Collaborative filtering

- Recommender systems analyze user preferences to suggest products, movies, or services, significantly impacting sales for companies. Two main types:
  1. **Collaborative Filtering** – "Users like you also liked..."
  2. **Content-Based Filtering** – "This item is similar to the ones you liked..."
- The example used involves predicting movie ratings based on user interactions, showcasing how users rate different movies.

Making recommendations

Predicting movies rating

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

- $n_u = \text{no. of users} \rightarrow n_u = 4$
- $n_m = \text{no. of movies} \rightarrow n_m = 5$
- $r(i,j) = 1 \text{ if user } j \text{ has rated movie } i$

$$R = \begin{bmatrix} 1 & 0 & \dots & 1 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}_{n_m \times n_u}$$

  - e.g. $r(1,1) = 1; r(3,1) = 0$
- $y^{(i,j)} = \text{rating given by user } j \text{ to the movie } i \ \ (\text{defined only if } r(i,j) = 1)$

$$Y = \begin{bmatrix} 7 & 1 & \dots & 2.5 \\ 5.6 & ? & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ ? & 9.2 & \dots & 1 \end{bmatrix}_{n_m \times n_u}$$

  - e.g. $y^{(3,2)} = 4$

The recommender system aims to predict how users would rate movies they haven't watched. For example, if Alice hasn't rated "Cute Puppies of Love," the system will analyze the ratings of similar users (like Bob or Carol) to estimate a likely rating for her.

Using per-item features

What if we have features of the movies?

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

- $n = 2 : \text{no. of features}$
- x_1 is the amount of romance and x_2 is the amount of action, ...
- $x^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}; x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$

→ **For user 1:** predict rating for movie i as : $w^{(1)} \cdot x^{(i)} + b^{(1)}$ (just **Linear Regession**).

Eg. $w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$; $b^{(1)} = 0$; $x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$ $\quad \Rightarrow \quad$ $w^{(1)} \cdot x(3) + b^{(1)} = 4.95$

→ **For user j:** predict rating for movie i as : $w^{(j)} \cdot x^{(i)} + b^{(j)}$.

$\Rightarrow$ This is look like **Linear Regression model** for each of the 4 users in the dataset.

Cost function

How we can formulate the cost function?

- $r(i,j)$: 1 if user j rated movie i, 0 otherwise.
- $y^{(i,j)}$: Actual rating user j gave movie i (if defined)
- $w^{(j)}, b^{(j)}$: Parameters (weights, bias) for user j
- $x^{(i)}$: Feature vector for movie i
- $m^{(j)}$: Number of movies rated by user j

Goal: Learn $w(j), b(j)$ so the predictions are close to the actual ratings given.

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2m^{(j)}} \sum_{i\,:\,r(i,j)=1}^{m} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

Remember $n : \text{number of features}$. Then remove $m^{(j)}$, the regularize cost function for user j:

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum_{i\,:\,r(i,j)=1}^{m} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

To learn for all parameters $w^{(j)}, b^{(j)}, \ldots, w^{(n_u)}, b^{(n_u)}$ for all users:

$$J\begin{pmatrix} w^{(1)}, \ldots, w^{(n_u)} \\ b^{(1)}, \ldots, b^{(n_u)} \end{pmatrix} = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(\underbrace{w^{(j)} \cdot x^{(i)} + b^{(j)}}_{f(x)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

Collaborative filtering algorithm

💡 **Context:** When building a recommendation system (like predicting movie ratings), you often use features to help predict how a user might rate an item. For example, you might describe movies by how "romantic" or "action-packed" they are, with numerical values like x_1 (romance) and x_2 (action). If these features are given, you can do linear regression to predict user ratings.

The Problem: What if you ***don't*** know those features for movies upfront? You only have the users' ratings for various movies, but you don't have values for x_1 or x_2.

The Idea Shown

- We have ratings from multiple users (Alice, Bob, Carol, Dave) for several movies.
- Each user has their own preferences, represented by parameters $w^{(j)}$ (vectors), which are *user-specific* weights indicating how much the user cares about romance vs. action.

  For example:

  - Alice's $w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ (she only cares about romance).
  - Carol's $w^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$ (she only cares about action).
- We want to **learn the movie features** $x^{(i)}$ so that when we take the dot product $w^{(j)} \cdot x^{(i)}$, we get the users' actual ratings for movie i.

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

Normal linear regression isn't suitable for learning item features

- **Normal Linear Regression** typically deals with predicting **outputs** from **input features** for a **single** user or dataset.

  You have a set of features given and want to find the best parameters (weights) to predict an output. But it does **not** try to learn features themselves from the data.
- **Collaborative Filtering**, on the other hand, uses **rating** data from **many users** over **many items** (e.g., movies).

  Because multiple users rate the same items, there is enough data diversity to simultaneously learn:

  - The user preferences (weights w)
  - The hidden item features x

    Both are **unknown** and learned from the data.
- If you only had **one user** (which is the case in normal linear regression), you have very limited data—only the ratings from that one user.

  This provides **insufficient information** to figure out the features of items. You essentially have one equation and multiple unknowns per item, so the problem is not **solvable**.

- Collaborative filtering **leverages the fact that multiple users have rated the same items**.

  This creates a rich matrix of ratings, allowing the algorithm to “reverse engineer” both what features the items have and how much each user cares about each feature.

Example for 1 user:

Movie	User 1 Rating
Movie A	5
Movie B	3
Movie C	? (unknown)

You want to figure out features of these movies, for example, how "romantic" or "action" each movie is, to predict the unknown rating for **Movie C**.

- With **only one user**, you have 3 unknowns (romance and action features for 3 movies = 6 unknowns), but only 2 known ratings.
- The problem becomes **underdetermined**—there are infinitely many ways to assign features and weights to match the two ratings.
- So, with one user, you **don’t have enough data** to learn features for movies.

Multiple users:

Movie	User 1	User 2	User 3
Movie A	5	4	0
Movie B	3	2	5
Movie C	?	3	4

- Now, from these many ratings, we have much more information (constraints) about each movie.
- We can try to find movie features (e.g., romance, action) and user preferences to explain all these ratings together.
- The system can use the fact that User 1, 2, and 3 have rated the same movies to **reverse engineer** the movie features x and user preferences w.
- This way, the unknown rating for Movie C by User 1 can be predicted more accurately.

So basically, collaborative filtering is like solving a big system of equations involving all users and items together, while normal linear regression solves a simpler problem with insufficient data to learn new features.

How to Guess the Movie Features $x^{(i)}$

- Take the ratings for a specific movie (like "**Love at last**").

  Example: Alice and Bob rate it 5 (they love romance), Carol and Dave rate it 0 (they care about action, and this movie is not action).

- Since Alice and Bob’s w emphasize **romance**, and Carol and Dave’s w emphasize **action**, a reasonable guess is:

  $x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. $\Rightarrow$ This means "***Love at last***" is a purely ***romance*** movie.

→ Similarly, for "Nonstop car chases," a guess might be $x^{(4)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, indicating it is an action movie.

Cost function for learning feature $x^{(i)}$

Given $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots, w^{(n_u)}, b^{(n_u)}$. To learn one single feature $x^{(i)}$:

$$J(x^{(i)}) = \frac{1}{2} \sum_{j\,:\,r(i,j)=1}^{n_m} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2$$

- Minimize this cost function for **all movies** and **users** simultaneously.
- Use regularization to avoid overfitting.
- Use gradient descent to update all w, b and x values iteratively.
- This way, we **learn both user preferences (weights w, b) and movie features x from just rating data**, without needing pre-specified features.

To learn one single feature $x^{(i)}$:

$J(x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2$

Putting all together, the cost function for w, b, x

$$J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{()} \right)^2$$

Gradient Descent

$$\text{repeat} \left[\begin{array}{l} w_i^{(j)} = w_i^{(j)} - \alpha \dfrac{\partial}{\partial w_i^{(j)}} J(w, b, x) \\ b^{(j)} = b^{(j)} - \alpha \dfrac{\partial}{\partial b^{(j)}} J(w, b, x) \\ x_k^{(i)} = x_k^{(i)} - \alpha \dfrac{\partial}{\partial x_k^{(i)}} J(w, b, x) \end{array} \right]$$

💡 **Summarize:** Uses only the rating patterns of users and items (no extra features required).

- The algorithm:
  1. Learn **user preferences** as a vector (what kind of things they like).
  2. Learn **item features** as a vector (what kind of qualities the item has).
  3. Prediction = dot product of user vector × item vector.
- Optimization: Use **gradient descent** to minimize cost.

Binary labels: favs, likes and clicks

Instead of explicit ratings (e.g., 1 to 5 stars), many recommender systems work with **binary labels** indicating whether a user liked or engaged with an item (e.g., watched a movie fully, clicked a product, or liked a post).

Example data for binary labels:

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	1	1	0	0
Romance forever	1	?	?	0
Cute puppies of love	?	1	0	?
Nonstop car chases	0	0	1	1
Swords vs. karate	0	0	1	?

Example applications

1. Did user j purchase an item after being shown?
2. Did user j fav/like an item?
3. Did user j spend at least 30sec with an item?
4. Did user j click on an

- 1 means the user **engaged with or liked** the item.
- 0 means the user **did not engage** (e.g., stopped early, did not click).
- Question mark $(?)$ means the user has **not yet interacted** with or been exposed to the item.

From regression to binary classification

Previously:

- Predict $y^{(i,j)}$ as $w^{(j)} \cdot x^{(i)} + b^{(j)}$

For binary labels:

- Predict that the probability of $y^{(i,j)} = 1$ is given by $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$ with $g(z) = \frac{1}{1+e^{-z}}$

Cost function for binary application

Previous cost function:

$$\frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(\underbrace{w^{(j)} \cdot x^{(i)} + b^{(j)}}_{f(x)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (w_k^{(j)})^2$$

Loss for binary labels $y^{(i,j)}$ **:** $f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

Single example:

For all examples:

You have the following table of movie ratings:

Movie	Elissa	Zach	Barry	Terry
Football Forever	5	4	3	?
Pies, Pies, Pies	1	?	5	4
Linear Algebra Live	4	5	?	1

Refer to the table above for question 1 and 2. Assume numbering starts at 1 for this quiz, so the rating for Football Forever by Elissa is at $(1, 1)$

What is the value of n_u?

Answer: 4

What is the value of $r(2, 2)$:

Answer: 0

In which of the following situations will a collaborative filtering system be the most appropriate learning algorithm (compared to linear or logistic regression)?

- You run an online bookstore and collect the ratings of many users. You want to use this to identify what books are "similar" to each other (i.e., if a user likes a certain book, what are other books that they might also like?) 👍👍
- You're an artist and hand-paint portraits for your clients. Each client gets a different portrait (of themselves) and gives you 1-5 star rating feedback, and each client purchases at most 1 portrait. You'd like to predict what rating your next customer will give you.
- You subscribe to an online video streaming service, and are not satisfied with their movie suggestions. You download all your viewing for the last 10 years and rate each item. You assign each item a genre. Using your ratings and genre assignment, you learn to predict how you will rate new movies based on the genre.
- You manage an online bookstore and you have the book ratings from many users. You want to learn to predict the expected sales volume (number of books sold) as a function of the average rating of a book.

Explain: You can find "similar" books by learning feature values using collaborative filtering.

For recommender systems with binary labels y, which of these are reasonable ways for defining when y should be 1 for a given user j and item i? (Check all that apply.)

- [x] ~~y is 1 if user j purchases item iii (after being shown the item)~~
- [ ] y is 1 if user j has been shown item i by the recommendation engine
- [x] ~~y is 1 if user j fav/likes/clicks on item i (after being shown the item)~~
- [ ] y is 1 if user j has not yet been shown item i by the recommendation engine