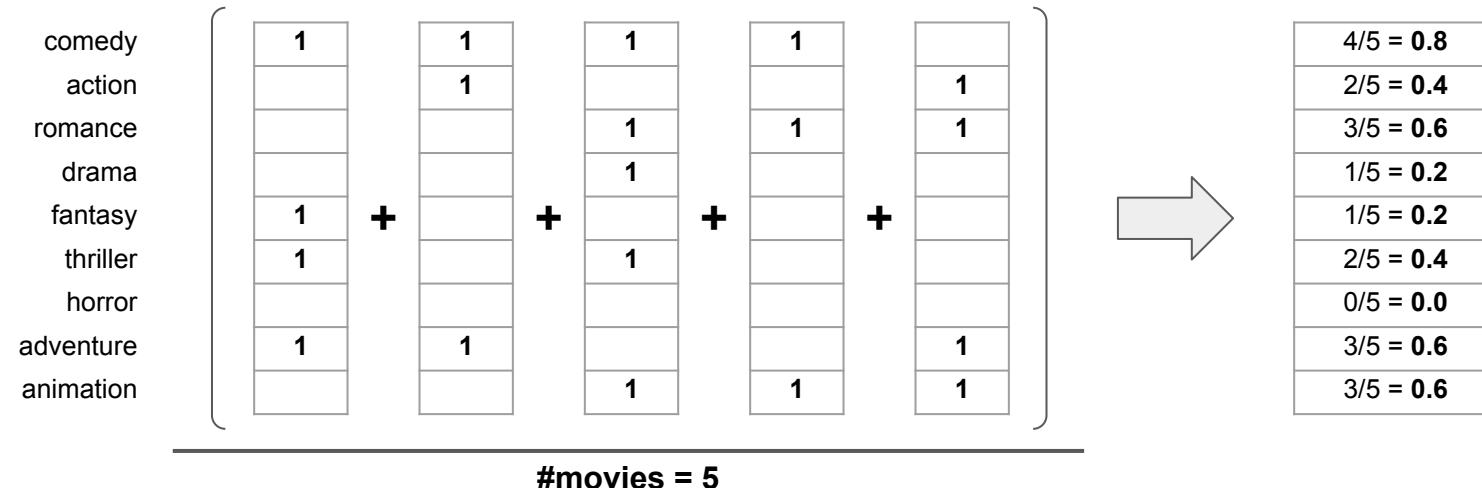
# User-Item Similarities — Binary Utility Matrix

- $R_{uv} \in \{0, 1\}$  — for example,  $R_{uv} = 1$  if

- User u bought movie v
- User u watched movie v

} Implicit rating that  $u$  liked  $v$  (no explicit ratings available here; but also no implicit dislikes!)

- Simple Average



# User-Item Similarities — Real-Valued Utility Matrix

- $R_{uv} \in \mathbb{R}$ — for example,  $R_{uv} \in \{1.0, 1.5, 2.0, 2.5, \dots, 5.0\}$  star rating
  - Explicit rating of user  $u$  for movie  $v$
  - Important: semantic interpretation — ratings express both likes and dislikes (despite all ratings positive)
  - Use rating as weights for features for a weighted aggregation

|                    |            |            |            |            |            |
|--------------------|------------|------------|------------|------------|------------|
| comedy             | 1          | 1          | 1          | 1          |            |
| action             |            | 1          |            |            | 1          |
| romance            |            |            | 1          |            | 1          |
| drama              |            |            | 1          |            |            |
| fantasy            | 1          |            |            |            |            |
| thriller           | 1          |            |            |            |            |
| horror             |            |            | 1          |            |            |
| adventure          | 1          | 1          |            | 1          |            |
| animation          |            |            | 1          |            | 1          |
| <b>u's ratings</b> | <b>1.5</b> | <b>2.0</b> | <b>4.5</b> | <b>5.0</b> | <b>4.0</b> |

## Intuition

- The user likes romantic and animated movies
- The user dislikes fantasy and adventure movies

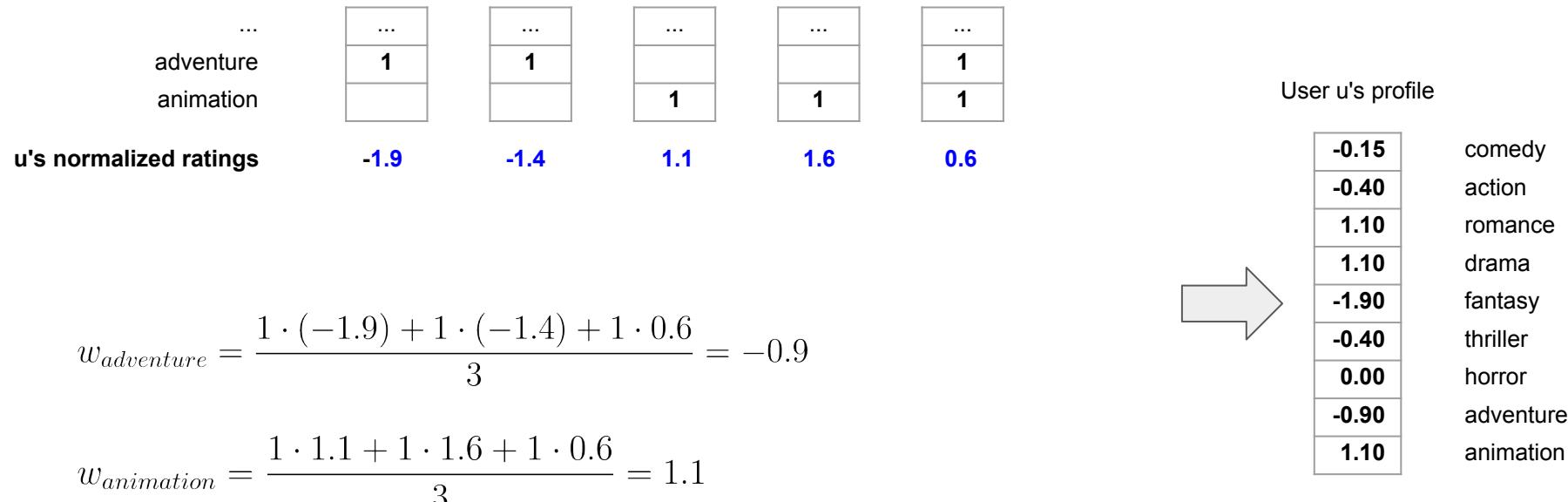
# User-Item Similarities — Real-Valued Utility Matrix

- Step 1: Normalize ratings
  - Subtract average user rating from each movie rating
  - Converts ratings into positive (liked) and negative (disliked) scale
  - Distinguishes "generous" users (mostly rate highly and a 3.0 is a low rating) from more "grumpy" users (mostly rate low and a 3.0 is a high rating)

|                        |      |      |     |     |     |   |
|------------------------|------|------|-----|-----|-----|---|
| comedy                 | 1    | 1    | 1   | 1   | 1   | Average user rating                           |
| action                 |      |      |     |     |     |   |
| romance                |      |      |     |     |     |   |
| drama                  |      |      |     |     |     |   |
| fantasy                | 1    |      |     |     |     | $\frac{1.5 + 2.0 + 4.5 + 5.0 + 4.0}{5} = 3.4$ |
| thriller               | 1    |      |     |     |     |   |
| horror                 |      |      |     |     |     |   |
| adventure              | 1    | 1    |     |     |     |   |
| animation              |      |      | 1   | 1   | 1   |   |
| u's normalized ratings | -1.9 | -1.4 | 1.1 | 1.6 | 0.6 |   |

# User-Item Similarities — Real-Valued Utility Matrix

- Step 2: Calculate weighted features for user profile
  - The weights are the normalized weights



# Quick Quiz

Given the example calculation above what should we **ensure** to get good profiles?

$$w_{adventure} = \frac{1 \cdot (-1.9) + 1 \cdot (-1.4) + 1 \cdot 0.6}{3} = -0.9$$

**A**

A user has given mostly high ratings to different movies

**B**

Enough movies of the same genre have been rated

**C**

A user has rated enough movies of the same genre

**D**

A user's ratings for the same genre need to be diverse

# User-Item Similarities

- Pairwise item similarity  $sim(u, v)$

- $u$  — user profile;  $v$  — item profile
- Suitable metric: cosine similarity (note that user and item profiles can have different magnitudes)

→ Recommend items  $v_i$  to user  $u$  with max. similarities  $sim(u, v_i)$

- Practical considerations

- Top  $k$  most similar items always the same → add some randomization for diversity  
(the set of top  $k$  most similar items might only change over time if the user rates more items)
- Top  $k$  most similar items might include items the user has already rated → remove those items  
(in practice, recommending known items not uncommon — e.g., YouTube recommendations)
- More sophisticated ways to aggregate item profiles to user profiles conceivable  
(for example: ignore underrepresented features, e.g., if a user rated only one comedy movie)

# Content-Based Recommender System — Pros & Cons

- **Pros**

- Recommendations for user  $u$  do not depend on other users  
(this also allows for good recommendations for users with very unique tastes)
- Recommendations can also include new or unpopular items (i.e., with no or very few ratings)
- Good explainability (features that had most effect on the high similarity)

- **Cons**

- Cold-start problem: How to build a profile for new users?  
(naive approach: recommend generally popular items to new users)
- Finding good features (and values!) for items a non-trivial task  
(Question: Are genres a good feature set to represent movies?)
- Overspecialization: By default, no recommendations outside a user's profile  
(in practice: add some randomization into the recommendation process)

# Outline

- **Overview**
  - Motivation
  - Naive / alternative approaches
- **Content-based recommendation systems**
  - Pairwise item-item similarity
  - User-item similarity
- **Collaborative filtering (CF)**
  - **Memory-based CF**
  - Model-based CF

# Collaborative Filtering

- Idea: Utilize the opinions of others
  - Recommend items that other user with similar tastes/preferences/needs have liked
  - Does not require item or user-specific features
- Two perspectives
  - User-based — two users are similar if they rated the same items similarly
  - Item-based — two items are similar if they are equally rated by users

# User-Based CF — Calculating Similarities

- Example: movie ratings

- How much might Bob like the movie "Heat"?

|        | Clueless | Heat | Jarhead | Big | Rocky |
|--------|----------|------|---------|-----|-------|
| Alice  | 2        | 4    | 5       | 0   | 1     |
| Bob    | 1        | ???  | 4       | 0   | 2     |
| Claire | 1        | 0    | 4       | 3   | 0     |
| Dave   | 5        | 1    | 2       | 0   | 5     |
| Erin   | 1        | 5    | 3       | 0   | 3     |

- Intuitions given the dataset

- Alice and Bob have similar tastes, so Bob might rate "*Heat*" similar to Alice
- Claire and Bob have similar tastes, but Claire has not rated "*Heat*"
- Dave and Bob have very different tastes, so Dave's opinion about "*Heat*" shouldn't matter  
(if anything, it should be an indicator that Bob will like "*Heat*"; usually not relevant in practice)

→ How can we capture and quantify these intuitions?

# User-Based CF — Calculating Similarities

- Represent all users by their rating vectors  $v$

- Rows of rating matrix

$$r_A = (2, 4, 5, 0, 1)^T$$

$$r_B = (1, 0, 4, 0, 2)^T$$

$$r_C = (1, 0, 4, 3, 0)^T$$

$$r_D = (5, 1, 2, 0, 5)^T$$

$$r_E = (1, 5, 3, 0, 3)^T$$

Using cosine similarity 

$$\text{sim}(r_A, r_B) = 0.77$$

$$\text{sim}(r_D, r_B) = 0.67$$

Too similar to capture our intuition

|        | Clueless | Heat | Jarhead | Big | Rocky |
|--------|----------|------|---------|-----|-------|
| Alice  | 2        | 4    | 5       | 0   | 1     |
| Bob    | 1        | ???  | 4       | 0   | 2     |
| Claire | 1        | 0    | 4       | 3   | 0     |
| Dave   | 5        | 1    | 2       | 0   | 5     |
| Erin   | 1        | 5    | 3       | 0   | 3     |

## Problem

- Missing values (0) are treated as negative
  - All ratings are positive values
- No explicit notion of dissimilarity  
(only less or more similar)

# User-Based CF — Calculating Similarities

- Idea: Normalize rating vectors
  - Mean-centering — subtract row mean from each rating vector
  - Missing values (0) now represent the average rating
  - Bad ratings (i.e., below average) now represented by negative values

|        | Clueless              | Heat                  | Jarhead               | Big                  | Rocky                 |
|--------|-----------------------|-----------------------|-----------------------|----------------------|-----------------------|
| Alice  | 2-3 = <b>-1</b>       | 4-3 = <b>1</b>        | 5-3 = <b>2</b>        | <b>0</b>             | 1-3 = <b>-2</b>       |
| Bob    | 1-2.33 = <b>-1.33</b> | <b>???</b>            | 4-2.33 = <b>1.67</b>  | <b>0</b>             | 2-2.33 = <b>-0.33</b> |
| Claire | 1-2.67 = <b>-1.67</b> | <b>0</b>              | 4-2.67 = <b>1.33</b>  | 3-2.67 = <b>0.33</b> | <b>0</b>              |
| Dave   | 5-3.25 = <b>1.75</b>  | 1-3.25 = <b>-2.25</b> | 2-3.25 = <b>-1.25</b> | <b>0</b>             | 5-3.25 = <b>1.75</b>  |
| Erin   | 1-3 = <b>-2</b>       | 5-3 = <b>2</b>        | 3-3 = <b>0</b>        | <b>0</b>             | 3-3 = <b>0</b>        |


$$\text{sim}(r_A, r_B) = 0.78$$
$$\text{sim}(r_D, r_B) = -0.65$$

Cosine similarity between mean-centered vectors → Pearson Correlation Coefficient

# User-Based CF — Predicting Ratings

- $\hat{R}_{uv}$  = weighted average of ratings from similar users
  - $N$  — set of  $k$  users most similar to  $u$  who have already rated item  $v$

$$\hat{R}_{uv} = \frac{\sum_{w \in N} sim(u, w) \cdot R_{wv}}{\sum_{w \in N} sim(u, w)}$$

- For the example

- With  $k = 2$

$$\hat{R}_{Bob,Heat} = \frac{0.78 \cdot 4 + 0.44 \cdot 5}{0.78 + 0.44} = 4.3$$

Alice                            Erin

|        | Clueless | Heat | Jarhead | Big | Rocky |
|--------|----------|------|---------|-----|-------|
| Alice  | 2        | 4    | 5       | 0   | 1     |
| Bob    | 1        | ???  | 4       | 0   | 2     |
| Claire | 1        | 0    | 4       | 3   | 0     |
| Dave   | 5        | 1    | 2       | 0   | 5     |
| Erin   | 1        | 5    | 3       | 0   | 3     |

# Item-Based CF

- Analog to user-based approach
  - For an item  $v$ , find the most similar items  
(2 items are similar, if their ratings across all users are similar)
  - $\hat{R}_{uv} = \text{weighted average of ratings of similar items}$
  - $M$  — set of  $k$  items most similar to  $v$  who have already been rated by  $u$

$$\hat{R}_{uv} = \frac{\sum_{i \in M} sim(i, v) \cdot R_{ui}}{\sum_{i \in M} sim(i, v)}$$

**Note:** Recall that in content-based recommender systems, measuring the similarity between items relied on item profiles / feature vectors. In case of item-item CF, the rating vector of an item represents its profile.

# Item-Based CF — Example

Calculate mean-centered rating vectors  
(here: columns of rating matrix)

$$v_C = (0, -1, -1, 3, -1)^T$$

$$v_H = (0.67, 0, 0, -2.33, 1.67)^T$$

$$v_J = (1.4, 0.4, 0.4, -1.6, -0.6)^T$$

$$v_B = (0, 0, 0, 0, 0)^T$$

$$v_R = (-1.75, -0.75, 0, 2.25, 0.25)^T$$

|        | Clueless | Heat | Jarhead | Big | Rocky |
|--------|----------|------|---------|-----|-------|
| Alice  | 2        | 4    | 5       | 0   | 1     |
| Bob    | 1        | ???  | 4       | 0   | 2     |
| Claire | 1        | 0    | 4       | 3   | 0     |
| Dave   | 5        | 1    | 2       | 0   | 5     |
| Erin   | 1        | 5    | 3       | 0   | 3     |

Calculate distances between "Heat" and all other movies Bob has already rated

$$\text{sim}(v_H, v_C) = -0.85$$

$$\text{sim}(v_H, v_J) = 0.55$$

$$\text{sim}(v_H, v_R) = -0.69$$

Even for k=2, there is only 1 movie with a positive Pearson correlation coefficient



$$\widehat{R}_{uv} = \frac{0.55 \cdot 4}{0.55} = 4$$

Jarhead

# Collaborative Filtering — User-Based vs. Item-Based

- In theory: user-based and item-based are dual approaches
  - In practice: item-based typically outperforms user-based
    - Items are "simpler" than users
    - Items can be more easily described
    - User can have very varied tastes
- 
- Item-item similarity typically more meaningful

# Outline

- **Overview**
  - Motivation
  - Naive / alternative approaches
- **Content-based recommendation systems**
  - Pairwise item-item similarity
  - User-item similarity
- **Collaborative filtering (CF)**
  - Memory-based CF
  - Model-based CF

# Model-Based Collaborative Filtering

- Latent factor models

- Latent representation:  $k$ -dimensional vector for each user  $u$  and item  $v$
- Learn latent representations from the data  
(in contrast to content-based systems where feature vectors are constructed)
- Estimate unknown ratings  $\hat{R}_{uv} = w_u^T h_v$

- Approach: Matrix Factorization

- Put all user vectors into a matrix  $W$
- Put all item vectors into a matrix  $H$

→ Find  $W, H$  such that  $R = WH$

or at least approximately

# Matrix Factorization — Basic Setup

- Given: ratings matrix  $R$

- $m$  — number of users  $|W|$
- $n$  — number of items  $|H|$

$$\left[ \begin{array}{c} R \\ m \times n \end{array} \right] = \left[ \begin{array}{c} W \\ m \times k \end{array} \right] \times \left[ \begin{array}{c} H \\ k \times n \end{array} \right]$$

- Hyperparameter  $k$

- Size of latent representations

# Finding Matrices W, H

- Minimize loss function

$$L = \sum_{R_{uv} > 0} e_{uv} = \sum_{R_{uv} > 0} (R_{uv} - \hat{R}_{uv})^2 = \sum_{R_{uv} > 0} (R_{uv} - w_u^T h_v)^2$$

→ with regularization:

$$L = \sum_{R_{uv}} (R_{uv} - w_u^T h_v)^2 + \lambda(\|w_u\|^2 + \|h_v\|^2)$$

- Using Gradient Descent

Calculate gradients

$$\frac{\partial e_{uv}}{\partial w_u} = -2(R_{uv} - w_u^T h_v)h_v + 2\lambda w_u$$

$$\frac{\partial e_{uv}}{\partial h_v} = -2(R_{uv} - w_u^T h_v)w_u + 2\lambda h_v$$



Update rules

$$w_u \leftarrow w_u - \eta \frac{\partial e_{uv}}{\partial w_u}$$

$$h_v \leftarrow h_v - \eta \frac{\partial e_{uv}}{\partial h_v}$$

# Finding Matrices $W, H$ — Algorithm

**Input** : rating matrix  $R^{(m \times n)}$ , latent vector size  $k$ , #iterations  $T$

**Initialization** :  $W^{(m \times k)}$ ,  $H^{(k \times n)}$  with values 0..1

**for** 1 **to**  $T$

**for all**  $u, v$  **with**  $R_{uv} > 0$

$$w_u \leftarrow w_u + \eta [2(R_{uv} - w_u^T h_v)h_v - 2\lambda w_u]$$

$$h_v \leftarrow h_v + \eta [2(R_{uv} - w_u^T h_v)w_u - 2\lambda h_h]$$

**return**  $W, H$

# Finding Matrices W, H — Example

|                | v <sub>1</sub> | v <sub>2</sub> | v <sub>3</sub> | v <sub>4</sub> | v <sub>5</sub> | v <sub>6</sub> | v <sub>7</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| u <sub>1</sub> | 4              | 0              | 0              | 5              | 1              | 0              | 0              |
| u <sub>2</sub> | 5              | 5              | 4              | 0              | 0              | 0              | 0              |
| u <sub>3</sub> | 0              | 0              | 0              | 2              | 4              | 5              | 0              |
| u <sub>4</sub> | 0              | 3              | 0              | 0              | 0              | 0              | 3              |

Calculate W, H



k = 100

$\lambda = 0.1$

#iterations = 10k

W · H

|                | v <sub>1</sub> | v <sub>2</sub> | v <sub>3</sub> | v <sub>4</sub> | v <sub>5</sub> | v <sub>6</sub> | v <sub>7</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| u <sub>1</sub> | 3.9            | 3.5            | 3.4            | 4.8            | 1              | 2.4            | 3.1            |
| u <sub>2</sub> | 4.9            | 4.8            | 4              | 3.7            | 3.2            | 5.3            | 4.4            |
| u <sub>3</sub> | 3.4            | 3.3            | 3.6            | 2              | 3.8            | 4.9            | 3.9            |
| u <sub>4</sub> | 3.1            | 2.9            | 3.2            | 2.5            | 2.3            | 3.8            | 3              |

- Effects of regularization

- Increase  $\lambda$ : worse fit of known ratings, "smoother" values for all ratings
- Decrease  $\lambda$ : better fit of known ratings, more "extreme" values of unknown ratings

# Collaborative Filtering — Pros & Cons

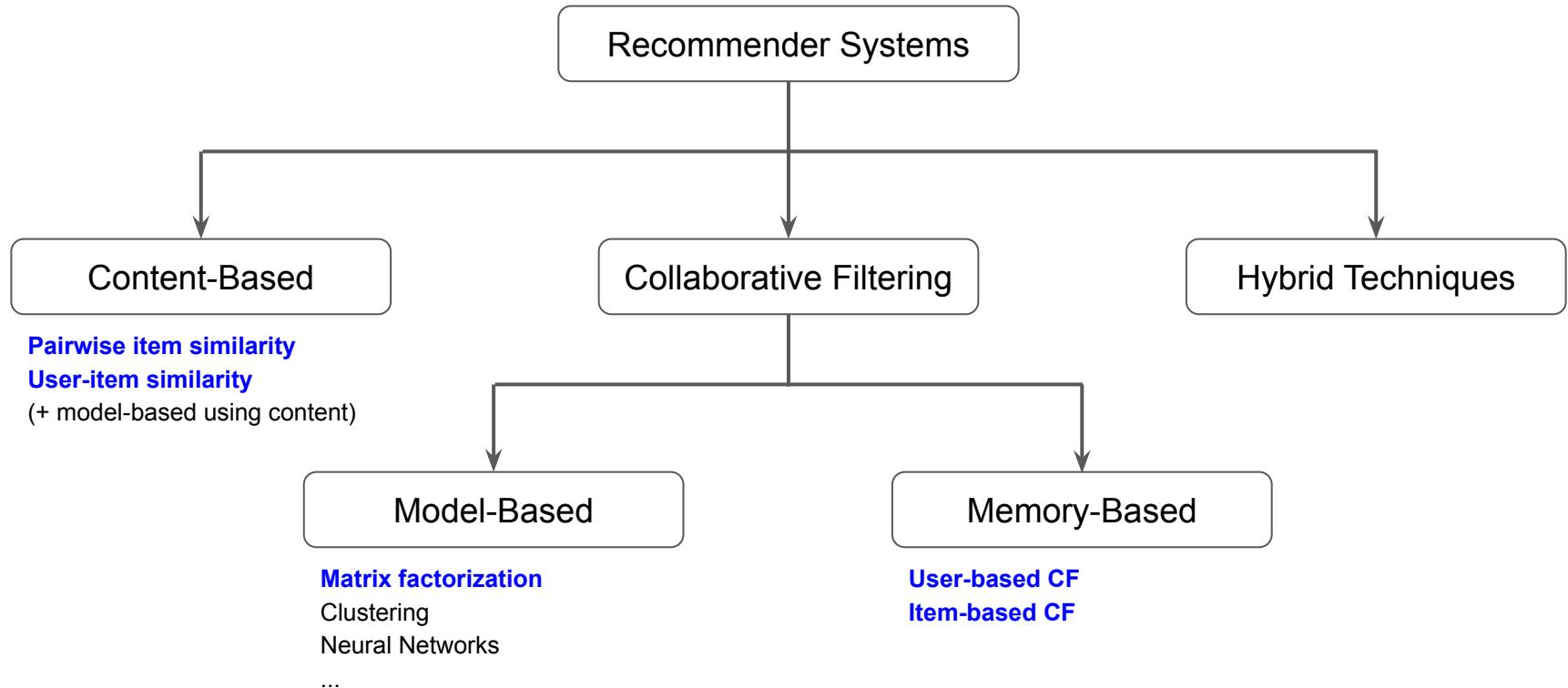
- Pros

- No need to find and create good features  
(such as genres for movies)
- Intuitive approach

- Cons

- Similarity calculations rely on sufficient number of ratings
- Cold-start problem in case of new users or items
- Popularity bias: user with very unique tastes likely to get subpar recommendations  
(because even the  $k$  most similar users will not be truly very similar)
- Naive implementation very expensive: Finding  $k$  most similar users/items  $\in O(|R|)$   
(optimization techniques needed: e.g. clustering of users/items to limit search space)

# Recommender Systems — Summary



# Quick Quiz

What does **not** affect the recommendations made by Collaborative Filtering?

**A**

A new movie is added

**B**

A new user is added

**C**

A user rates a movie

**D**

A movie's details get updated

# Quick Quiz

What does generally affect the recommendations made by Collaborative Filtering the **least**?

**A**

An old user has rated an old movie

**B**

An old user has rated a new movie

**C**

A new user has rated an old movie

**D**

A new user has rated a new movie

# The Dangers of (Over-)Personalization

- 2 infamous side-effects

(particularly when recommending news or social media posts)

- Filter bubbles
- Echo chambers

- Core problems

- No incentive for service providers to ensure (sufficient) diversity
- Users do not know what content is shown and why (or why not!)

**Why is TikTok creating filter bubbles based on your race?**

**How social media filter bubbles and algorithms influence the election**

**When Algorithms Decide Whose Voices Will Be Heard**

*Social Media Giants Support Racial Justice. Their Products Undermine It.*

**Facebook reportedly ignored its own research showing algorithms divided users**

# Outline

- Overview
  - Motivation
  - Naive / alternative approaches
- Content-based recommendation systems
  - Pairwise item-item similarity
  - User-item similarity
- Collaborative filtering (CF)
  - Memory-based CF
  - Model-based CF

# Summary

- **Recommender systems**
  - More specifically: personalized recommender systems
  - Integral component of many online platforms
  - User: find relevant items + providers: present relevant items
  - BUT: risk of over-personalized recommendations
- **Implementing recommender systems**
  - Wide range of data mining techniques applicable
  - No "one-size-fits-all" solutions
  - In practice, hybrid approaches most successful

# Solutions to Quick Quizzes

- Slide 13: C
- Slide 34: C
- Slide 54: D
- Slide 55: A

# **CS5228: Knowledge Discovery and Data Mining**

## Lecture 9 — Graph Mining

# Course Logistics

- **Deadline Reminders**
  - Submission of A4: Nov 14, 11.59 pm
  - Submission of project report: Nov 14, 11.59 pm
- **Project**
  - Check your TEAMMATES result

# Quick Recap

- **Recommender Systems**

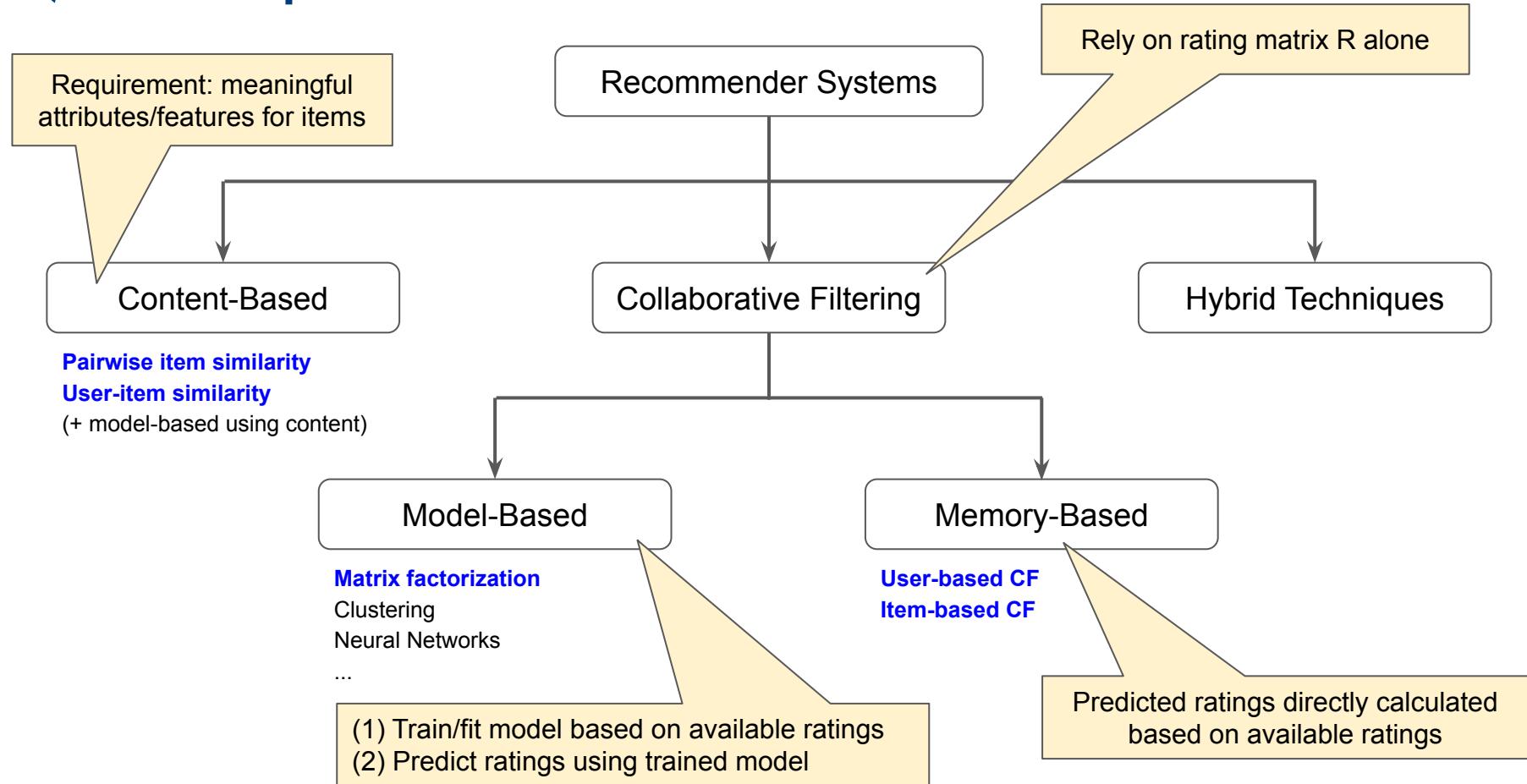
- Problem: information overload (too many items: products, songs, movies, news articles, restaurants, etc.)
- Goal: identify relevant items matching a user's preferences

- **Basic setup**

- Set of users  $U$ , set of items  $V$
- Rating matrix  $R$  with  $|U|$  rows and  $|V|$  columns
- Matrix element  $R_{uv}$ :  $u$ 's rating of  $v$  (e.g., 1-5 stars, binary 0/1)
- If available: information (features) about each item

|       | items |   |   |   |   |   |   |
|-------|-------|---|---|---|---|---|---|
| users |       | 2 | 5 |   | 1 |   |   |
|       |       |   | 4 |   | 5 | 5 |   |
|       |       |   | 4 | 2 |   |   |   |
|       | 2     |   | 3 |   | 2 |   | 3 |
|       |       |   |   |   | 4 |   |   |
|       | 5     |   | 5 |   |   | 2 |   |
|       |       |   |   |   |   | 4 | 3 |
|       | 3     |   |   |   | 1 |   |   |
|       |       |   | 2 |   | 2 |   |   |
|       |       |   | 5 |   | 1 |   | 4 |

# Quick Recap



# Outline

- **Graph Mining**
  - Application Examples
  - Basic definitions
- **Community Detection**
  - Basic definition & goals
  - Overview to different algorithms
- **Centrality**
  - Basic definition & goals
  - Overview to different algorithms
- **Summary**

# Graphs are Everywhere

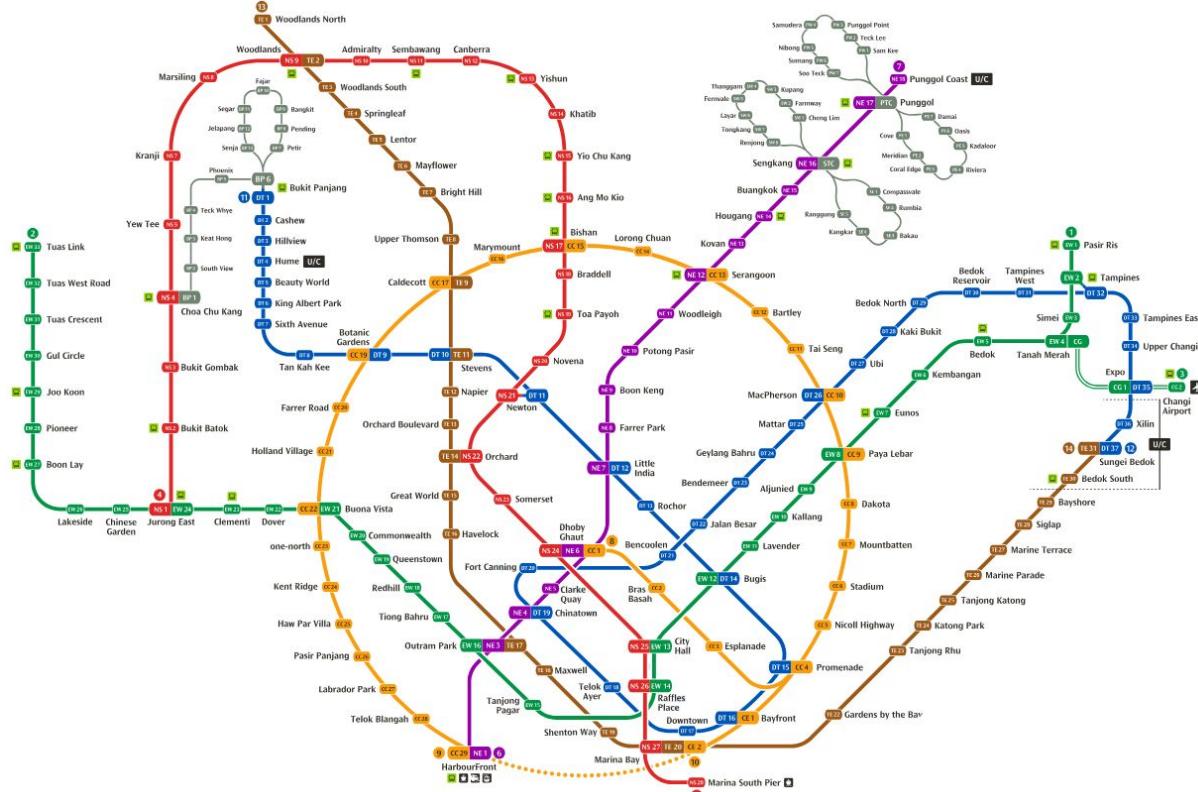
## Transportation Networks

**Nodes:** MRT stations

**Edges:** Direct train connections between stations

### Applications:

- Find shortest travels
- Find busy stations
- Plan bus routes



# Graphs are Everywhere

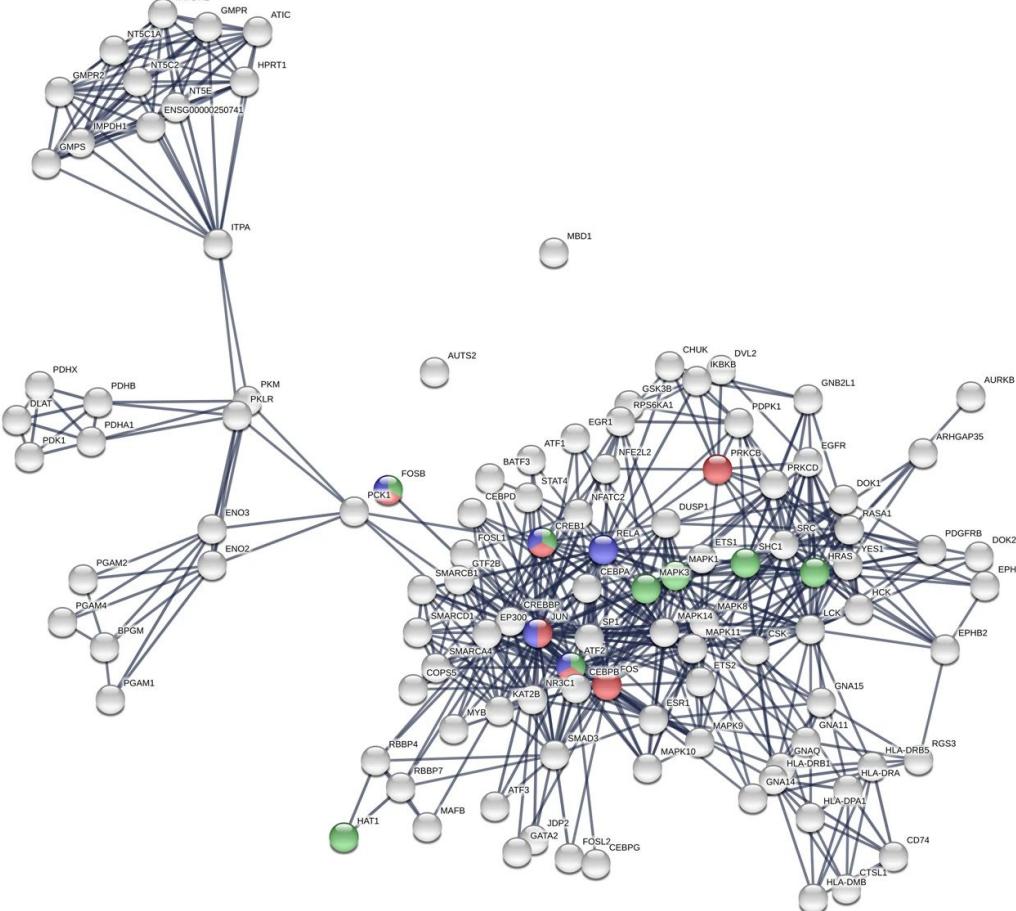
# Protein-Protein Interactions

## Nodes: Proteins

**Edges:** Physical interactions between two proteins

## **Applications:**

- Understanding of diseases  
(disease prognosis, disease susceptibility)
  - Drug discovery



# Graphs are Everywhere

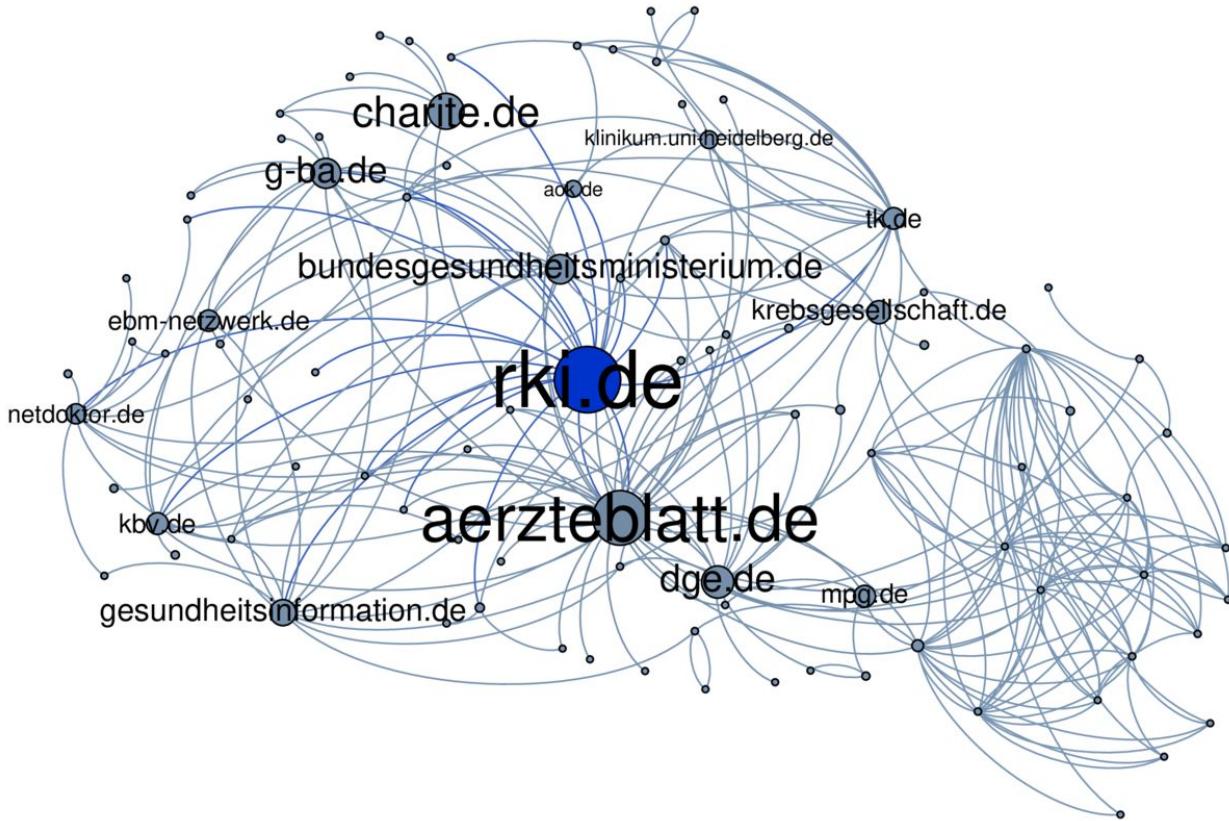
## Web Graph

**Nodes:** Web pages

**Edges:** Hyperlinks

### Applications:

- Identify authoritative sites
- Effective Web search



# Graphs are Everywhere

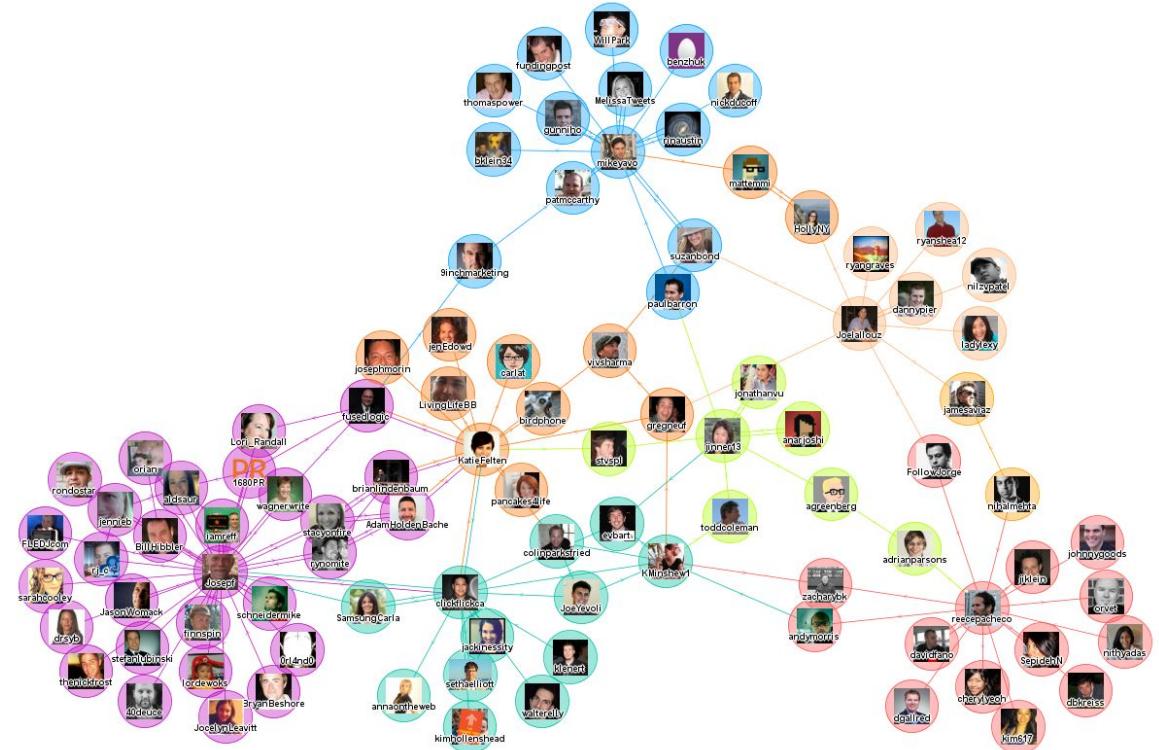
## Social Networks

**Nodes:** People, users

**Edges:** friendship relationships

### Applications:

- Identify communities
- Find influential people
- Friendship recommendations
- Effective Information diffusion  
(e.g., advertising, political campaigns)



# Outline

- **Graph Mining**
  - Application Examples
  - **Basic definitions**
- **Community Detection**
  - Basic definition & goals
  - Overview to different algorithms
- **Centrality**
  - Basic definition & goals
  - Overview to different algorithms
- **Summary**

# Graphs — Mathematical Definition

- Graph: Formalism for representing **relationships** between **items**

- A graph is a tuple  $G = (V, E)$

- Set of vertices (or nodes)  $V$

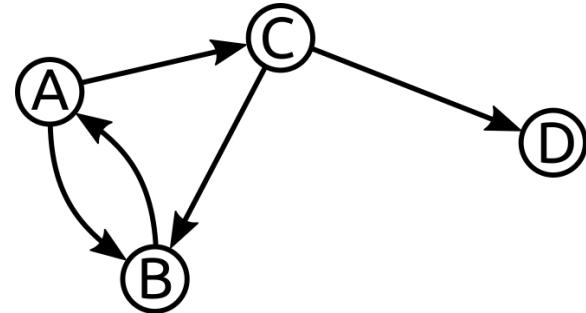
$$V = \{v_1, v_2, \dots, v_n\}$$

- Set of edges  $E$

$$E = \{e_1, e_2, \dots, e_m\}$$

where an edge is a pair of vertices

$$e_i = (v_j, v_k)$$



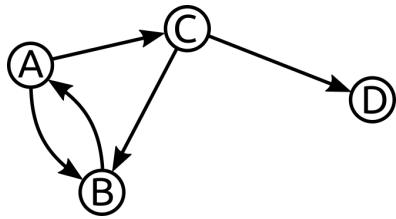
$$V = \{A, B, C, D\}$$

$$E = \{(A, B), (A, C), (C, D), (B, A), (C, B)\}$$

# Types of Graphs

|            | undirected   | directed  |
|------------|--|---|
| unweighted | <p>friendships on Facebook<br/>directly connected MRT stations</p>                 | <p>hyperlinks between web pages<br/>followers on Twitter</p>                |
| weighted   | <p>co-authorship with number of papers<br/>transport network with travel times</p> | <p>#retweets between Twitter users<br/>borrower network with money owed</p> |

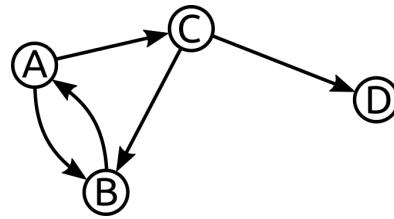
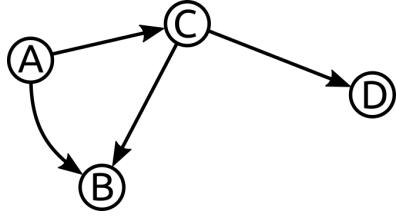
# Types of Graphs



Cyclic Graph

vs.

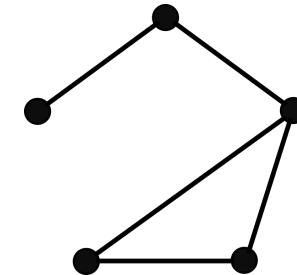
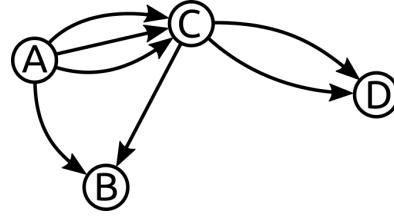
Acyclic Graph



(Simple) Graph

vs.

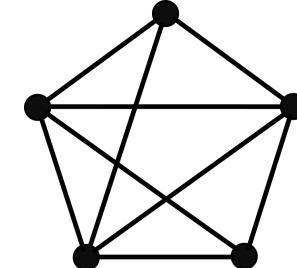
Multigraph



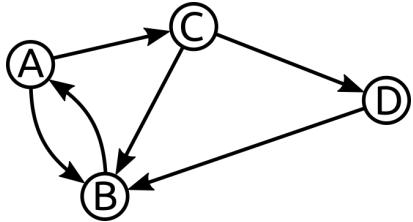
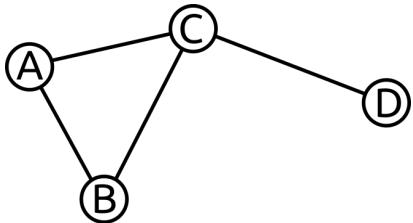
Sparse Graph

vs.

Dense Graph

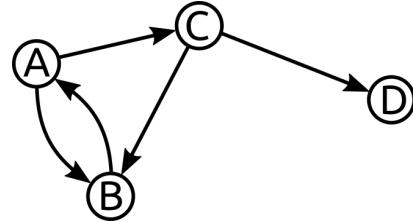


# Types of Graphs



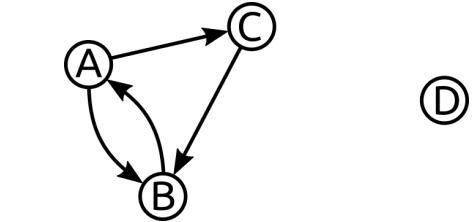
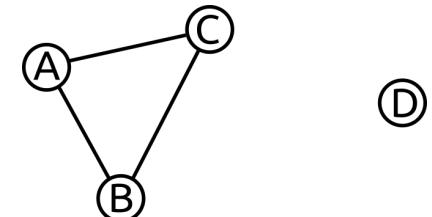
## (Strongly) Connected Graph

There exists a path from each node to every other node



## Weakly Connected Graph

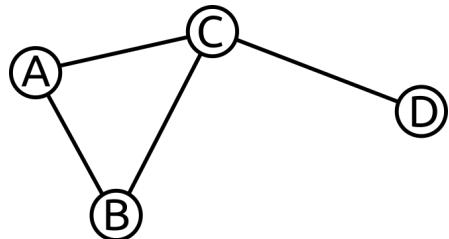
A directed graph where the underlying undirected graph is (strongly) connected



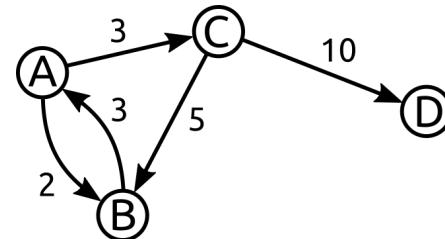
## Disconnected Graph

A graph that is not connected

# Representing Graphs — Adjacency Matrix



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$A = \begin{bmatrix} 0 & 2 & 3 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 5 & 0 & 10 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

# Quick Quiz

When representing the **MRT** network as a graph  $G$  which **type of graph** will  $G$  definitely be?

A

Connected

B

Undirected

C

Weighted

D

Unweighted

# Quick Quiz

Which type of graph is **not suitable** to represent the SBS bus network? Why?

A

Sparse

B

Undirected

C

Weighted

D

Unweighted

# Overview

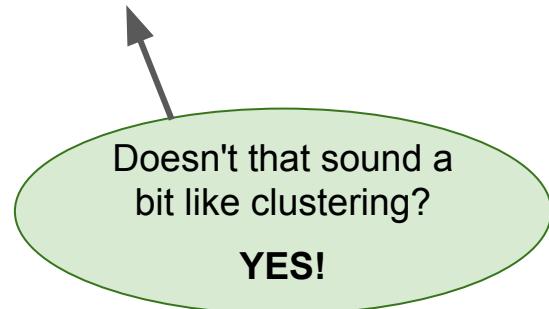
- **Graph Mining**
  - Application Examples
  - Basic definitions
- **Community Detection**
  - Basic definition & goals
  - Overview to different algorithms
- **Centrality**
  - Basic definition & goals
  - Overview to different algorithms
- **Summary**

# Community Detection

- No formal definition of a community
  - From "Networks: An Introduction" (Mark Newman):

*"Loosely stated, [community detection] is the problem of finding the natural divisions of a network into groups of vertices such that there are many edges within groups and few edges between groups. What exactly we mean by "many" or "few," however, is debatable, [...]"*

- Wide range of applications
  - Identifying groups in social networks
  - Recommendation systems
  - Market segmentation
  - Outlier/anomaly detection



# Community Detection — Modularity

- Modularity  $Q \in [-1/2, 1]$  of an undirected graph  $G$  with adjacency matrix  $A$ 
  - Measures the relative density of edges inside communities with respect to edges outside communities
  - Optimizing modularity is NP-hard → Practical algorithms based on heuristics

$$Q = \frac{1}{2m} \sum_{vw} \left[ A[v, w] - \frac{k_v k_w}{2m} \right] \underbrace{\delta(c_v, c_w)}_{\delta(c_v, c_w) = \begin{cases} 1 & , c_v = c_w \\ 0 & , \text{otherwise} \end{cases}}$$

Diagram illustrating the components of the modularity formula:

- $A[v, w]$  — weight of edge between nodes  $v$  and  $w$
- $k_i$  — sum of the weights of edges attached to node  $i$
- $c_i$  — community of a node  $i$
- $m$  — sum of weights of all edges

# Community Detection — Louvain Algorithm

- Method for the optimization of modularity

**Initialization:** Each node is a community

**Repeat** until no further change

## Phase 1: Modularity Optimization

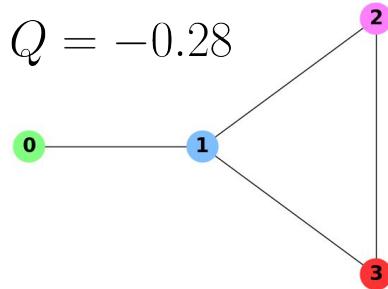
- For each node  $v$ , check if moving it to an adjacent community improves modularity
- Move  $v$  to community that maximizes modularity

## Phase 2: Graph Aggregation

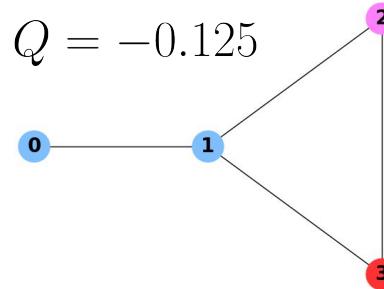
- Represent each community as a new node
- Update weights between new nodes

# Louvain Algorithm — Modularity Optimization

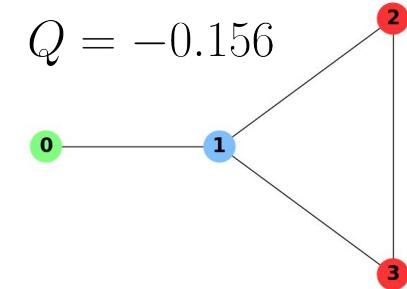
Input: Each node a community



Moving community 0 to community 1

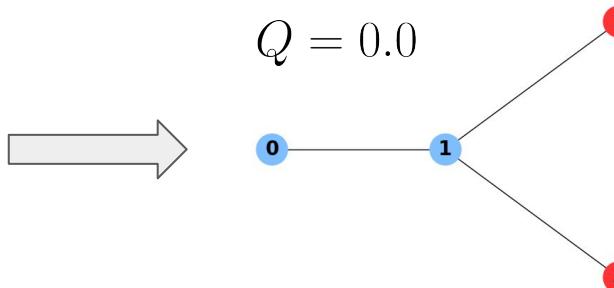


Moving community 2 to community 3



→ Move community 0 to community 1 as it maximizes modularity

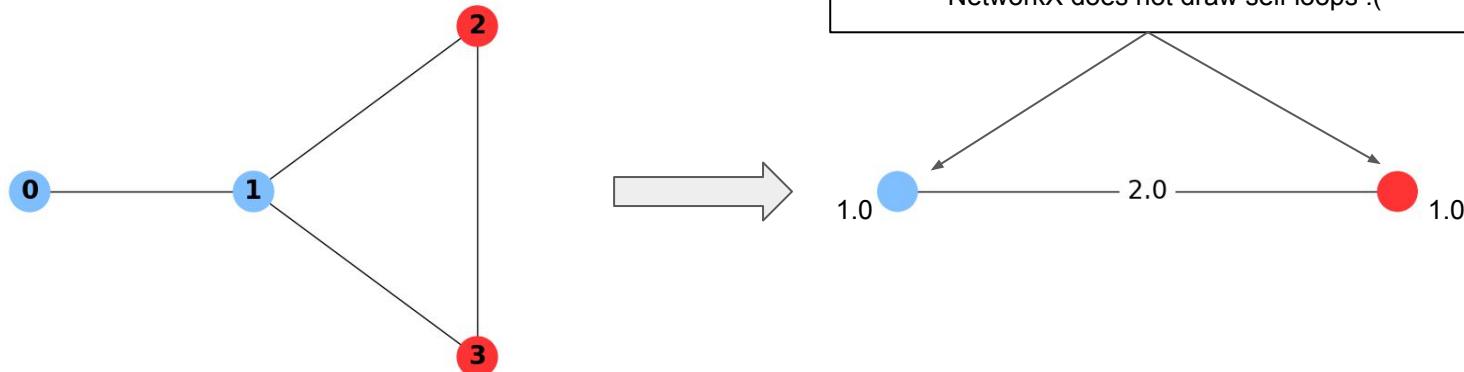
Output after  
all iterations



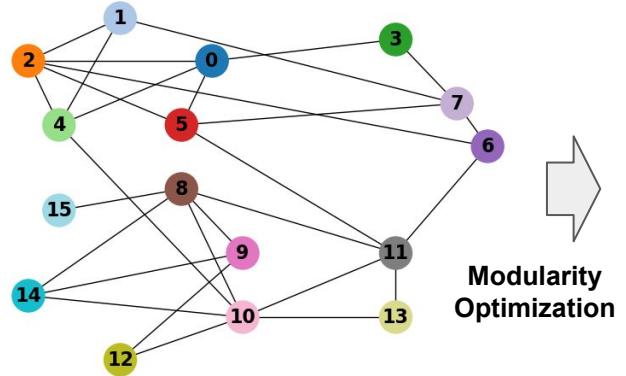
Note: Moving a node may not be permanent but can change again in later iteration before convergence.

# Louvain Algorithm — Graph Aggregation

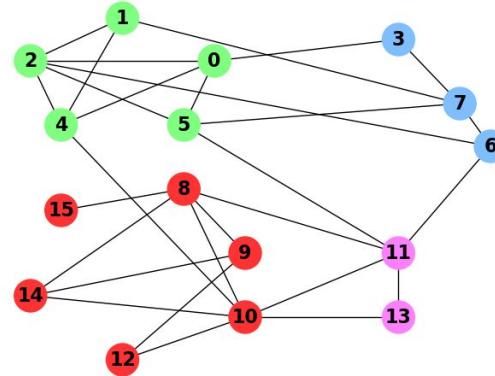
- 2 main steps
  - Merge all nodes of a community into a new single node
  - Aggregate edge weights accordingly



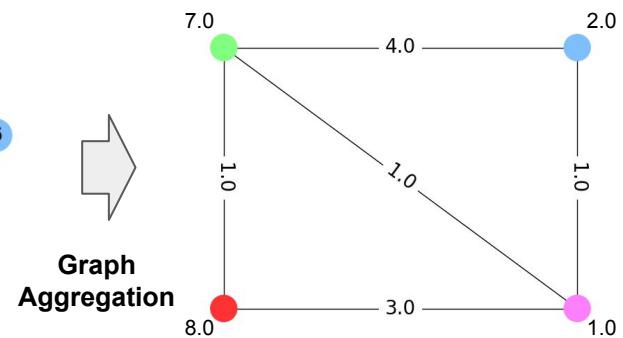
# Louvain Algorithm — Full Example



Modularity Optimization



Modularity Optimization



Graph Aggregation

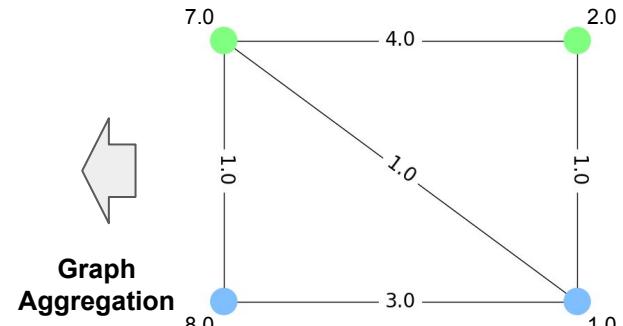
Modularity Optimization

No change in  
communities

→ Done!



Modularity  
Optimization



Graph  
Aggregation

# Louvain Algorithm — Remarks

- **Heuristic**
  - Optimizes modularity locally on all nodes
  - No guarantees for optimal modularity globally
    - (in practice often superior to other methods)
- **Performance optimization**
  - Phase 1 (Modularity Optimization) requires to calculate change in modularity  $\Delta Q$ 
    - (difference between modularity of G before and after moving communities)
  - Calculating  $\Delta Q$  can be done based on local changes in community assignments
    - (does not require the recalculate the modularity G after each change)

Full details in paper: [Fast Unfolding of Communities in Large Networks](#) (Blondel et al., 2008)

# Community Detection — Girvan-Newman Algorithm

- **Divisive hierarchical approach**
  - Start with whole graph representing a community
  - Iteratively remove edges until community is split into 2 sub-communities  
(continue splitting sub-communities recursively if needed)
- Criteria for removing edges: **Edge Betweenness Centrality**
  - Sum of fraction of all-pairs shortest paths that pass through an edge e

$$c_B(e) = \sum_{v,w \in V} \frac{\sigma(v, w|e)}{\sigma(v, w)}$$

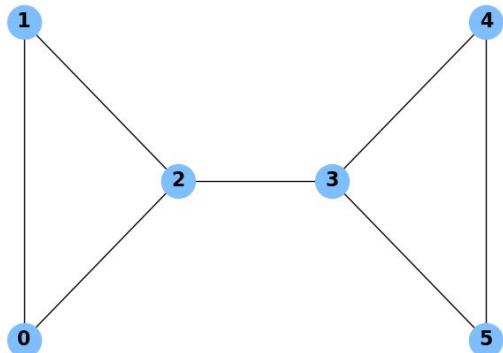
number of shortest paths from v to w going through edge e

number of shortest paths from v to w

Full details in paper: [On Variants of Shortest-Path Betweenness Centrality and their Generic Computation](#) (Brandes et al., 2008)

# Girvan-Newman Algorithm — Edge Betweenness Centrality

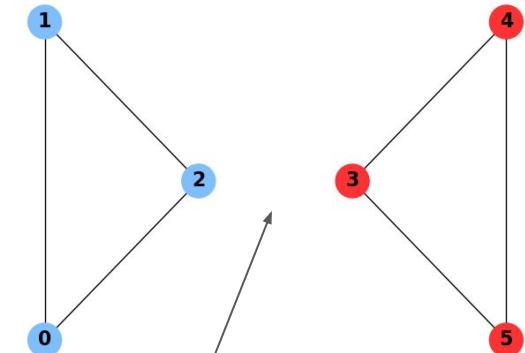
Input community



Betweenness Centrality  
for all edges

| $e$           | $c_B(e)$     |
|---------------|--------------|
| (0, 1)        | 0.067        |
| (0, 2)        | 0.267        |
| (1, 2)        | 0.267        |
| <b>(2, 3)</b> | <b>0.600</b> |
| (3, 4)        | 0.267        |
| (3, 5)        | 0.267        |
| (4, 5)        | 0.067        |

Output communities



Remove edge  $e$  with max.  $c_B(e)$

Continue recursively until community is split  
(not needed in this simple example)

# Girvan-Newman Algorithm

- Algorithm splits a graph  $G(V, E)$  into 2 disconnected components

**Repeat**

Calculate  $c_B(e)$  for all  $e \in E$

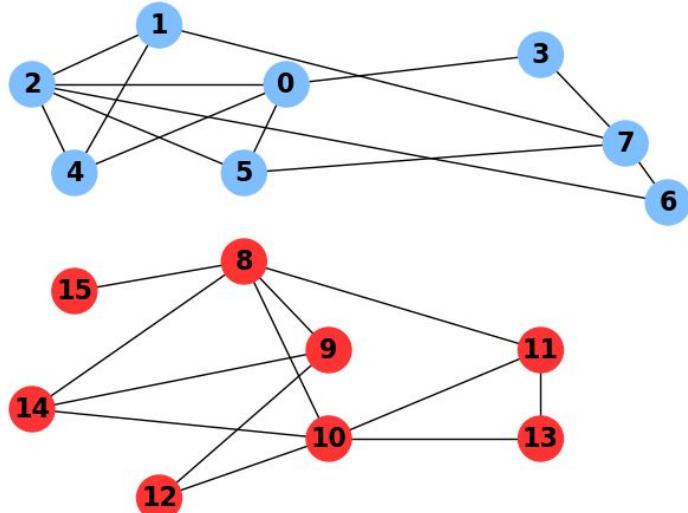
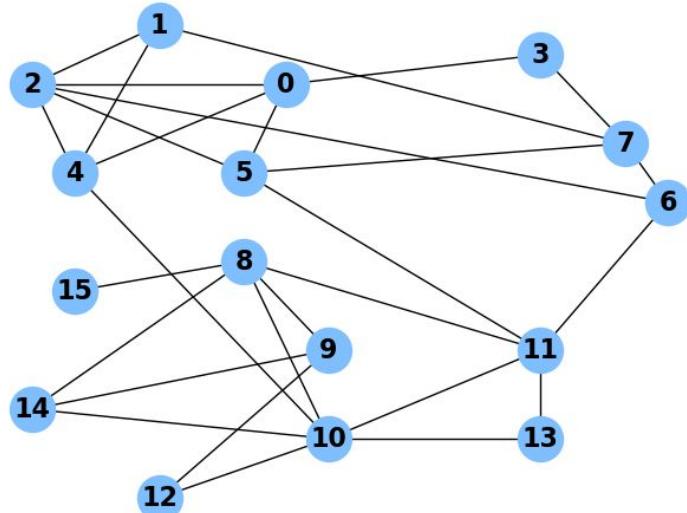
Remove edge from with max.  $c_B(e)$

**Until**  $G$  is split into 2 components

- Recursive step

- Apply algorithm to each new component
- Stops if a component contains only single node  
(or early stop based on user specifications)

# Girvan-Newman Algorithm — Full Example



# Girvan-Newman Algorithm — Remarks

- Complexity Analysis
  - Core concept of algorithm: Edge Betweenness Centrality
  - Requires to solve the **All-Pairs Shortest Path (APSP)** problem
  - Various algorithms and complexities depending on type of graph  
(directed vs. undirected, cyclic vs. acyclic, with or without negative weights, etc.)

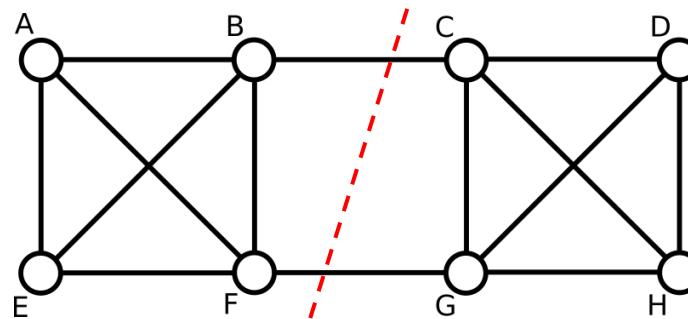
| Time Complexity            |
|----------------------------|
| $n^3$                      |
| $n^3(\log n)/\log n^{1/3}$ |
| $n^3(\log n/\log n)^{1/2}$ |
| $n^3/(\log n)^{1/2}$       |
| $n^3(\log n/\log n)^{5/7}$ |
| $n^3 \log n/\log n$        |
| $n^3(\log n)^{1/2}/\log n$ |
| $n^3/\log n$               |
| $n^3(\log n/\log n)^{5/4}$ |
| $n^3(\log n)^3/(\log n)^2$ |
| $n^3(\log n)/(\log n)^2$   |

Table taken from: [A Survey of Shortest-Path Algorithms](#)  
(Madkour et al., 2017) —  $n = |V|$ , number of nodes

# Karger's Algorithm for Min-Cut

- **Min-Cut Problem**

- Given a graph  $G$ , cut  $G$  into 2 components such that the number of edges between both components is minimal



→  $|\text{Min-Cut}| = 2$

- Fundamental problem in graph theory → many existing algorithms  
(with varying focus and support for different graph types — e.g., directed vs. undirected)

# Karger's Algorithm for Min-Cut

- Karger's algorithm
  - Randomized method to find Min-Cut
  - Applicable to undirected graphs with positive weights  
(this includes unweighted graphs where the weight can be considered 1)

**While**  $|V| > 2$

Randomly pick a remaining edge  $e = (v, u)$

Merge/contract  $v$  and  $u$  into a new node

- Update edges to neighbors of  $v$  and  $u$
- Remove self-loops

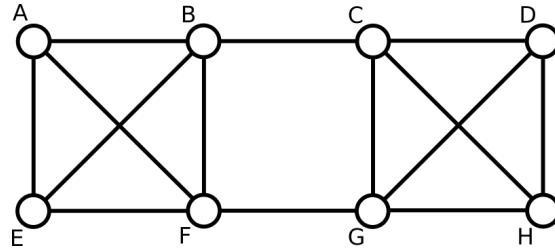
**Return** edges between the final 2 nodes as Min-Cut

**Intuition:** Edges that are in the Min-Cut have a lower probability to get picked!

Basic runtime:  $O(|V|^2)$

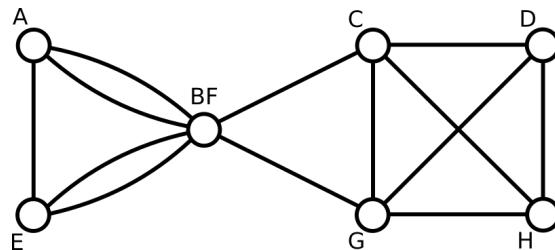
(but further optimizations exists)

# Karger's Algorithm for Min-Cut — Example



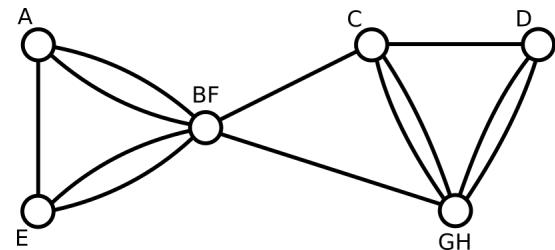
Number of edges: 14

Pick  $e = (B, F)$  with  $P(e) = 1/14$



Number of edges: 13

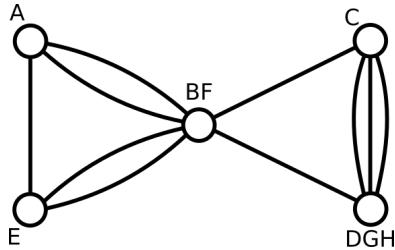
Pick  $e = (G, H)$  with  $P(e) = 1/13$



Number of edges: 12

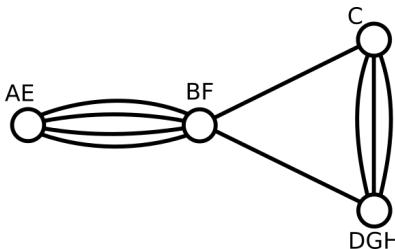
Pick  $e = (D, GH)$  with  $P(e) = 1/6$

# Karger's Algorithm for Min-Cut — Example



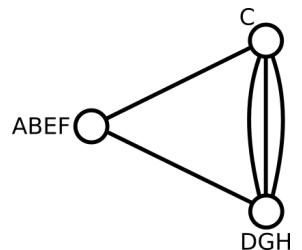
Number of edges: 10

Pick  $e = (A, E)$  with  $P(e) = 1/10$



Number of edges: 9

Pick  $e = (AE, BF)$  with  $P(e) = 4/9$



Number of edges: 5

Pick  $e = (C, DGH)$  with  $P(e) = 3/5$



2 node left → Done, with  $|\text{Min-Cut}| = 2$

# Karger's Algorithm for Min-Cut — Analysis

- What is the probability that the algorithm finds the "correct" Min-Cut?
- For an undirected graph  $G = (V, E)$ , with  $n = |V|$  and  $m = |E|$

The average degree of a node is  $\frac{1}{n} \sum_{v \in V} \text{degree}(v) = \frac{2m}{n}$

The size of the Min-Cut is limited to  $|Min-Cut| \leq \frac{2m}{n}$  since  $\forall v \in V : |Min-Cut| \leq \text{degree}(v)$

→  $P(\text{randomly selected edge is in Min-Cut}) \leq \frac{\frac{2m}{n}}{m} = \frac{2}{n}$

# Karger's Algorithm for Min-Cut — Analysis

- Let  $P(\text{success}) = P(\text{final cut is Min-Cut})$

$$P(\text{success}) = P(\text{1st selected edge not in Min-Cut}) \times \\ P(\text{2nd selected edge not in Min-Cut}) \times \dots$$

$$\begin{aligned} P(\text{success}) &\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \dots \left(1 - \frac{2}{4}\right) \left(1 - \frac{2}{3}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-3} \cdot \dots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ &= \frac{2}{n(n-1)} = \binom{n}{2}^{-1} \end{aligned}$$

That's a rather low probability :(  
→ Run algorithm multiple times!  
But how often?

# Karger's Algorithm for Min-Cut — Analysis

- If the algorithm is run  $k$  times and take the smallest cut found
  - What is the probability  $P(\text{failure})$  that it is not a Min-Cut?

$$P(\text{failure}) = \left[1 - \binom{n}{2}^{-1}\right]^k$$

$$\text{with } k = \binom{n}{2} \ln n \quad \rightarrow \quad P(\text{failure}) \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n}$$

**Note:** For any  $x \geq 1$ :

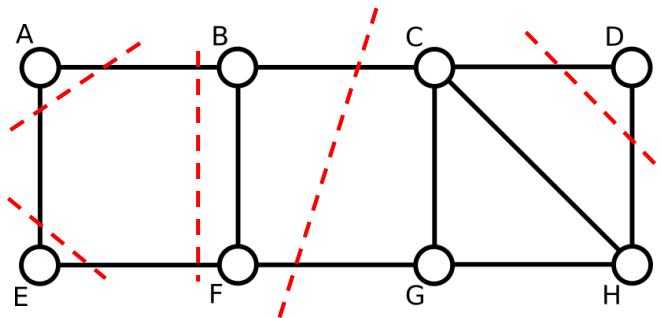
$$\frac{1}{4} \leq \left(1 - \frac{1}{x}\right)^{cx} \leq \left(\frac{1}{e}\right)^c$$

With  $k \in O(n^2 \log n)$  → Total runtime of Karger's Algorithm is in  $O(n^4 \log n)$

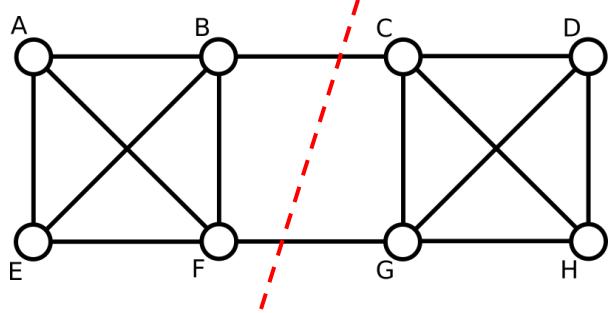
(with a Min-Cut with a high probability)

# Karger's Algorithm for Min-Cut — Remarks

- In general, a graph has multiple possible Min-Cuts



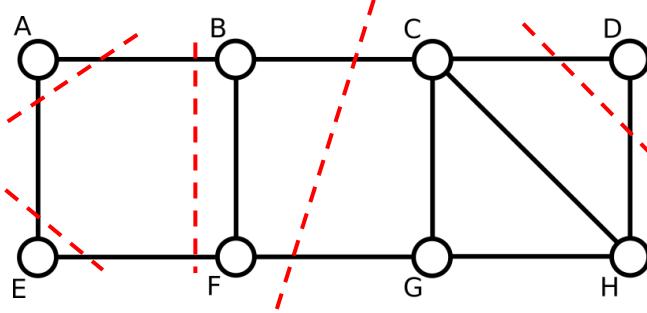
- Choice of Min-Cut often application-specific, e.g.,
  - Favor Min-Cuts where the 2 components are of similar size (e.g., similar number of nodes)
  - Ignore Min-Cuts where the size of a component is below a threshold



```

|Min-Cut| = 5 -- Components: A,B ### C,D,E,F,G,H
|Min-Cut| = 4 -- Components: A,C,D,E,F,G,H ### B
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 5 -- Components: A,B,C,D,E,F ### G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 4 -- Components: A,B,D,E,F,G,H ### C
|Min-Cut| = 5 -- Components: A,F ### B,C,D,E,G,H
|Min-Cut| = 4 -- Components: A,B,C,E,F,G ### D,H
|Min-Cut| = 5 -- Components: A,B,C,D,E,F ### G,H
|Min-Cut| = 4 -- Components: A,B,C,E,F ### D,G,H
|Min-Cut| = 3 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 4 -- Components: A,B,C,E,F,G ### D,H
|Min-Cut| = 4 -- Components: A,B,C,D,E,G,H ### F
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H

```



```

|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 2 -- Components: A,E ### B,C,D,F,G,H
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,H ### G
|Min-Cut| = 3 -- Components: A,B,C,E,F,G ### D,H
|Min-Cut| = 2 -- Components: A,B,C,D,E,F,H ### G
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,H ### G
|Min-Cut| = 3 -- Components: A,B,C,D,E,F,G ### H
|Min-Cut| = 2 -- Components: A ### B,C,D,E,F,G,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,C,E,F,G,H ### D
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H
|Min-Cut| = 2 -- Components: A,B,E,F ### C,D,G,H

```

# Overview

- **Graph Mining**
  - Application Examples
  - Basic definitions
- **Community Detection**
  - Basic definition & goals
  - Overview to different algorithms
- **Centrality**
  - Basic definition & goals
  - Overview to different algorithms
- **Summary**

# Centrality

- **Centrality — Centrality measures**

- Quantify the **importance** of a node given its topological position in a graph  
(centrality is commonly assigned to nodes but can be extended to edges, cf. Edge Betweenness Centrality)
- Different measures favor different "flavours" of importance

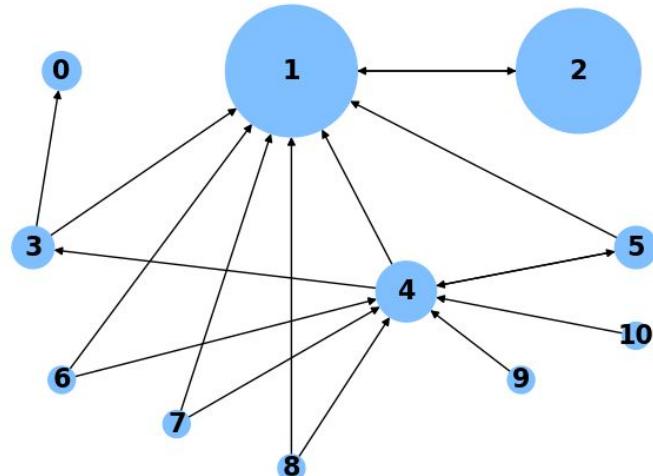
## → What makes a node important?

(or more important compared to other nodes)

- **Wide range of applications**

- Identify influential social network users
- Identify superspreaders of diseases
- Identify key infrastructure nodes in infrastructure networks

Example: PageRank of 10 web pages



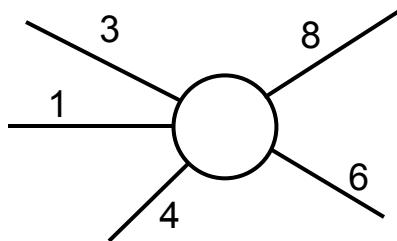
# Degree Centrality

- Centrality of a node only dependent on direct neighborhood edges

Undirected graph

Sum of weights of connected edges

$$c_d(v_i) = \sum_{v_j \in V} A[i, j]$$

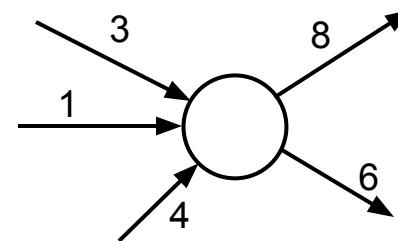


$$c_d(v) = 22$$

Directed graph

Sum of weights of incoming edges

$$c_{d\_in}(v_i) = \sum_{v_j \in V} A[j, i]$$



$$c_{d\_in}(v) = 8$$

Sum of weights of outgoing edges

$$c_{d\_out}(v_i) = \sum_{v_j \in V} A[i, j]$$

$$c_{d\_out}(v) = 14$$

Note: For unweighted graphs,  $A[i,j] = 1$  for existing edges → sum of weights = edge count

# Degree Centrality — Pros & Cons

- **Pros**

- Simple measure → easy and fast to calculate
- For many applications "good enough"

- **Cons**

- Local measure — does not take any extended topological information of a node into account
- Treats all connected edges of a node equally (i.e., neighboring source or target node does not matter)
- Depending on application, easy to manipulate

## Examples of manipulation attacks

- Create fake pages with links to a target page to bump up its search result rank
- Create fake followers on social media to increase reputation and attract sponsors
- Create fake feedback to create on e-commerce sites to increase reputation

# Eigenvector Centrality

- Idea: Centrality of a node depends on the centrality of its neighbors
  - Basic definition applicable to undirected graph
  - Recursive definition — How to calculate it? (well, it's in the name)

$$c_{ev}(v_i) = \frac{1}{\lambda} \sum_{v_j \in V} [A[i, j] \cdot c_{ev}(v_j)]$$

$\lambda$  — some normalization constant

In matrix form:  $\lambda c_{ev} = A c_{ev}$

$c_{ev}$  — vector of centrality degrees of all nodes

Common Eigenvector equation:

$$A\vec{x} = \lambda\vec{x}$$

All  $\vec{x}$  solving this equation are the eigenvectors of  $A$  and  $\lambda$  are their corresponding eigenvalues:

$$(A - \lambda I)\vec{x} = 0$$

$c_{ev}$  is the largest eigenvector of adjacency matrix  $A$ !

# Eigenvector Centrality — Power Iteration / Power Method

- Power Iteration — numerical method to find largest eigenvector of a matrix
  - Solving eigenvector equation analytically not tractable for large matrices

**Input:** matrix  $M$ , error threshold  $\varepsilon$ , #max. iterations  $T$

**Initialization:**  $t = 0$ ,  $x_0 = [1/|V|, 1/|V|, \dots, 1/|V|]$

**Repeat**

$$t = t+1$$

$$x_t = Ax_{t-1}$$

$$x_t = x_t / \|x_t\| \quad \# \text{ Normalize vector}$$

$$\delta = \|x_t - x_{t-1}\| \quad \# \text{ Calculate difference}$$

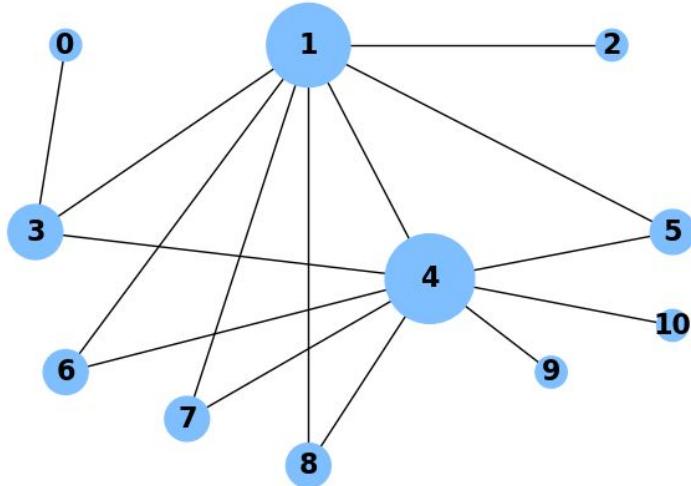
**Until**  $\delta < \varepsilon$  **or**  $t > T$

**Return**  $x_t$

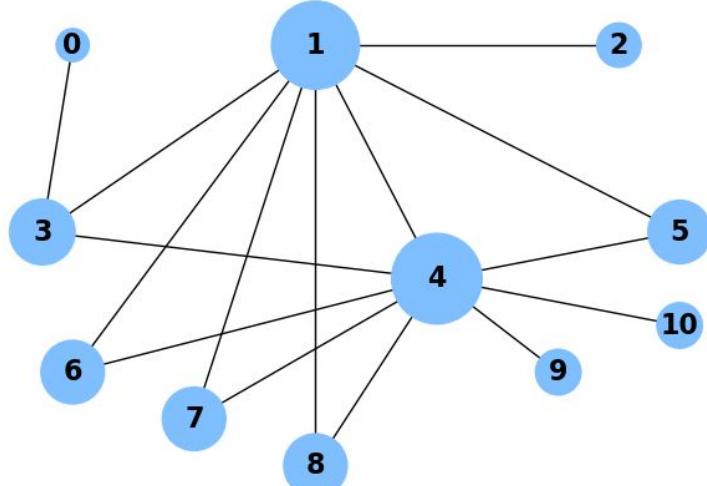
# Degree vs. Eigenvector Centrality

(size of nodes reflect centrality scores)

Degree



Eigenvector



- Low-degree nodes benefit from connections to high-degree nodes

# PageRank

- Goal: Find important pages in the web graph
  - The set of vertices  $V$  is set of all (indexed) web pages
  - An edge  $e_{ij} \in E$  indicates that there is a hyperlink for page  $i$  to page  $j$

- PageRank — a page  $v_j$  has a high PageRank if

- Many other pages link to  $v_j$
- Those pages linking to  $v_j$  have
  - (a) a high PageRank themselves
  - (b) not many other outgoing links



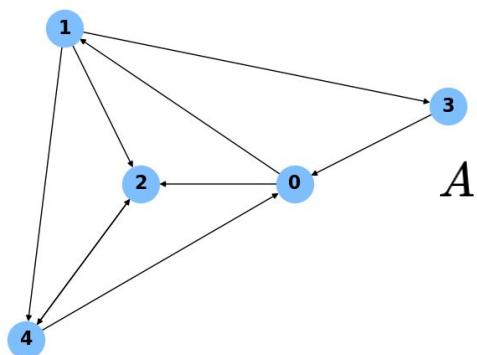
$$PR(v_j) = \sum_{i \rightarrow j} \frac{1}{\text{outdegree}(i)} PR(v_i)$$



Matrix notation for all vertices

$$PR(v) = M \cdot PR(v)$$

# PageRank — Transition Matrix M



Adjacency matrix A

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- Transpose A
- Normalize columns  
(column entries sum up to 1)

$$\xrightarrow{\hspace{1cm}} M =$$

Transition matrix M

$$M = \begin{bmatrix} 0 & 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 1 & 0 & 0 \end{bmatrix}$$

**Note:** M is a so-called column-stochastic matrix.

# Computing PageRank Scores

- PageRank is an Eigenvector problem
  - Solvable with Power Iteration Method

$$\lambda PR(v) = M \cdot PR(v)$$



Since  $M$  is column-stochastic,  
the largest eigenvalue  $\lambda = 1$ .

$$PR(v) = M \cdot PR(v)$$

Common Eigenvector equation:

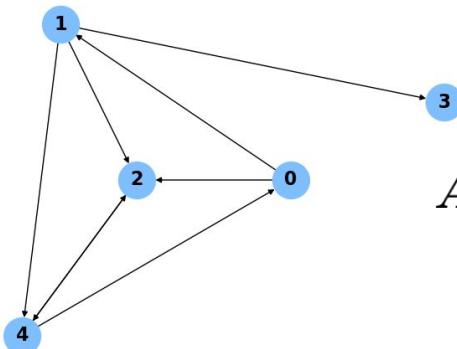
$$A\vec{x} = \lambda\vec{x}$$

All  $\vec{x}$  solving this equation are the eigenvectors  
of  $A$  and  $\lambda$  are their corresponding eigenvalues:

$$\vec{x} \quad \lambda \quad (A - \lambda I) \vec{x} = 0$$

# Computing PageRank Scores — Challenge

- Requirement: Graph must be strongly connected
  - Each node can be reached from any other node
  - Requirement does not hold for the web graph → dangling nodes
  - Computing PageRank scores using Power Iteration Method not working



Adjacency matrix A

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

- Transpose A
- Normalize columns  
(column entries sum up to 1?)

$$\xrightarrow{\hspace{1cm}} M =$$

Transition matrix M

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 1 & 0 & 0 \end{bmatrix}$$

# Computing PageRank Scores — Handling Dangling Nodes

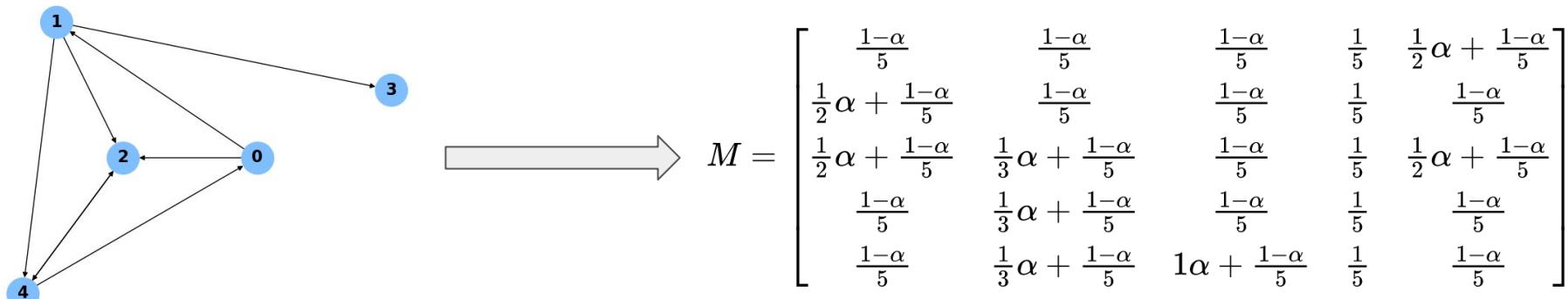
- PageRank implements **Random Surfer** model
    - $\alpha$  probability of following a link on a page to the next
    - $(1 - \alpha)$  probability of jumping to a random page (i.e., not following a link)
- Any page can be reached, whether linked or not!

$$PR(v) = M \cdot PR(v) \iff PR(v) = \alpha \cdot M \cdot PR(v) + (1 - \alpha) \cdot E$$

$$\text{with } E = \begin{bmatrix} 1/|V| \\ 1/|V| \\ \vdots \\ 1/|V| \end{bmatrix}$$

# Random Surfer Model — Effects on Transition Matrix

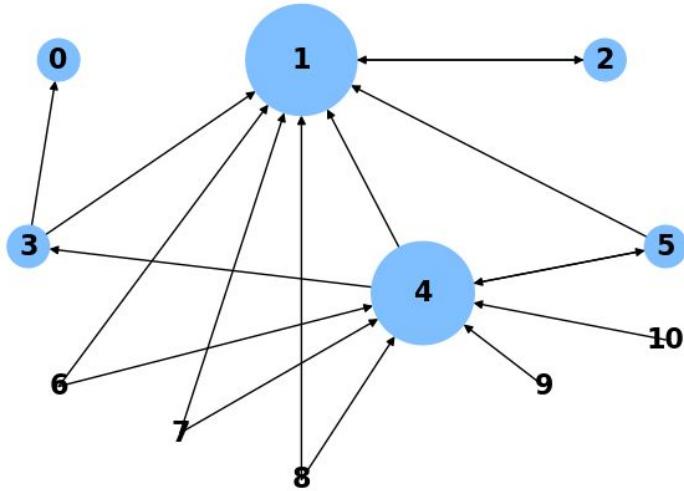
- Random Surfer model introduces "virtual edges"
  - Each node can be reached from any other → virtual links → fully connected graph
  - PageRank scores can be calculated using Power Iteration Method without problems



**\*Note:** Matrix  $M$  is not materialized, but this is the matrix that reflects the PageRank formula!

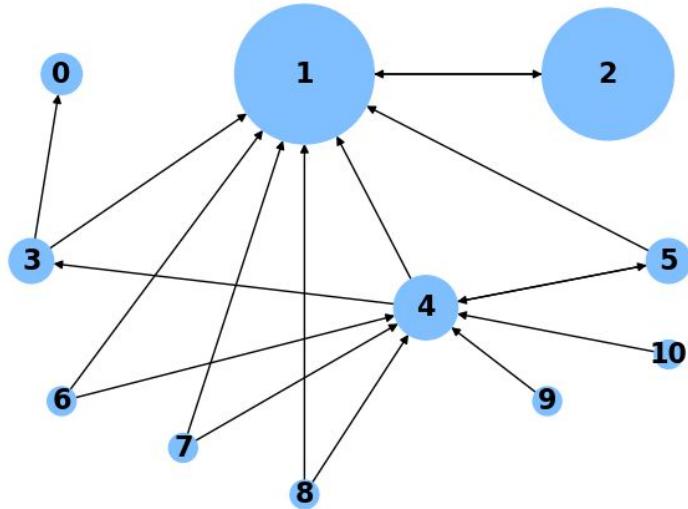
# InDegree vs. PageRank (size of nodes reflect centrality scores)

InDegree



- Nodes 1 and 4 very similar centrality scores
- Node 2 with only a low centrality score
- Scores of 0 for nodes with no incoming edges

PageRank



- Clear difference between Nodes 1 and 4
- Node 2 with a high centrality score since linked to from Node 1 with a very high score
- Non-zero scores for nodes with no incoming edges

# Quick Quiz

Given a Graph with 10 nodes and a setting of  $\alpha=0.85$ , what is the **lowest possible** PageRank score?

$$c_{pr}(v_i) = \alpha M c_{pr}(v_i) + (1 - \alpha)E$$

A

0.0

B

0.015

C

0.085

D

0.15

# Eigenvector-Based Measures — Remarks

- Measuring centrality by solving an Eigenvector problem
  - Recursive definition of centrality very intuitive
  - Many other similar measures (e.g., HITS, SALSA, Katz)
  - Many application-specific extensions to basic measures  
(e.g., personalization of PageRank where random jumps are no longer uniform)
  - More complex calculation than for local measures but calculation of largest Eigenvectors very scalable through parallelization

# Closeness Centrality

- Intuition: A node  $t$  is central if the distance to all other nodes is small
  - Small distance to node  $t$  = short paths from all other nodes to  $t$
  - For directed paths, the closeness of node  $t$  can differ greatly when considering incoming or outgoing edges for calculating distances
  - Basic definition applicable to unweighted graph  
(generalized definitions for weighted graphs have been proposed)

$$c_{cl}(v) = \frac{N - 1}{\sum_{w \in V} d(w, v)}$$

$N$  — number of nodes from which  $v$  is reachable

$d(w, v)$  — length of shortest path from  $w$  to  $v$

**Note:** For directed graphs, this definition calculates closeness using the nodes' incoming edges — more common case.  
To consider outgoing edges,  $d(w, v)$  becomes  $d(v, w)$ , and  $N$  becomes the number of nodes reachable from  $v$ .

# Betweenness Centrality

- Intuition: A node  $t$  is central if many shortest paths between all other nodes pass through node  $t$ 
  - Removing such nodes would cause the most "disruption" in a graph
  - Directly applicable to directed/undirected and weighted/unweighted graphs  
(since the notion of shortest path is well-defined for all graph types)

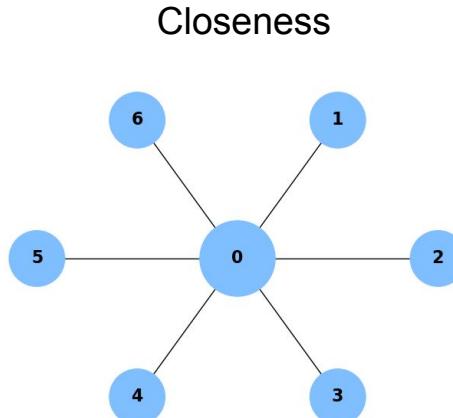
$$c_b(v) = \sum_{s,t \in V; s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

number of shortest paths from  $s$  to  $t$  passing through node  $v$

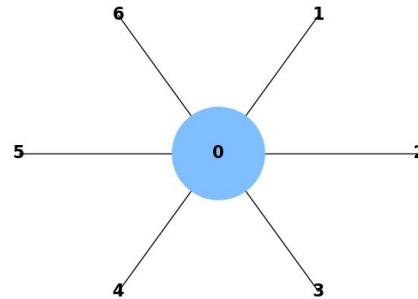
total number of shortest paths from  $s$  to  $t$

# Closeness vs. Betweenness (size of nodes reflect centrality scores)

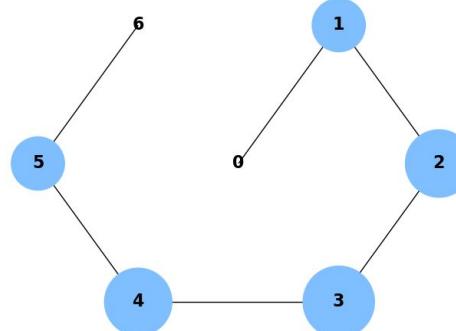
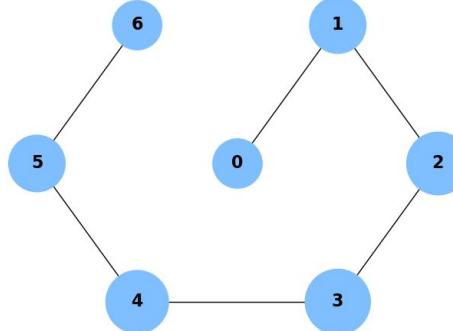
Star network graph



Betweenness

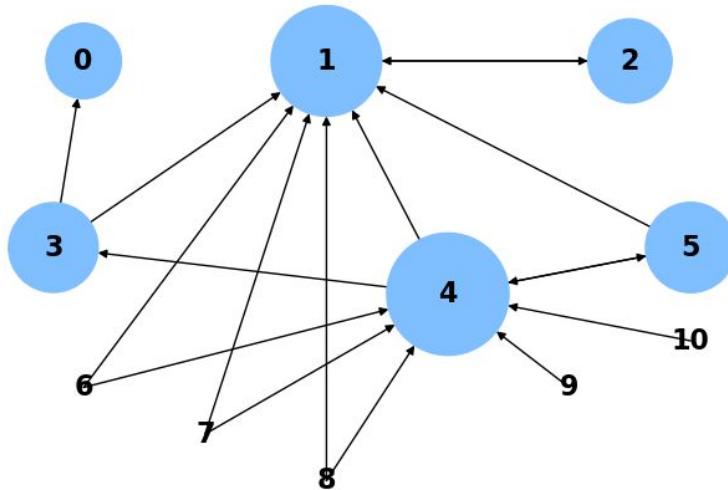


Sequence graph

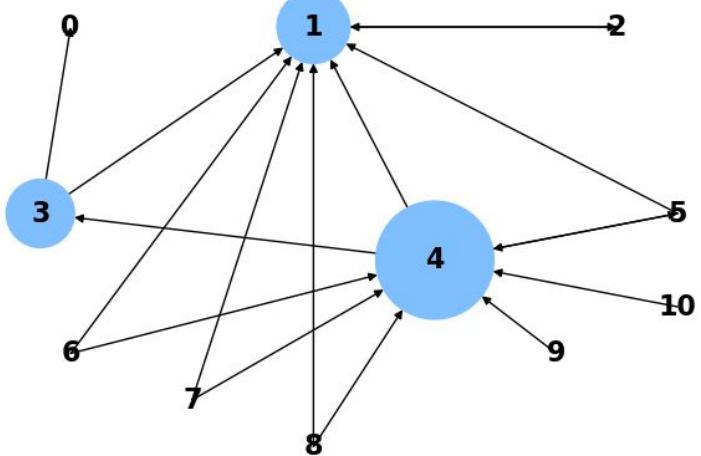


# Closeness vs. Betweenness (size of nodes reflect centrality scores)

Closeness



Betweenness



# Closeness & Betweenness — Remarks

- Distance-based measures

- Both measures rely on the notion of shortest paths between nodes
- Requires to solve the **All-Pairs Shortest Path (APSP)** problem
- Various algorithms and complexities depending on type of graph  
(directed vs. undirected, cyclic vs. acyclic, with or without negative weights, etc.)

| Time Complexity            |
|----------------------------|
| $n^3$                      |
| $n^3(\log n)/\log n^{1/3}$ |
| $n^3(\log n/\log n)^{1/2}$ |
| $n^3/(\log n)^{1/2}$       |
| $n^3(\log n/\log n)^{5/7}$ |
| $n^3 \log n/\log n$        |
| $n^3(\log n)^{1/2}/\log n$ |
| $n^3/\log n$               |
| $n^3(\log n/\log n)^{5/4}$ |
| $n^3(\log n)^3/(\log n)^2$ |
| $n^3(\log n)/(\log n)^2$   |

Table taken from: [A Survey of Shortest-Path Algorithms](#)  
(Madkour et al., 2017) —  $n = |V|$ , number of nodes

# Centrality — Discussion

- Centrality = importance of nodes on a graph
  - Important concept of graph mining with many applications
  - Wide range of proposed measures that differ in their definition of a node's importance
  - Not all measures are applicable (or suitable) to all types of graphs
- Overview to a selected set of popular measures
  - Local measures — Degree, InDegree, OutDegree
  - Eigenvector-based measures — Eigenvector Centrality, PageRank
  - Distance-based measures — Closeness, Betweenness

# Quick Quiz

Given an **undirected** and **connected** graph G, which centrality measure can yield **scores of 0**?

A

Eigenvector

B

Closeness

C

Degree

D

Betweenness

# Quick Quiz

Which MRT station has the highest  
**Closeness centrality** score?

(graph does not include LRT stations)

A

Dhoby Ghaut

B

Buona Vista

C

City Hall

D

Stevens

# Quick Quiz

Which MRT station has the highest  
**Betweenness centrality** score?

(graph does not include LRT stations)

A

Bugis

B

Botanic Gardens

C

Clementi

D

Outram Park

# Solutions to Quick Quizzes

- Slide 16: A
- Slide 17: B
- Slide 54: B
- Slide 62: D
- Slide 63: A
- Slide 64: B

# **CS5228: Knowledge Discovery and Data Mining**

## Lecture 7 — Dimensionality Reduction

# Outline

- **Dimensionality Reduction**
  - Motivation
  - Naive approaches
- Dimensionality Reduction Techniques
  - PCA — Principal Component Analysis
  - LDA — Linear Discriminant Analysis
  - t-SNE — t-distributed Stochastic Neighbor Embedding

# Dimensionality Reduction — Motivation

- High number of dimensions = high number of data features
  - $m$  ("mass") — number of data points
  - $V$  ("volume") — data space described by dimensions

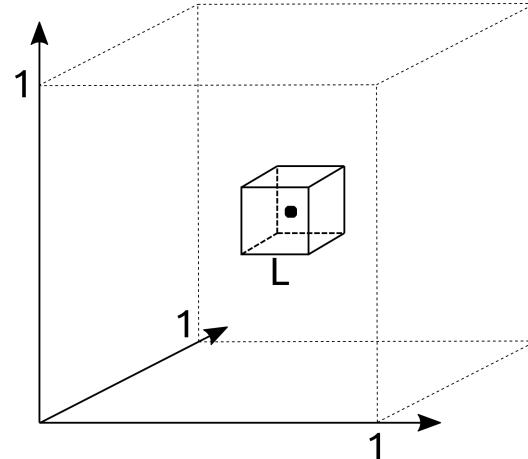
$$\text{density } \rho = \frac{m}{V} \qquad \rightarrow \text{High-dimensional data quickly becomes very sparse!}$$

- Effects of many features and data sparsity
  - Higher computational cost
  - Problematic for any method that requires statistical significance → high risk of overfitting ("good" data points and noise/outliers look more and more indistinguishable)
  - Obscures similarities between data points  
(points similar in low dimensions but not in high dimensions)

# Curse of Dimensionality

- **Effect of high dimensions** (i.e., many features)
  - Data points tend to never be close together
  - Average distance between points converges
- **Intuition**
  - Assume  $N$  data points uniformly distributed within a unit cube with  $d$  dimensions
  - Let  $L$  be the length of the smallest cube containing the  $k$ -NN of a data point

$$L^d \approx \frac{k}{N} \Rightarrow L \approx \left(\frac{k}{N}\right)^{\frac{1}{d}}$$



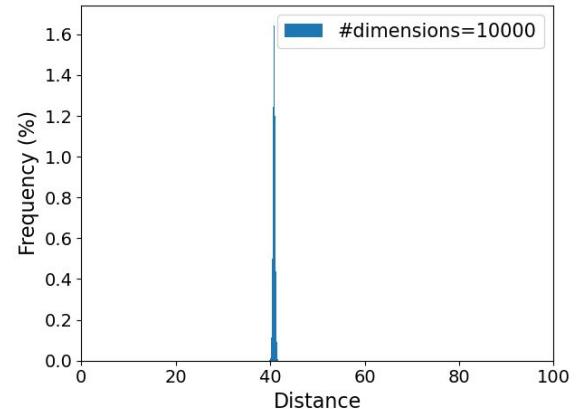
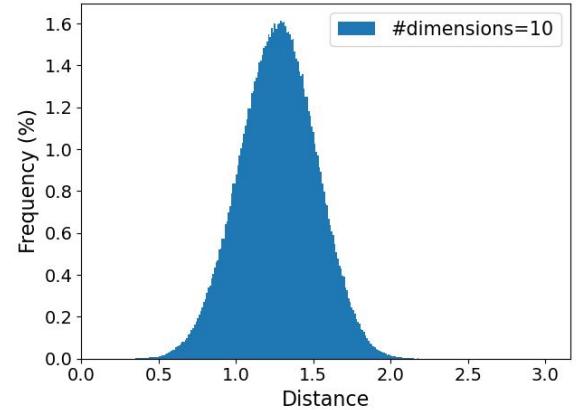
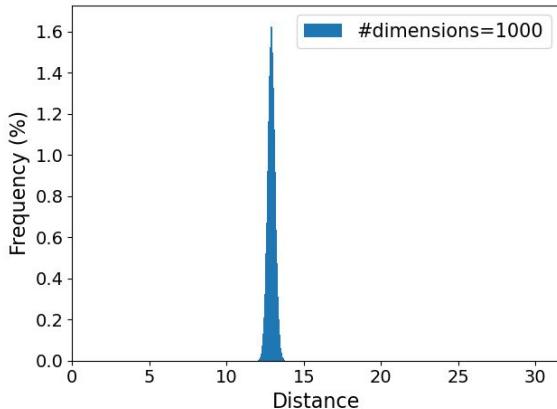
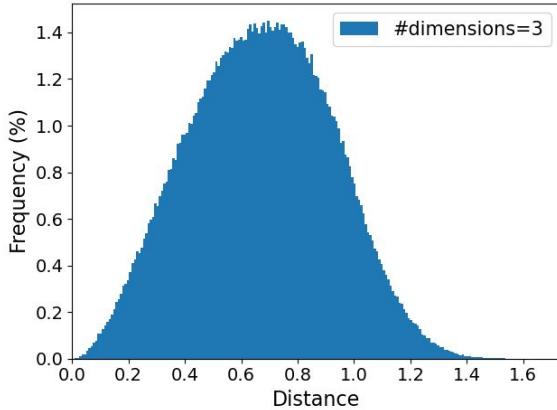
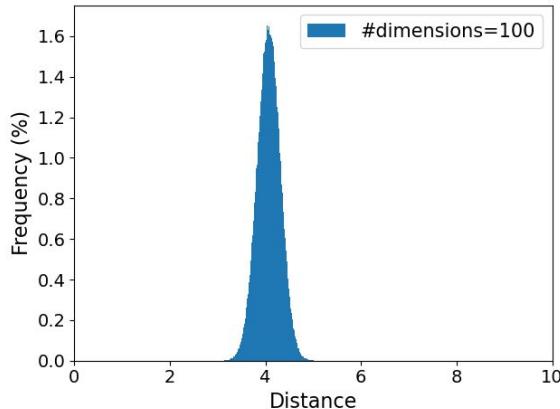
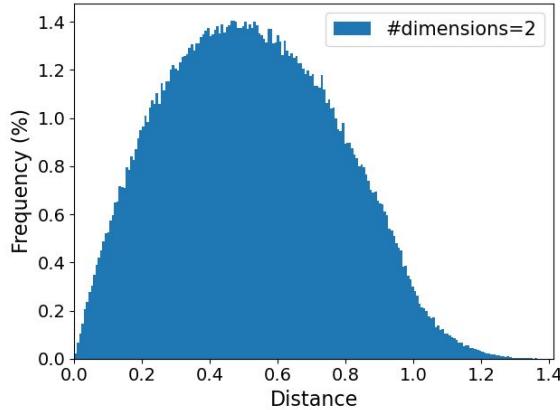
N=1,000, k=10

| $d$    | $L$   |
|--------|-------|
| 2      | 0.100 |
| 3      | 0.215 |
| 10     | 0.631 |
| 100    | 0.955 |
| 1,000  | 0.995 |
| 10,000 | 0.999 |

The cube with the  
k-NN is almost the  
whole unit cube!

# Curse of Dimensionality

Distribution of pairwise distances between 1,000 random data points and different number of dimensions



# Dimensionality Reduction — Feature Selection

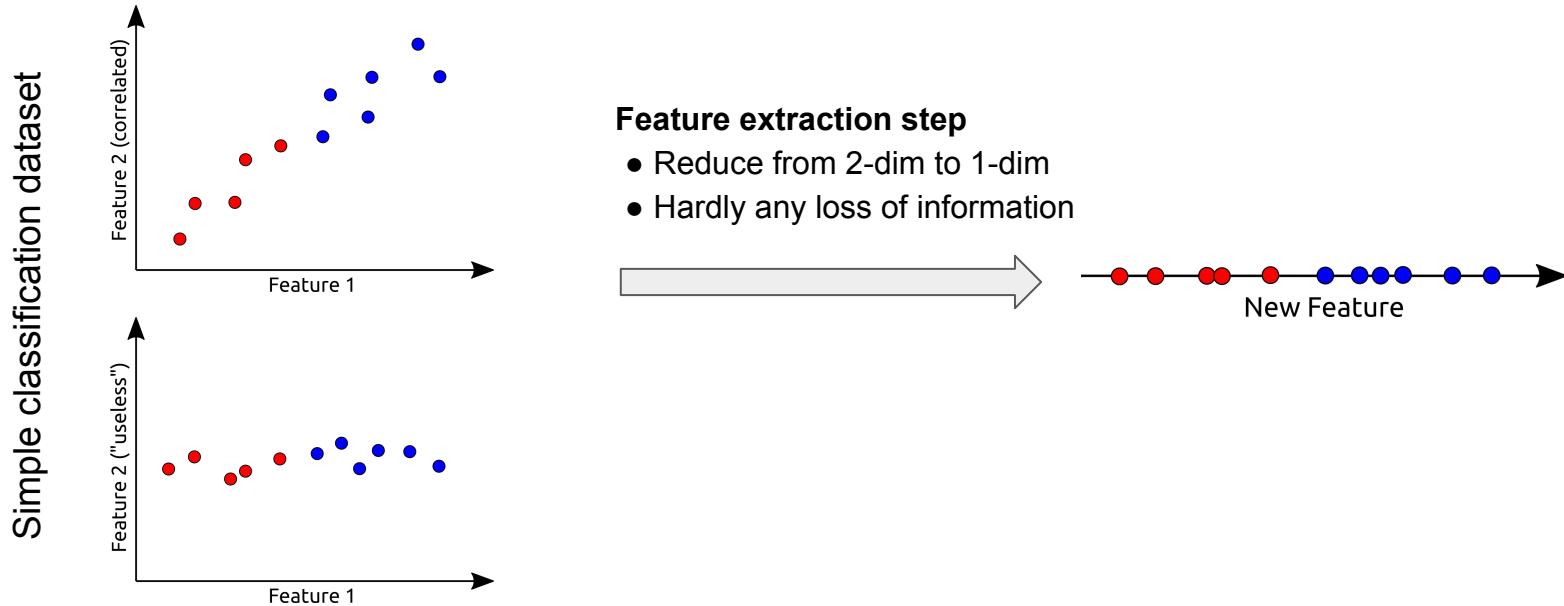
- Feature Selection — Removal of features before analysis  
(alternatively: keep only a certain subset of features)
- Common feature selection
  - Remove "unimportant" features based on expert knowledge  
(e.g., birth date of a person is unlikely to affect his/her spending behavior)
  - Remove features that might introduce ethical biases  
(e.g., sexual orientation and ethnicity for credit card approval)
  - Remove features with very low variance → not useful
  - Remove features that are strongly correlated with other features → not needed  
(basic method: calculate pairwise correlations and remove one of the features in case of high correlation)
  - Remove features with low ranks w.r.t. to their power to discriminate data points  
(basic approach of Decision Trees where feature near the root node yield purer subtrees)

# Feature Selection — Pros & Cons

- Pros
  - Relatively straightforward to implement
  - Does not change features themselves → effects on analysis results are preserved
- Cons
  - Selecting important features based on expert knowledge often not trivial/obvious
  - Finding meaningful thresholds — e.g., min. variance or correlation — not obvious

# Dimensionality Reduction — Feature Extraction

- Feature Extraction — basic idea
  - Generate new features as form of summary of original features
  - Feature extraction algorithms utilize discriminatory power and correlations among features



# Outline

- Dimensionality Reduction
  - Motivation
  - Naive approaches
- Dimensionality Reduction Techniques
  - PCA — Principal Component Analysis
  - LDA — Linear Discriminant Analysis
  - t-SNE — t-distributed Stochastic Neighbor Embedding

# Principal Component Analysis (PCA)

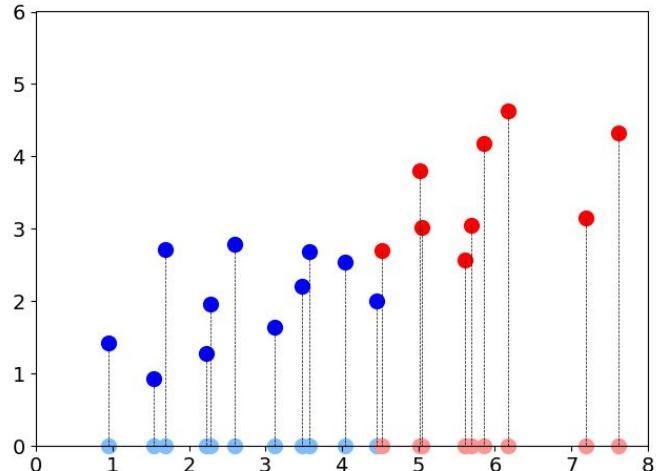
- PCA — Dimensionality reduction through linear transformation
  - New output features are a linear combination of original input features
  - Transforms data to a new coordinate systems
  - Unsupervised approach (independent from any kind of class labels)
- Basic setup
  - Dataset  $X$  ( $n$  samples,  $d$  features)
  - Find matrix  $W$  to transform  $X$  into a  $p$ -dimensional dataset ( $p < d$ )

$$\begin{bmatrix} X \\ n \times d \end{bmatrix} \begin{bmatrix} W \\ d \times p \end{bmatrix} = \begin{bmatrix} P \\ n \times p \end{bmatrix}$$

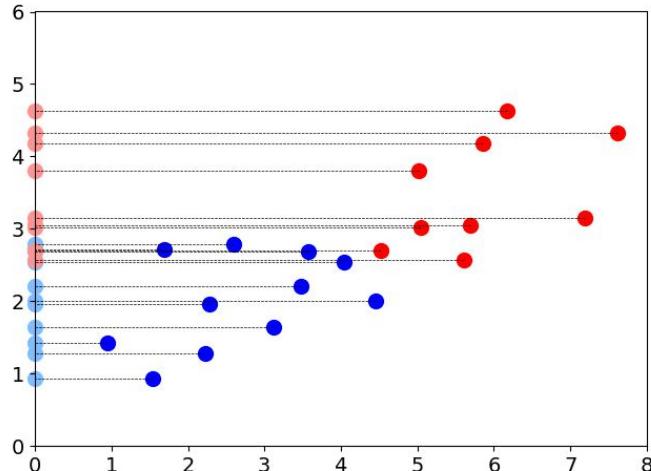
→ What makes a good transformation matrix  $W$  and how to find it?

# PCA — Intuition Using Naive Transformations

Mapping of data to x-axis:  $W = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



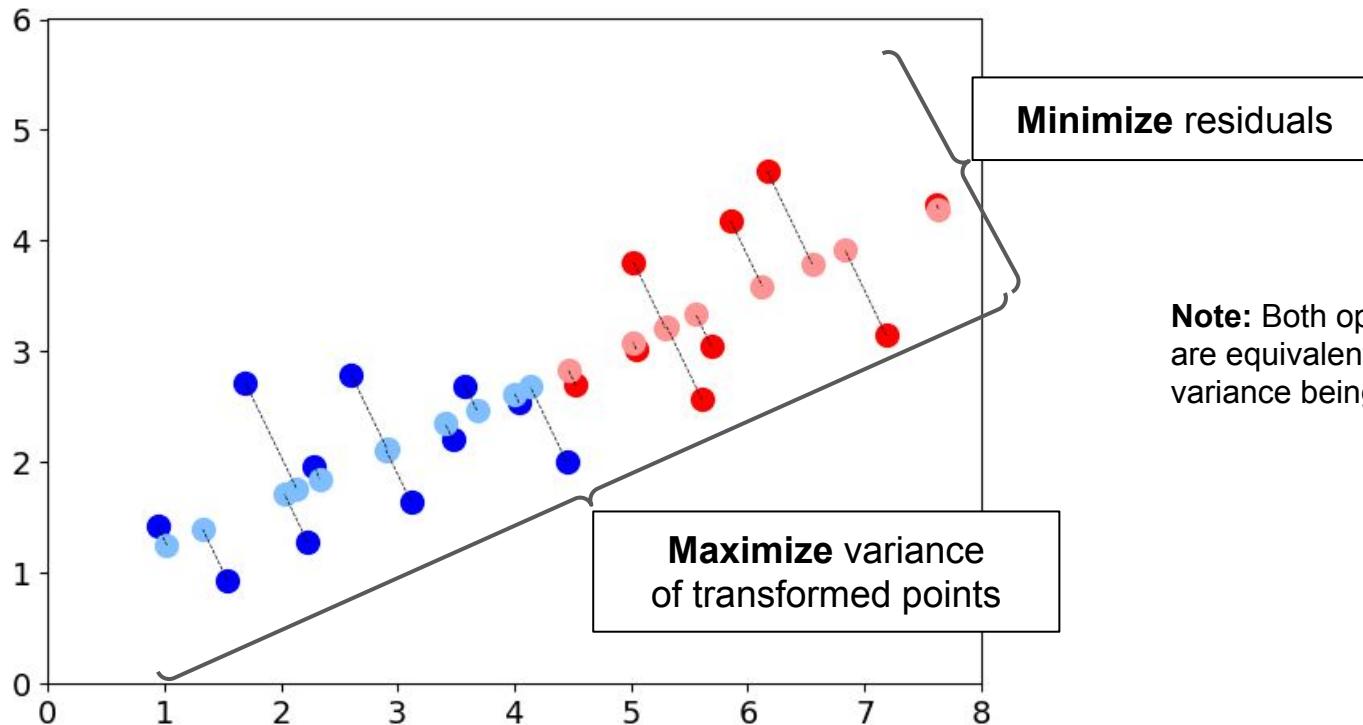
Mapping of data to x-axis:  $W = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



Which transformation is the better one? Why?

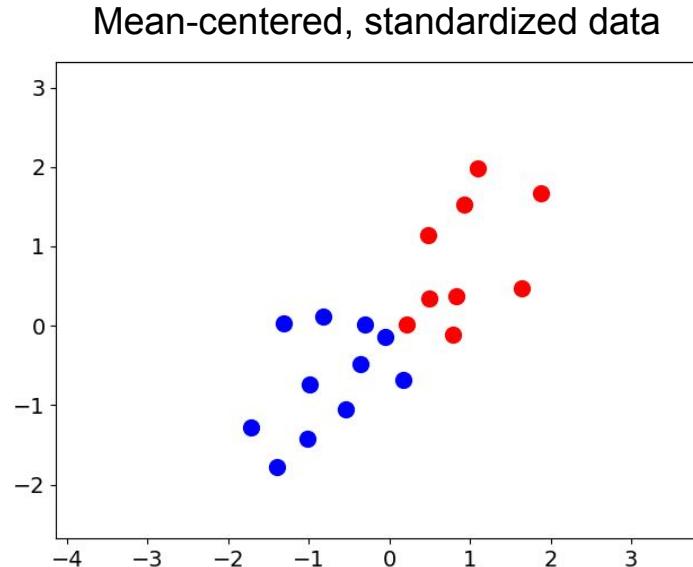
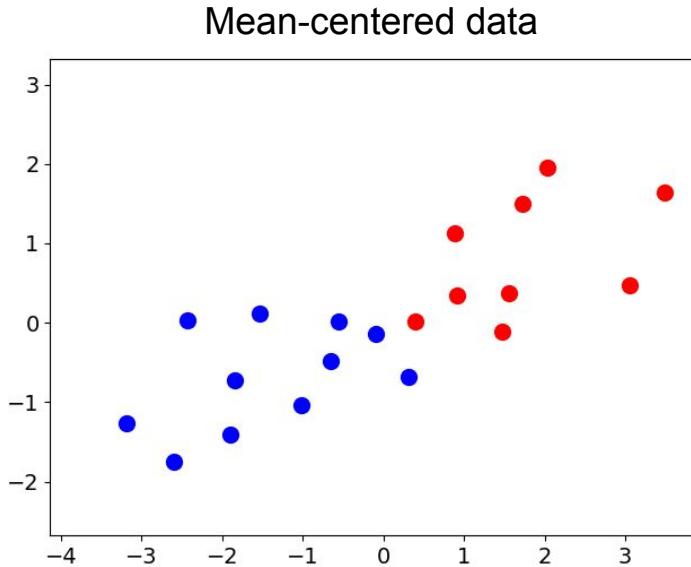
Is there an even better transformation?

# PCA — Equivalent Objectives for Finding Transformation



# PCA — Data Normalization

- Data normalization steps
  - Mean-centering — does not affect results but makes the math much easier
  - Standardizing (divide by standard deviation) — optional; will affect the results



# PCA — Finding the 1st Principal Component

- 1st Principal Component of data  $X$

- Unit vector  $w_1$  that maximizes the variance of transformed data

$$w_1 = \underset{\|w\|=1}{\operatorname{argmax}} \underbrace{\frac{1}{n} \sum_i (p_i - 0)^2}_{\text{Variance of transformed data}}$$

$p_i$  — data point  $x_i$  after transformation

0 because of mean-centered data

# PCA — Finding the 1st Principal Component

- 1st Principal Component of data  $X$

- Unit vector  $w_1$  that maximizes the variance of transformed data

$$w_1 = \underset{\|w\|=1}{\operatorname{argmax}} \underbrace{\frac{1}{n} \sum_i (p_i - 0)^2}_{\text{Variance of transformed data}}$$

$p_i$  — data point  $x_i$  after transformation

0 because of mean-centered data

$$\begin{aligned} w_1 &= \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} \sum_i (x_i \cdot w)^2 \\ &= \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} \|Xw\|^2 \\ &= \underset{\|w\|=1}{\operatorname{argmax}} \frac{1}{n} w^T X^T X w \\ &= \underset{\|w\|=1}{\operatorname{argmax}} w^T \frac{X^T X}{n} w \\ &= \underset{\|w\|=1}{\operatorname{argmax}} w^T C_X w \end{aligned}$$

→  $C_X$  is the covariance matrix of  $X$

**Note:** Sometimes  $C_X = X^T X$  (instead of  $C_X = X^T X/n$ ). In this case  $C_X$  is the unnormalized covariance matrix (scatter matrix). This only affects the magnitude of the eigenvalues but the eigenvectors for  $W$ .

# PCA — Finding the 1st Principal Component

$$w_1 = \underset{\|w\|=1}{\operatorname{argmax}} w^T C_X w$$

$$= \underset{\|w\|=1}{\operatorname{argmax}} \frac{w^T C_X w}{w^T w}$$

Note that  $w^T w = \|w\| = 1$

*Rayleigh Quotient*

→  $w_1$  is the largest eigenvector of the covariance matrix  $C_X$

# PCA — Finding the k-th Principal Component

- Subtract (k-1) principal components from  $X$

$$X_k = X - \sum_{s=1}^{k-1} X w_s w_s^T$$

- k-th Principal Component of data  $X_k$

- Unit vector  $w_k$  that maximizes the variance of transformed data — after transforming  $X_k$

$$w_k = \underset{\|w\|=1}{\operatorname{argmax}} w^T \frac{X_k^T X_k}{n} w = \underset{\|w\|=1}{\operatorname{argmax}} w^T C_x^{(k)} w = \underset{\|w\|=1}{\operatorname{argmax}} \frac{w^T C_x^{(k)} w}{w^T w}$$

→  $w_k$  is the largest eigenvector of the covariance matrix  $C_x^{(k)}$

# PCA — The Math

$$\frac{dJ(w)}{dw} = \frac{\frac{d(w^T C w)}{dw} \cdot w^T w - w^T C w \cdot \frac{d(w^T w)}{dw}}{(w^T w)^2}$$

$$= \frac{w^T (C + C^T)}{w^T w} - \frac{w^T C w \cdot 2w}{(w^T w)^2}$$

$$= \frac{2Cw}{w^T w} - \frac{w^T C w \cdot 2w}{(w^T w)^2}$$

$$= \frac{2}{w^T w} \left( Cw - \frac{w^T C w}{w^T w} w \right)$$

$$J(w) = \frac{w^T C w}{w^T w}$$

Quotient Rule

$$\frac{x^T A x}{dx} = x^T (A + A^T)$$

$C$  is symmetric  $\rightarrow C + C^T = 2C$

scalar value!

$$\frac{dJ(w)}{dw} = \frac{2}{w^T w} (Cw - J(w)w) \stackrel{!}{=} 0 \quad \Leftrightarrow \quad Cw = \overbrace{J(w)w}$$

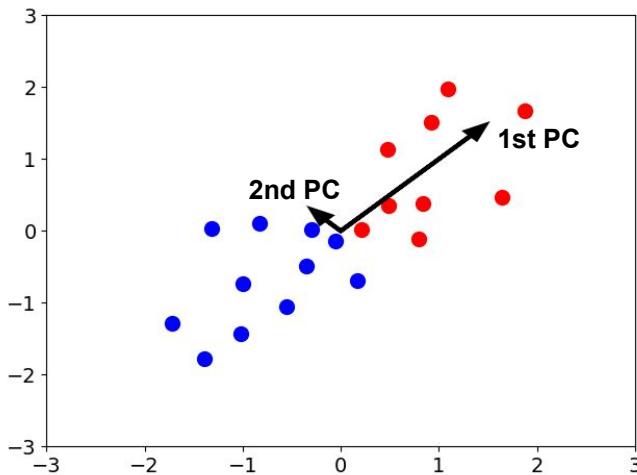
$\rightarrow$  Eigenvalue problem!

# PCA — Getting all Principal Components

- Mathematical convenience

- Largest eigenvector of  $C_X^{(k)}$  = k-largest eigenvector of  $C_X$

→ Principal Components of  $X$  = eigenvectors of covariance matrix  $C_X$



## Interpretation

- 1st PC points into the direction of maximum variance
- 2nd PC points into the direction of maximum variance after 1st PC removed from dataset  $X$
- ...

# PCA in Python Code (using numpy library)

```
1 def normalize(X):
2     X -= np.mean(X, 0) # Mean-center the data
3     X /= np.std(X, 0) # Standardize data (optional; in line with sklearn.decomposition.PCA)
4     return X
5
6
7 def covariance(X):
8     return np.dot(X.T, X) / X.shape[0] # Assumes mean-centered data matrix X
9
10
11 def pca(X, num_pc=None):
12     # Prepare data
13     X = normalize(X)
14     C = covariance(X)
15     # Calculate all eigenvectors and eigenvalues
16     eigenval, eigenvec = np.linalg.eigh(C)
17     # find the the num_pc largest eigenvalues
18     top_idx = np.argsort(eigenval)[::-1][:num_pc]
19     # Create transformation matrix W using eigenvectors with the largest eigenvalues
20     W = eigenvec[:, top_idx]
21     # Return the transformed data
22     return np.dot(X, W)
```

# PCA — Transforming Original Dataset X

- $C_x$  is a  $(d \times d)$  matrix  $\rightarrow d$  eigenvectors and eigenvalues
  - How to choose  $1 \leq p \leq d$  to get transformation matrix  $W$  of shape  $(d \times p)$ ?

- **Explained Variance Ratio**

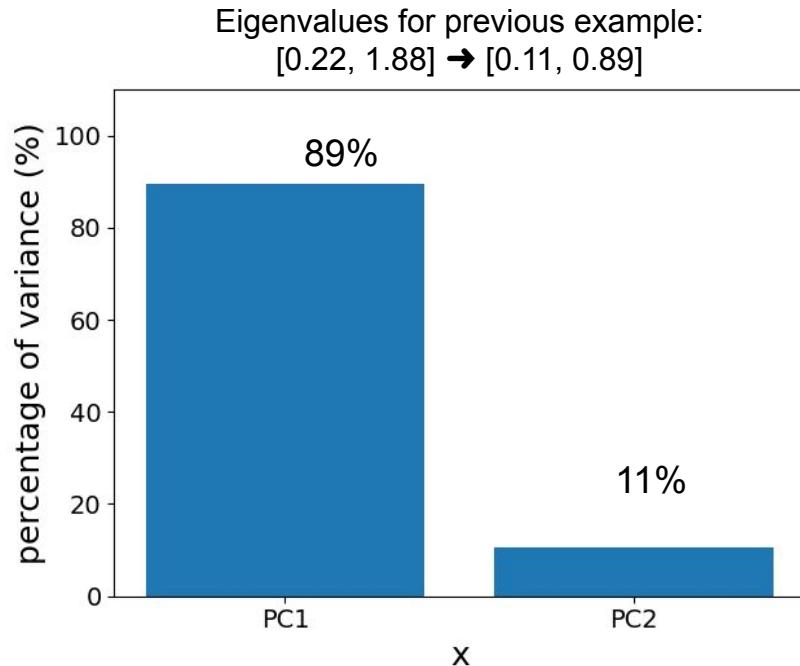
- Percentage of variance that is attributed by each principal component
  - normalized eigenvalues
- Choose  $p$  such that  $p$  largest PCs explain a minimum amount of variance

$$W = \begin{bmatrix} | \\ w_1 \\ | \end{bmatrix}$$

Using only PC1

$$W = \begin{bmatrix} | & | \\ w_1 & w_2 \\ | & | \end{bmatrix}$$

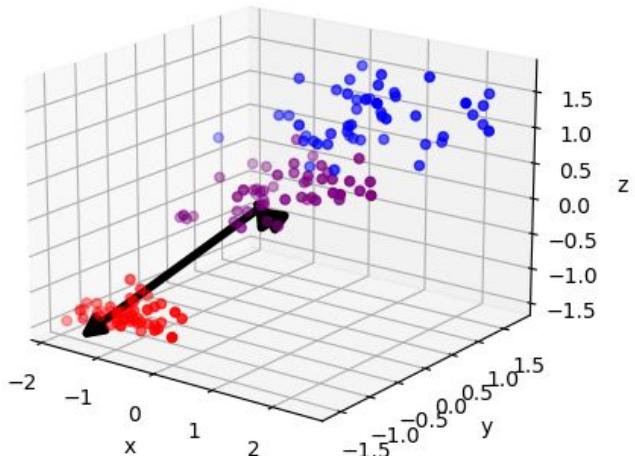
Using PC1 and PC2



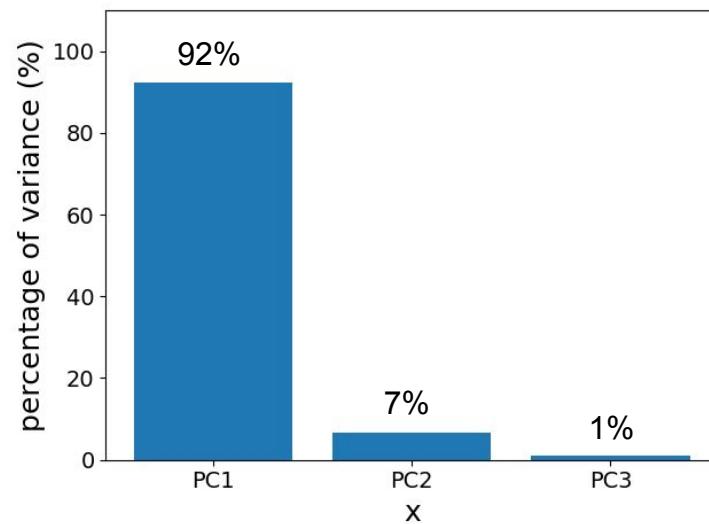
# PCA — Full Example (IRIS dataset)

- IRIS dataset
  - Only 3 (out of 4) features considered — only to allow for easy visualization

Dataset with the 3 principal components



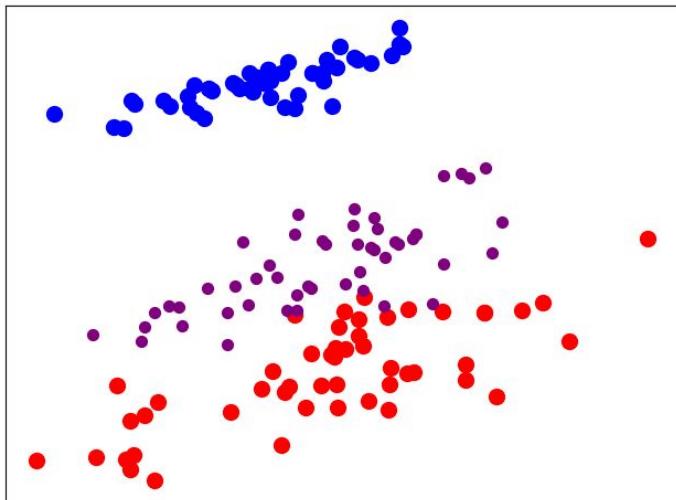
Explained variance of the 3 PCs



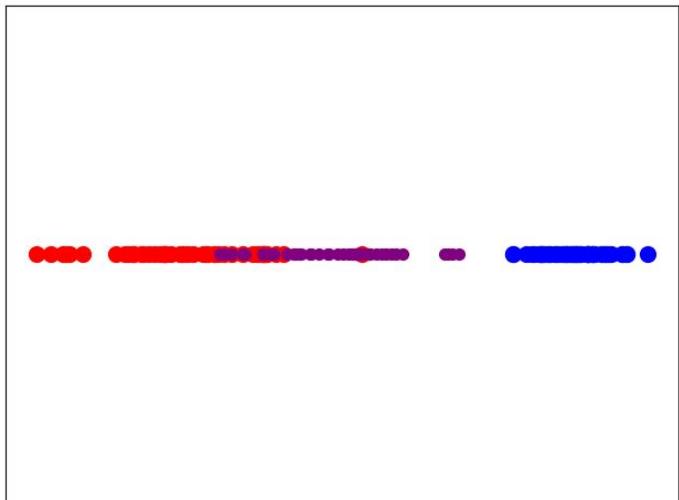
# PCA — Full Example (IRIS dataset)

- Transformation of  $X$  using principal components

Using PC1 and PC2  
(99% of variance explained)



Using only PC1  
(92% of variance explained)



# PCA — Pros & Cons

- **Pros**

- Intuitive — exploit knowledge about correlated and low-variance features
- Can significantly reduce the amount of data
- Improves performance of algorithms and reduces risk of overfitting
- Visualization of high-dimensional data (even just when applied during EDA)

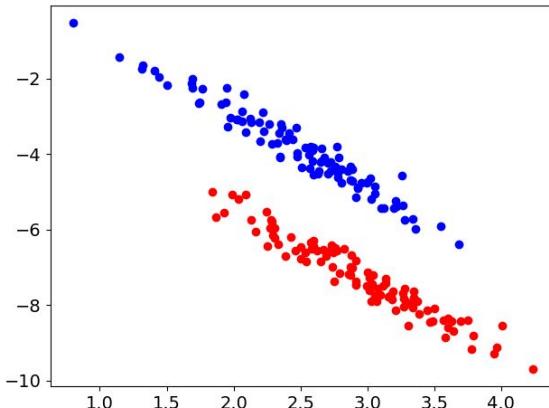
- **Cons**

- Most fundamentally: loss of information
- Assumes linear correlations among features
- Assumes that large variance equals high importance (does not always have to be the case)
- Does not take class labels into account (in case of classification tasks)

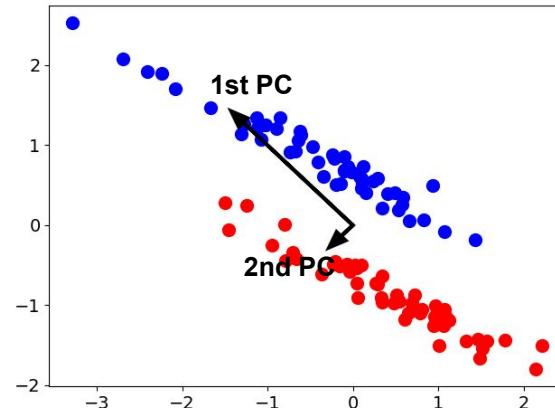
# PCA — Limitation for Classification Datasets

- PCA applied to labelled datasets for classification (pathological example)
  - PCA maximizes w.r.t. the variance of the whole dataset
  - PCA ignores any information from the class labels

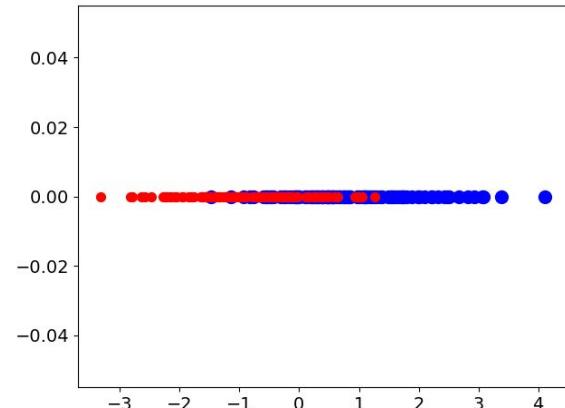
2-class dataset



principal components



transformation using 1st PC



# Outline

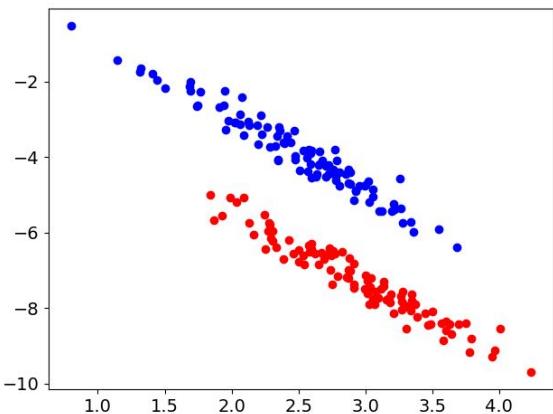
- Dimensionality Reduction
  - Motivation
  - Naive approaches
- Dimensionality Reduction Techniques
  - PCA — Principal Component Analysis
  - LDA — Linear Discriminant Analysis
  - t-SNE — t-distributed Stochastic Neighbor Embedding

# Linear Discriminant Analysis (LDA)

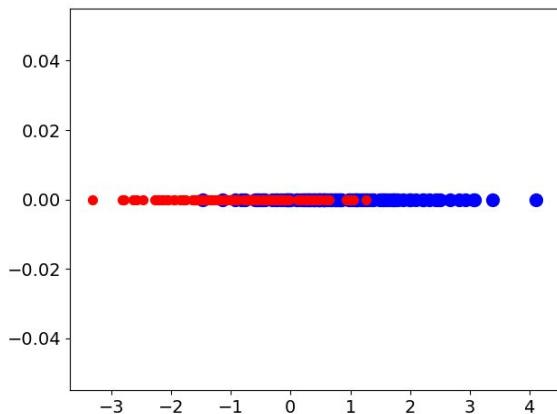
- Similarity to PCA
  - Linear transformation technique
  - Output: Matrix  $W$  to transform dataset  $X$  into a lower-dimensional space
- Main difference: 2 optimization objectives
  - Minimize variance of transformed points within each class  
(recall that PCA maximizes the variance across the whole dataset)
  - Maximize separation between classes

# LDA vs. PCA — Example

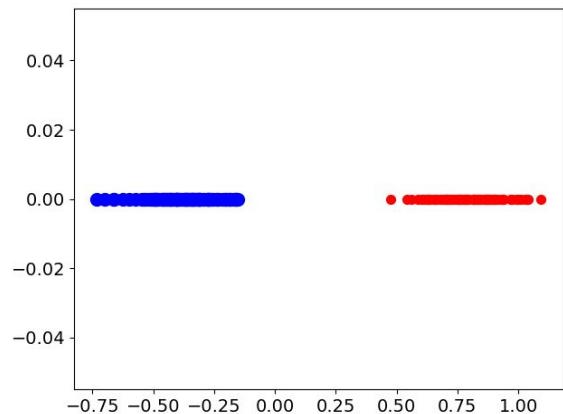
2-class dataset



PCA



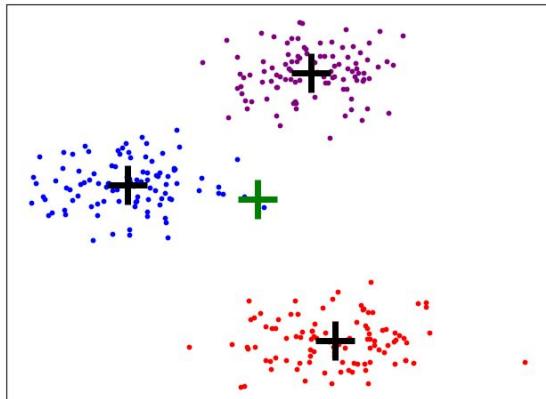
LDA



# LDA Concepts — Between-Class Variance

- Variance of distances between class means and (overall) mean

- $C$  — number of classes
- $\mu^{(i)}$  — class mean vector  
(mean of data points of class  $i$ )
- $\mu$  — (overall) mean vector  
(mean of all data points)



$$\begin{aligned} [m^{(i)} - m]^2 &= [w^T \mu^{(i)} - w^T \mu]^2 \\ &\quad \text{projected mean vectors} \\ &= w^T \underbrace{[\mu^{(i)} - \mu] [\mu^{(i)} - \mu]^T}_{\text{scatter of class means}} w \\ &= w^T S_B^{(i)} w \\ &\quad \downarrow \text{over all classes} \\ &= w^T S_B w, \text{ with } S_B = \sum_{i=1}^C n^{(i)} S_B^{(i)} \end{aligned}$$

# LDA ( $S_B$ ) in Python Code (using numpy library)

```
1 def calc_S_b(X, y):
2     C = np.unique(y) # C = set of class labels
3     d = X.shape[1]   # d = number of features
4
5     mu = np.mean(X, axis=0) # Calculate overall mean
6
7     S_b = np.zeros((d, d)) # Initialize S_b matrix
8
9     for c in C:
10         # n_i number of samples labeled c
11         n_i = X[y==c,:].shape[0]
12         # mu_i mean of samples labeled c
13         mu_i = np.mean(X[y==c], axis=0)
14         # Calculate difference vector
15         # (= mean-centering class means)
16         diff = (mu_i - mu).reshape(-1, 1)
17         # Add scatter of difference vector
18         S_b += n_i * np.dot(diff, diff.T)
19
20     return S_b
21
```

$$= w^T \underbrace{[\mu^{(i)} - \mu] [\mu^{(i)} - \mu]^T}_\text{scatter of class means} w$$

$$= w^T S_B^{(i)} w$$

 over all classes

$$= w^T S_B w, \text{ with } S_B = \sum_{i=1}^C n^{(i)} S_B^{(i)}$$

# LDA Concepts — Within-Class Variance

- Variance of data points of the same class

$$\begin{aligned} \sum_{j=1}^{n^{(i)}} \left[ p_j^{(i)} - m^{(i)} \right]^2 &= \sum_{j=1}^{n^{(i)}} \left[ w^T x_j^{(i)} - w^T \mu^{(i)} \right]^2 \\ &= \sum_{j=1}^{n^{(i)}} w^T \underbrace{\left[ x_j^{(i)} - \mu^{(i)} \right] \left[ x_j^{(i)} - \mu^{(i)} \right]^T}_\text{scatter of data points of class } i w \\ &= w^T S_W^{(i)} w \\ &\quad \downarrow \text{over all classes} \\ &= w^T S_W w, \text{ with } S_W = \sum_{i=1}^C S_W^{(i)} \end{aligned}$$

# LDA ( $S_W$ ) in Python Code (using numpy library)

```
1 def calc_S_w(X, y):
2     C = np.unique(y)    # C = set of class labels
3     d = X.shape[1]      # d = number of features
4
5     S_w = np.zeros((d, d))  # Initialize S_w matrix
6
7     for c in C:
8         # Get all samples labeled c
9         X_c = X[y==c]
10        # Mean-center the data
11        X_c -= np.mean(X_c, 0)
12        # Add scatter matrix of X_c
13        S_w += np.dot(X_c.T, X_c)
14
15    return S_w
```

$$= \sum_{j=1}^{n(i)} w^T [x_j^{(i)} - \mu^{(i)}] [x_j^{(i)} - \mu^{(i)}]^T w$$

$$= w^T S_W^{(i)} w$$

↓ over all classes

$$= w^T S_W w, \text{ with } S_W = \sum_{i=1}^C S_W^{(i)}$$

# LDA — Optimization Objective

Maximize:  $J(w) = \frac{w^T S_B w}{w^T S_W w}$

scatter of projected class means  
scatter of projected data points (per class)

*Generalized Rayleigh Quotient*

→ Generalized eigenvalue problem:  $S_W^{-1} S_B w = J(w)w$

scalar value!

Optimal projection vectors =

eigenvectors of largest the eigenvalues of matrix  $S_W^{-1} S_B$

# LDA — The Math

$$J(w) = \frac{w^T A w}{w^T B w}$$

$$\frac{dJ(w)}{dw} = \frac{\frac{d(w^T A w)}{dw} \cdot w^T B w - w^T A w \cdot \frac{d(w^T B w)}{dw}}{(w^T B w)^2}$$

Quotient Rule

$$= \frac{w^T (A + A^T)(w^T B w) - w^T (B + B^T)(w^T A w)}{(w^T B w)^2}$$

$$\frac{x^T A x}{dx} = x^T (A + A^T)$$

$$= \frac{2Aw(w^T B w) - 2Bw(w^T A w)}{(w^T B w)^2}$$

$X$  is symmetric  $\rightarrow X + X^T = 2X$

$$\frac{dJ(w)}{dw} = \frac{2}{w^T B w} (Aw - J(w)Bw) \stackrel{!}{=} 0 \quad \Leftrightarrow \quad \begin{aligned} Aw &= \overbrace{J(w)Bw}^{\text{scalar value!}} \\ B^{-1}Aw &= J(w)w \end{aligned}$$

→ Generalized Eigenvalue problem!

# Note on the Number of Eigenvectors

- Definition of  $S_B$  includes two constraints

- $S_B$  is the sum of  $C$  matrices of rank 1 or less

- The mean  $\mu$  is constraint by  $\mu = \frac{1}{C} \sum_{i=1}^C \mu_i$

→  $S_B$  has rank of  $(C-1)$  or less

→ only  $(C-1)$  eigenvectors are non-zero! (the respective eigenvalues are 0)

**Note:** In practice, these remaining  $d-c+1$  eigenvalues are only very close to zero due to floating pointing imprecisions

# LDA — Algorithm

1. Calculate mean vectors  $\mu$  and  $\mu^{(i)}$  for all C classes
2. Calculate scatter matrices  $S_w$  and  $S_B$
3. Calculate eigenvectors and eigenvalues of  $S_w^{-1}S_B$
4. Select  $p$  eigenvectors  $w_p$  with the largest eigenvalues

$$\mathbf{W} = \begin{bmatrix} | & | & \dots & | \\ w_1 & w_2 & \dots & w_p \\ | & | & \dots & | \end{bmatrix} \quad \text{with } p \leq C-1$$

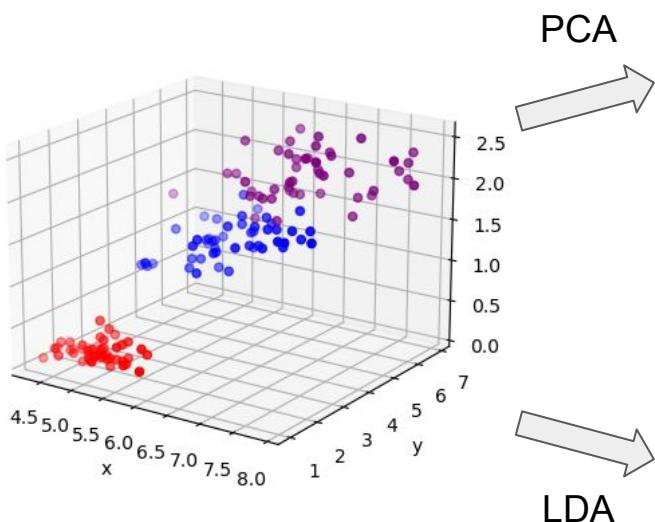
5. Project dataset X into new space via  $XW$

# LDA in Python Code (using numpy library)

$$S_W^{-1} S_B$$

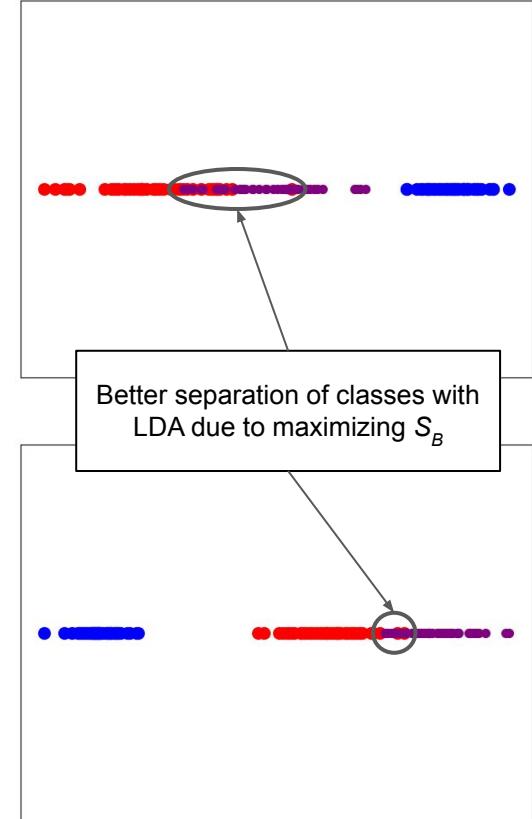
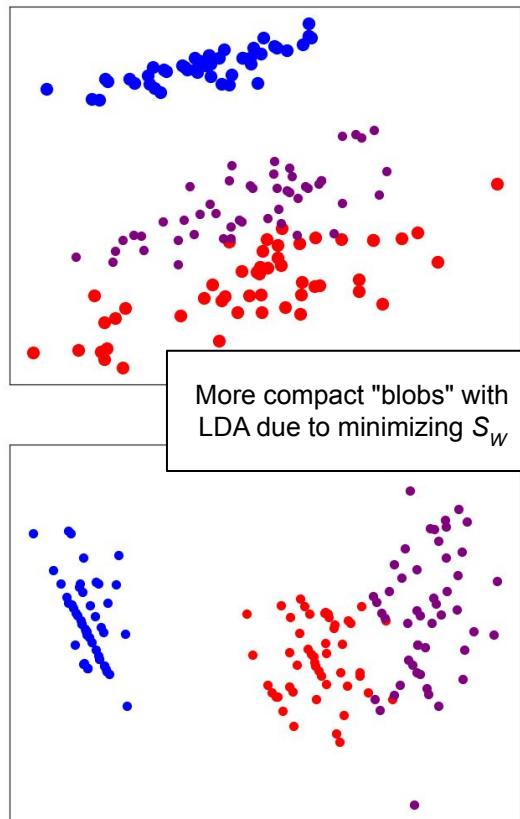
```
1 def lda(X, y, num_pc=None):
2     # Calculate scatter matrices
3     S_w = calc_S_w(X, y)
4     S_b = calc_S_b(X, y)
5     # Calculate all eigenvectors and eigenvalues
6     eigenval, eigenvec = np.linalg.eig(np.linalg.inv(S_w).dot(S_b))
7     # find the the num_pc largest eigenvalues
8     top_idx = np.argsort(eigenval)[::-1][:num_pc]
9     # Create transformation matrix W using eigenvectors with the largest eigenvalues
10    W = eigenvec[:, top_idx]
11    # Return the transformed data
12    return np.dot(X, W)
```

# LDA — Full Example (IRIS dataset)



PCA

LDA



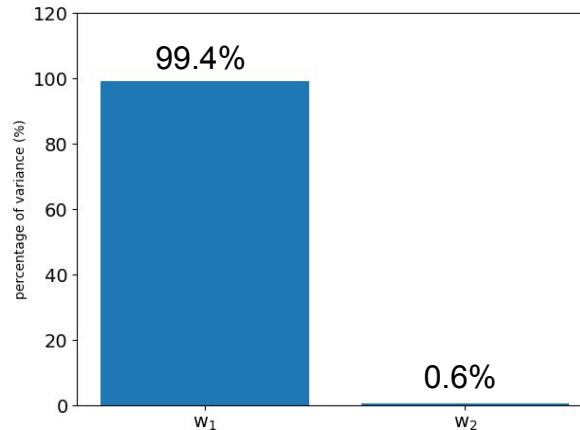
# LDA — Full Example (IRIS dataset)

- Resulting  $d$  eigenvalues (sorted)

|    |               |
|----|---------------|
| 1. | 26.9          |
| 2. | 0.17          |
| 3. | $2.03e^{-15}$ |

$\left. \right\} C-1 = 3-1 = 2$  non-zero eigenvalues  
 $\left. \right\} d-C+1 = 3-3+1 = 1$  zero eigenvalues

- Choosing  $p \leq C-1$  using Explained Variance Ratio



→  $p=1$  a good choice

# LDA — Pros & Cons

- Pros

- Intuitive extension to PCA yielding similar benefits  
(reduction in amount of data, reduced risk of overfitting, visualization, etc.)
- Consideration of class labels (generally more suitable than PCA for labelled dataset)

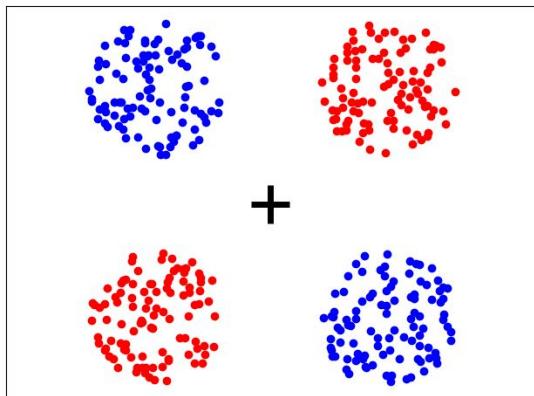
- Cons

- Similar cons as PCA (loss of information, assumes linear correlations, etc.)
- Assumes unimodal Gaussian distributions
- Assumes that means are the most discriminant features

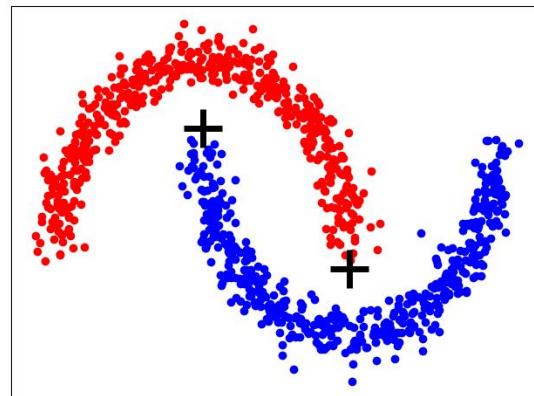
# LDA — Problematic Cases

- Data distributions that are (significantly) non-unimodal Gaussian

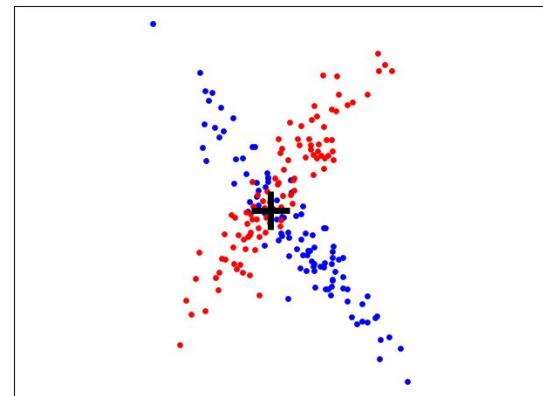
bimodal with  $\mu = \mu_1 = \mu_2$



non-Gaussian



unimodal but  $\mu = \mu_1 = \mu_2$



# Outline

- Dimensionality Reduction
  - Motivation
  - Naive approaches
- Dimensionality Reduction Techniques
  - PCA — Principal Component Analysis
  - LDA — Linear Discriminant Analysis
  - t-SNE — t-distributed Stochastic Neighbor Embedding

# t-Distributed Stochastic Neighbor Embedding (t-SNE)

- t-SNE — Non-linear dimensionality reduction technique
  - Unsupervised approach (independent from any kind of class labels)
  - Iterative algorithm:
    - 1) Start with random lower-dimensional representation  $Y$
    - 2) Change  $Y$  until a loss function converges to a minimum
- Intuition behind t-SNE
  - Convert Euclidean distances in  $X$  and  $Y$  to conditional probabilities  
(e.g., if data points  $x_i$  and  $x_j$  are close  $\rightarrow$  conditional probability  $p_{ij}$  should be high)
  - Iteratively change  $Y$  such that both probability distributions become more similar
- Optimization objective: Points that are close in  $X$  are also close in  $Y$

# t-SNE — Convert Euclidean Distances to Cond. Probs

- Mathematical formulation
  - For data points in  $X$

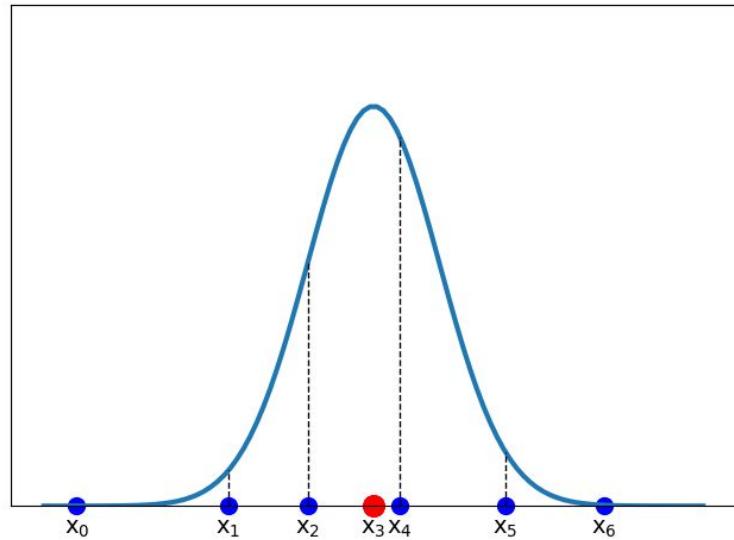
$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}$$

- Visual explanation (for  $i = 3$ )
  - Assume  $d$ -dim Gaussian centered at  $x_3$
  - Calculate  $p_{j|3}$  proportional to Gaussian (proportional to heights of dashed lines from each point  $x_j$ )

| $p_{4 3}$ | $p_{2 3}$ | $p_{5 2}$ | $p_{1 3}$ | $p_{6 3}$ | $p_{0 3}$ |
|-----------|-----------|-----------|-----------|-----------|-----------|
| 0.84      | 0.16      | ~0.0      | ~0.0      | ~0.0      | ~0.0      |

## Interpretation

$p_{j|i}$  is the probability that  $x_i$  would pick  $x_j$  as neighbor



# t-SNE

- Calculate joint probabilities

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

Joint probabilities  $P = \{p_{ij}\}$  for previous example

$$\begin{bmatrix} 0 & 0.071 & 0 & 0 & 0 & 0 & 0 \\ 0.071 & 0 & 0.09 & 0 & 0 & 0 & 0 \\ 0 & 0.09 & 0 & 0.057 & 0.09 & 0 & 0 \\ 0 & 0 & 0.057 & 0 & 0.129 & 0.001 & 0 \\ 0 & 0 & 0.009 & 0.129 & 0 & 0.025 & 0 \\ 0 & 0 & 0 & 0.001 & 0.025 & 0 & 0.117 \\ 0 & 0 & 0 & 0 & 0 & 0.117 & 0 \end{bmatrix}$$

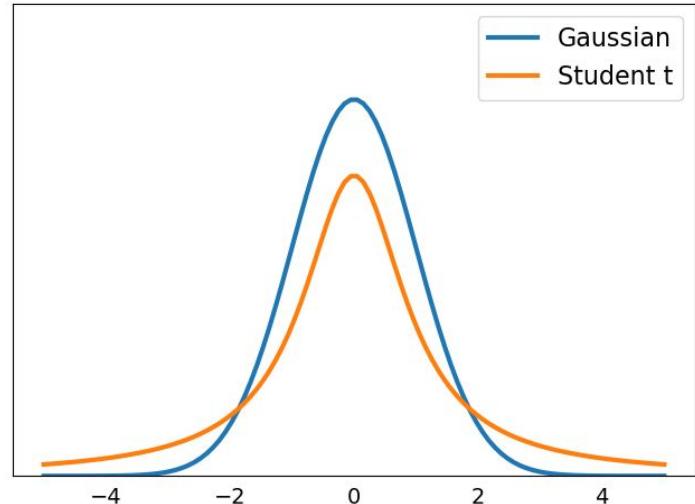
Assume  $Q = \{q_{ij}\}$  being the joint probabilities of data points  $y_i$  in  $Y$

→ How to modify all  $y_i$  such that  $P \approx Q$ ?

# t-SNE

- t-SNE uses a Student t distribution
  - Heavier tails yield better results in practice
  - 1 degree of freedom

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_i - y_j\|^2)^{-1}}$$



- Loss function: Kullback-Leibler divergence between P and Q

$$L = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

**Note:** The KL divergence is a measure of how one probability distribution is different from another probability distribution

# t-SNE — Minimizing Loss

- Minimize  $L$  using Gradient Descent

$$L = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$\frac{\partial L}{\partial y_i} = \dots = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$

# t-SNE — Basic Algorithm

**Input** : Dataset  $X$ , number of iterations  $T$ , learning rate  $\eta$

**Initialization** : Sample  $Y^{(0)} = \{y_1, y_2, \dots, y_n\}$  from  $\mathcal{N}(0, 10^{-4}\mathbf{I})$

Calculate  $p_{j|i}$  and  $p_{ij}$  for all  $(x_i, x_j)$ -pairs

**for**  $t = 1$  **to**  $T$  **do**

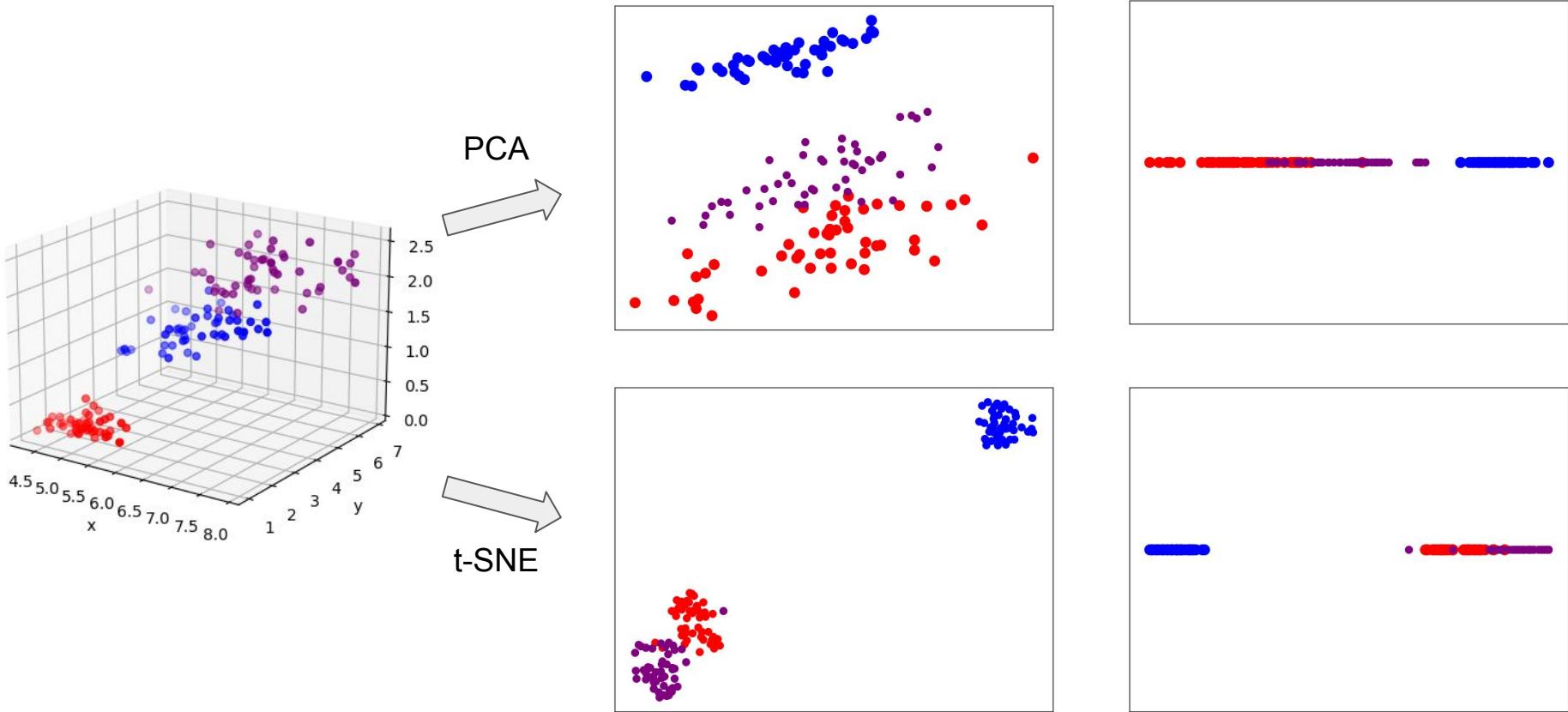
    Calculate  $q_{ij}$  for all  $(y_i, y_j)$ -pairs

    Calculate gradients  $\frac{\partial L}{\partial y_i}$

    Update  $y_i^t \leftarrow y_i^{t-1} - \eta \frac{\partial L}{\partial y_i}$

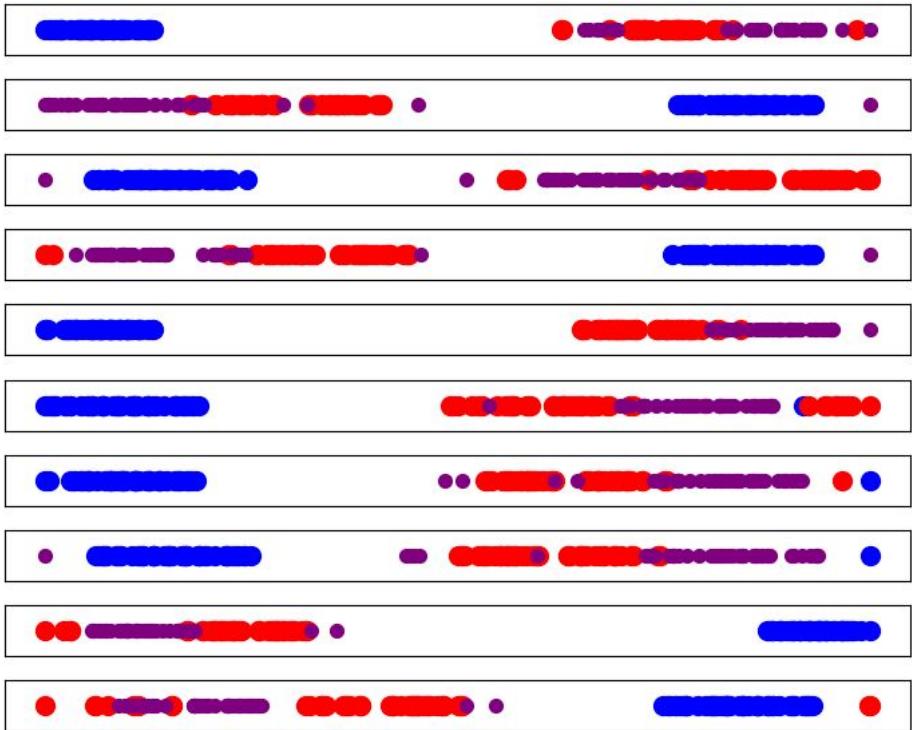
**return**  $Y^{(T)}$

# t-SNE — Full Example (IRIS dataset)



# t-SNE — Non-Determinism

- t-SNE is non-deterministic
  - $Y^{(0)}$  is randomly sampled
  - Different runs will generally yield different projections
  - In practice, perform multiple runs to get an understanding of the data

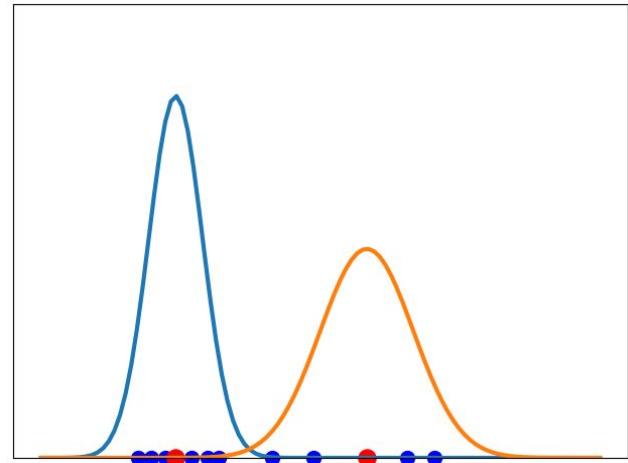


# t-SNE — Calculating $p_{j|i}$ (implementation detail)

- How to pick the values for  $\sigma_i$ ?

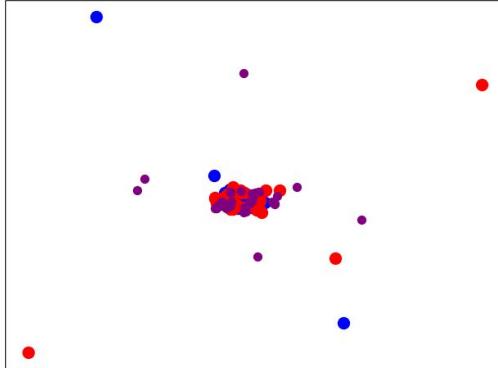
$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}$$

- Intuition: Set  $\sigma_i$  based on density around  $x_i$ 
  - High density → smaller  $\sigma_i$  / Low density → larger  $\sigma_i$
  - Controls how many  $x_j$  with effective  $p_{j|i}$
- Calculate best based  $\sigma_i$  on hyperparameter perplexity
  - Larger perplexity: more neighbors have effective  $p_{j|i}$
  - Common perplexity values in practice: 5..50

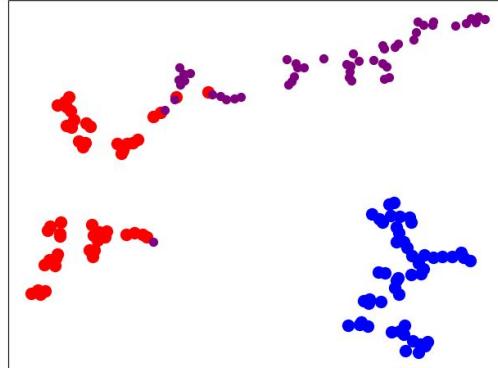


# t-SNE — Effects of Perplexity Parameter

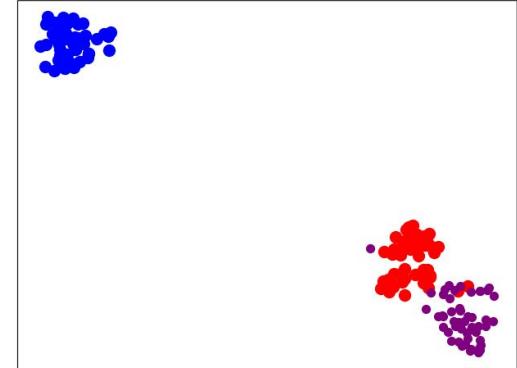
perplexity = 1



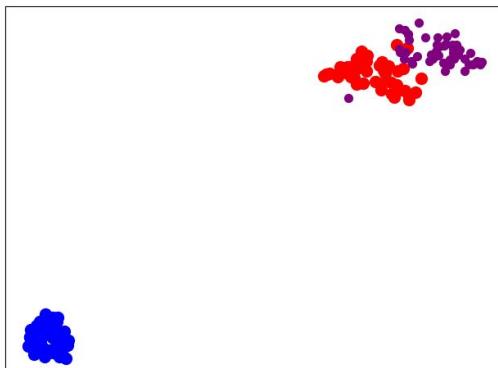
perplexity = 5



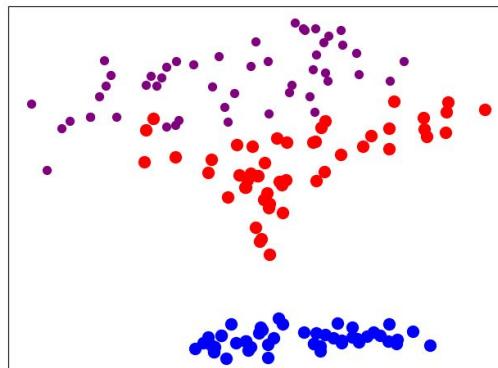
perplexity = 25



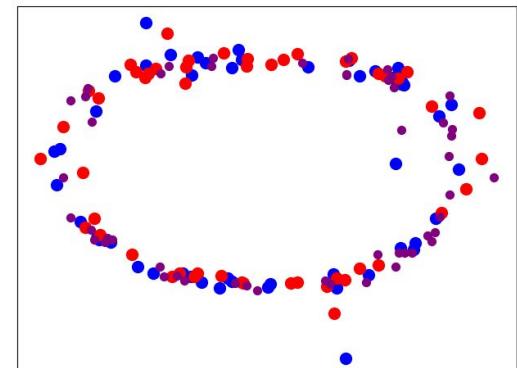
perplexity = 50



perplexity = 100



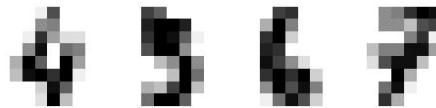
perplexity = 200



# t-SNE — Pros & Cons

- Pros
  - Can handle non-linear data
  - Works very well for data visualization
- Cons
  - Computational very expensive on very high-dimensional data (compared to, e.g., PCA)
  - Non-deterministic behavior; might need multiple runs
  - Several hyperparameters affecting the output (perplexity, learning rate, number of iterations, initialization)

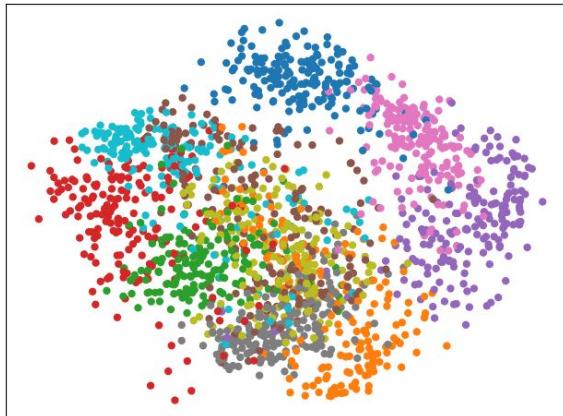
# Example — Digits Dataset



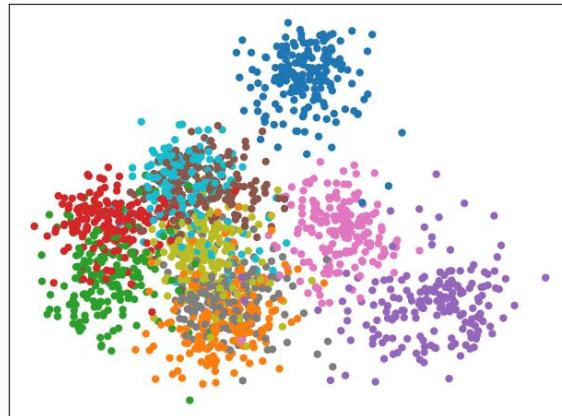
- **Digits Dataset**

- 1,797 handwritten digits 0, 1, 2, ..., 9  
(~180 samples for class)
- 8×8 pixels → 64 features  
(integer grayscale value 0..16)

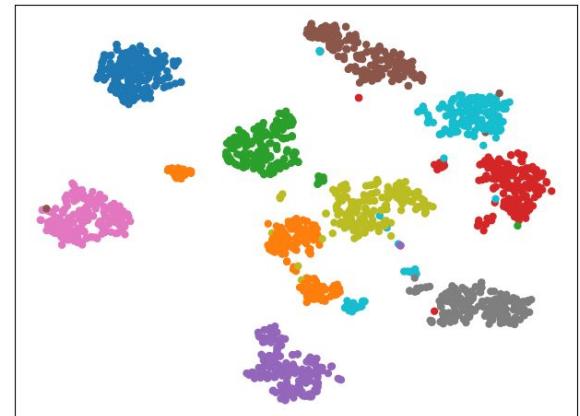
PCA



LDA



t-SNE



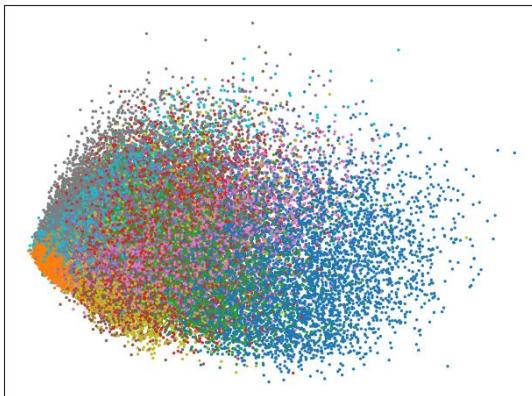
# Example — MNIST

- Digits Dataset

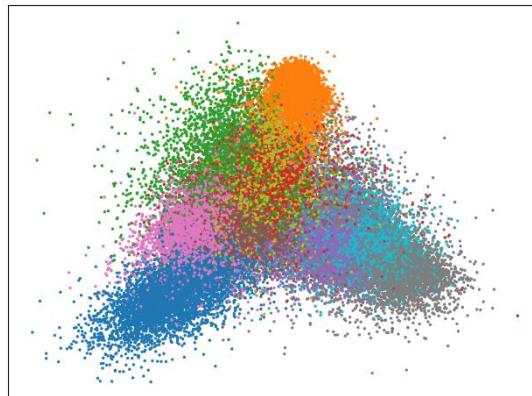
- 60k handwritten digits 0, 1, 2, ..., 9  
(~6k samples for class)
- 28×28 pixels → 784 features  
(integer grayscale values 0..255)

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

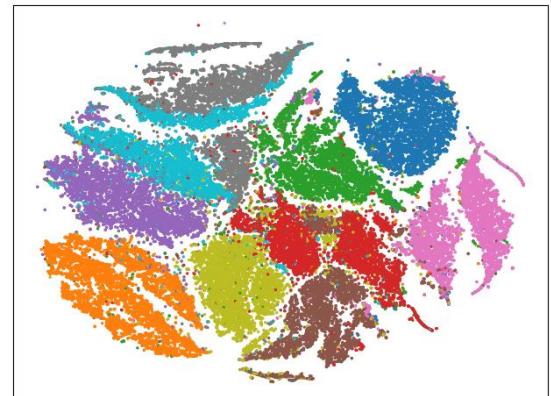
PCA (4.9 sec)



LDA (5.1 sec)

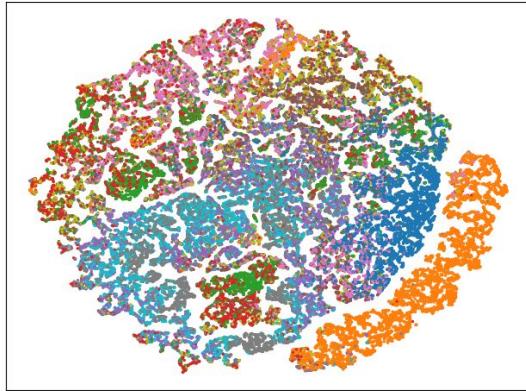


t-SNE (~58 min!)

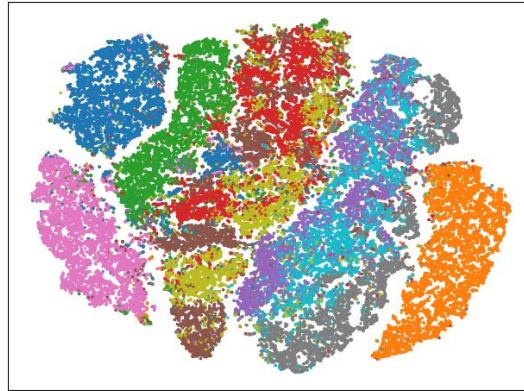


# PCA + t-SNE

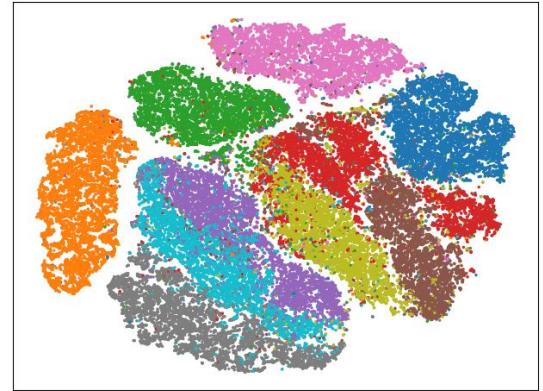
PCA/3 + t-SNE (2:04 min)



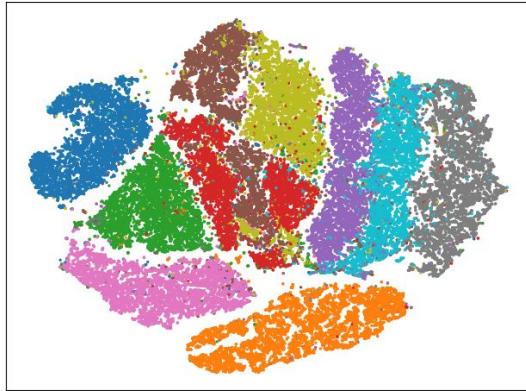
PCA/7 + t-SNE (2:30 min)



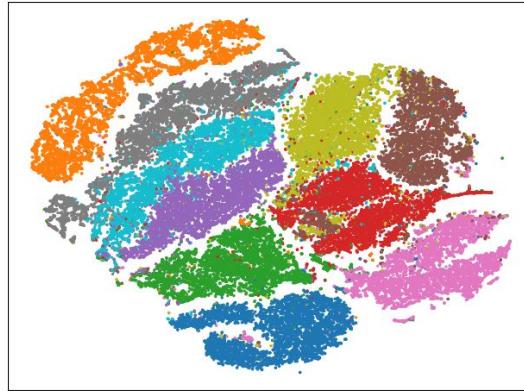
PCA/14 + t-SNE (2:56 min)



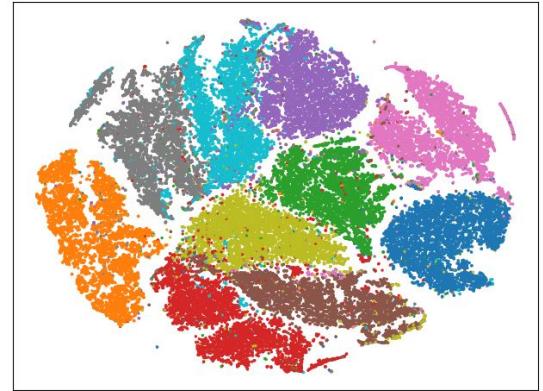
PCA/28 + t-SNE (4:31 min)



PCA/56+ t-SNE (8:16 min)



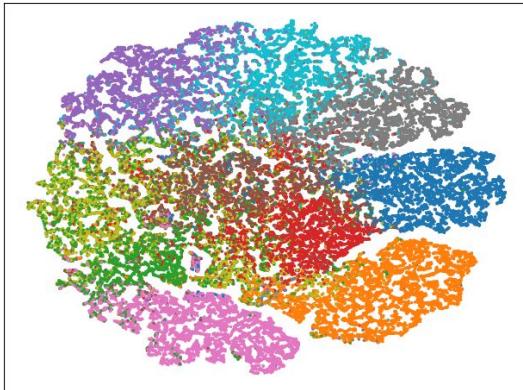
PCA/128 + t-SNE (14:00 min)



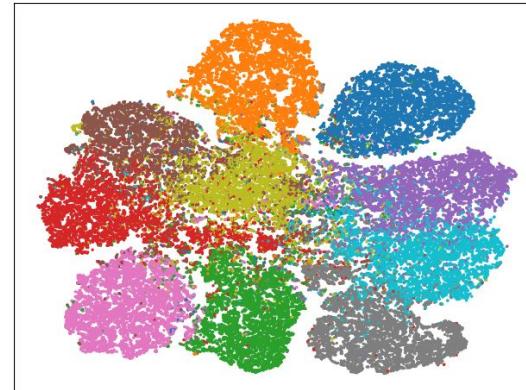
# LDA + t-SNE

- Recall restriction of LDA (compared to PCA)
  - Only up to ( $C-1$ ) non-zero eigenvalues
  - Range of dimensionality reduction by:  $1 \leq p \leq 9$  for MNIST

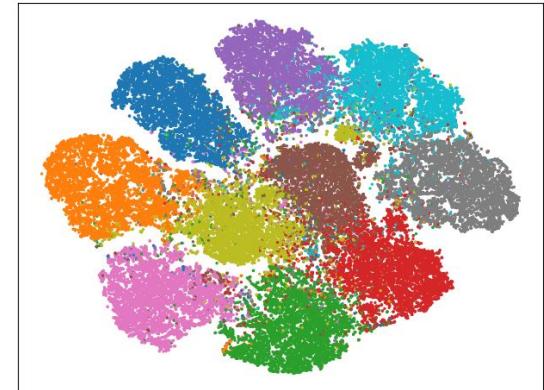
**LDA/3 + t-SNE** (2:30 min)



**LDA/6+ t-SNE** (2:41 min)



**LDA/9 + t-SNE** (2:55 min)

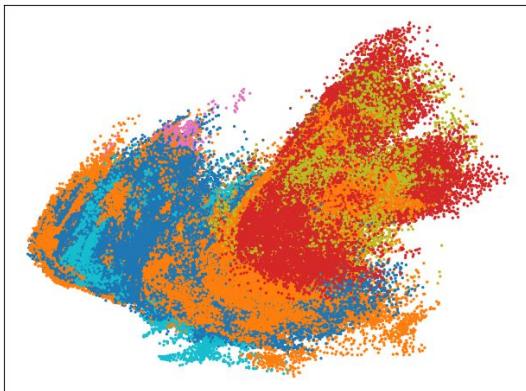


# Example — Forest Cover Type

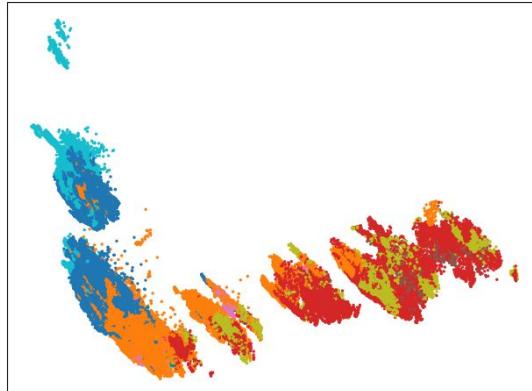
- Forest cover type classification dataset

- 581k samples of 30×30m cells with 1 of 7 forest types  
(Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir, Krummholz)
- 54 features (elevation, aspect, slope, etc.)

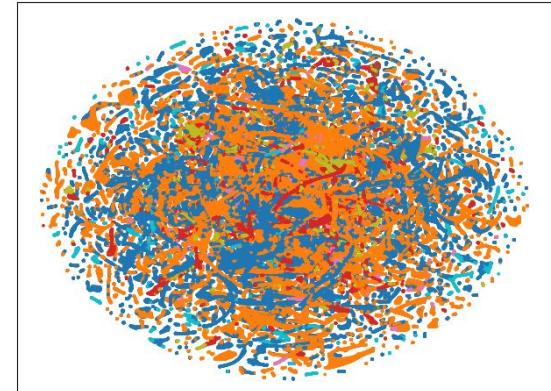
PCA (1.2sec)



LDA (3.2sec)

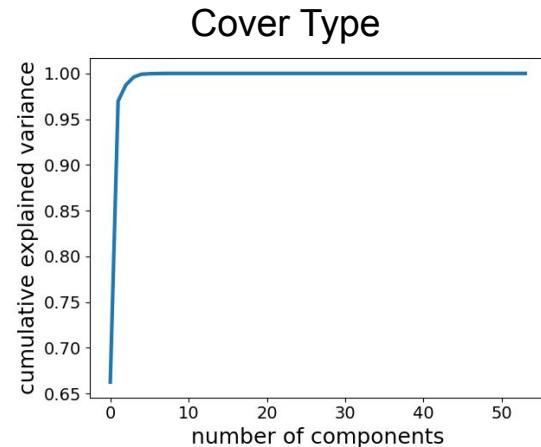
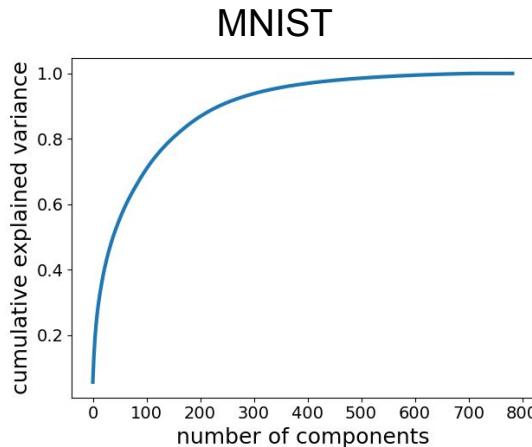
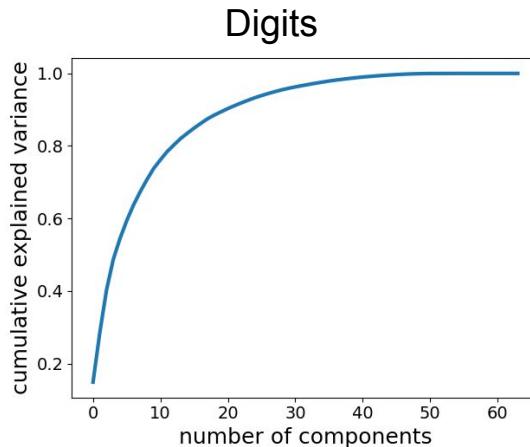


t-SNE (~30min)



# Example — Explained Variance (PCA)

- Cumulative explained variance
  - Data distribution more homogeneous for Digits and MNIST
  - Cover type dataset with many features have only 0 values (low/no variance)



# Summary

- Problem: high-dimensional dataset (i.e., large number of features)
  - Higher risk of overfitting
  - Higher computational costs
- Two basic types of countermeasures
  - Feature selection — "manually" remove subset of features
  - Feature extraction — create new features based on original features
- Dimensionality reduction techniques for feature extraction
  - Linear techniques: PCA (unsupervised) and LDA (supervised)
  - Probabilistic techniques: t-SNE

# **CS5228: Knowledge Discovery and Data Mining**

## Lecture 11 — Data Stream Mining

# Course Logistics

- Reminder for submission deadlines
  - A4: Nov 14, 11.59 pm
  - Project Report: Nov 4, 11.59 pm
- Project submission
  - Submission = project report (PDF, max 8 pages) + source code
  - Only 1 submission per team needed
  - Submission files should include team name
- Last Lecture Quiz
  - Friday, Nov 15, ~19:15 (30 min)
  - MCQs/MRQs, Lectures 7-11

# Quick Recap — Graph Mining

- **Community Detection**

- Identification of "interesting" subgraphs  
(≈ nodes in subgraph more tightly compared to other nodes)
- Similar to the task of clustering  
(clustering algorithms can be adopted to find communities)

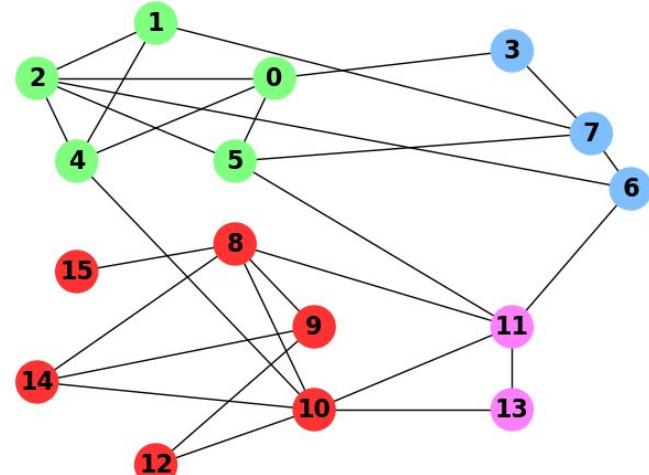
- No single definition of "community"

→ Many algorithms for community detection

- Similarity between nodes (e.g., AGNES)
- Density-based (modularity + Louvain algorithm)
- Split-based (Edge Betweenness, Min-Cut)

Girvan Newman algorithm

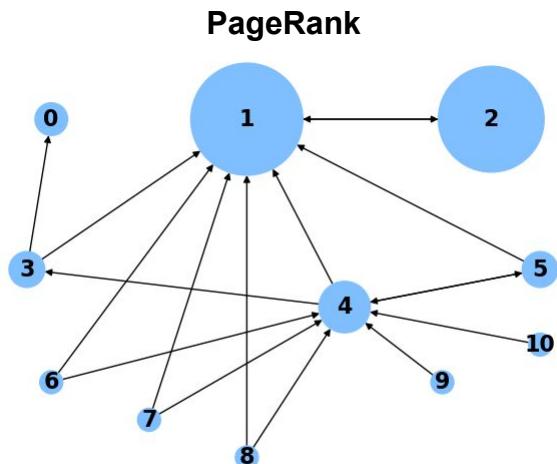
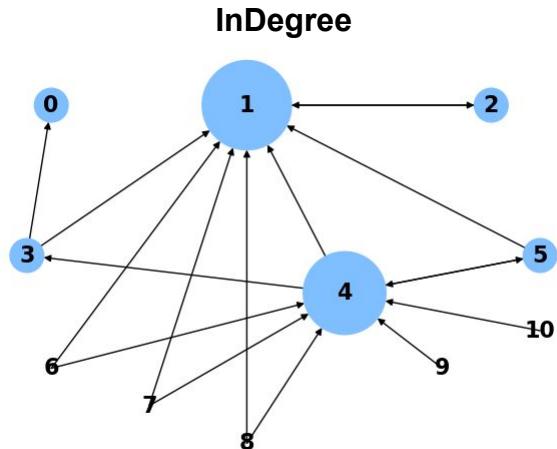
Karger's algorithm



Shared focus: **connectedness**

# Quick Recap — Graph Mining

- **Centrality = importance of a node**
  - Based on a node's topological position in a graph
  - Different centrality measures focusing on different topological features
  - Not all measures applicable to all types of graphs
- **Popular centrality measures covered**
  - Local measures (Degree, InDegree, OutDegree)
  - Eigenvector-based measures (Eigenvector Centrality, PageRank)
  - Distance-based or path-based measures (Closeness, Betweenness)



# Outline

- **Motivation**
  - Basic setup
  - Example Applications
- **Core Techniques**
  - Sampling
  - Filtering
  - Counting (distinct items)
- **Summary**

# Data Stream Mining

- Data Mining so far
    - Access to complete dataset (at the same time)
    - Virtually unlimited storage and computing resources  
(also: runtime of algorithms typically not that important, compared to the results)
    - Support of arbitrary complex patterns
  - Now: data items arrive one-by-one  
in real time...like a stream
    - All data never fully available
    - Often very high arrival speeds
    - Often limited amount of resources
    - Often time-critical decisions  
(common: execution in main memory only)
- 
- Focus on simple patterns (but not too simple)

# Quick Quiz

Given is a stream of temperature sensor values in °C.

What is the only **non-trivial** pattern to monitor?

A

temp > 100 °C

B

Average of all temperature values

C

Median of all temperature values

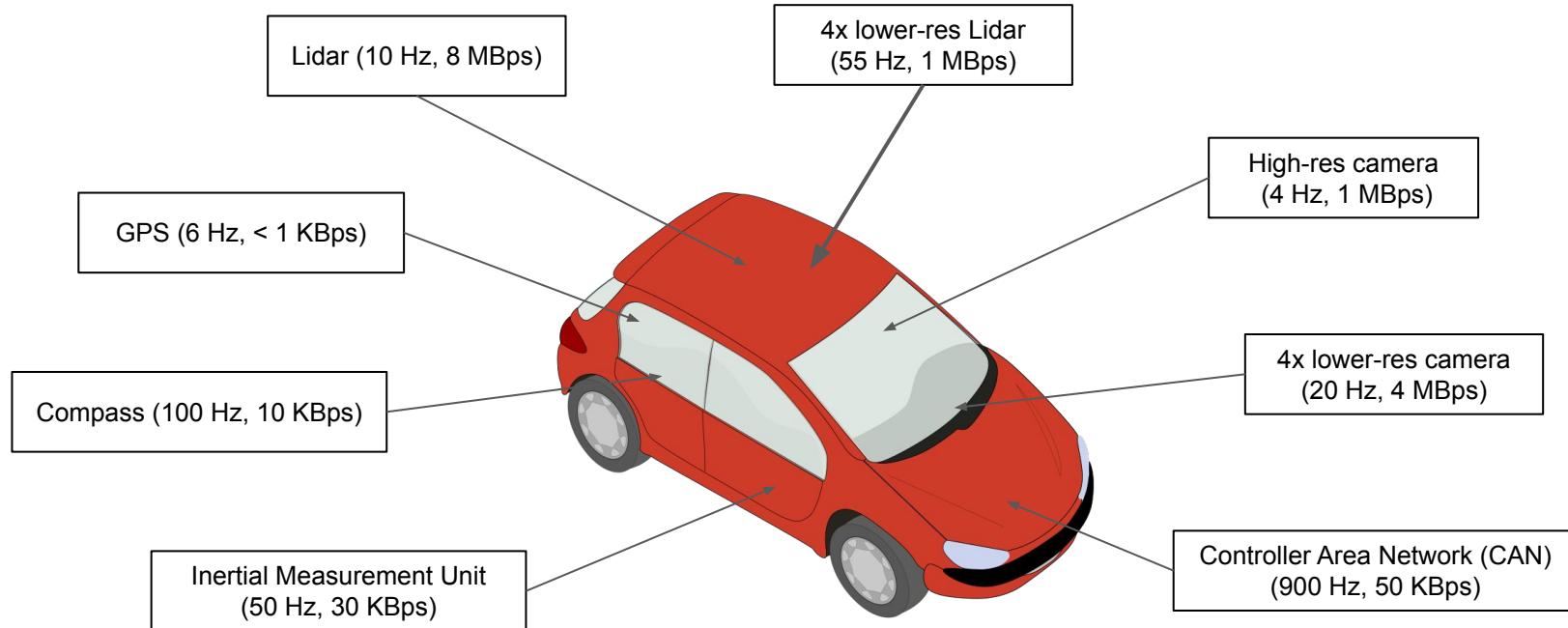
D

Maximum range of all temperature values

# (Self-Driving) Cars

- Sensor data generated during a 40-minutes trip: 30 GB (13 MBps)

Source: [Exploring big volume sensor data with Vroom](#) (Moll et al., 2017)



# Smart Cities

- Example: Lamppost-as-a-Platform (LaaP)

Source: <https://www.developer.tech.gov.sg/technologies/sensor-platforms-and-internet-of-things/lamppost-as-a-platform>



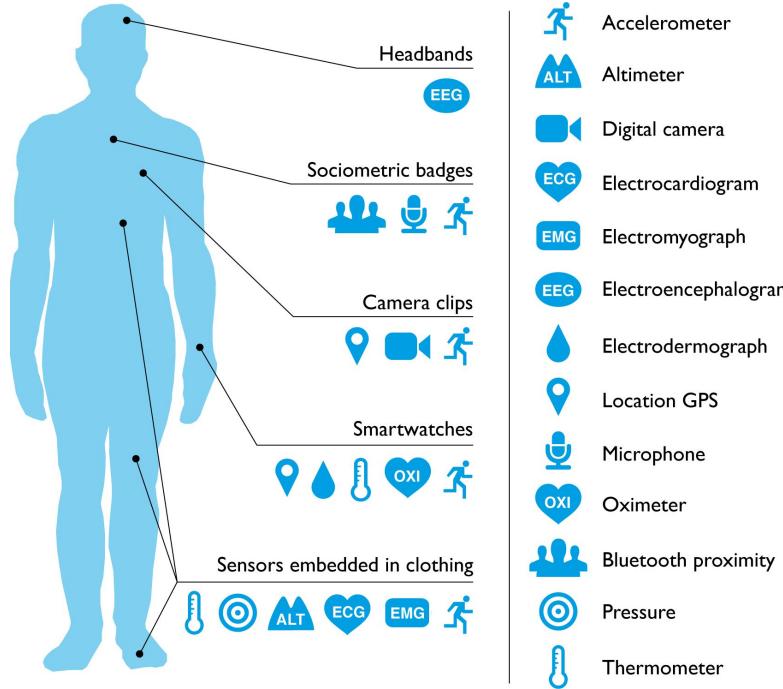
- Camera
- Temperature
- Humidity
- Gas (e.g., carbon monoxide)
- Air quality
- Rain

# Health Monitoring

- Consumer Health Wearables

Source: <https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.1001953>

- Smartphones
- Smartwatches
- Fitness bands
- Body cameras



# Social Media

- Example: Daily data volume on Facebook

Source: <https://blog.wishpond.com/post/115675435109/40-up-to-date-facebook-facts-and-stats>

- 4.5 billion Likes
- 17 billion location-tagged posts
- 350 million photos uploaded
- 4.75 billion items shared
- 10 billion messages sent



# Quick Sidenotes

- Covered techniques and algorithms not specific to streams
  - Applicable to many use cases involving large volumes of data
- Not of interest here: sliding window approaches
  - Keep the  $k$  most recent data items
  - Ignore all older items (delete from window)

} → window = complete dataset (snapshot)
- Commonly used throughout the lecture: hash functions
  - We will assume "well-behaved" hash functions  
(same input → same output, minimize duplication, avalanche effect)
  - No deeper discussion what this means here

# Outline

- Motivation
  - Basic setup
  - Example Applications
- Core Techniques
  - Sampling
  - Filtering
  - Counting (distinct items)
- Summary

# Sampling

- Sampling — basic definition

- Process of selecting members of a population of interest (e.g., HDB residents)
- Consideration of whole population typically impractical (e.g., too costly to survey all HDB residents)
- Goal: statistical analysis of population (e.g., average happiness, most common complaints)

- Sampling data streams

- Population = stream of incoming data items
- Time and/or resource constraint generally make it impossible to consider all data items
- Relevant patterns based on statistical analysis



→ Core challenge: How to get a representative sample?

# Problems with Naive Approach

- Toy example setup
  - Stream of search queries — items are tuples (user, query, time)
  - Goal: Fraction of queries issued more than once (here: twice) in the last 24h (by the average user)
  - Restriction only 10% of all tuples should be stored
- Naive approach: Store latest tuple with a probability of 1/10
  - Let  $s$  = number of time a user issued a query **once**
  - Let  $d$  = number of time a user issued a query **twice**

**Correct estimate:**  
(assuming all tuples)

$$\frac{d}{s + d}$$

**Estimate based on  
naive sample:**

$$\frac{d}{10s + 19d}$$

# Problems with Naive Approach — Explanation

- The issue: of  $d$  queries issued twice

- $d/100$  will be both in the sample

$$\left(\frac{1}{10}\right) \left(\frac{1}{10}\right) \cdot d = \frac{1}{100}d$$

- $18d/100$  will be only once in the sample

$$2 \cdot \left(\frac{1}{10}\right) \left(\frac{9}{10}\right) \cdot d = \frac{18}{100}d$$

- Fractions of queries issued twice in the sample:

$$\frac{\frac{1}{100}d}{\frac{1}{100}d + \frac{18}{100}d + \frac{1}{10}s} = \frac{d}{10s + 19d}$$

# Sample with a Given Probability

- General Approach
  - Given: items/tuples with  $n$  components, e.g., (user, query, time)
  - Select subset of components as key on which the selection of the sample is based  
(for toy example: key = "user" → consider only 10% of users instead of arbitrary tuples)
  - Choice of key depends on goal of analysis
- Question: How to obtain a sample consisting of any fraction  $a/b$  of keys?  
(in other words: How to decide whether to keep or discard a new tuple?)
- Common implementation via hash function
  - Define hash function  $h$  that maps  $h(key)$  in to  $b$  buckets  $1..b$
  - For each new tuple, if  $h(key) \leq a$  (with  $a \leq b$ ), add tuples to sample

# Sample with a Given Size Limit — Reservoir Sampling

- Goal: Maintain a uniform random sample of fixed size  $B$ 
  - Uniform random = each item has the same probability to be sampled
  - Allows to approximate basic statistics such as mean, variance, median, etc.
- Basic algorithm
  - Input stream of items  $\{a_1, a_2, \dots\}$
  - Maximum reservoir size  $B$
  - It can be shown that each item in the reservoir was sampled with the same probability  $B/t$  (at any time  $t$ )

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
- 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

# Reservoir Sampling — Example

- Initialization:  $t = 0, B = 3$

Stream:



|         |          |
|---------|----------|
| $t = 1$ | <b>A</b> |
| $t = 2$ | <b>B</b> |
| $t = 3$ | <b>C</b> |
| $t = 4$ | <b>A</b> |
| $t = 5$ | <b>C</b> |
| $t = 6$ | <b>B</b> |
| $t = 7$ | <b>B</b> |
| $t = 8$ | <b>A</b> |
| $t = 9$ | <b>C</b> |

Reservoir:

|  |
|--|
|  |
|  |
|  |

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
  - 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

# Reservoir Sampling — Example

- $t = 3$

Stream:

|         |   |
|---------|---|
| $t = 1$ | A |
| $t = 2$ | B |
| $t = 3$ | C |
| $t = 4$ | A |
| $t = 5$ | C |
| $t = 6$ | B |
| $t = 7$ | B |
| $t = 8$ | A |
| $t = 9$ | C |



Reservoir:

|   |
|---|
| A |
| B |
| C |

Just fill up the reservoir...

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
- 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

# Reservoir Sampling — Example

- $t = 4$

Stream:

|  |   |
|--|---|
| $t = 1$  | A |
| $t = 2$  | B |
| $t = 3$  | C |
|  $t = 4$ | A |
| $t = 5$  | C |
| $t = 6$  | B |
| $t = 7$  | B |
| $t = 8$  | A |
| $t = 9$  | C |

Reservoir:

|                |
|----------------|
| A              |
| B              |
| <del>C</del> A |

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
- 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

Probability  $B/t = 3/4$

→ Randomized decision: add  $a_4 = A$  to reservoir

→ Replace random element — here  $B_3 = C$  with A

# Reservoir Sampling — Example

- $t = 5$

Stream:

|  |   |
|--|---|
| $t = 1$  | A |
| $t = 2$  | B |
| $t = 3$  | C |
| $t = 4$  | A |
|  $t = 5$ | C |
| $t = 6$  | B |
| $t = 7$  | B |
| $t = 8$  | A |
| $t = 9$  | C |

Reservoir:

|                |
|----------------|
| A              |
| <del>B C</del> |
| A              |

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
- 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

Probability  $B/t = 3/5$

→ Randomized decision: add  $a_5 = C$  to reservoir

→ Replace random element — here  $B_2 = B$  with C

# Reservoir Sampling — Example

- $t = 6$

Stream:

|  |          |
|--|----------|
| $t = 1$  | <b>A</b> |
| $t = 2$  | <b>B</b> |
| $t = 3$  | <b>C</b> |
| $t = 4$  | <b>A</b> |
| $t = 5$  | <b>C</b> |
|  $t = 6$ | <b>B</b> |
| $t = 7$  | <b>B</b> |
| $t = 8$  | <b>A</b> |
| $t = 9$  | <b>C</b> |

Reservoir:

|          |
|----------|
| <b>A</b> |
| <b>C</b> |
| <b>A</b> |

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
- 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

Probability  $B/t = 3/6$

→ Randomized decision: discard  $a_6 = B$

# Reservoir Sampling — Example

- $t = 7$

Stream:

|         |   |
|---------|---|
| $t = 1$ | A |
| $t = 2$ | B |
| $t = 3$ | C |
| $t = 4$ | A |
| $t = 5$ | C |
| $t = 6$ | B |
| $t = 7$ | B |
| $t = 8$ | A |
| $t = 9$ | C |



Reservoir:

|                |
|----------------|
| A              |
| C              |
| <del>A</del> B |

- 1) Add  $a_t$  with  $t \leq B$  to reservoir
- 2) When receiving  $a_t$  with  $t > B$ :  
with probability  $B/t$ , replace random item in reservoir with new item  $a_t$

Probability  $B/t = 3/7$

→ Randomized decision: add  $a_7 = B$  to reservoir

→ Replace random element — here  $B_3 = A$  with B

# Proof Sketch — All Elements in $B$ sampled with $B/t$

- Obvious case:  $i = t$ 
  - $a_i$  was inserted into  $B$  with probability  $B/t$  (direct result from algorithm)
- Otherwise:  $i < t$ 
  - Observation: in step  $i$ , an item gets replaced with probability  $B/i * 1/B = 1/i$
  - Probability of an item in reservoir

$$B/i \cdot \left(1 - \frac{1}{i+1}\right) \cdot \left(1 - \frac{1}{i+2}\right) \cdot \dots \cdot \left(1 - \frac{1}{t}\right) = B/t$$

The diagram illustrates the derivation of the formula for the probability of an item being in the reservoir. It consists of four rectangular boxes at the bottom, each containing a probability statement, with arrows pointing upwards to the corresponding term in the product formula above.

- The first box contains: "Probability that  $a_i$  was inserted in B". An arrow points from this box to the first term  $B/i$ .
- The second box contains: "Probability that  $a_i$  was NOT replaced in B at step  $(i+1)$ ". An arrow points from this box to the second term  $\left(1 - \frac{1}{i+1}\right)$ .
- The third box contains: "Probability that  $a_i$  was NOT replaced in B at step  $(i+2)$ ". An arrow points from this box to the third term  $\left(1 - \frac{1}{i+2}\right)$ .
- The fourth box contains: "Probability that  $a_i$  was NOT replaced in B at step  $t$ ". An arrow points from this box to the final term  $\left(1 - \frac{1}{t}\right)$ .

# Outline

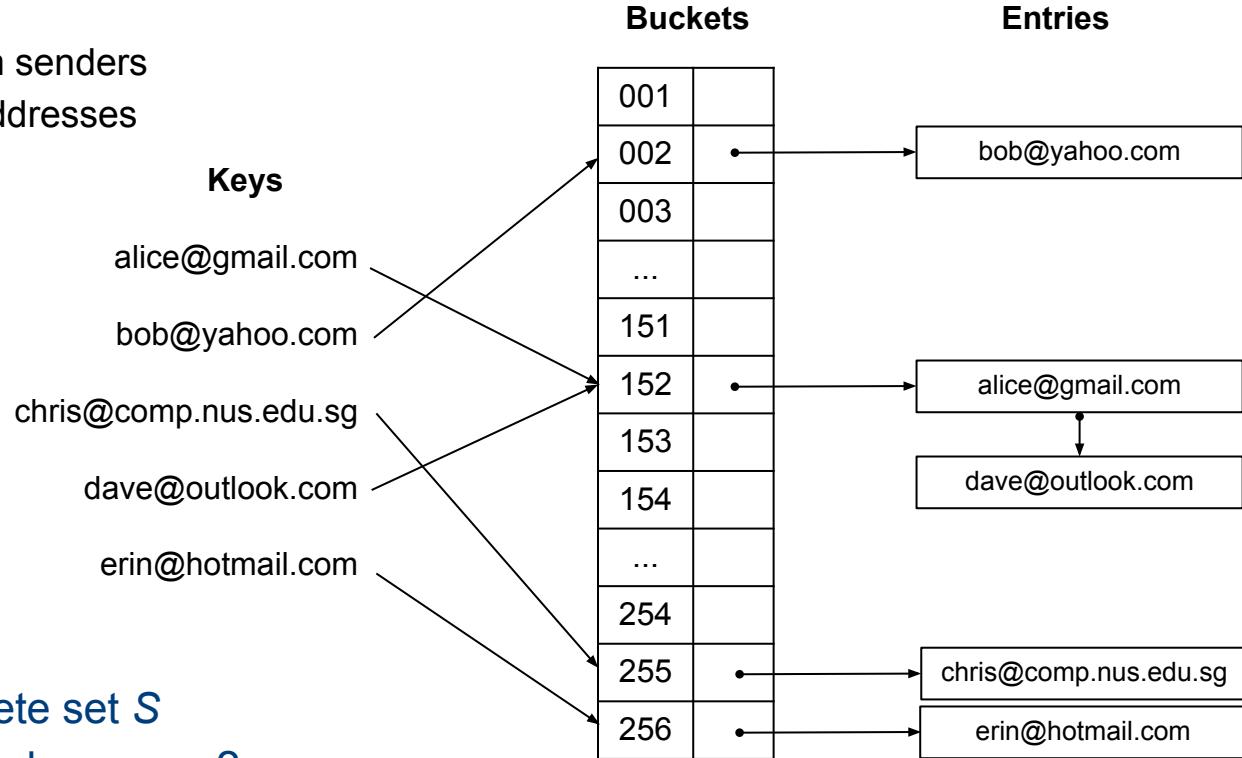
- Motivation
  - Basic setup
  - Example Applications
- Core Techniques
  - Sampling
  - Filtering
  - Counting (distinct items)
- Summary

# Filtering Data Streams

- Goal: Only accept items that meet certain criterion
- Simple: criterion is a property of item that can be (easily) calculated, e.g.:
  - Search queries with more than 2 keywords
  - Sensor values with valid status code
- Challenge: criterion involves lookup for membership in a (very large) set  $S$ , e.g.:
  - Emails with sender addresses that are whitelisted (spam filter)
  - Page visits to a predefined set of websites
  - Tweets from a selected group of users

# Basic Solution — Create Hash Table for Set S

- Example: spam filter
  - Only accept emails from senders with whitelisted email addresses



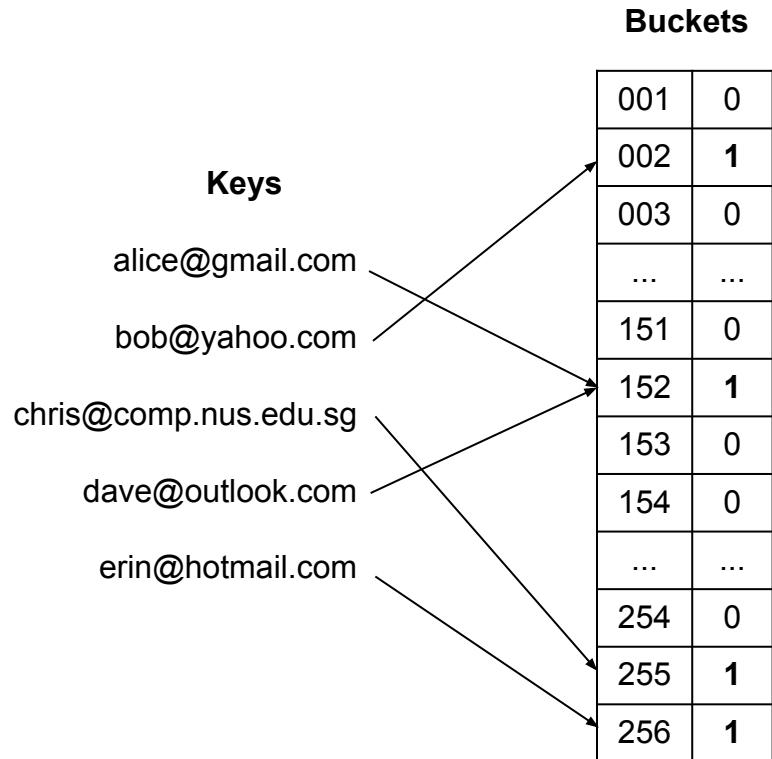
- Requires to store complete set S
- What if there is not enough memory?

# Hashing without Storing $S$

- Create lookup table
    - Create bit array  $B$  with  $1..n$  bits and  $B[i] = 0$
    - Choose hash function  $h(key) \in [1, n]$
    - For each key  $s \in S$  set  $B[h(s)] = 1$
  - Filter step for new data item with key  $k$ 
    - Accept if  $B[h(k)] = 1$ , discard otherwise
- Problem: False Positives
- $a \in S, b \notin S$  and  $h(a) = h(b)$



How bad is it?



# Quick Quiz

Can **false negatives** occur?

**A**

No

**B**

Yes, but the probability is negligible

**C**

Yes, but they do not matter for  
the application context.

**D**

Yes, and that's why  
this is a bad approach.

# False Positive Rate — Analysis

Probability of  $B[i] = 1$  after inserting **one key** from  $S$ :

$$\frac{1}{|B|}$$

Probability of  $B[i] = 0$  after inserting **one key** from  $S$ :

$$1 - \frac{1}{|B|}$$

Probability of  $B[i] = 0$  after inserting **all keys** from  $S$ :

$$\left(1 - \frac{1}{|B|}\right)^{|S|} = \left(1 - \frac{1}{|B|}\right)^{|B|\frac{|S|}{|B|}} \approx e^{-\frac{|S|}{|B|}}$$

Probability of  $B[i] = 1$  after inserting **all keys** from  $S$ :

$$1 - e^{-\frac{|S|}{|B|}}$$



Probability of  $B[h(s)] = 1$  with key  $s \notin S$ :

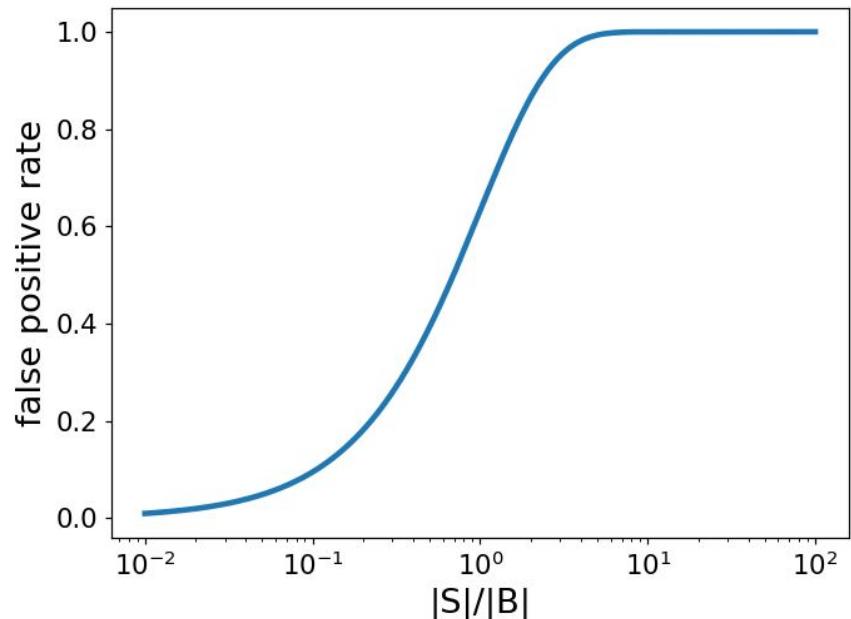
(probability of a false positive)

$$1 - e^{-\frac{|S|}{|B|}}$$

$$e^p = \left[ \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n \right]^p$$

# False Positive Rate — Visualization

- Effect of ratio  $|S|/|B|$  on false positive rate
  - Example calculation
    - $|S| = 10^9$  whitelisted email addresses
    - $|B| = 8 \cdot 10^9$  (1 GB of main memory)
- $|S|/|B| = 1/8$
- False positive rate:  $1 - e^{-1/8} = 11.75\%$



**How to reduce false positive rate?**  
(without increasing the size of bit array B)

# Bloom Filters

- Bloom Filters — basic idea

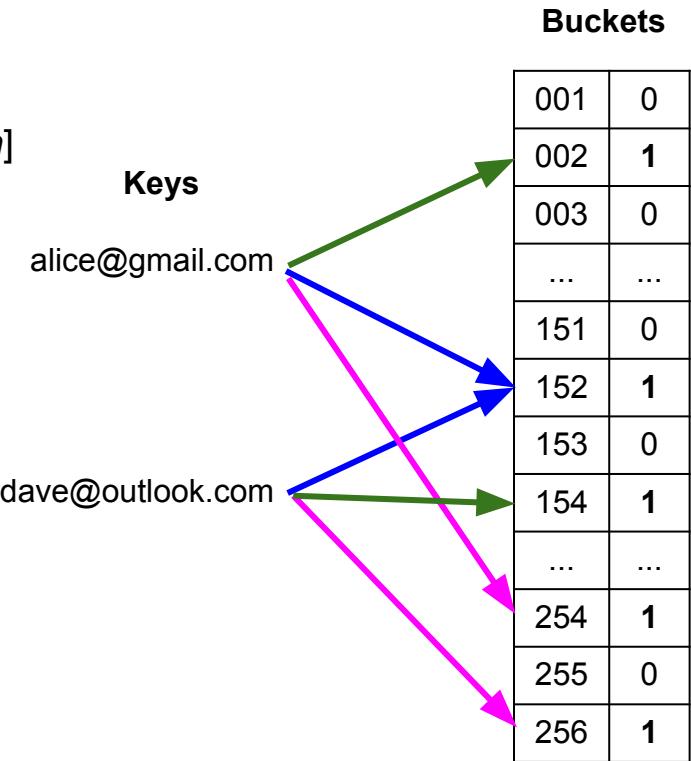
- Create bit array  $B$  with  $1..n$  bits and  $B[i] = 0$
- Choose  $m$  independent hash functions  $h_i(key) \in [1, n]$
- For each key  $s \in S$  set  $B[h_i(s)] = 1$ , with  $1 \leq i \leq m$

- Example

- 3 hash functions:  $h_1$ ,  $h_2$ ,  $h_3$

- Filter step for new data item with key  $k$

- Accept if  $B[h_i(k)] = 1$  for all  $1 \leq i \leq m$ ,  
discard otherwise



# False Positive Rate — Analysis

Probability of  $B[h(s)] = 1$  with key  $s \notin S$ :  
(probability of a false positive)

$$1 - e^{-\frac{|S|}{|B|}}$$



from 1 to  $m$  hash functions

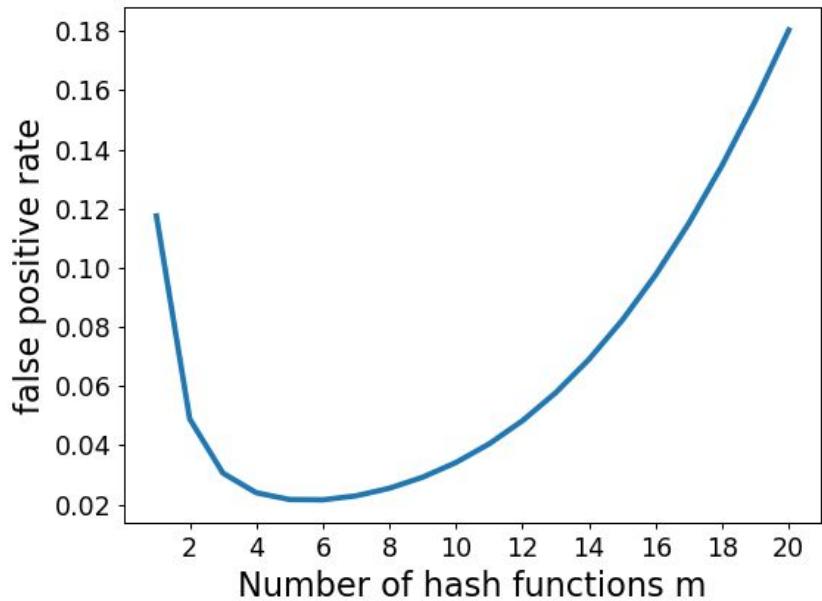
Probability of  $B[h_i(s)] = 1$  ( $1 \leq i \leq m$ ) with key  $s \notin S$ :  
(probability of a false positive)

$$\left(1 - e^{-m \frac{|S|}{|B|}}\right)^m$$

# False Positive Rate — Visualization

- Effect of number of hash functions  $m$  on false positive rate
  - Here,  $|S|/|B| = 1/8$  (see previous example)
- Global optimum — intuition: large  $m$ 
  - More hash results need all be 1
  - But: also more 1s in bit array
- Optimal value for  $m$

$$m_{best} = \frac{|B|}{|S|} \ln 2$$



$$m_{best} = \frac{8}{1} \ln 2 = 5.5 \approx 6$$

→ False positive rate: **2.2%**

# Bloom Filter — Discussion

- **Memory and space consumption**
  - Time complexity:  $O(m)$
  - Space complexity:  $O(|B|)$
- **Limitation: no support of removing keys from  $S$** 
  - E.g.: not able to remove a whitelisted email address
  - Only workaround: rebuild lookup table (bit array) from scratch

Why?

# Extension — Counting Bloom Filters

- Counting Bloom Filters

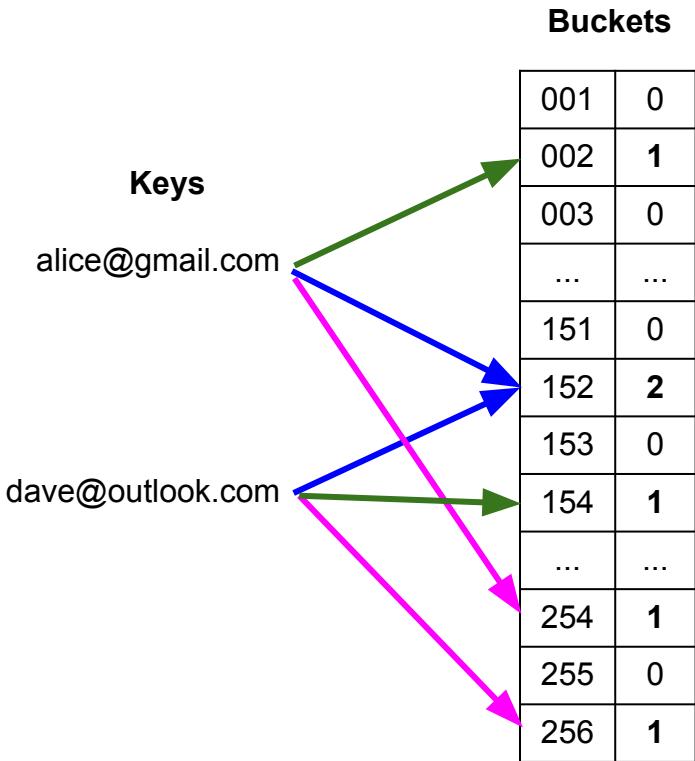
- Replace bits for counters (typically 3-4 bits per counter)
- Increased size of lookup table (3-4 times)

- Operations

- Insert  $s$ :  $B[h_i(s)] \leftarrow 1$ , with  $1 \leq i \leq m$
- Delete  $s$ :  $B[h_i(s)] \leftarrow 0$ , with  $1 \leq i \leq m$
- Lookup  $k$ : Accept if  $B[h_i(k)] > 0$  for all  $1 \leq i \leq m$

- False positive rate

- Same as normal Bloom Filter:  $\left(1 - e^{-m\frac{|S|}{B}}\right)^m$



# Outline

- Motivation
  - Basic setup
  - Example Applications
- Core Techniques
  - Sampling
  - Filtering
  - **Counting** (distinct items)
- Summary

# Counting Unique/Distinct Elements

- Applications

- Number of unique Facebook visitors (identified by user id)
- Number of unique website visitors (identified by IP address)
- Number of unique words in a (large) text document

- Straightforward solution

- Maintain set  $S$  of items seen so far
- Number distinct elements  $\rightarrow |S|$

→ What if set  $S$  can grow very large?

```
1 S = set()
2
3 with open('data/ip-only-nasa-access.log') as file:
4     for line in file:
5         ip = line.strip() # Get IP address
6         S.add(ip)          # Add IP address to set
7
8 print('|S| = {}'.format(len(S)))
|S| = 7637
```

# Flajolet-Martin Algorithm

- Approximation approach
  - Estimate distinct count in an unbiased way
  - Accept errors but minimize their probability
- Basic algorithm

- (1) Choose hash function that maps each of the  $n$  elements to at least  $\log_2 n$  bits
- (2) For each element  $s$  in the stream
  - (a) Calculate hash  $h(s)$  → bit string of length  $\log_2 n$
  - (b) Let  $r(s) =$  number of trailing 0s in  $h(s)$
  - (c) Keep track of largest  $r(s)$  →  $R$
- (3) Return estimate for distinct count as  $2^R$

} Example:  $2^{32}$  IPv4 addresses  
→ at least 32 bits

# Flajolet-Martin Algorithm — Example

| a: IPv4 address | h(a)                                      |
|-----------------|---|
| 13.66.139.0     | 01010100110100100110111000001001          |
| 157.48.153.185  | 1100101001001001001001011011111 <u>00</u> |
| 157.48.153.185  | 1100101001001001001001011011111 <u>00</u> |
| 216.244.66.230  | 0001110001010101100111010001001 <u>0</u>  |
| 54.36.148.92    | 010101010001010110010111000011 <u>00</u>  |
| 92.101.35.224   | 101000111000011000000000100000001         |
| 73.166.162.225  | 00100100100111100100101101011 <u>000</u>  |
| 73.166.162.225  | 00100100100111100100101101011 <u>000</u>  |
| 54.36.148.108   | 00001100010100011000000001010001          |
| 54.36.148.1     | 011001011000101000101100100001 <u>00</u>  |
| 162.158.203.24  | 10111000010010010011100010111 <u>0</u>    |
| 157.48.153.185  | 1100101001001001001011011011 <u>00</u>    |
| 157.48.153.185  | 1100101001001001001011011011 <u>00</u>    |
| ...             | ...                                       |

R = 3 (largest number of trailing 0s so far)

→ Estimate for distinct count:  $2^3 = 8$

# Quick "Quiz"

Someone is telling you that s/he flipped a fair coin **3 times** and got **3 Heads** after ***k* tries**?

Which **number of tries** is the most believable?

**A**

$k = 1$

**B**

$k = 10$

**C**

$k = 100$

**D**

$k = 1,000$

# Flajolet-Martin Algorithm — Intuition & Proof Sketch

- Basic intuition

- More distinct elements → more different hash values → "unusual" hash values more likely
- "Unusual" hash value = hash value with rare bit pattern (e.g., number of trailing 0s)

Probability that  $h(a)$  ends in **at least**  $k$  trailing 0s

$$\underbrace{\frac{1}{2} \cdot \frac{1}{2} \cdot \dots \cdot \frac{1}{2}}_{k \text{ factors}} = \frac{1}{2^k} = s^{-k}$$

Probability that  $h(a)$  ends in **less than**  $k$  trailing 0s

$$1 - \frac{1}{2^k}$$

Given  $m$  distinct elements, probability that  
**all**  $h(a)$  end in **less than**  $k$  trailing 0s

$$\left(1 - \frac{1}{2^k}\right)^m$$

Given  $m$  distinct elements, probability that  $R \geq k$   
(i.e., at least one of the  $m$  elements has  $h(a)$  with at least  $k$  trailing 0s)

$$1 - \left(1 - \frac{1}{2^k}\right)^m$$

# Flajolet-Martin Algorithm — Proof Sketch

Given  $m$  distinct elements, probability that  $\mathbf{R} \geq \mathbf{k}$   
(i.e., at least one of the  $m$  elements has  $h(a)$  with at least  $k$  trailing 0s)

$$e^p = \left[ \lim_{n \rightarrow \infty} \left( 1 + \frac{1}{n} \right)^n \right]^p$$

$$1 - \left( 1 - \frac{1}{2^k} \right)^m = 1 - \left( 1 - \frac{1}{2^k} \right)^{\frac{2^k m}{2^k}} \approx 1 - e^{-\frac{m}{2^k}}$$

**Case 1:**  $2^k \ll m \rightarrow 1 - e^{-\frac{m}{2^k}} \approx 1 - 0 = 1$

**Case 2:**  $2^k \gg m \rightarrow 1 - e^{-\frac{m}{2^k}} \approx 1 - \underbrace{\left( 1 - \frac{m}{2^k} \right)}_{\text{First 2 terms of the Taylor expansion of } e^x} \approx \frac{m}{2^k} \approx 0$

$$e^x = 1 + \frac{x}{1!}, \text{ if } x \ll 1$$

First 2 terms of the  
Taylor expansion of  $e^x$

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

# Flajolet-Martin Algorithm — Proof Sketch

- Interpretation

**Case 1:**  $2^k \ll m \rightarrow P(R \geq k) \approx 1$

The probability to get an  $h(a)$  with  
**enough trailing 0s** is rather high

**Case 2:**  $2^k \gg m \rightarrow P(R \geq k) \approx 0$

The probability to get an  $h(a)$  with  
**too many trailing 0s** is rather low

}  $\rightarrow R$  is typically in the right ballpark

# Flajolet-Martin Algorithm — Problems & Extensions

- Obvious problem with basic Flajolet-Martin algorithm — given  $2^R$ 
  - An estimate is always a power of 2
  - If  $R$  is off by just 1, estimates doubles or halves
- Practical solution: use multiple hash functions  $h_i(a)$  — e.g.:
  - $p \cdot q$  hash functions  $\rightarrow p \cdot q R$  values  $\rightarrow p \cdot q$  estimates for the distinct counts
  - Put all estimates into  $p$  groups, each of size  $q$
  - Calculate median of each group  $\rightarrow p$  medians
  - Calculate the mean over all  $p$  medians

# Outline

- Motivation
  - Basic setup
  - Example Applications
- Core Techniques
  - Sampling
  - Filtering
  - Counting (distinct items)
- Summary

# Summary

- Data streams — main challenges
  - Large data volumes + high arrival speeds
  - Limited resources + real-time requirements



→ patterns = statistical analysis

- Common tasks on streams

(or very large datasets in general)

- Sampling
- Filtering
- Counting  
(distinct elements)



→ trade-off: speed / resource-efficiency vs. accuracy / errors

# Solutions to Quick Quizzes

- Slide 7: C
- Slide 30: A
- Slide 42: B