上海科技大学
ShanghaiTech University

CS182: Introduction to Machine Learning – Recommendation Systems + Ensemble Methods: Boosting

Yujiao Shi
SIST, ShanghaiTech
Spring, 2025

Learning Paradigms

上海科技大学 ShanghaiTech University

Paradigm	Data
Supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ $\quad \mathbf{x} \sim p^*(\cdot)$ and $y = c^*(\cdot)$
$\hookrightarrow$ Regression	$y^{(i)} \in \mathbb{R}$
$\hookrightarrow$ Classification	$y^{(i)} \in \{1, \dots, K\}$
$\hookrightarrow$ Binary classification	$y^{(i)} \in \{+1, -1\}$
$\hookrightarrow$ Structured Prediction	$\mathbf{y}^{(i)}$ is a vector
Unsupervised	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ $\quad \mathbf{x} \sim p^*(\cdot)$
$\hookrightarrow$ Clustering	predict $\{z^{(i)}\}_{i=1}^{N}$ where $z^{(i)} \in \{1, \dots, K\}$
$\hookrightarrow$ Dimensionality Reduction	convert each $\mathbf{x}^{(i)} \in \mathbb{R}^M$ to $\mathbf{u}^{(i)} \in \mathbb{R}^K$ with $K << M$
Semi-supervised	$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_1} \cup \{\mathbf{x}^{(j)}\}_{j=1}^{N_2}$
Online	$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), (\mathbf{x}^{(3)}, y^{(3)}), \dots\}$
Active Learning	$\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ and can query $y^{(i)} = c^*(\cdot)$ at a cost
Imitation Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}), (s^{(2)}, a^{(2)}), \dots\}$
Reinforcement Learning	$\mathcal{D} = \{(s^{(1)}, a^{(1)}, r^{(1)}), (s^{(2)}, a^{(2)}, r^{(2)}), \dots\}$

Outline for Today

上海科技大学
ShanghaiTech University

We'll talk about two distinct topics:

1. **Recommender Systems**: produce recommendations of what a user will like
   (i.e. the solution to a particular type of task)
2. **Ensemble Methods**: combine or learn multiple classifiers into one
   (i.e. a family of algorithms)

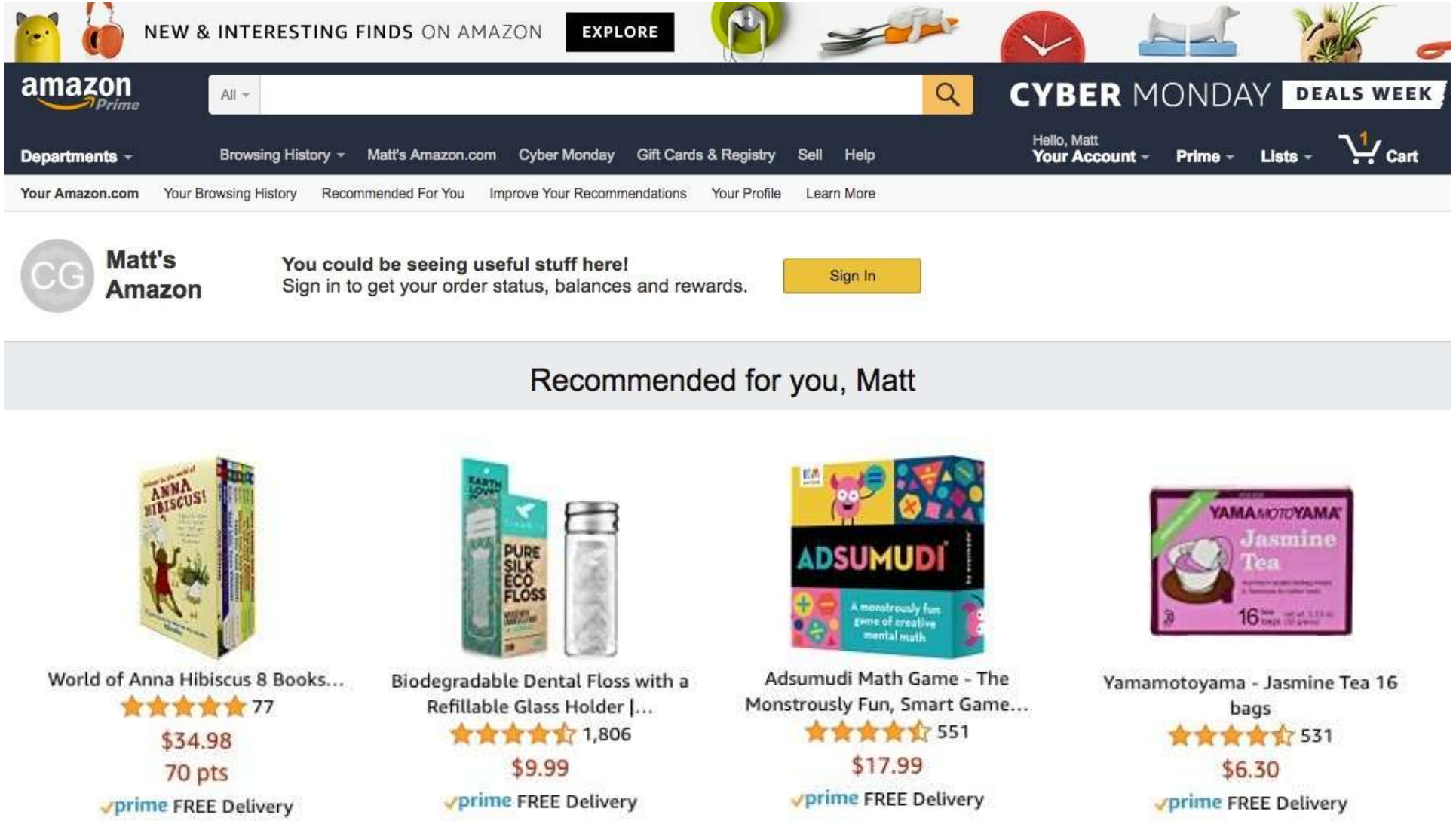
We'll use a prominent example of a recommender systems (the Netflix Prize) to motivate both topics…

RECOMMENDER SYSTEMS

A Common Challenge:

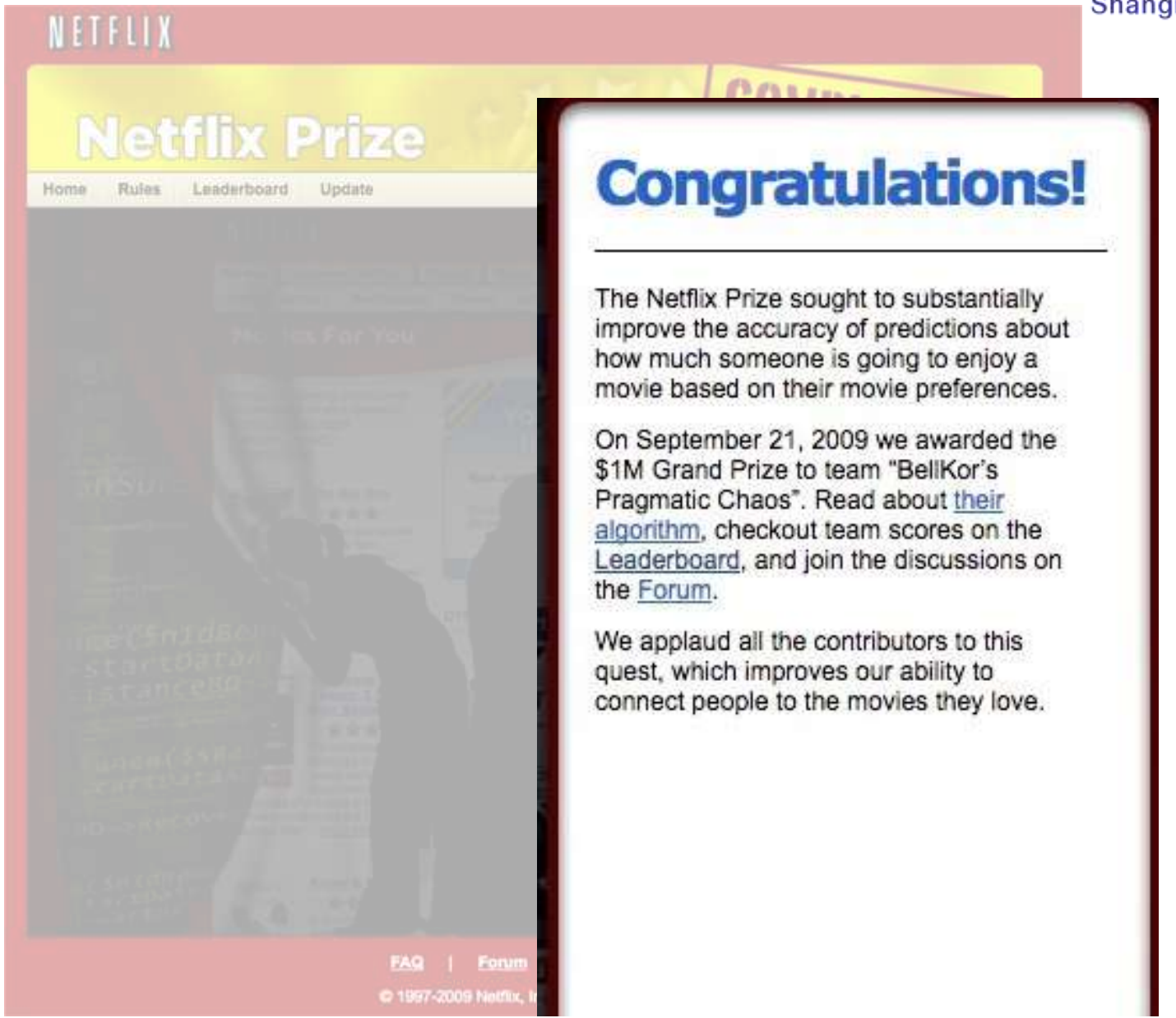
- Assume you're a company selling **items** of some sort: movies, songs, products, etc.
- Company collects millions of **ratings** from **users** of their **items**
- To maximize profit / user happiness, you want to **recommend** items that users are likely to want

Recommender Systems

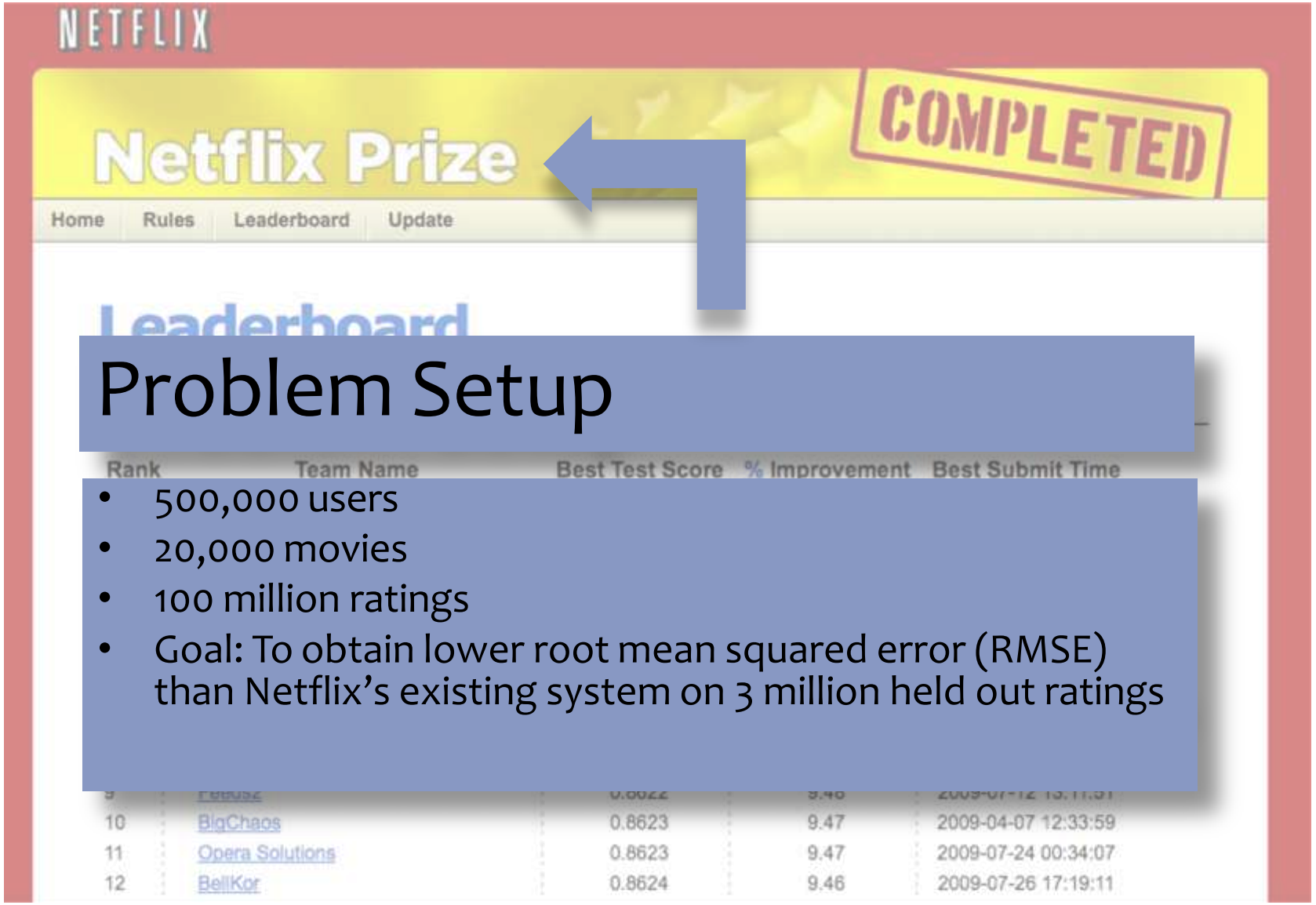


Recommender Systems

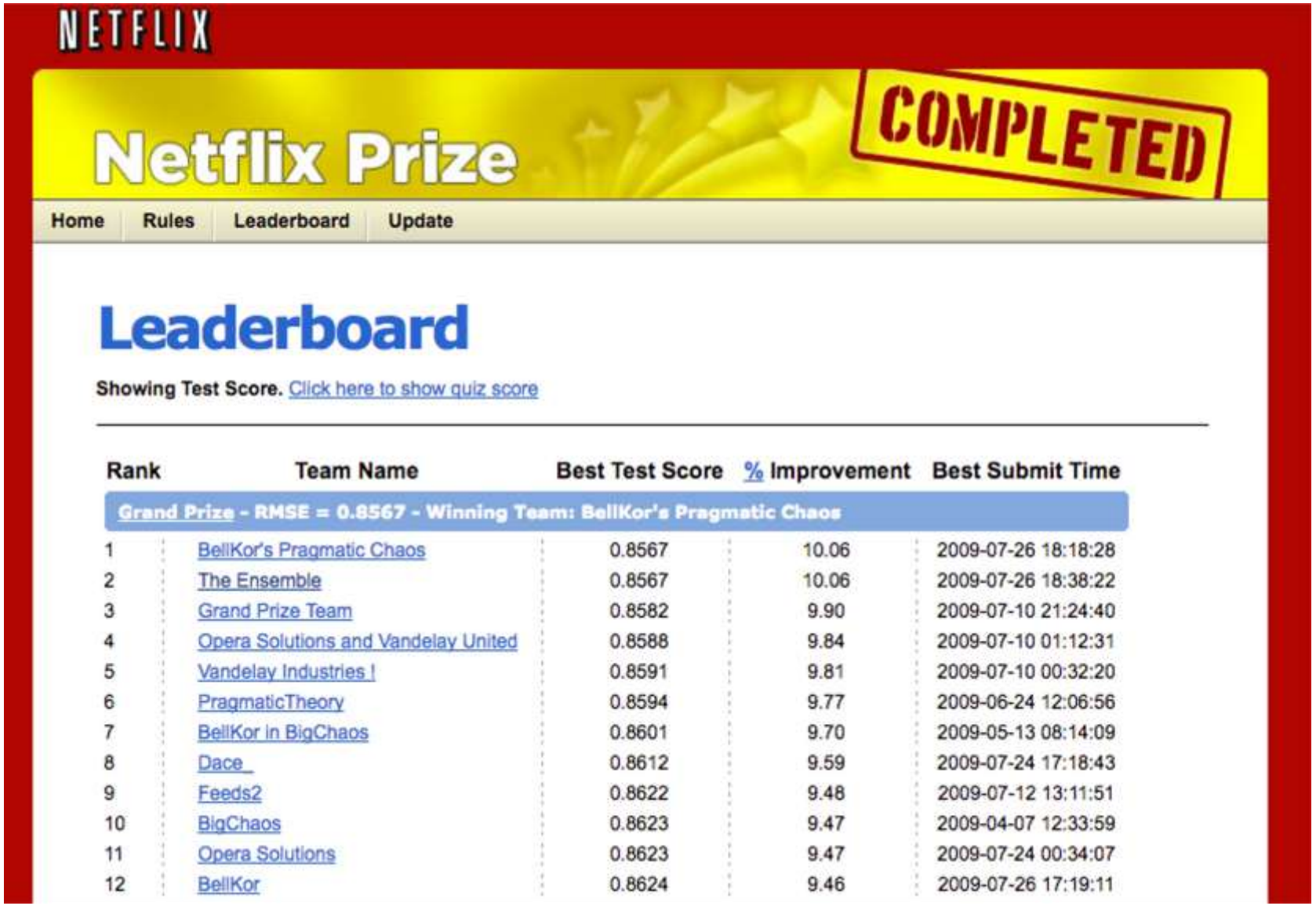
Recommender Systems



Recommender Systems



Recommender Systems



Recommender Systems

- **Setup:**
  - Items:

    movies, songs, products, etc. (often many thousands)
  - Users:

    watchers, listeners, purchasers, etc. (often many millions)
  - Feedback:

    5-star ratings, not-clicking 'next', purchases, etc.
- **Key Assumptions:**
  - Can represent ratings numerically as a user/item matrix
  - Users only rate a small number of items (the matrix is sparse)

	Doctor Strange	Star Trek: Beyond	Zootopia
Alice	1		5
Bob	3	4	
Charlie	3	5	2

Two Types of Recommender Systems

Content Filtering

- *Example*: Pandora.com music recommendations (Music Genome Project)
- **Con:** Assumes access to side information about items (e.g. properties of a song)
- **Pro:** Got a new item to add? No problem, just be sure to include the side information

Collaborative Filtering

- *Example*: Netflix movie recommendations
- **Pro:** Does not assume access to side information about items (e.g. does not need to know about movie genres)
- **Con:** Does not work on new items that have no ratings

COLLABORATIVE FILTERING

Collaborative Filtering

上海科技大学 ShanghaiTech University

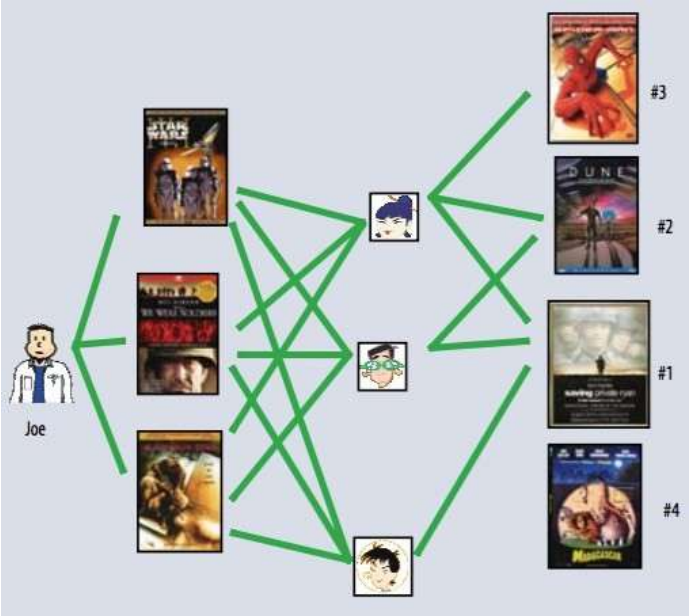
- **Everyday Examples of Collaborative Filtering...**
  - Bestseller lists
  - Top 40 music lists
  - The "recent returns" shelf at the library
  - Unmarked but well-used paths thru the woods
  - The printer room at work
  - "Read any good books lately?"
  - …
- **Common insight:** personal tastes are correlated
  - If Alice and Bob both like X and Alice likes Y then Bob is more likely to like Y
  - especially (perhaps) if Bob knows Alice

Slide from William Cohen

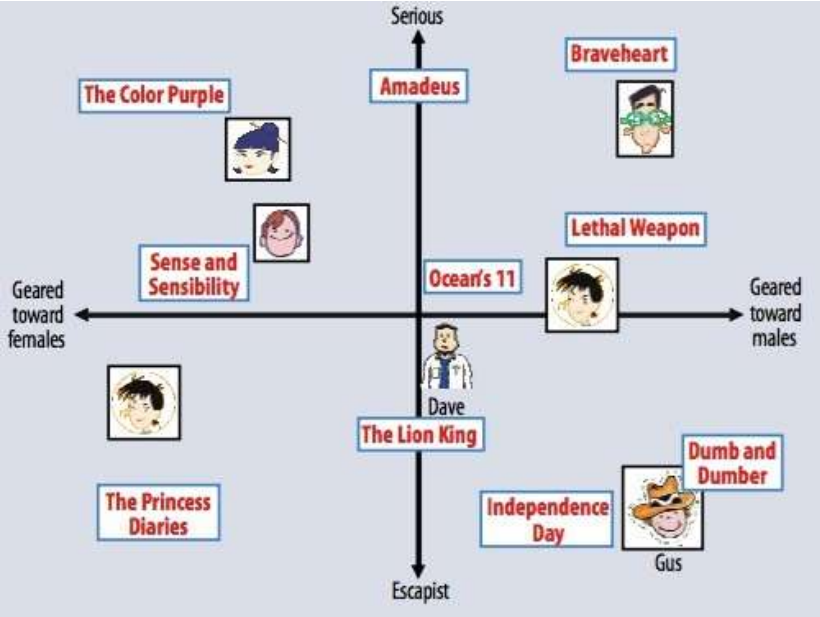
Two Types of Collaborative Filtering

上海科技大学 ShanghaiTech University

1. Neighborhood Methods



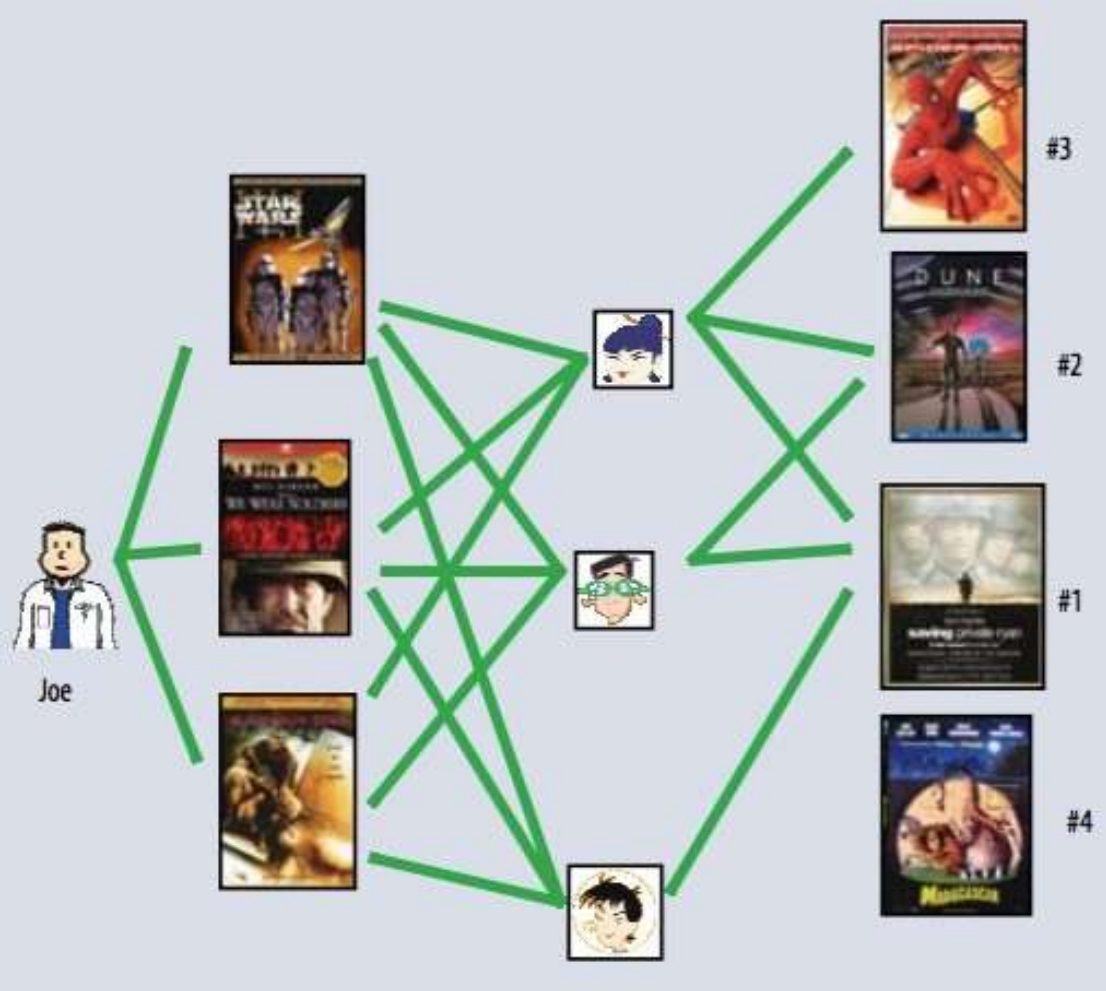
2. Latent Factor Methods



Figures from Koren et al. (2009)

Two Types of Collaborative Filtering

1. Neighborhood Methods



- In the figure, assume that a green line indicates the movie was **watched**

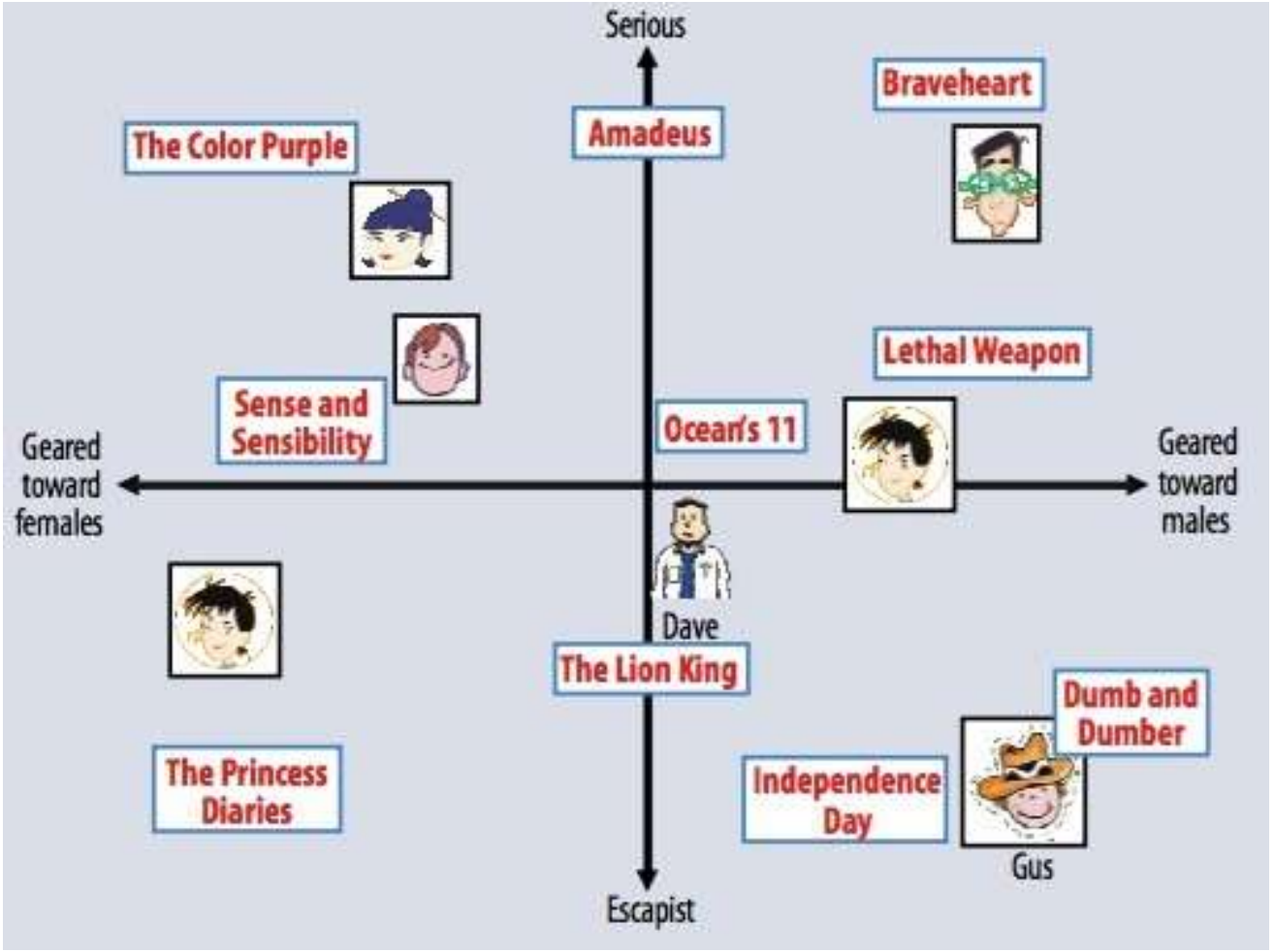
- **Algorithm:**
  1. **Find neighbors** based on similarity of movie preferences
  2. **Recommend** movies that those neighbors watched

Figures from Koren et al. (2009)

Two Types of Collaborative Filtering

2. Latent Factor Methods

- Assume that both movies and users live in some **low-dimensional space** describing their properties
- **Recommend** a movie based on its **proximity** to the user in the latent space
- **Example Algorithm:** Matrix Factorization



Figures from Koren et al. (2009)

Recommending Movies

上海科技大学
ShanghaiTech University

Question:

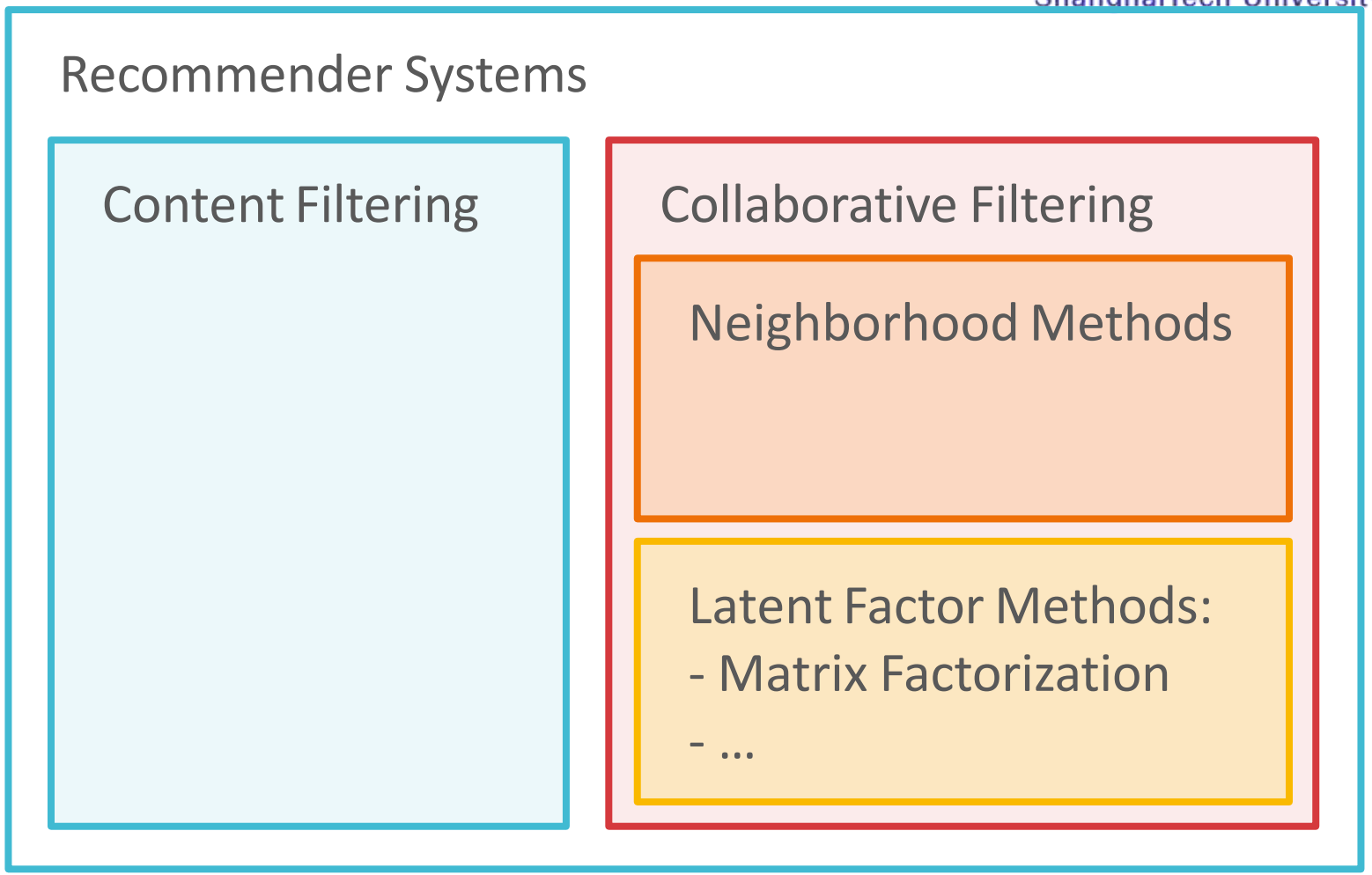
Applied to the Netflix Prize problem, which of the following methods *always* requires side information about the users and movies?

Select all that apply

A. principal component analysis
B. collaborative filtering
C. latent factor methods
D. ensemble methods
E. content filtering
F. neighborhood methods
G. recommender systems

Answer:

Summary Thus Far



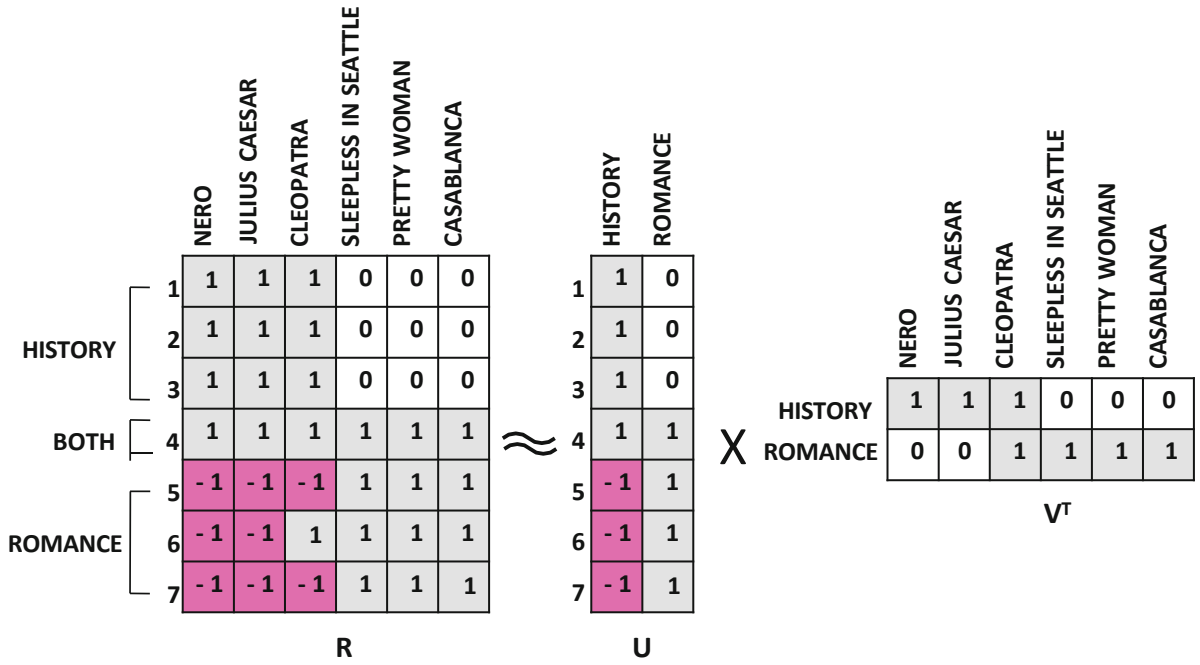
MATRIX FACTORIZATION

Matrix Factorization

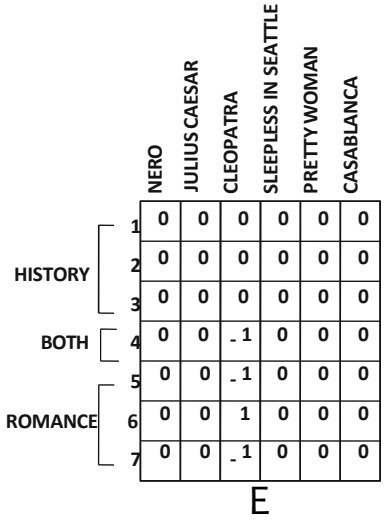
上海科技大学
ShanghaiTech University

- Many different ways of factorizing a matrix
- We'll consider three:
    1. Unconstrained Matrix Factorization
    2. Singular Value Decomposition
    3. Non-negative Matrix Factorization
- MF is just another example of a **common recipe**:
    1. define a model
    2. define an objective function
    3. optimize with SGD

Example: MF for Netflix Problem



(a) Example of rank-2 matrix factorization

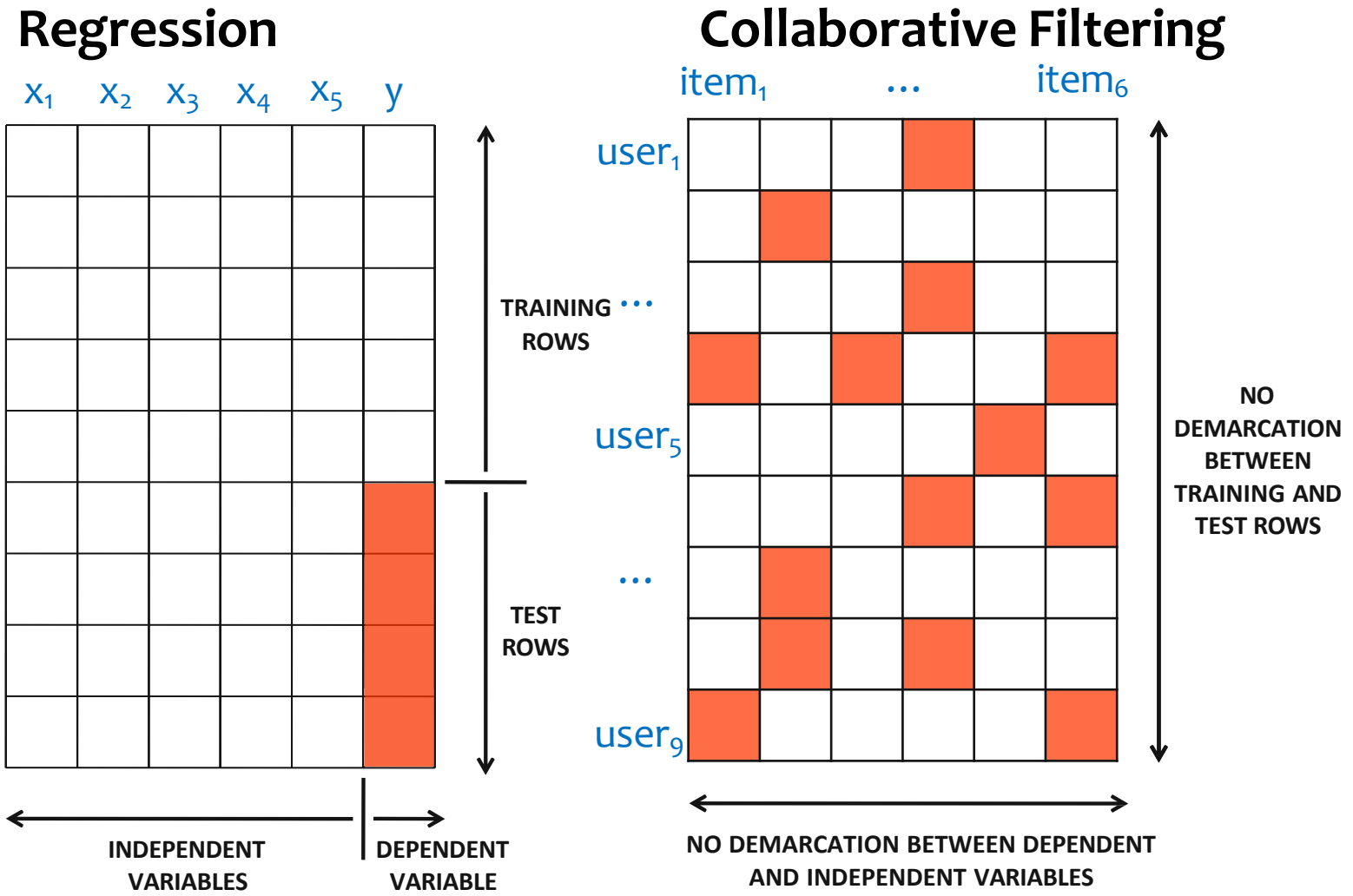


(b) Residual matrix

Figures from Aggarwal (2016)

Regression vs. Collaborative Filtering

Goal of each problem is to predict the values of the missing squares 



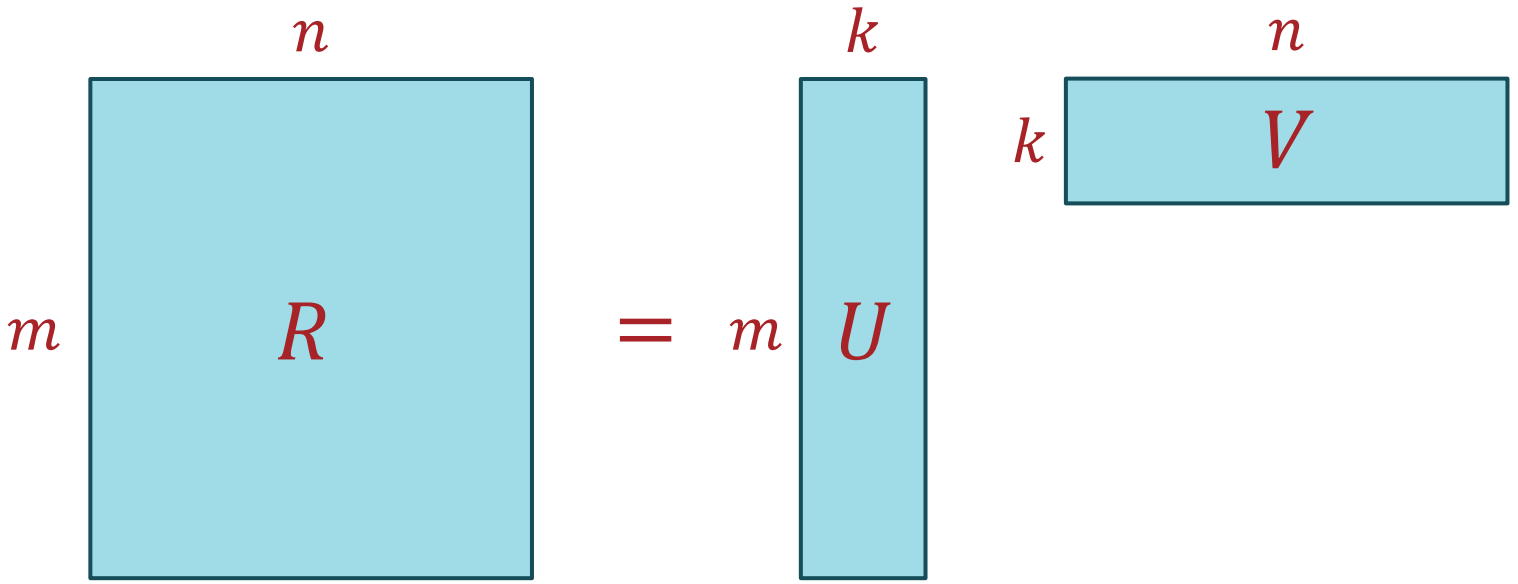
Figures from Aggarwal (2016)

Matrix Factorization

- **High-level idea**: Decompose the ratings matrix, R, into the product of two (low-dimensional) matrices:
  - U, corresponding to users and
  - V, corresponding to items
- To do so, we're going to follow our usual recipe for learning:
  1. define a model
  2. define an objective function
  3. optimize the objective with SGD

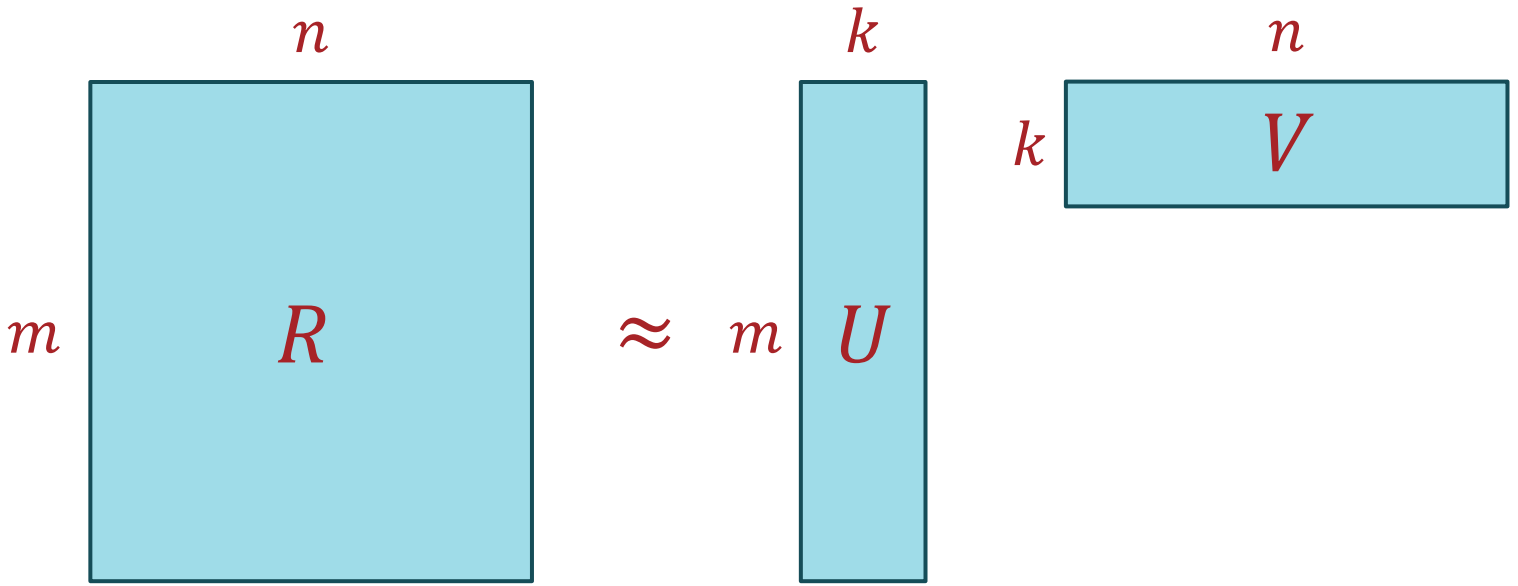
- **Insight:** if $R \in \mathbb{R}^{m \times n}$ has *rank* $k \ll \min(m, n)$, then
  $\exists\, U \in \mathbb{R}^{m \times k}$ and
  $V \in \mathbb{R}^{n \times k}$ such that $R = UV^T$

Low-rank Matrix Factorization



Low-rank Matrix Factorization

- **Idea:** even if $R \in \mathbb{R}^{m \times n}$ has *rank* $l > k$, then there still $\exists\, U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$ such that $R \approx UV^T$
- **Approach:** pick some arbitrary (typically small) k and *learn* rank-k matrices U and V such that $R \approx UV^T$



UNCONSTRAINED MATRIX FACTORIZATION

Low-rank Matrix Factorization

- **Observation:** R is just a bunch of real-valued ratings
- **Idea:** minimize the mean-squared error
  - Let $E = R - UV^T$
  - Objective function:

$$J(U,V) = \frac{1}{2}\|E\|_2^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} E_{i,j}^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(R_{i,j} - U_{j,}.V_{.,i}^T\right)^2$$

where $U_{j,}.$ is the j^{th} row of U and $V_{.,i}^T$ is the i^{th} column of V^T

- **Problem:** the objective above is only defined if R is *fully-observed* i.e., we have ratings from every user for every item

Partially Observed Low-rank Matrix Factorization

- **Observation:** R is just a bunch of real-valued ratings
- **Idea:** minimize the mean-squared error
  - Let $E = R - UV^T$ and let $Z = \{(i,j): R_{i,\mathrm{j}} \text{ is known}\}$
  - Objective function:

$$J(U,V) = \frac{1}{2}\|E\|_2^2 = \frac{1}{2}\sum_{(i,j)\in Z} E_{i,j}^2 = \frac{1}{2}\sum_{(i,j)\in Z}\left(R_{i,j} - U_{j,\cdot}V_{\cdot,i}^T\right)^2$$

where $U_{\mathrm{j},\cdot}$ is the j^{th} row of U and $V_{\cdot,i}^T$ is the i^{th} column of V^T

- **Interpretation:** Z is the "training dataset"; we can learn U and V via SGD by sampling a "data point" from Z and computing the gradients w.r.t. to that single rating.

SGD for Partially-Observed Low-rank Matrix Factorization

- while not converged:
  - sample (i, j) from Z
  - compute $E_{i,\mathrm{j}} = R_{i,\mathrm{j}} - U_{\mathrm{j},}.V_{\cdot,i}^T$
  - update $U_{\mathrm{j},}.$ and $V_{\cdot,i}^T$:
    - $U_{\mathrm{j},}. \leftarrow U_{\mathrm{j},}. - \eta \nabla_{U_{\mathrm{j},}.} J_{i,\mathrm{j}}(U, V)$
    - $V_{\cdot,i}^T \leftarrow V_{\cdot,i}^T - \eta \nabla_{V_{\cdot,i}^T} J_{i,\mathrm{j}}(U, V)$

where $J_{i,\mathrm{j}}(U, V) = \frac{1}{2}\left(R_{i,\mathrm{j}} - U_{\mathrm{j},}.V_{\cdot,i}^T\right)^2$

$$\nabla_{U_{\mathrm{j},}.} J_{i,\mathrm{j}}(U, V) = -E_{i,\mathrm{j}} V_{\cdot,i}^T$$

$$\nabla_{V_{\cdot,i}^T} J_{i,\mathrm{j}}(U, V) = -E_{i,\mathrm{j}} U_{\mathrm{j},}.$$

Regularized SGD for Partially-Observed Low-rank Matrix Factorization

- while not converged:
  - sample (i, j) from Z
  - compute $E_{i,\mathrm{j}} = R_{i,\mathrm{j}} - U_{\mathrm{j},}.V_{\cdot,i}^{T}$
  - update $U_{\mathrm{j},}.$ and $V_{\cdot,i}^{T}$:
    - $U_{\mathrm{j},}. \leftarrow U_{\mathrm{j},}. - \eta \nabla_{U_{\mathrm{j},}.} J_{i,\mathrm{j}}(U, V)$
    - $V_{\cdot,i}^{T} \leftarrow V_{\cdot,i}^{T} - \eta \nabla_{V_{\cdot,i}^{T}} J_{i,\mathrm{j}}(U, V)$

where $J_{i,\mathrm{j}}(U, V) = \frac{1}{2}\left(R_{i,\mathrm{j}} - U_{\mathrm{j},}.V_{\cdot,i}^{T}\right)^2 + \frac{\lambda}{2}\left(\|U\|_2^2 + \|V\|_2^2\right)$

$$\nabla_{U_{\mathrm{j},}.} J_{i,\mathrm{j}}(U, V) = -E_{i,\mathrm{j}} V_{\cdot,i}^{T} + \lambda U_{\mathrm{j},}.$$

$$\nabla_{V_{\cdot,i}^{T}} J_{i,\mathrm{j}}(U, V) = -E_{i,\mathrm{j}} U_{\mathrm{j},}. + \lambda V_{\cdot,i}$$

SGD for Partially-Observed Low-rank Matrix Factorization

- while not converged:
  - sample (i, j) from Z
  - compute $E_{i,\mathrm{j}} = R_{i,\mathrm{j}} - U_{\mathrm{j},\cdot} V_{\cdot,i}^T$
  - update $U_{\mathrm{j},\cdot}$ and $V_{\cdot,i}^T$:
    - $U_{\mathrm{j},\cdot} \leftarrow U_{\mathrm{j},\cdot} - \eta \nabla_{U_{\mathrm{j},\cdot}} J_{i,\mathrm{j}}(U, V)$
    - $V_{\cdot,i}^T \leftarrow V_{\cdot,i}^T - \eta \nabla_{V_{\cdot,i}^T} J_{i,\mathrm{j}}(U, V)$

with user and item bias terms O_{j} and P_i respectively:

$$U = \begin{bmatrix} U_{1,\cdot} & O_1 & 1 \\ U_{2,\cdot} & O_2 & 1 \\ \vdots & \vdots & \vdots \\ U_{m,\cdot} & O_m & 1 \end{bmatrix} \text{ and } V^T = \begin{bmatrix} V_{\cdot,1}^T & V_{\cdot,2}^T & \cdots & V_{\cdot,n}^T \\ 1 & 1 & \cdots & 1 \\ P_1 & P_2 & \cdots & P_n \end{bmatrix}$$

Alternating Least Squares for Partially-Observed Low-rank Matrix Factorization

- **Insight:** if we knew either U or V^T, then solving for the other is easy! In fact, it is exactly the same as linear regression!

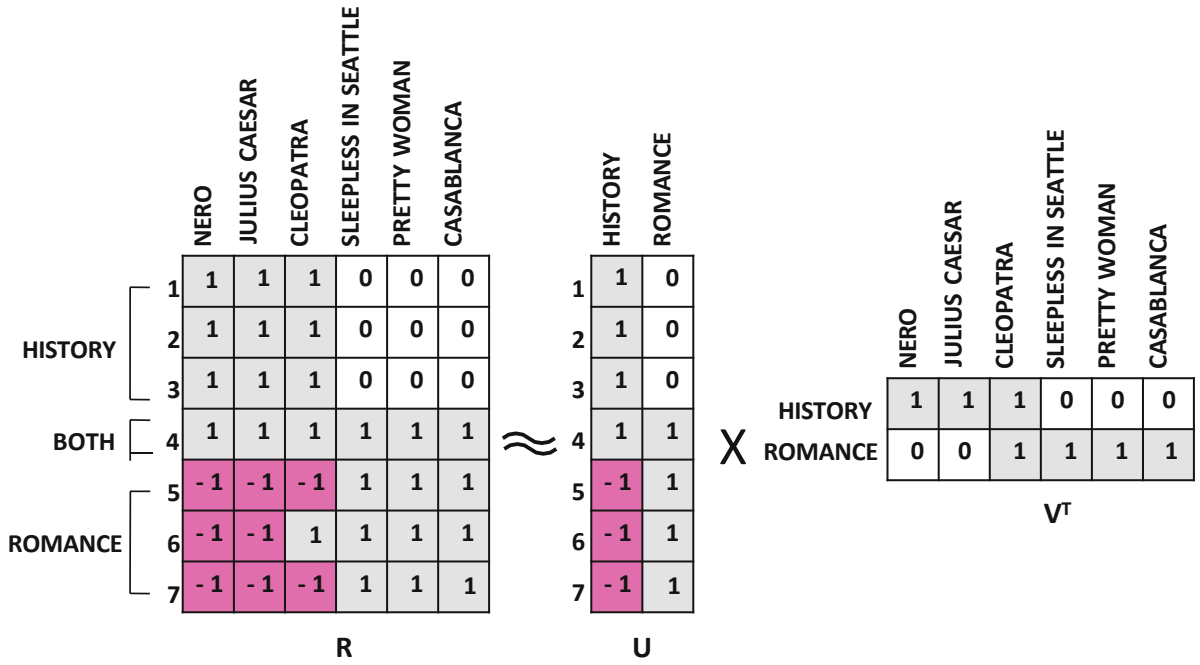
$$J(U,V) = \frac{1}{2}\sum_{(i,\mathrm{j})\in \mathrm{Z}} \left(R_{i,\mathrm{j}} - U_{\mathrm{j},\cdot}V^T_{\cdot,i}\right)^2$$

vs.

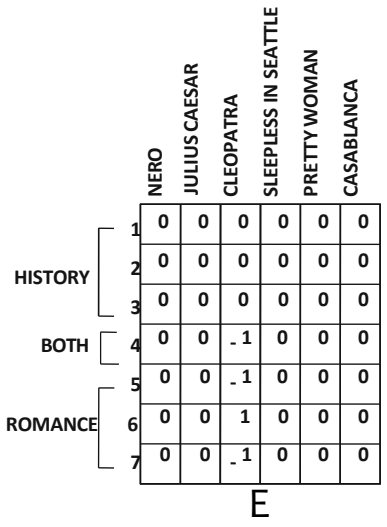
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{N} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

- initialize U and V^T
- while not converged:
  - Fix V^T and solve for U *exactly* using ordinary least squares
  - Fix U and solve for V^T *exactly* using ordinary least squares

Example: MF for Netflix Problem



(a) Example of rank-2 matrix factorization



(b) Residual matrix

Figures from Aggarwal (2016)

Matrix Factorization

Example Factors

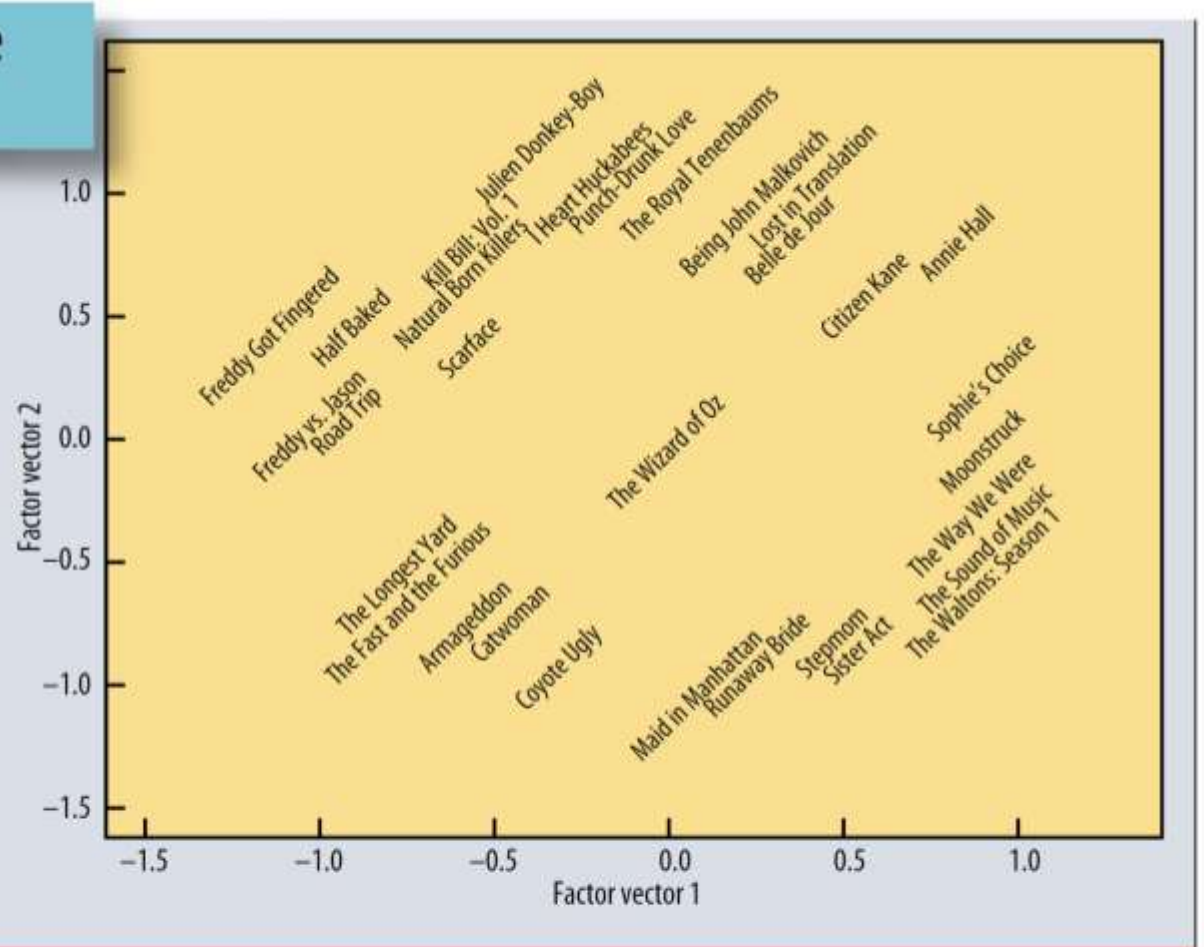


Figure 3. The first two vectors from a matrix decomposition of the Netflix Prize data. Selected movies are placed at the appropriate spot based on their factor vectors in two dimensions. The plot reveals distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky independent films.

Figure from Koren et al. (2009)

Matrix Factorization

Comparison of Optimization Algorithms

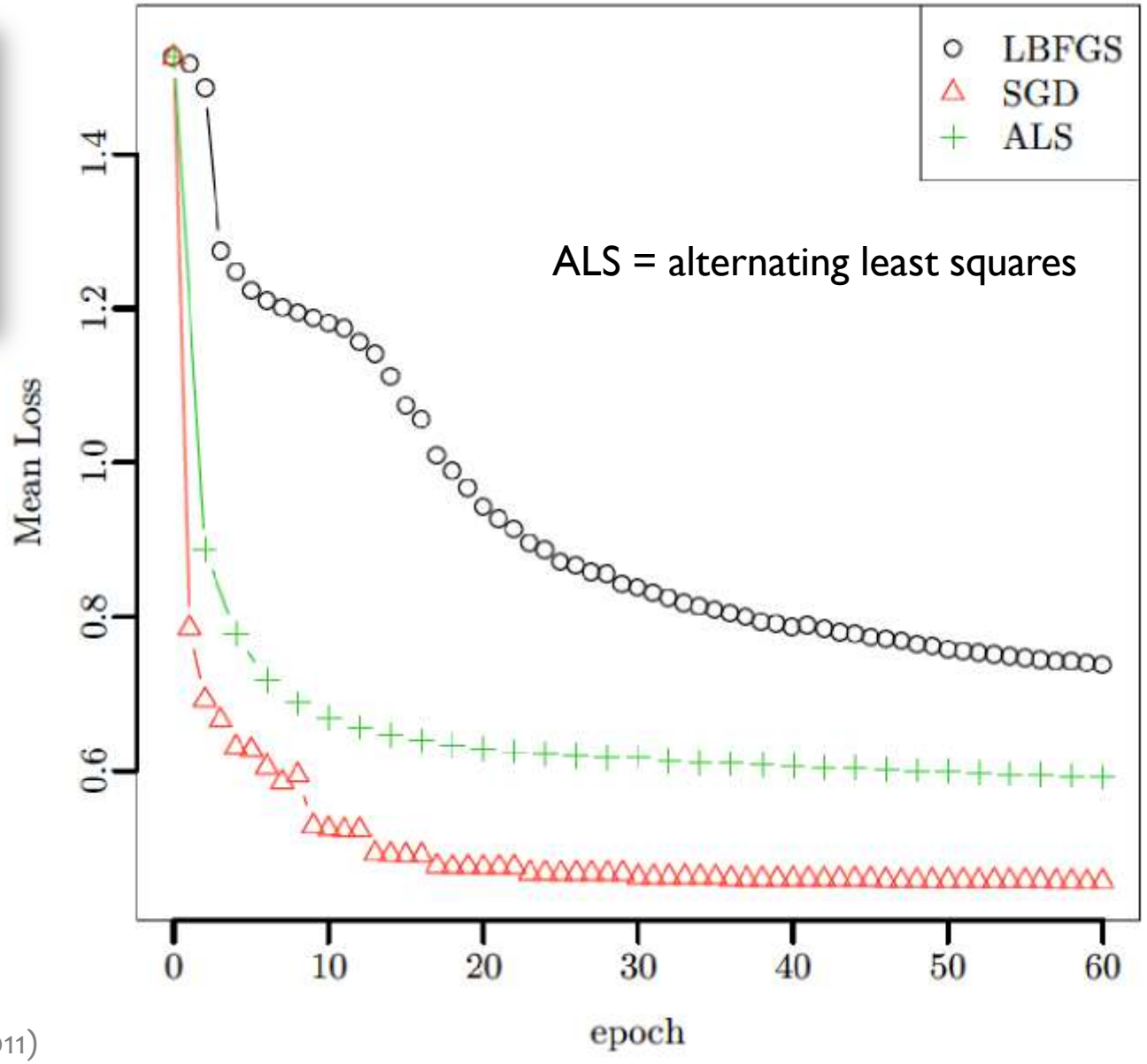


Figure from Gemulla et al. (2011)

Unconstrained Matrix Factorization

上海科技大学
ShanghaiTech University

Opt. Problem #1 (fully observed R)

Model Predictions:

Opt. Problem #2 (partially observed R)

Gradient Descent:

SVD FOR COLLABORATIVE FILTERING

Singular Value Decomposition for Collaborative Filtering

上海科技大学 ShanghaiTech University

For any arbitrary matrix $\mathbf{A}$, SVD gives a decomposition:

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$$

where $\mathbf{\Lambda}$ is a diagonal matrix, and $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices.

Suppose we have the SVD of our ratings matrix

$$R = Q\Sigma P^T,$$

but then we truncate each of Q, Σ, and P s.t. Q and P have only k columns and Σ is $k \times k$:

$$R \approx Q_k \Sigma_k P_k^T$$

For collaborative filtering, let:

$$U \triangleq Q_k \Sigma_k$$

$$V \triangleq P_k$$

$$\Rightarrow U, V = \underset{U,V}{\operatorname{argmin}} \frac{1}{2}||R - UV^T||_2^2$$

s.t. columns of U are mutually orthogonal

s.t. columns of V are mutually orthogonal

Theorem: If *R* fully observed and no regularization, the optimal UV^T from SVD equals the optimal UV^T from Unconstrained MF

NON-NEGATIVE MATRIX FACTORIZATION

Implicit Feedback Datasets

上海科技大学
ShanghaiTech University

- What information does a five-star rating contain?

- Implicit Feedback Datasets:
  - In many settings, users don't have a way of expressing *dislike* for an item (e.g. can't provide negative ratings)
  - The only mechanism for feedback is to "like" something
- Examples:
  - Facebook has a "Like" button, but no "Dislike" button
  - Google's "+1" button
  - Pinterest pins
  - Purchasing an item on Amazon indicates a preference for it, but there are many reasons you might *not* purchase an item (besides dislike)
  - Search engines collect click data but don't have a clear mechanism for observing dislike of a webpage

Examples from Aggarwal (2016)

Non-negative Matrix Factorization

- Constrained Optimization Problem:

$$U, V = \underset{U,V}{\operatorname{argmin}} \frac{1}{2}||R - UV^T||_2^2$$

$$\text{s.t. } U_{ij} \geq 0$$

$$\text{s.t. } V_{ij} \geq 0$$

Multiplicative Updates: simple iterative algorithm for solving just involves multiplying a few entries together

Summary

上海科技大学
ShanghaiTech University

- Recommender systems solve many **real-world** (*large-scale) **problems**
- Collaborative filtering by Matrix Factorization (MF) is an **efficient** and **effective** approach
- MF is just another example of a **common recipe:**
  1. define a model
  2. define an objective function
  3. optimize with your favorite black box optimizer
     (e.g. SGD, Gradient Descent, Block Coordinate Descent aka. Alternating Least Squares)

Learning Objectives

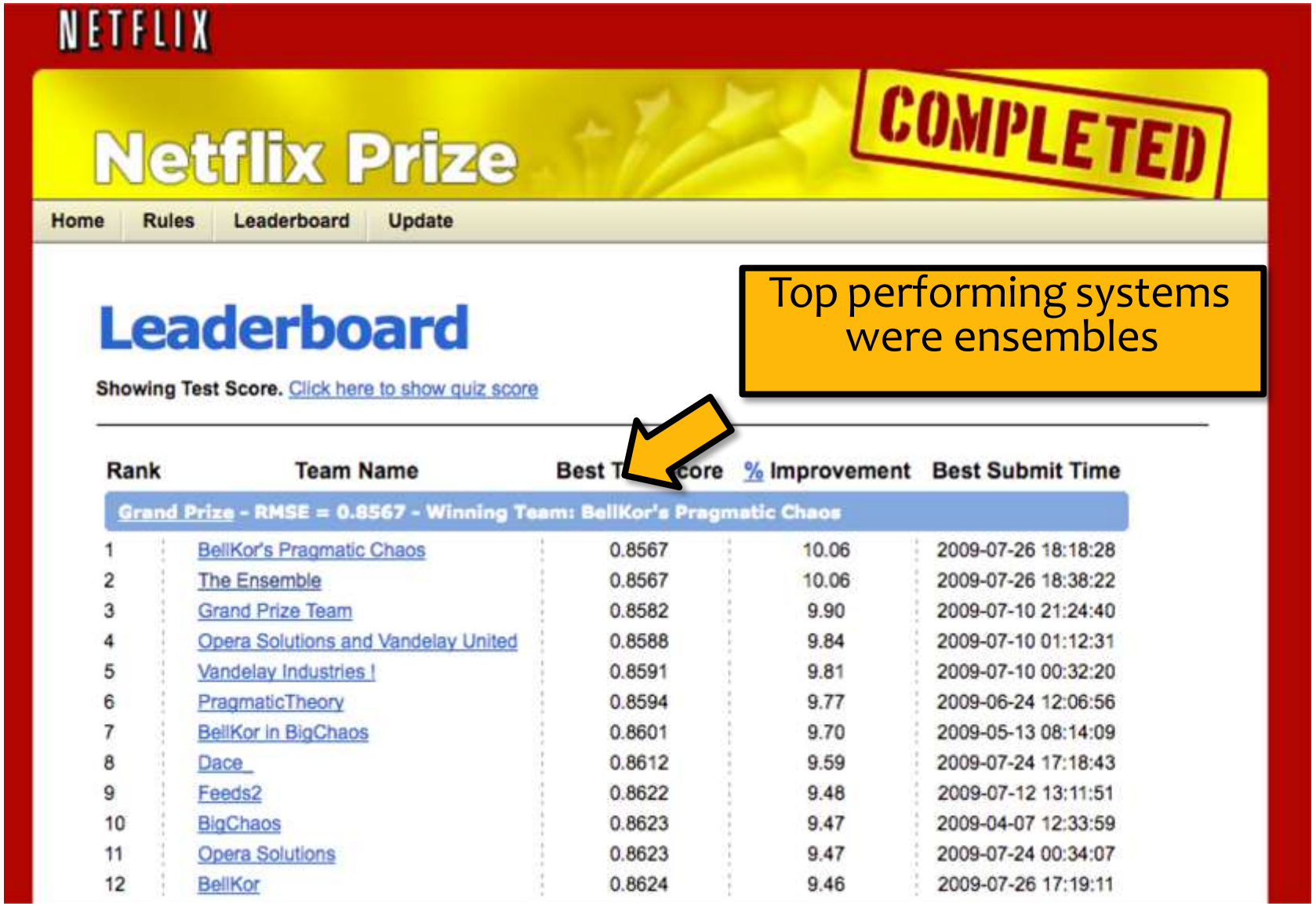
上海科技大学
ShanghaiTech University

Recommender Systems

You should be able to…

1. Compare and contrast the properties of various families of recommender system algorithms: content filtering, collaborative filtering, neighborhood methods, latent factor methods
2. Formulate a squared error objective function for the matrix factorization problem
3. Implement unconstrained matrix factorization with a variety of different optimization techniques: gradient descent, stochastic gradient descent, alternating least squares
4. Offer intuitions for why the parameters learned by matrix factorization can be understood as user factors and item factors

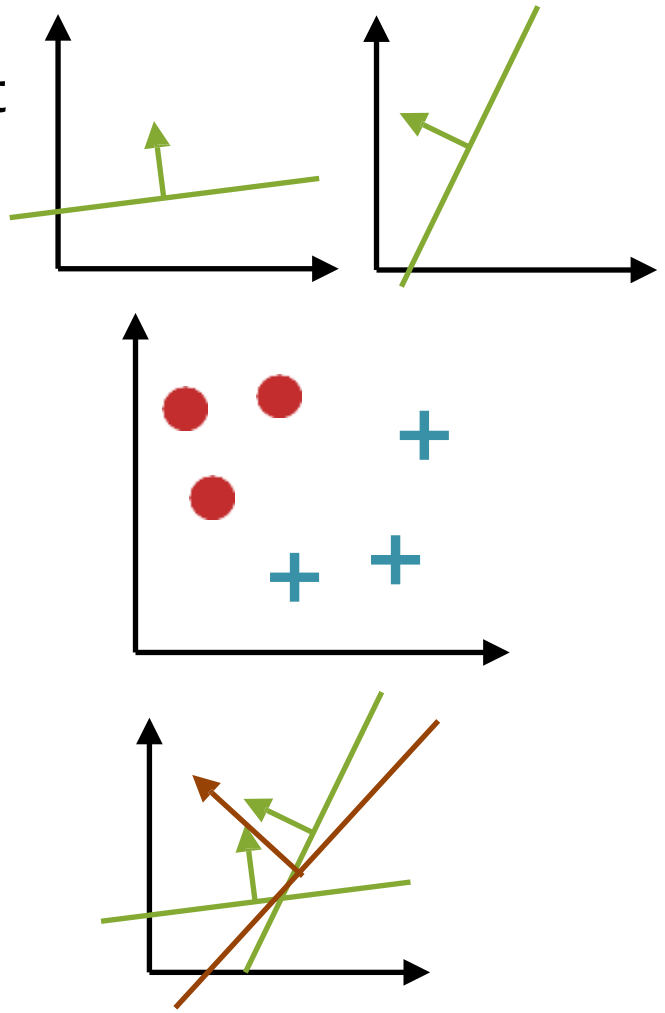
Recommender Systems



Weighted Majority Algorithm

(Littlestone & Warmuth, 1994)

- **Given:** pool *A* of binary classifiers (that you know nothing about)
- **Data:** stream of examples (i.e. online learning setting)
- **Goal:** design a new learner that uses the predictions of the pool to make new predictions
- **Algorithm:**
  - Initially weight all classifiers equally
  - Receive a training example and predict the (weighted) majority vote of the classifiers in the pool
  - Down-weight classifiers that contribute to a mistake by a factor of β



Weighted Majority Algorithm

(Littlestone & Warmuth, 1994)

上海科技大学 ShanghaiTech University

Suppose we have a pool of T binary classifiers $\mathcal{A} = \{h_1, \ldots, h_T\}$ where $h_t : \mathbb{R}^M \to \{+1, -1\}$. Let α_t be the weight for classifier h_t.

hyperparameter, $\beta \in (0, 1)$

Algorithm 1 Weighted Majority Algorithm

1: **procedure** WEIGHTEDMAJORITY($\mathcal{A}$, β)
2: Initialize classifier weights $\alpha_t = 1, \ \forall t \in \{1, \ldots, T\}$
3: **for** each training example $(\mathbf{x}, y)$ **do**
4: Predict majority vote class (splitting ties randomly)

$$\hat{h}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

5: **if** a mistake is made $\hat{h}(x) \neq y$ **then**
6: **for** each classifier $t \in \{1, \ldots, T\}$ **do**
7: If $h_t(x) \neq y$, then $\alpha_t \leftarrow \beta\alpha_t$

Weighted Majority Algorithm

(Littlestone & Warmuth, 1994)

Suppose we have a pool of T binary classifiers $\mathcal{A} = \{h_1, \ldots, h_T\}$ where $h_t : \mathbb{R}^M \rightarrow \{+1, -1\}$. Let α_t be the weight for classifier h_t.

hyperparameter, $\beta \in (0, 1)$

Algorithm 1 Weighted Majority Algorithm

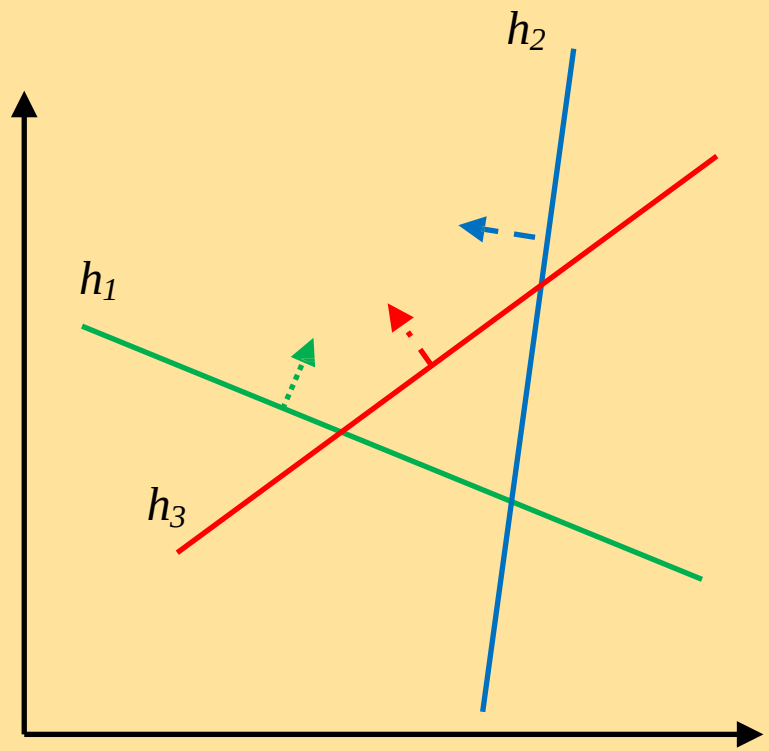
1: **procedure** WEIGHTEDMAJORITY($\mathcal{A}$, β)
2: Initialize classifier weights $\alpha_t = 1, \ \forall t \in \{1, \ldots, T\}$
3: **for** each training example $(\mathbf{x}, y)$ **do**
4: Predict majority vote class (splitting ties randomly)

$$\hat{h}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

5: **if** a mistake is made $\hat{h}(x) \neq y$ **then**
6: **for** each classifier $t \in \{1, \ldots, T\}$ **do**
7: If $h_t(x) \neq y$, then $\alpha_t \leftarrow \beta\alpha_t$

What does the majority vote decision boundary look like?

Suppose $\alpha_1 = 1, \alpha_2 = 1, \alpha_3 = 1$



Weighted Majority Algorithm

上海科技大学 ShanghaiTech University

Theorems (Littlestone & Warmuth, 1994)

For the general case where *WM* is applied to a pool $\mathcal{A}$ of algorithms we show the following upper bounds on the number of mistakes made in a given sequence of trials:

1. $O(\log |\mathcal{A}| + m)$, if one algorithm of $\mathcal{A}$ makes at most m mistakes.

2. $O(\log \frac{|\mathcal{A}|}{k} + m)$, if each of a subpool of k algorithms of $\mathcal{A}$ makes at most m mistakes.

3. $O(\log \frac{|\mathcal{A}|}{k} + \frac{m}{k})$, if the total number of mistakes of a subpool of k algorithms of $\mathcal{A}$ is at most m.

These are "mistake bounds" of the variety we saw for the Perceptron algorithm

ADABOOST

Comparison

上海科技大学
ShanghaiTech University

Weighted Majority Algorithm

- an example of an ensemble method
- assumes the classifiers are learned ahead of time
- only learns (majority vote) weight for each classifiers

AdaBoost

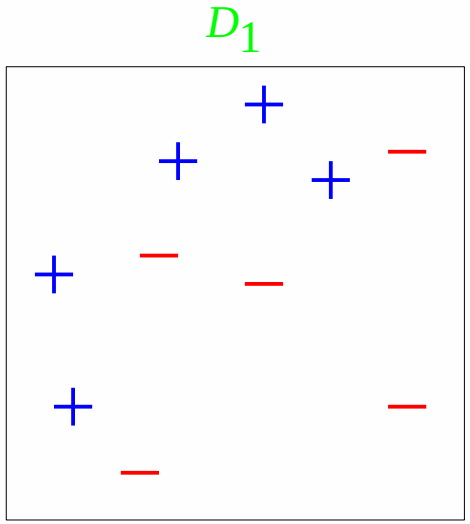
- an example of a boosting method
- simultaneously learns:
  - the classifiers themselves
  - (majority vote) weight for each classifiers

AdaBoost

- Definitions
  - Def: a **weak learner** is one that returns a hypothesis that is not much better than random guessing
  - Def: a **strong learner** is one that returns a hypothesis of arbitrarily low error

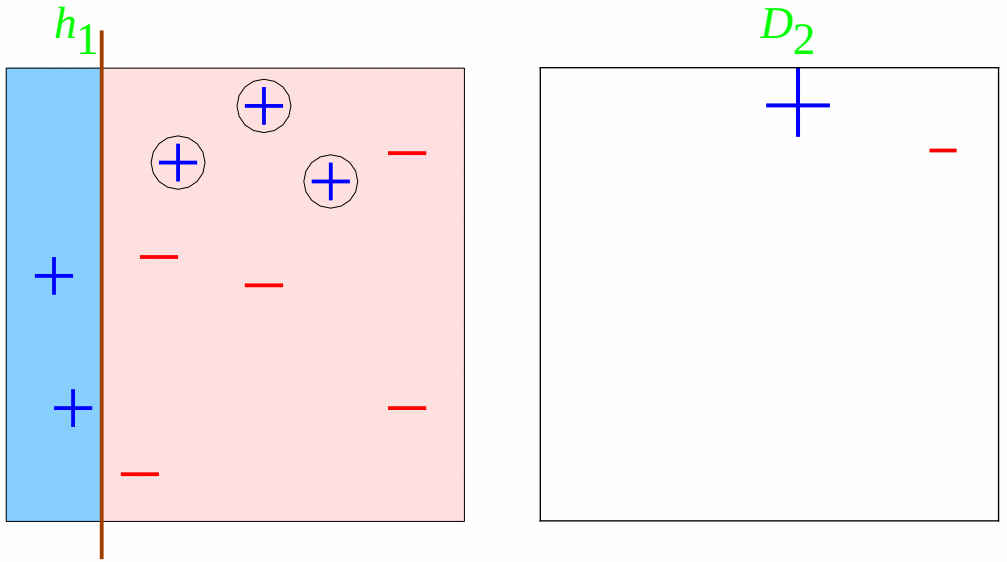
- AdaBoost answers the following question:
  - *Does that exist an efficient learning algorithm that can combine weak learners to obtain a strong learner?*

AdaBoost: Toy Example



weak classifiers = vertical or horizontal half-planes

AdaBoost: Toy Example

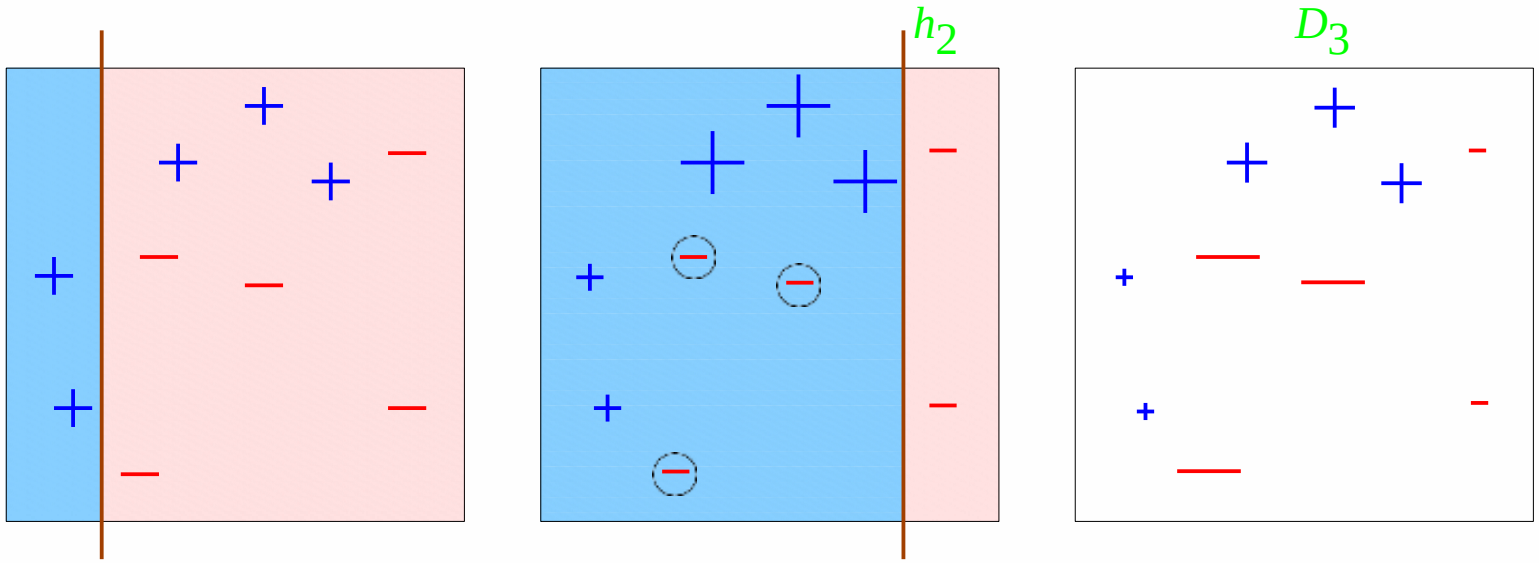


$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

Slide from Schapire NeurIPS Tutorial

AdaBoost: Toy Example

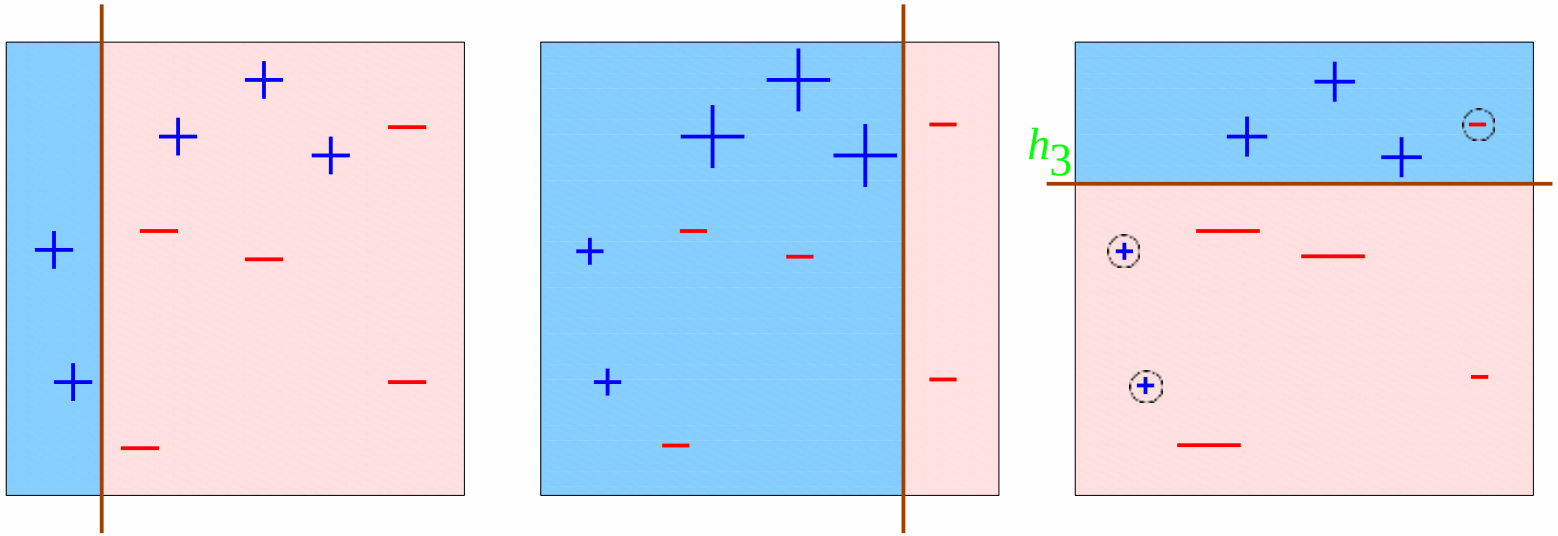


$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

Slide from Schapire NeurIPS Tutorial

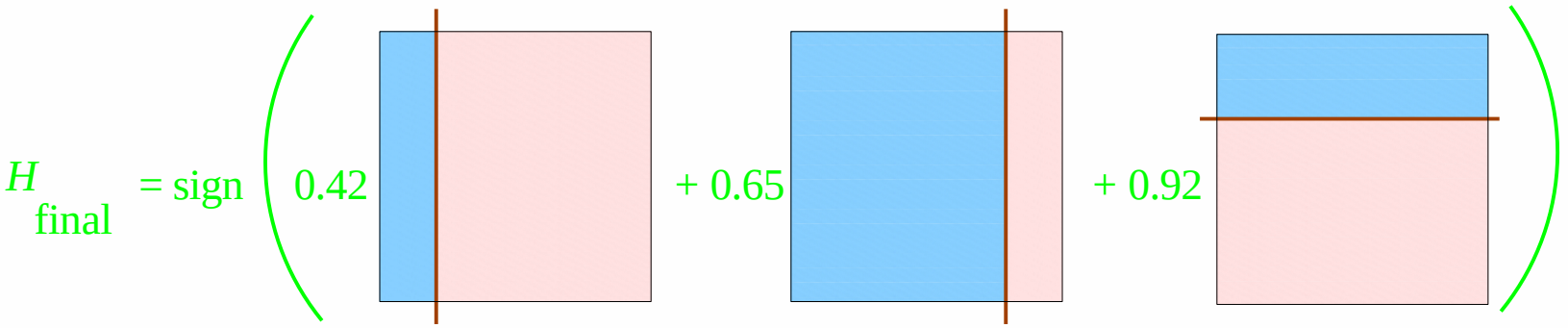
AdaBoost: Toy Example

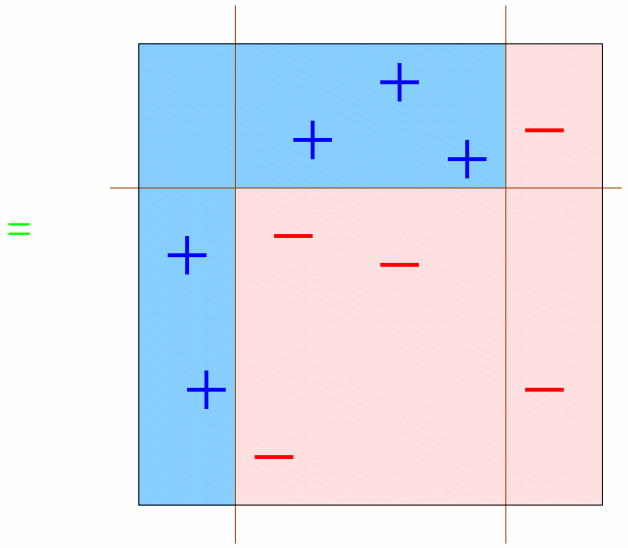


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

AdaBoost: Toy Example





AdaBoost

Algorithm 1 AdaBoost Algorithm

1: Given: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ where $\mathbf{x}_i \in \mathbb{R}^M, y_i \in \{-1, +1\}$

2: Initialize $D_1(i) = \frac{1}{N}$

3: **for** $t = 1, \ldots, T$ **do**

4: Train weak learner using distribution D_t.

5: Get weak hypothesis $h_t : \mathbb{R}^M \to \{-1, +1\}$ with error

$$\epsilon_t = P_{i \sim D_t}[h_t(\mathbf{x}_i) \neq y_i]$$

6: Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$. for high error, we get _____ α_t
for low error, we get _____ α_t

7: **for** $i = 1, \ldots, N$ **do**

8: Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\mathbf{x}_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases}$$

if correct, _____
if incorrect, _____
↳ for high error, _____
↳ for low error, _____

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

where normalization const. Z_t chosen s.t. D_{t+1} is a distribution.

9: Output the final hypothesis: $H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right)$.

Algorithm adapted from (Freund & Schapire, 1999)

AdaBoost

上海科技大学 ShanghaiTech University

Algorithm 1 AdaBoost Algorithm

1: Given: $(\mathrm{x}_1, y_1), \ldots, (\mathrm{x}_N, y_N)$ where $\mathrm{x}_i \in \mathrm{R}^M, y_i \in \{-1, +1\}$

2: Initialize $D_1(i) = \frac{1}{N}$

3: **for** $t = 1, \ldots, T$ **do**

4: Train weak learner using distribution D_t.

5: Get weak hypothesis $h_t : \mathrm{R}^M \to \{-1, +1\}$ with error

$$\epsilon_t = P_{i \sim D_t}[h_t(\mathrm{x}_i) \neq y_i]$$

6: Choose $\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$.

for high error, we get _____ α_t

for low error, we get _____ α_t

7: **for** $i = 1, \ldots, N$ **do**

8: Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\mathrm{x}_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(\mathrm{x}_i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(\mathrm{x}_i))}{Z_t}$$

if correct, _____

if incorrect, _____

↳ for high error, _____

↳ for low error, _____

where normalization const. Z_t chosen s.t. D_{t+1} is a distribution.

9: Output the final hypothesis: $H(\mathrm{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathrm{x})\right)$.

Algorithm adapted from (Freund & Schapire, 1999)

AdaBoost: Theory

(Training) Mistake Bound

The most basic theoretical property of AdaBoost concerns its ability to reduce the training error. Let us write the error ϵ_t of h_t as $\frac{1}{2} - \gamma_t$. Since a hypothesis that guesses each instance's class at random has an error rate of $1/2$ (on binary problems), γ_t thus measures how much better than random are h_t's predictions. Freund and Schapire [23] prove that the training error (the fraction of mistakes on the training set) of the final hypothesis H is at most

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_t \gamma_t^2\right). \quad (1)$$

Thus, if each weak hypothesis is slightly better than random so that $\gamma_t \geq \gamma$ for some $\gamma > 0$, then the training error drops exponentially fast.

Figure from (Freund & Schapire, 1999)

AdaBoost: Theory

Generalization Error

Freund and Schapire [23] showed how to bound the generalization error of the final hypothesis in terms of its training error, the sample size N, the VC-dimension d of the weak hypothesis space and the number of boosting rounds T. (The VC-dimension is a standard measure of the "complexity" of a space of hypotheses. See, for instance, Blumer et al. [5].) Specifically, they used techniques from Baum and Haussler [4] to show that the generalization error, with high probability, is at most

$$\hat{\Pr}\left[H(x) \neq y\right] + \tilde{O}\left(\sqrt{\frac{Td}{N}}\right)$$

where $\hat{\Pr}\left[\cdot\right]$ denotes empirical probability on the training sample. This bound suggests that boosting will overfit if run for too many rounds, i.e., as T becomes large. In fact, this sometimes does happen. However, in early experiments, several authors [9, 15, 36] observed empirically that boosting often does *not* overfit, even when run for thousands of rounds. Moreover, it was observed that AdaBoost would sometimes continue to drive down the generalization error long after the training error had reached zero, clearly contradicting the spirit of the bound above. For instance, the left

Figure from (Freund & Schapire, 1999)

AdaBoost

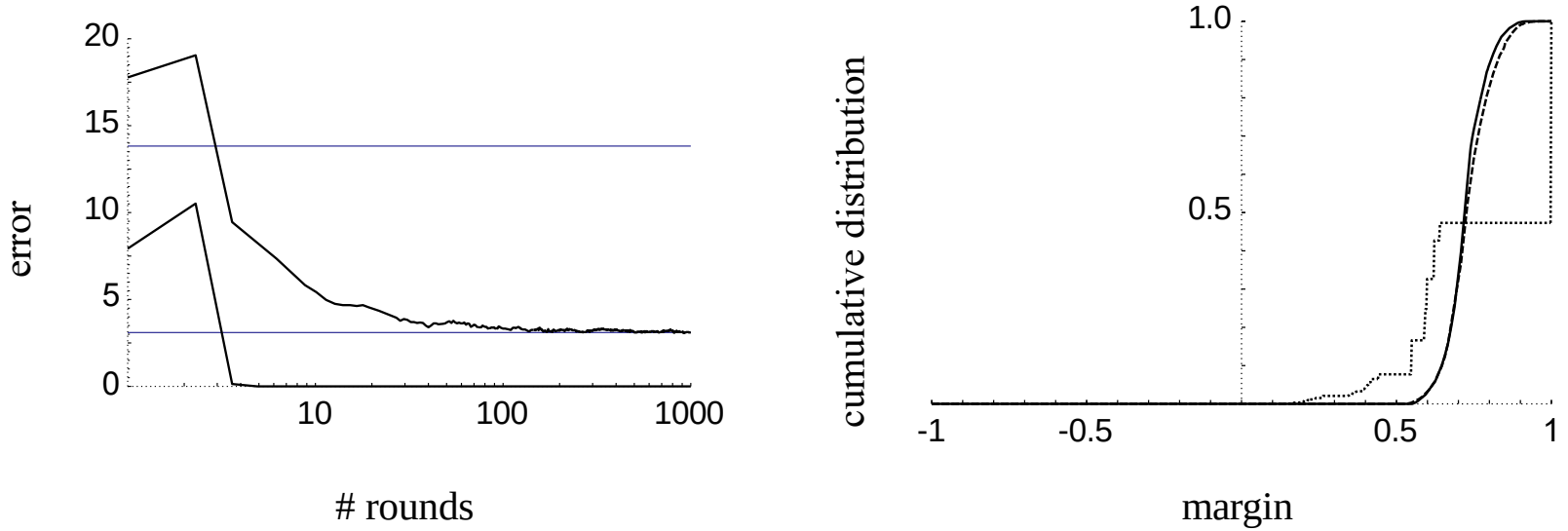


Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [41]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. *Right*: The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

Figure from (Freund & Schapire, 1999)

Ensemble Methods: Boosting

You should be able to…

1. Explain how a weighted majority vote over linear classifiers can lead to a non-linear decision boundary
2. Implement AdaBoost
3. Describe a surprisingly common empirical result regarding Adaboost train/test curves