10-301/601: Introduction to Machine Learning Lecture 23 – Recommender Systems

Matt Gormley & Henry Chai

4/9/25

Front Matter

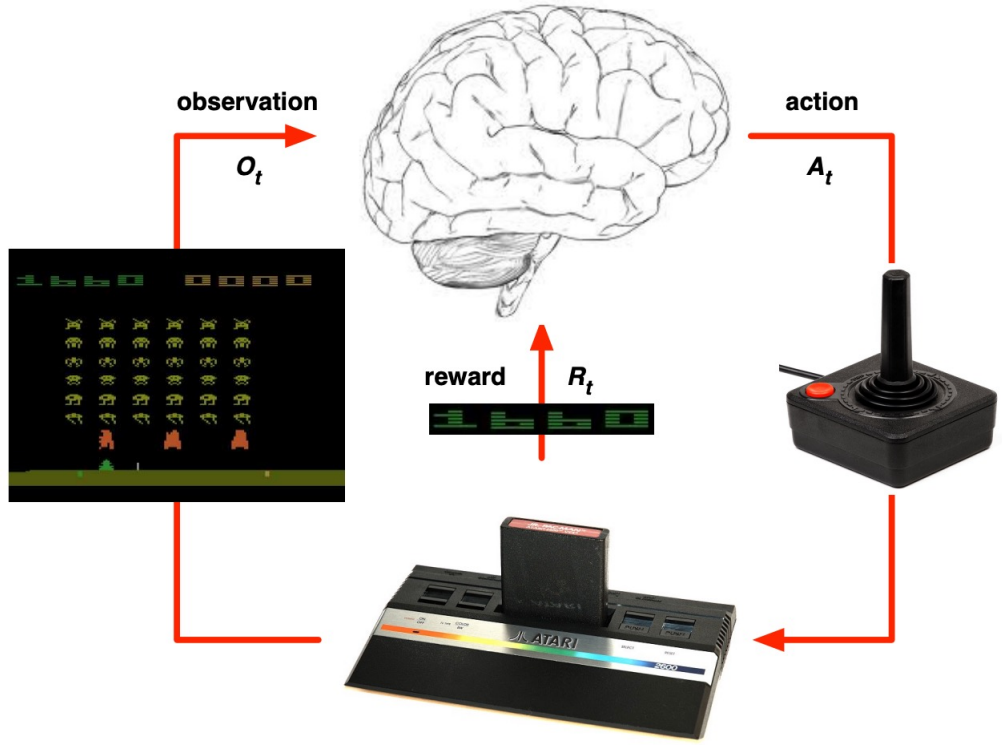
- Announcements
  - HW8 released 4/8, due 4/16 at 11:59 PM

Q: So what kinds of things can we do with all this RL stuff anyway?

- A: I mean really the same things that you or I do, like play video games!

Playing Atari with Deep RL:

- Agent observes the pixels on the screen
- Reward is tied to the score
- Actions are the joystick inputs
- No knowledge about the rules/dynamics of the game!



Source: https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf

Q: So what kinds of things can we do with all this RL stuff anyway?

- A: I mean really the same things that you or I do, like play video games!

Source: https://www.youtube.com/watch?v=V1eYniJ0Rnk&t=2s&ab_channel=TwoMinutePapers

Q: So what kinds of things can we do with all this RL stuff anyway?

- A: I mean really the same things that you or I do, like play video games!

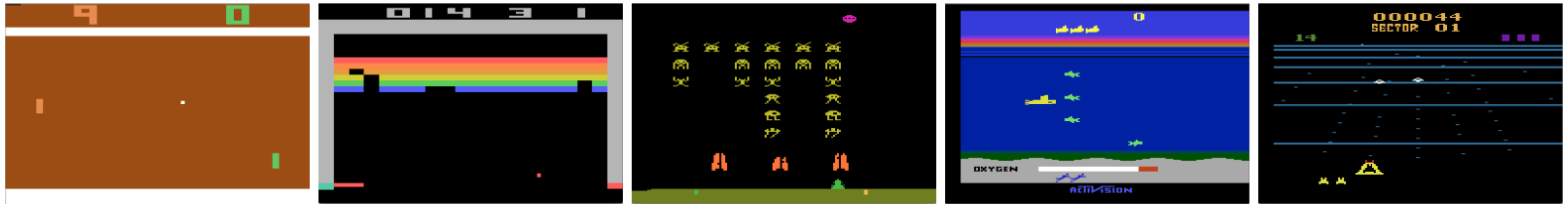


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

Source: https://arxiv.org/pdf/1312.5602

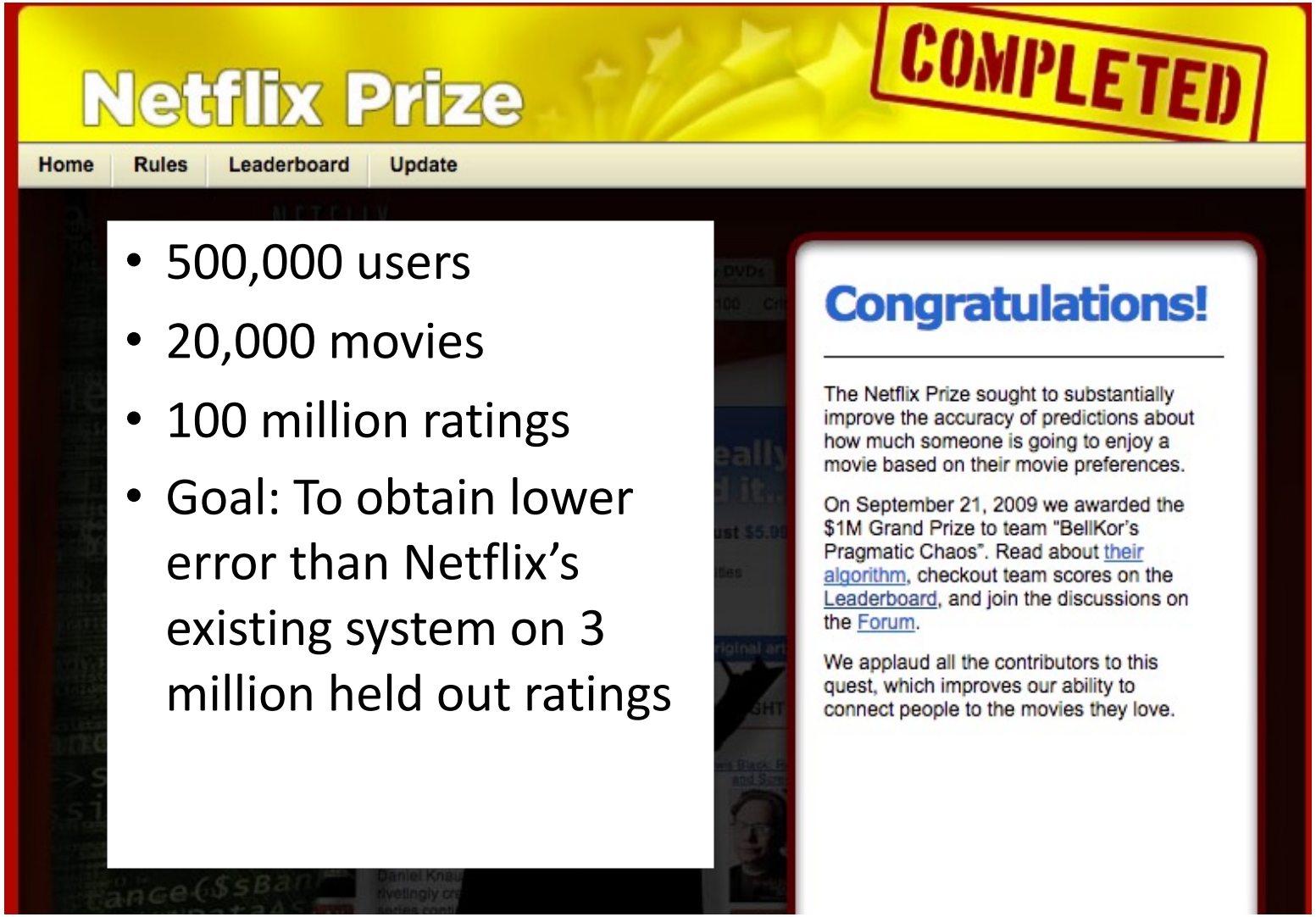
Outline

Today, we're going to introduce two distinct topics:

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

We'll use a prominent example of a recommender systems to motivate both topics…

The Netflix Prize



Source: https://web.archive.org/web/20090926213457/http://www.netflixprize.com/leaderboard

Recommender Systems

- **Setup:**
  1. Items:
     movies, songs,
     products, etc.
     (often many thousands)
  2. Users:
     watchers, listeners,
     buyers, etc.
     (often many millions)
  3. 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 (very) sparse

	Dune: Part 2	Anora	Snow White
Alice	5		1
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 metadata or "side information" about items (e.g. properties of a song)
- **Pro:** Can make recommendations of new items, without previous ratings

Collaborative Filtering

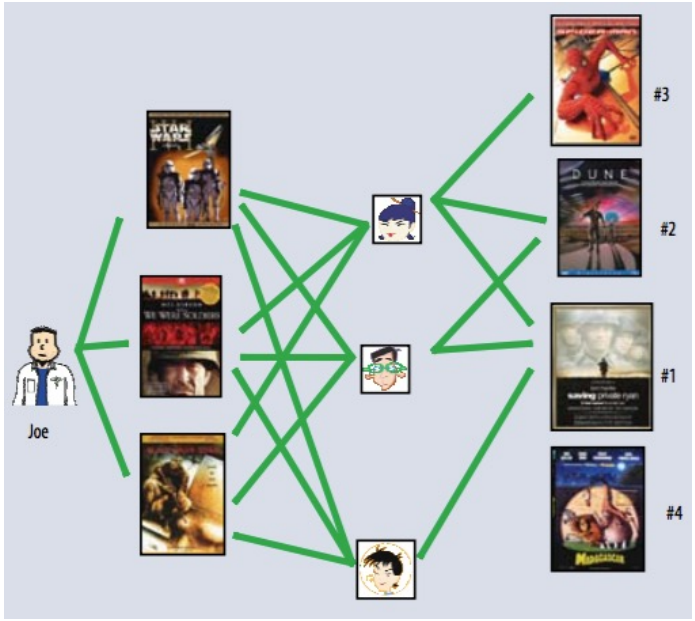
- *Example*: Netflix movie recommendations
- **Pro:** Does not require access to side information about items (e.g. does not need to know about movie genres or actors)
- **Con:** Does not work on new items that have no ratings

Collaborative Filtering

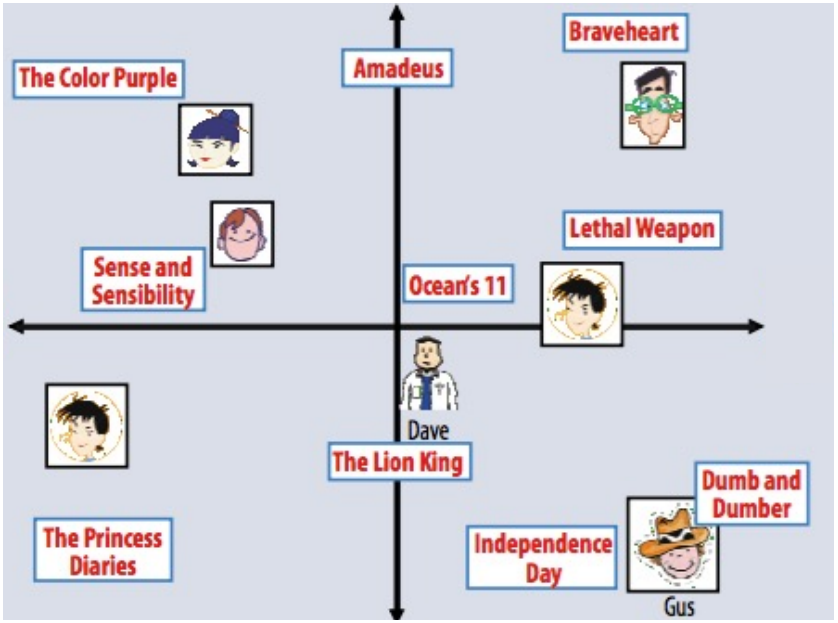
- Collaborative filtering is everywhere!
- Examples:
  - Bestseller lists
  - Top 40 music lists
  - The “recent returns” shelf at the library
  - Unmarked but well-used paths thru the woods
  - “Read any good books lately?”
  - …
- **Insight:** personal tastes are correlated
  - If Alice and Bob both like X and Alice likes Y then Bob is more likely to like Y
    - especially (perhaps) if Bob knows Alice

Two Types of Collaborative Filtering

1. Neighborhood Methods

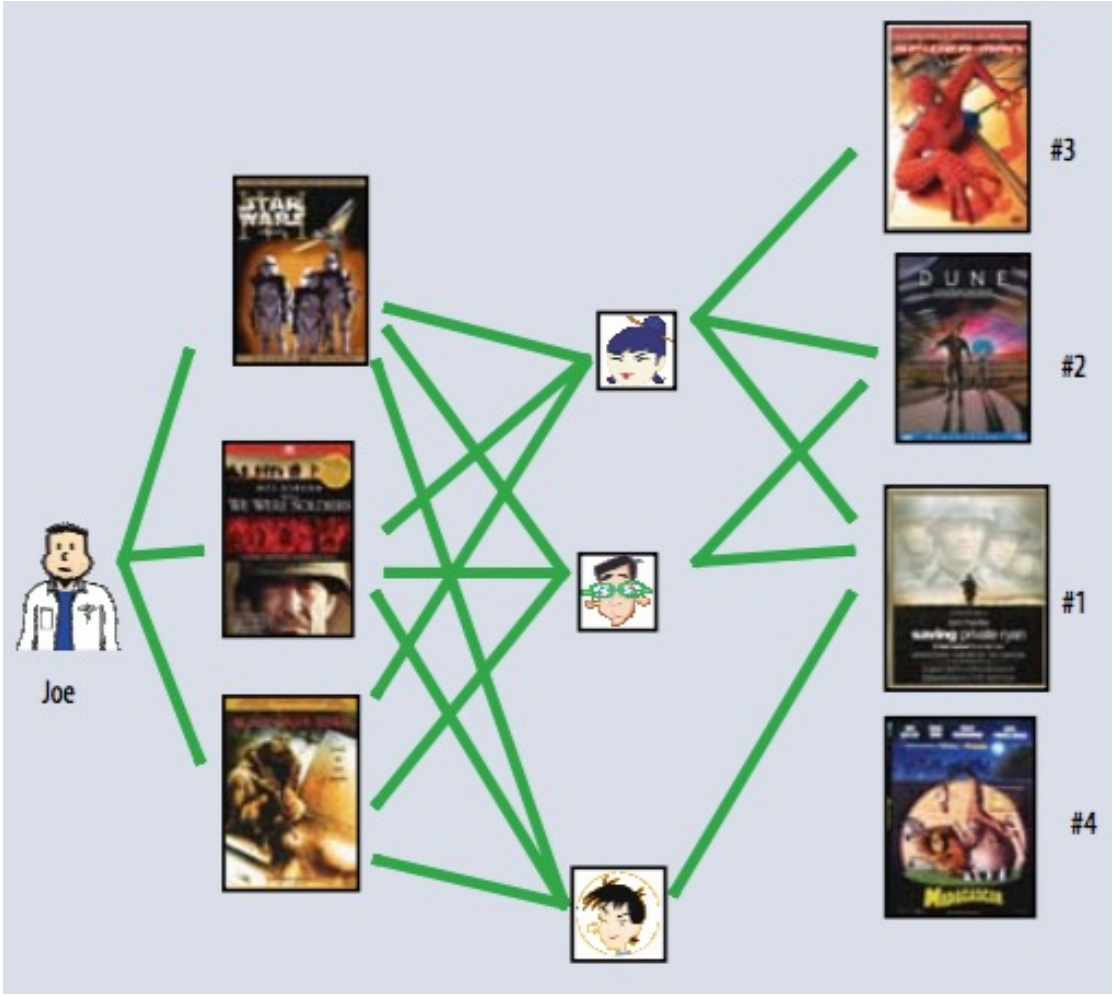


2. Latent Factor Methods



Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5197422

Neighborhood Methods



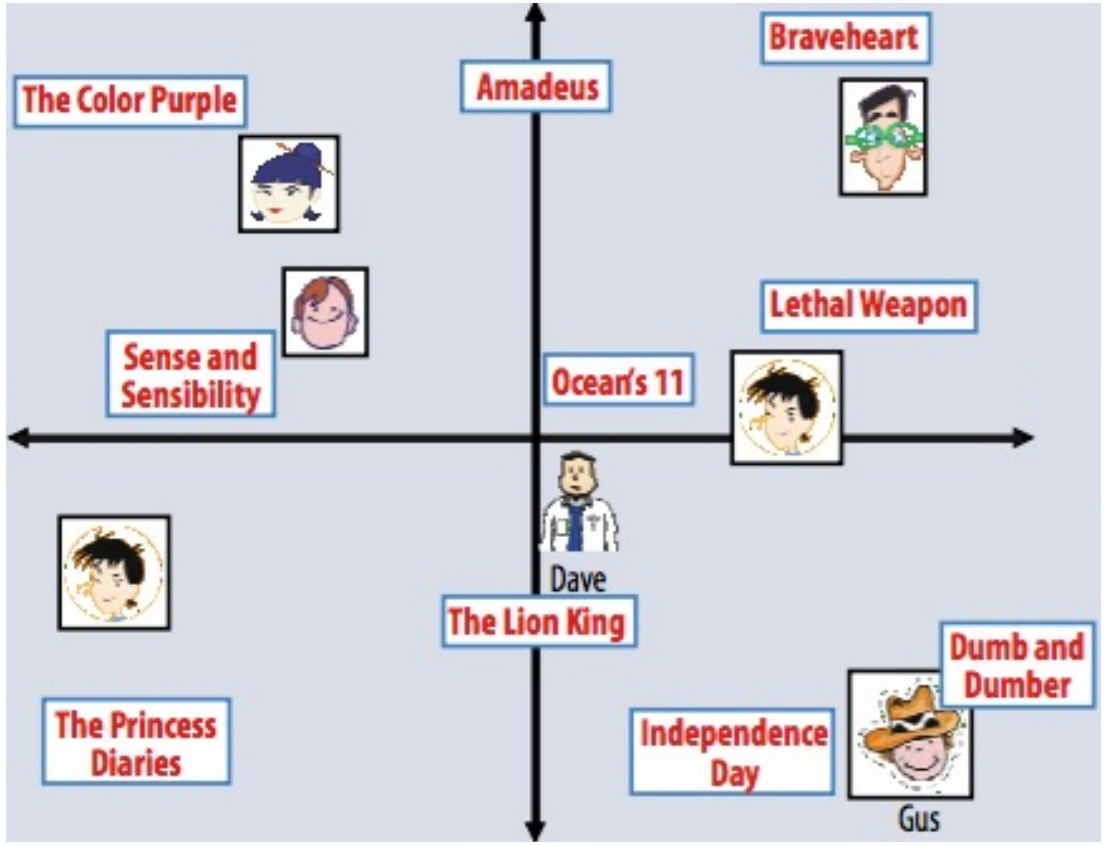
In the figure, a green line indicates the movie was **liked**

Algorithm:

1. **Find neighbors** based on similarity of movie preferences
2. **Recommend** movies that those neighbors also liked

Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5197422

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

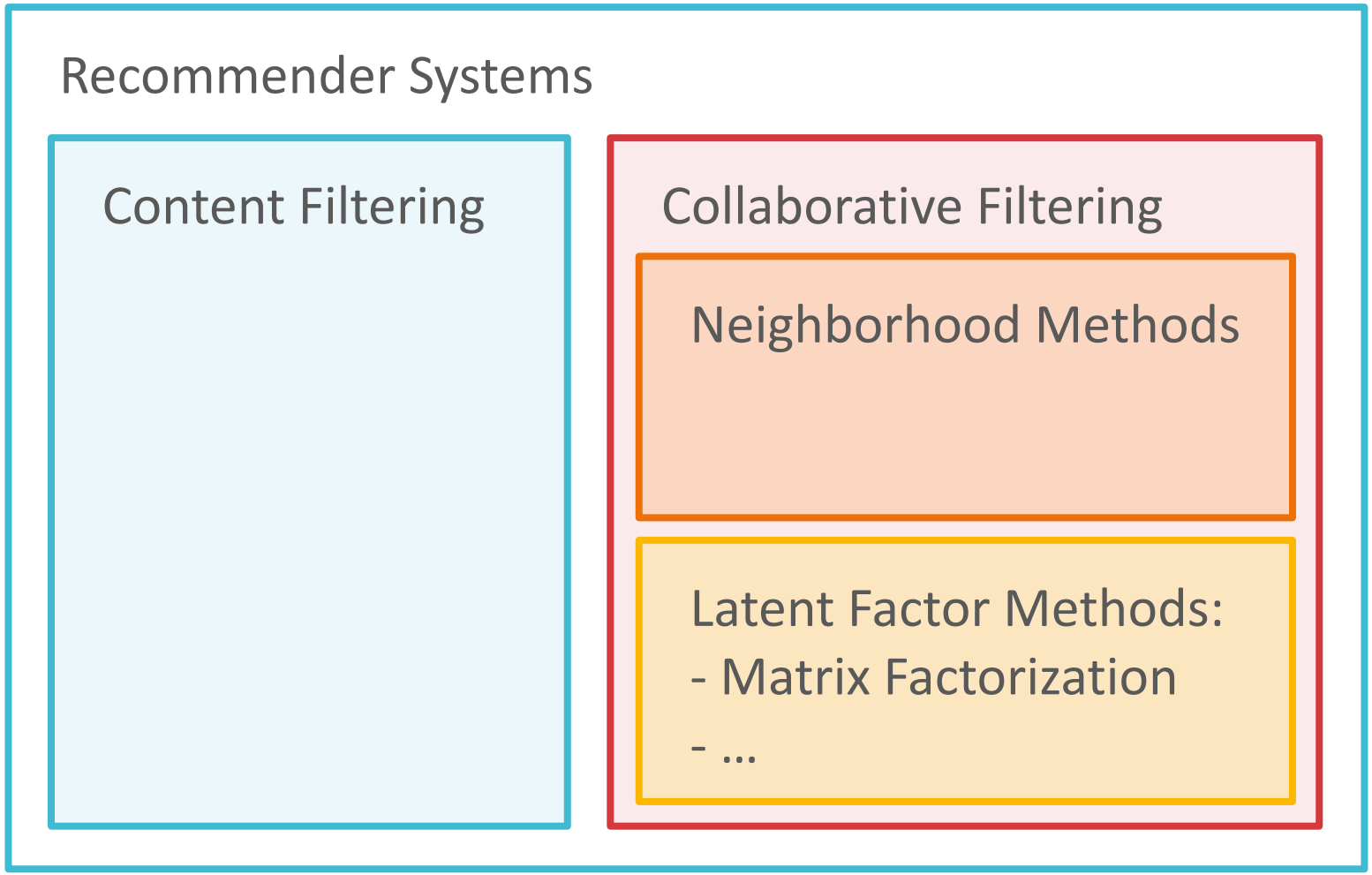
Source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5197422

Poll Question 1:

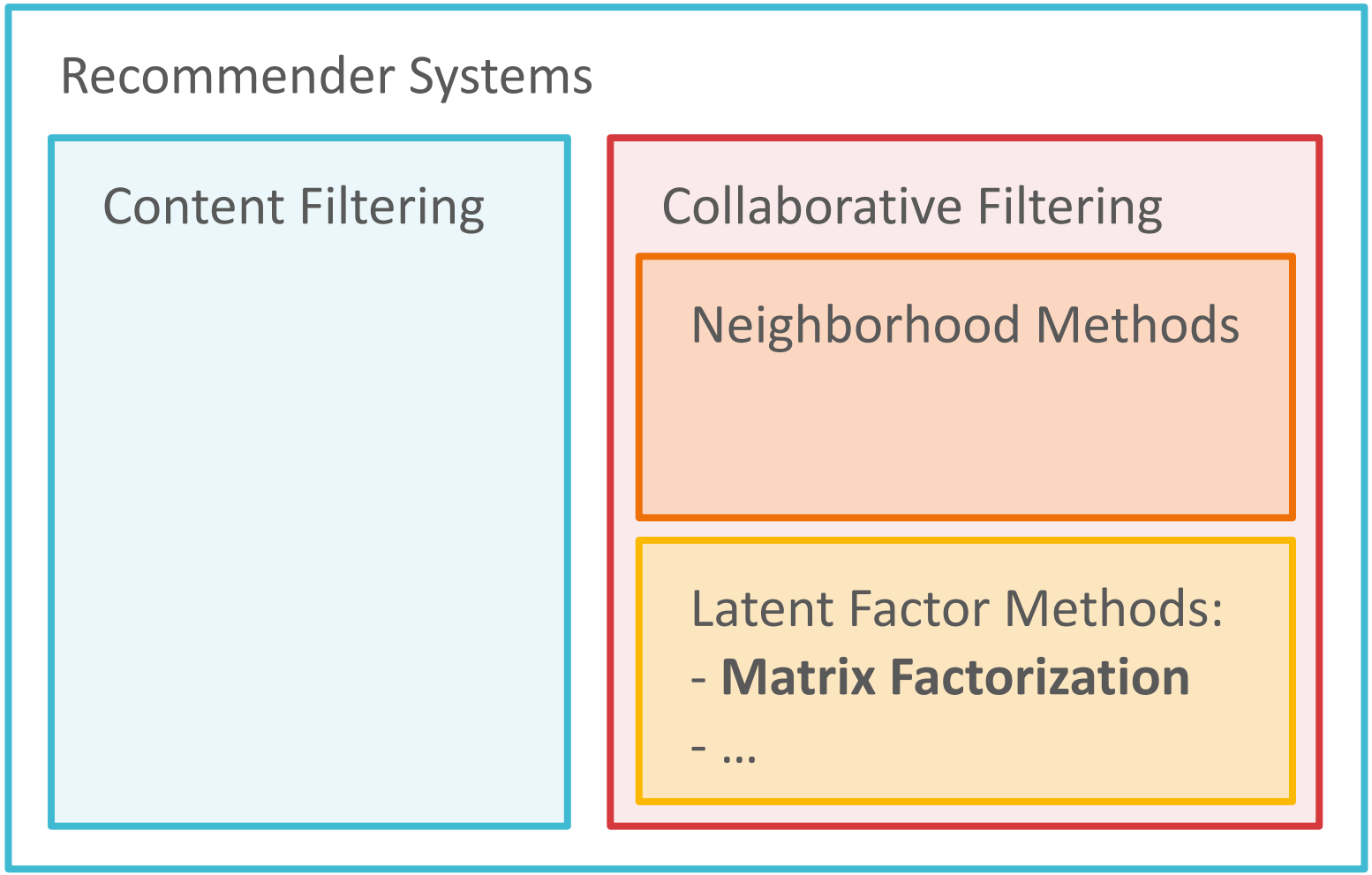
Which of the following methods always requires side information? Select all that apply.

A. ensemble methods **(TOXIC)**
B. collaborative filtering
C. latent factor methods
D. ensemble methods
E. content filtering
F. neighborhood methods
G. recommender systems

Summary Thus Far



Summary Thus Far

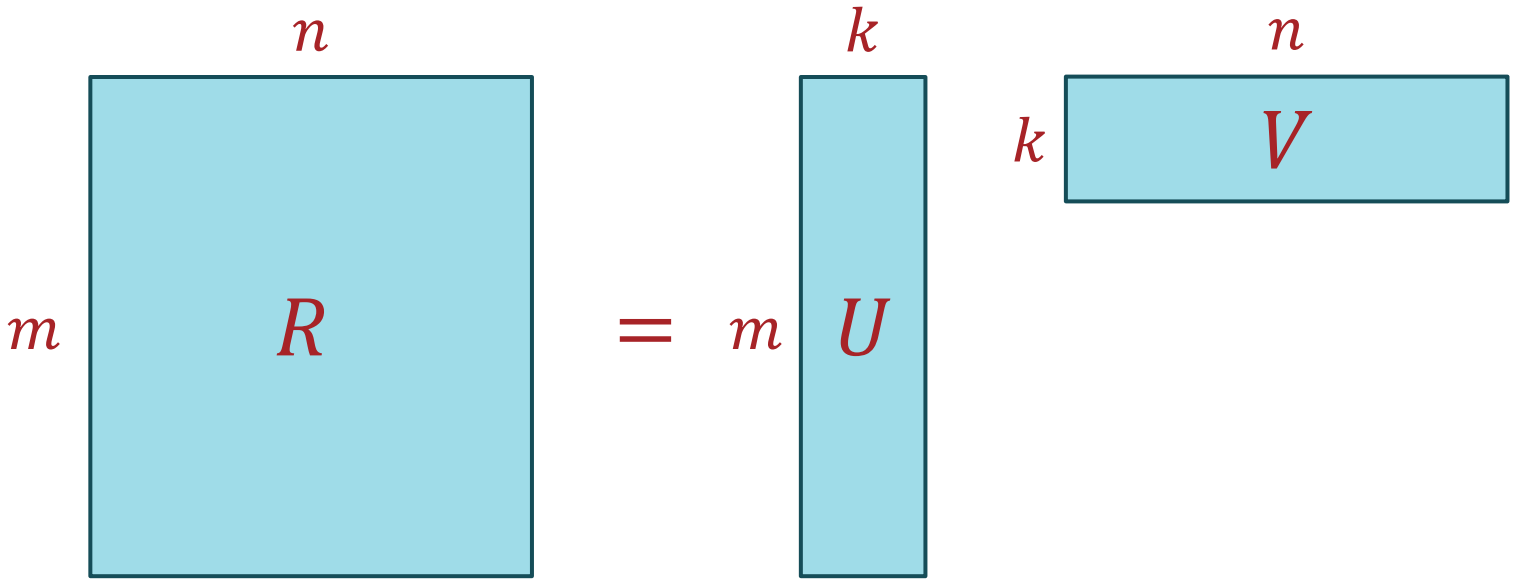


Matrix Factorization

- **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

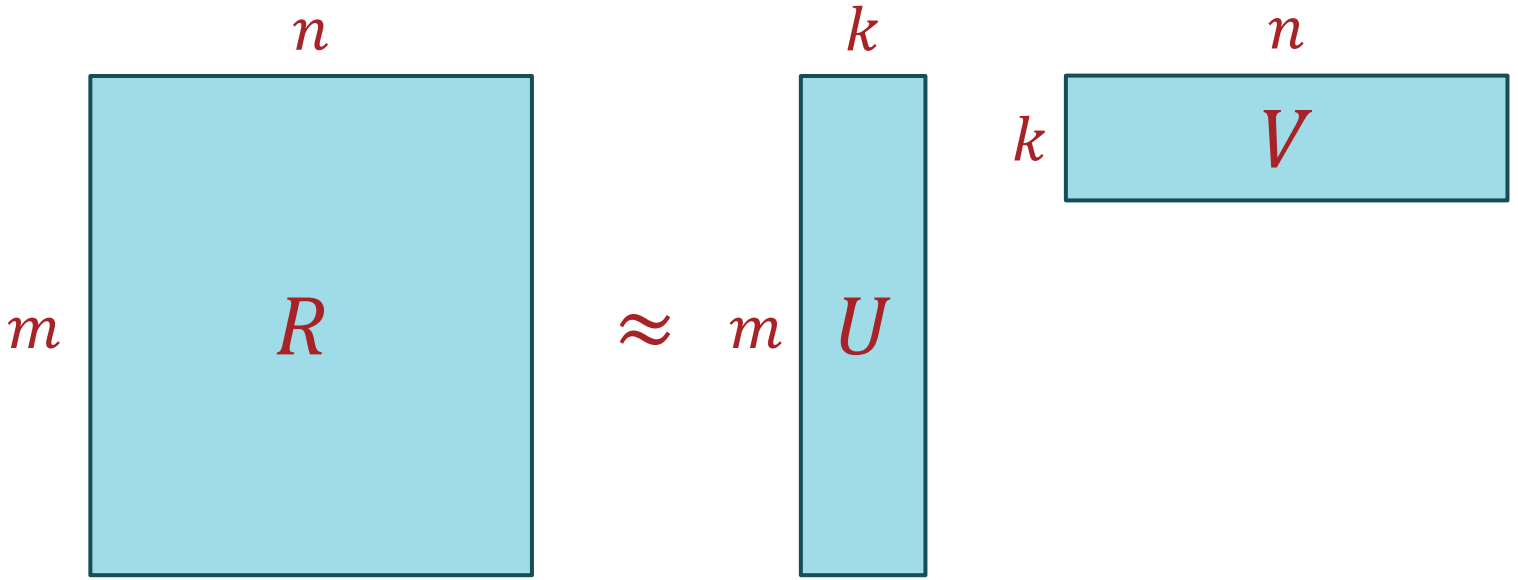
Low-rank Matrix Factorization

- **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

- **Idea:** even if $R \in \mathbb{R}^{m \times n}$ has *rank* $l > k$, then there still exist $U \in \mathbb{R}^{m \times k}$ 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 $R \approx UV^T$



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:

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

where $U_{j,\cdot}$ is the j^{th} row of U and $V_{\cdot,i}^T$ 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 $Z = \{(i,j): R_{i,j} \text{ is known}\}$
  - 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 $U_{j,\cdot}$ 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 $E_{i,j} = R_{i,j} - U_{j,.}V_{.,i}^T$
  - update $U_{j,.}$ and $V_{.,i}^T$:
    - $U_{j,.} \leftarrow U_{j,.} - \eta \nabla_{U_{j,.}} J_{i,j}(U, V)$
    - $V_{.,i}^T \leftarrow V_{.,i}^T - \eta \nabla_{V_{.,i}^T} J_{i,j}(U, V)$

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

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

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

Regularized SGD for Partially-Observed Low-rank Matrix Factorization

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

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

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

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

SGD for Partially-Observed Low-rank Matrix Factorization

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

with user and item bias terms O_j and P_i respectively:

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

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!

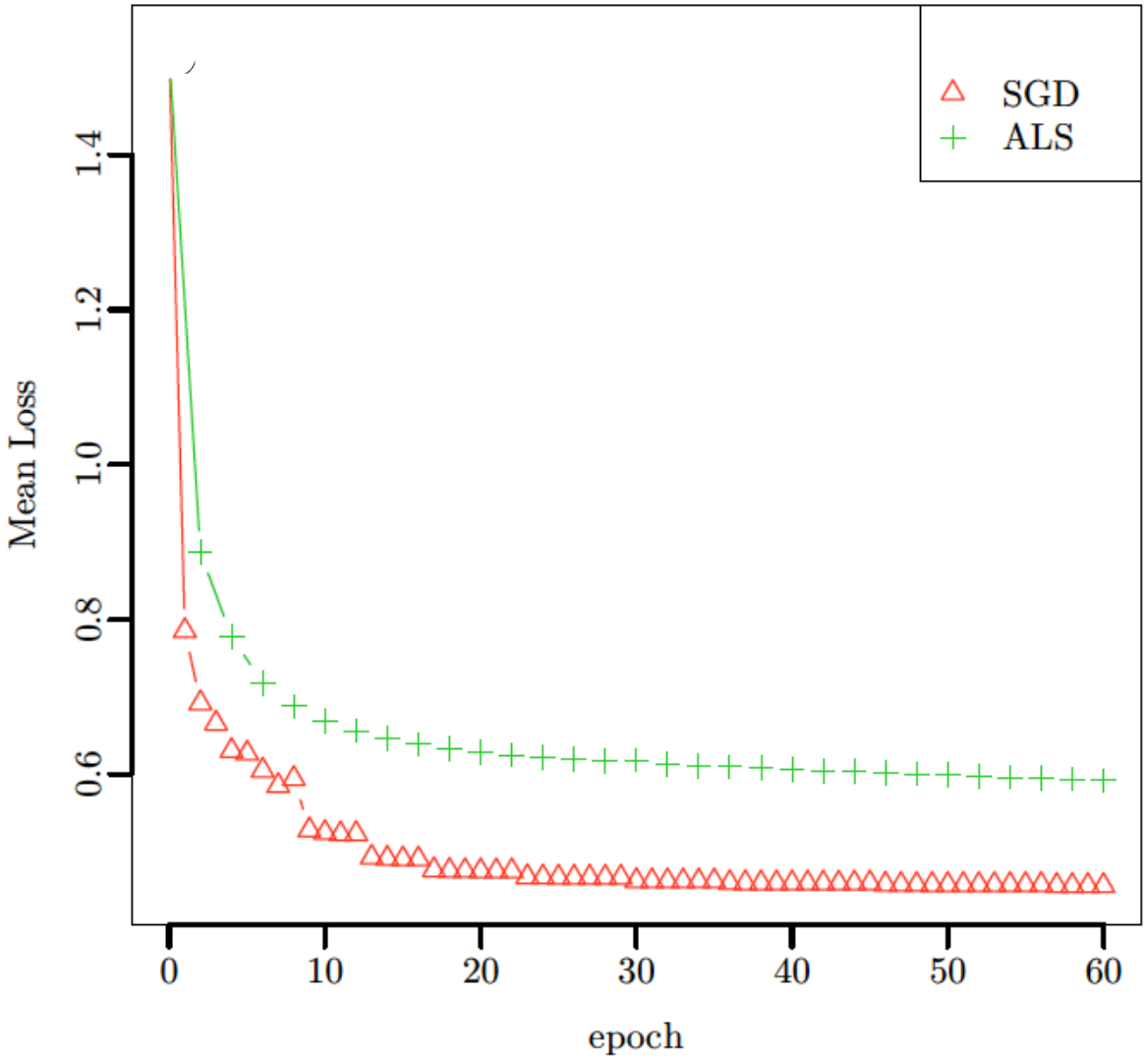
$$J(U,V) = \frac{1}{2}\sum_{(i,j)\in Z}\left(R_{i,j} - U_{j,}.V^T_{.,i}\right)^2$$

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

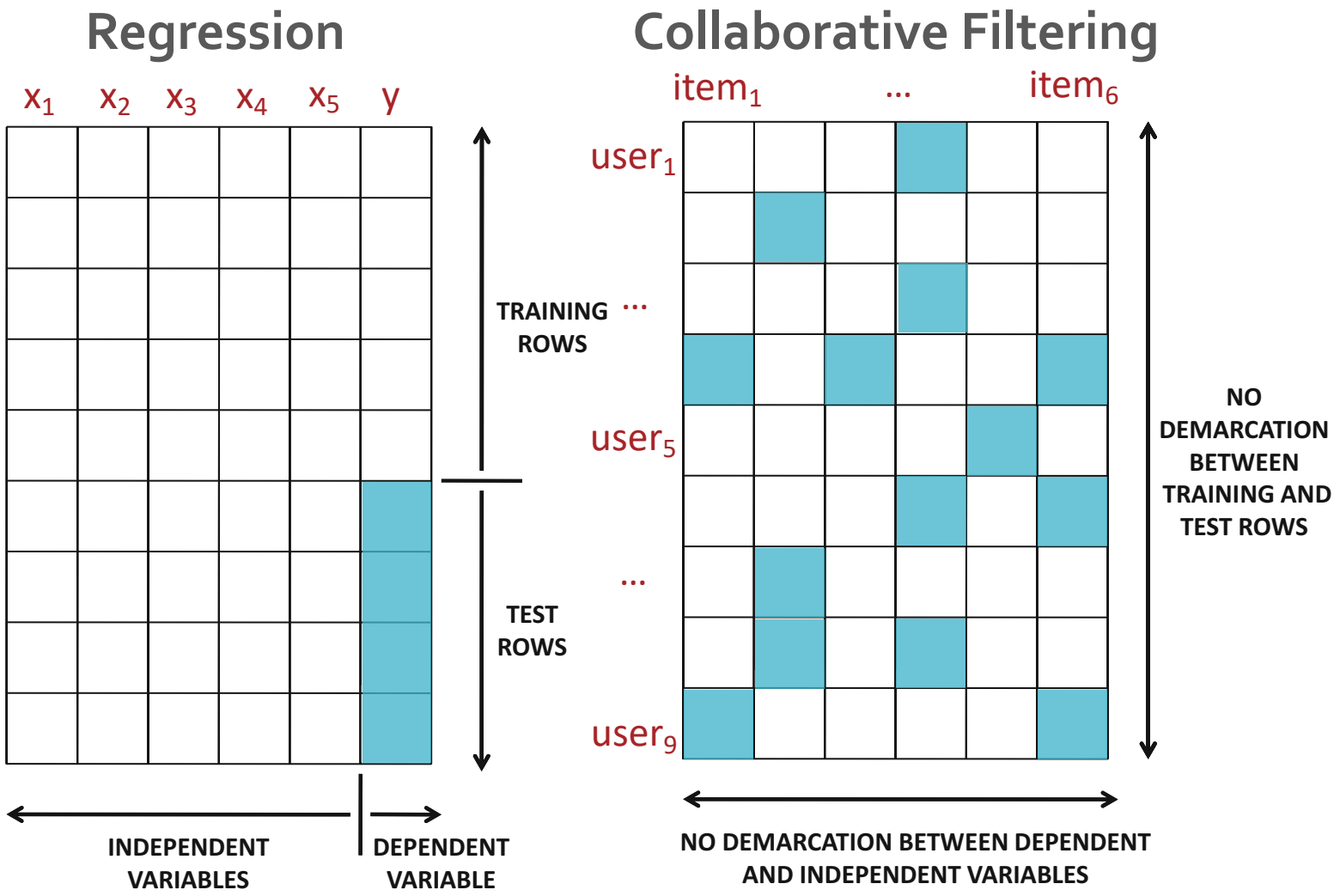
Low-rank Matrix Factorization: Comparison



Source: https://people.cs.umass.edu/~phaas/files/rj10482Updated.pdf

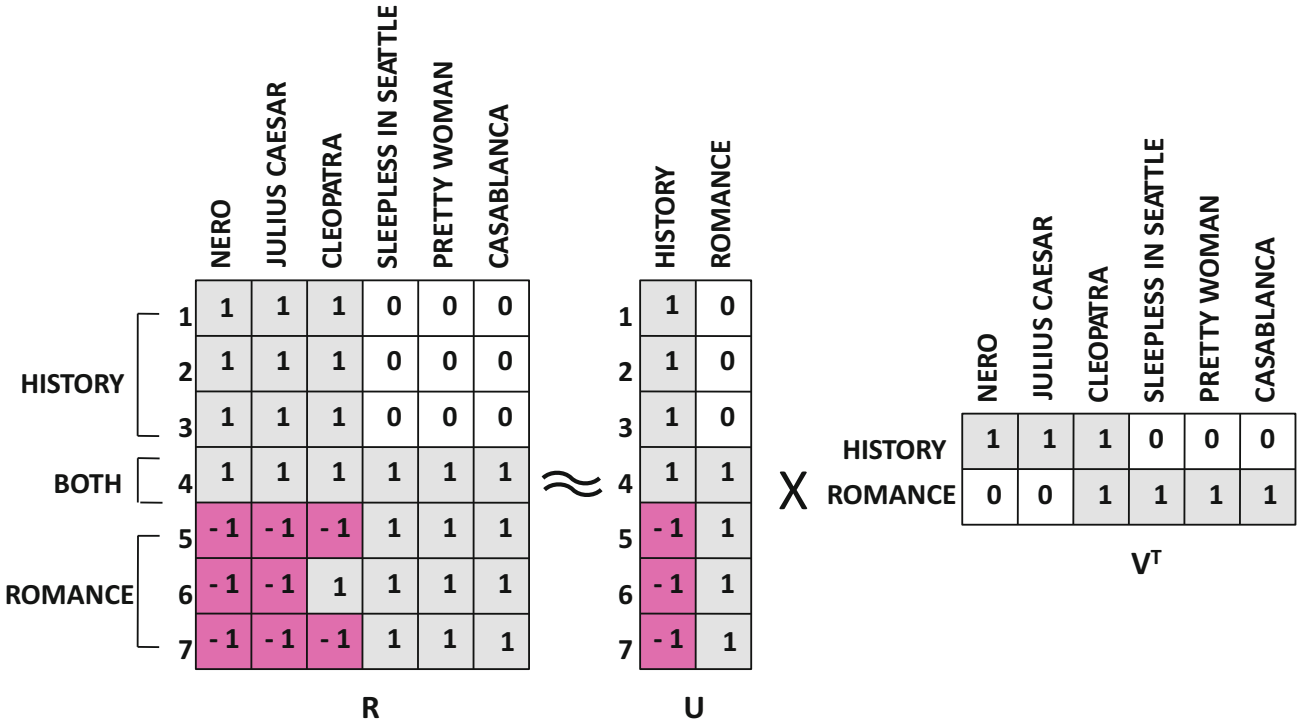
Regression vs. Collaborative Filtering

Goal: to predict the values of the missing squares 

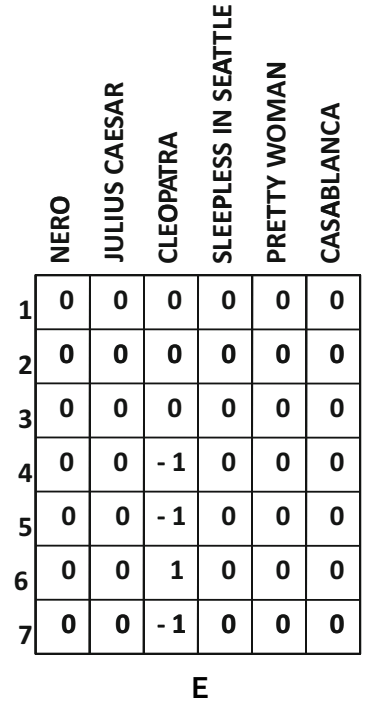


Source: https://link.springer.com/chapter/10.1007/978-3-319-29659-3_4#Bib1

Low-rank Matrix Factorization: Example



(a) Example of rank-2 matrix factorization



(b) Residual matrix

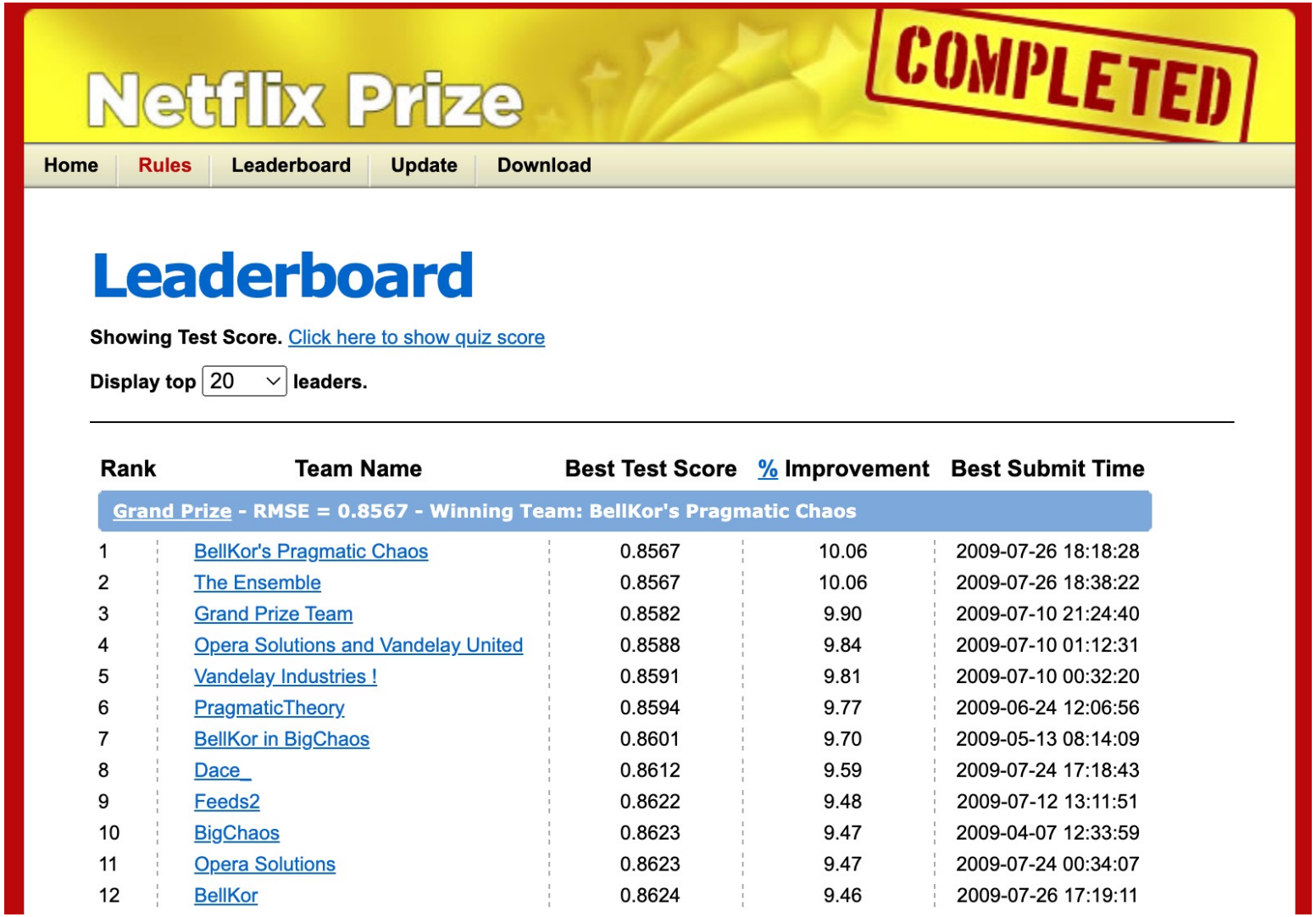
Source: https://link.springer.com/chapter/10.1007/978-3-319-29659-3_4#Bib1

Learning Objectives: 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

The Netflix Prize: Winners



Source: https://web.archive.org/web/20090926213457/http://www.netflixprize.com/leaderboard

Boosting

- An *ensemble method* combines the predictions of multiple “weak” hypotheses to learn a single, more powerful classifier
- Boosting is a *meta-algorithm*: it can be applied to a variety of machine learning models
  - Commonly used with decision trees

Ranking Classifiers (Caruana & Niculescu-Mizil, 2006)

Table 2. Normalized scores for each learning algorithm by metric (average over eleven problems)

MODEL	CAL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN	OPT-SEL
BST-DT	PLT	.843*	.779	**.939**	**.963**	**.938**	.929*	**.880**	**.896**	**.896**	**.917**
RF	PLT	.872*	.805	.934*	.957	.931	**.930**	.851	.858	.892	.898
BAG-DT	–	.846	.781	.938*	.962*	.937*	.918	.845	.872	.887*	.899
BST-DT	ISO	.826*	.860*	.929*	.952	.921	.925*	.854	.815	.885	.917*
RF	–	**.872**	.790	.934*	.957	.931	**.930**	.829	.830	.884	.890
BAG-DT	PLT	.841	.774	.938*	.962*	.937*	.918	.836	.852	.882	.895
RF	ISO	.861*	**.861**	.923	.946	.910	.925	.836	.776	.880	.895
BAG-DT	ISO	.826	.843*	.933*	.954	.921	.915	.832	.791	.877	.894
SVM	PLT	.824	.760	.895	.938	.898	.913	.831	.836	.862	.880
ANN	–	.803	.762	.910	.936	.892	.899	.811	.821	.854	.885
SVM	ISO	.813	.836*	.892	.925	.882	.911	.814	.744	.852	.882
ANN	PLT	.815	.748	.910	.936	.892	.899	.783	.785	.846	.875
ANN	ISO	.803	.836	.908	.924	.876	.891	.777	.718	.842	.884
BST-DT	–	.834*	.816	**.939**	**.963**	**.938**	.929*	.598	.605	.828	.851
KNN	PLT	.757	.707	.889	.918	.872	.872	.742	.764	.815	.837
KNN	–	.756	.728	.889	.918	.872	.872	.729	.718	.810	.830
KNN	ISO	.755	.758	.882	.907	.854	.869	.738	.706	.809	.844
BST-STMP	PLT	.724	.651	.876	.908	.853	.845	.716	.754	.791	.808
SVM	–	.817	.804	.895	.938	.899	.913	.514	.467	.781	.810
BST-STMP	ISO	.709	.744	.873	.899	.835	.840	.695	.646	.780	.810
BST-STMP	–	.741	.684	.876	.908	.853	.845	.394	.382	.710	.726
DT	ISO	.648	.654	.818	.838	.756	.778	.590	.589	.709	.774
DT	–	.647	.639	.824	.843	.762	.777	.562	.607	.708	.763
DT	PLT	.651	.618	.824	.843	.762	.777	.575	.594	.706	.761
LR	–	.636	.545	.823	.852	.743	.734	.620	.645	.700	.710
LR	ISO	.627	.567	.818	.847	.735	.742	.608	.589	.692	.703
LR	PLT	.630	.500	.823	.852	.743	.734	.593	.604	.685	.695
NB	ISO	.579	.468	.779	.820	.727	.733	.572	.555	.654	.661
NB	PLT	.576	.448	.780	.824	.738	.735	.537	.559	.650	.654
NB	–	.496	.562	.781	.825	.738	.735	.347	-.633	.481	.489

Source: https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf

Weighted Majority Algorithm (Littlestone & Warmuth, 1994)

- **Given**: a "pool" $\mathcal{A}$ of pre-trained binary classifiers (that you know nothing about) and a stream of data points (i.e., an online learning setting)
- **Goal:** design a new learner that uses the output of classifiers in the pool to make its predictions
- **Algorithm**:
  - Initially weight all classifiers equally
  - Receive a new data point 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)

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.

Algorithm 1 Weighted Majority Algorithm

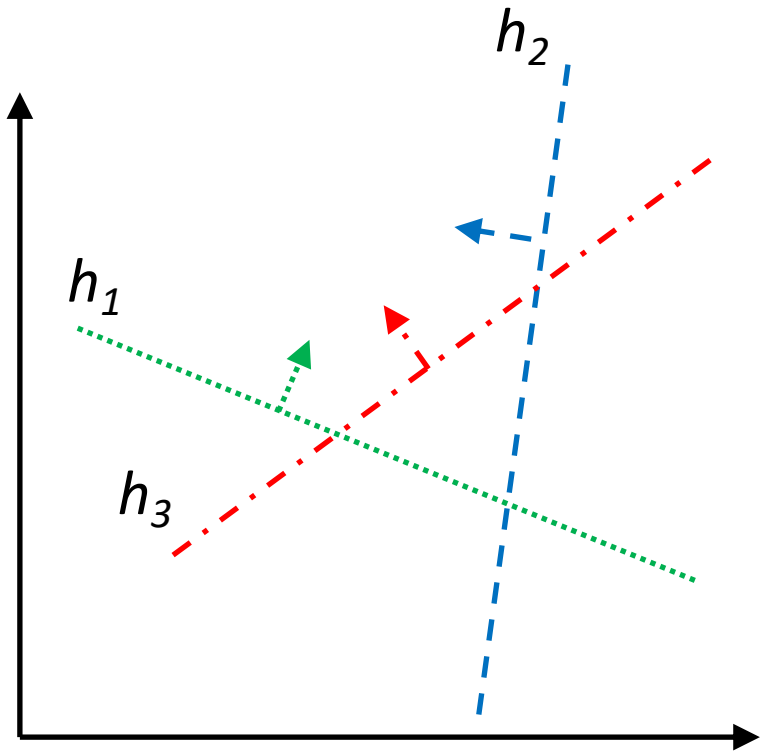
procedure WEIGHTEDMAJORITY($\mathcal{A}$, β)
  Initialize classifier weights $\alpha_t = 1, \ \forall t \in \{1, \ldots, T\}$
  for each training example $(\mathbf{x}, y)$ **do**
    Predict majority vote class (splitting ties randomly)

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

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

What does the weighted majority vote decision boundary look like for this pool of classifiers?

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



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.

Algorithm 1 Weighted Majority Algorithm

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

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

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

Weighted Majority Algorithm: Theory

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.

Source: https://www.sciencedirect.com/science/article/pii/S0890540184710091

Weighted Majority Algorithm vs. AdaBoost

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

- Intuition: iteratively reweight inputs, giving more weight to inputs that are difficult-to-predict correctly
- Analogy:
  - You all have to take a test (😱) …
  - … but you're going to be taking it one at a time.
  - After you finish, you get to tell the next person the questions you struggled with.
  - Hopefully, they can cover for you because…
  - … if "enough" of you get a question right, you'll all receive full credit for that problem