⇒ Recommendor systems

→ Recommendation is currently very popular application of ML. eg: Netflix, Amazon, Spotify, iTunes Genius

→ Really popular in industry, not so much in academia (:o).

→ There is an interesting idea that for some specific ~~algorithms~~ problems, there are certain algorithms that can automatically "learn" a good set of features for you!!

→ So, instead of trying to manually pick some features, there might be some settings where you might be able to have an algorithm, just to learn what feature to use, & the recommender system is just one example of that sort of setting.

→ Example motivation : Ranking movies

* Suppose we're trying to recommend movies to customers.
* We have a dataset as follows :

♫ Rating scale → [0 ; 5]

♫ ? → No rating found

	User 1	User 2	User 3
Movie 1	0	1	?
Movie 2	?	5	5

* Lets introduce a bit of notation :

n_u = no. of users

n_m = no. of movies

$r(i,j) = 1$ if user j has rated movie i

$y^{(i,j)}$ = Rating given by user j to movie i
(defined only if $r(i,j)=1$)

* For above example : $n_u = 3$; $n_m = 2$; $r(2,1) = 0$; $y^{(2,1)} = 1$
* Our objective is that given $r(i,j)$ & $y^{(i,j)}$, fill try to come up with a learning algo that fit/predict the missing values (?)

⇒ Content based recommendations

→ Using a similar example, how do we predict the missing entries?

→ One approach is to introduce 2 features x_1 & x_2 which represent how much romance or how much action a movie may have (on a scale of 0-1).

eg:

Movie	Alice(1) θ^1	Bob(2) θ^2	Carol(3) θ^3	x_1 (Romance)	x_2 (Action)
Fault in our Stars $x^{(1)}$	5	5	?	0.99	0
DDLJ $x^{(2)}$	5	?	1	1.0	0.01
The Notebook $x^{(3)}$	?	4	0	1.0	0
Die Hard $x^{(4)}$	0	0	5	0	1
Star Wars $x^{(5)}$	0	0	5	0.1	1

→ Formally, for convenience sake we also add a $x_0 = 1$ for every movie. ∴ for DDLJ $\equiv x^2 = \begin{bmatrix} x_0^2 \\ x_1^2 \\ x_2^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.0 \\ 0.01 \end{bmatrix}$

→ For filling the missing values, one approach is to do linear regression for every single user:

* For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^{n+1}$;

$n \rightarrow$ no. of categories for movies. For our previous example, $\theta^{(j)} \in \mathbb{R}^{2+1} \therefore \theta^{(j)} \in \mathbb{R}^3$

* Predict user j's rating for i^{th} movie as

$$\boxed{(\theta^{(j)})^T x^{(i)} \text{ stars.}}$$

eg: Suppl. in our prev. example.

For "The Notebook" → $x^3 = \begin{bmatrix} 1 \\ 1.0 \\ 0 \end{bmatrix}$

For Alice, we say we've learned $\theta^1 = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$

∴ Alice will rate The Notebook : $(\theta^1)^T x^3 = 1 \times 5 = 5$

⇒ Problem formulation

→ Some conventional notation :

* $r(i,j) = \begin{cases} 1 & \text{if user } j \text{ has rated movie } i \\ 0 & \text{otherwise} \end{cases}$

* $y^{(i,j)}$ = rating by user j on movie i (if defined)

* θ^j = Parameter vector for user j

* x^i = Feature vector for movie i

* For user j's movie i, predicted rating $(\theta^j)^T x^i$

* m^j = No. of movies rated by user j

⇒ Objective : Learn $\theta^{(j)}$

⇒ & To "learn" $\theta^{(j)}$ find values which minimize the following cost function $J(\theta^{(j)})$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right]^2$$

* The cost function is the same as the one used in linear regression : Root mean square error

* $\sum_{i:r(i,j)=1}$ ( ) → means sum over all values of i (all the available movies) when $r(i,j) = 1$ (all the films users has rated)

→ We can also add a regularization term to make our eqn look as follows:

$$\min_{\theta^j} \left(\frac{1}{2m^j} \sum_{i:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right]^2 + \frac{\lambda}{2m^j} \sum_{k=1}^{n} (\theta_k^j)^2 \right)$$

* The regularization term goes from $k=1$ through n; so θ^j ends up being an $(n+1)$ feature vector. Don't regularize over the bias terms (0).

→ To make this a little cleaner, get rid of the m^j term.

* $m^j \rightarrow$ constant; so won't affect our minimization routine

$$\min_{\theta^j} \left[\frac{1}{2} \sum_{i:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right]^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^j)^2 \right]$$

$J(\theta^j)$

→ To get the parameters for all our users, we do the following:

$$\min_{\theta^{(1)},\dots,\theta^{(n_u)}} \left(\frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right]^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \right)$$

$$J(\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)})$$

* In our recommendation system, we want to learn parameters for all the users. So, we add an extra summation term to this which means we determine the $\theta^{(j)}$ value for every user s.t. it minimizes the overall optimization Objective / cost function.

→ We can also write it as: $\min_{\theta^{(1)},\dots,\theta^{(n_u)}} J(\theta^{(1)},\dots,\theta^{(n_u)})$

⇒ Gradient descent updates

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \frac{\partial}{\partial \theta_k^{(j)}} J(\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)})$$

$$\rightarrow \frac{\partial}{\partial \theta_k^{(j)}} = \left(\sum_{i : r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right] x_k^{(i)} \right) + \lambda \theta_k^{(j)}$$

∴ The updates become :

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i : r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right] x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i : r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right] x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

→ The only diff. from the linear regression updates is that we've eliminated the $1/m$ term.

→ This approach is called content-based approach as we assume that we have features regarding the content (eg. how romantic or action heavy some movie is) which helps us identify things that make them appealing to a user.

However such features are usually not available & are really hard to find. So, this approach is not that practical!

⇒ Collaborative filtering

→ It can be very difficult to find features such as "amount of romance" or "amount of action" in a movie.

The collaborative filtering algo has a very interesting property — feature learning

i.e. it can learn for itself what features to use it needs to learn!

→ Now, let's make a diff. assumption: ~~We~~

Say, we before training we ask each user & find out how much each user likes romantic & action films & capture that into 2 vectors.

→ eg: Suppose we have a dataset:

Movie	Alice (1)	Bob (2)	Carol (3)	x_1 (Romance)	x_2 (Action)
Fault in Our Stars	5	5	0	?	?
DDLJ	5	?	?	?	?
The Notebook	?	4	0	?	?
Die Hard	0	0	5	?	?
Star Wars	0	0	5	?	?

* We've marked x_1 & x_2 as ? because unlike content based recommendations, we don't have any knowledge about the "contents" of a film.

* Suppose we poll each user & find out how much each user likes romantic & action films. This results in the following parameter set:

$$\theta^1 = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad \theta^2 = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \quad \theta^3 = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

* Each column of these θ vectors can represent anything, but these values will be DIFFERENT for each user acc. to their likings & choices. For convenience sake, assume $\theta_2 \rightarrow$ liking for romantic film & $\theta_3 \rightarrow$ liking for film

→ If we can get these parameters from the users we can infer the missing values from our table!!

eg: For the movie "Fault in our stars",

We know from the θ vectors that Alice & Bob love romantic films & Carol hates them.

(by using the θ values)

Based on the factors that

Alice & Bob loved the movie ~~[redacted]~~ (Fault in our stars)

& Carol hated it, so we can conclude that

"Fault in our stars" is a romantic film.

→ This is a bit simplified by what we're really asking is: "What feature vector should x^1 be so that"

* $(\theta^1)^T x^1 \approx 5$
* $(\theta^2)^T x^{(1)} \approx 5$
* $(\theta^3)^T x^1 \approx 0$

From this, we can guess x^1 may be $\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ → Romantic

⇒ Formalizing the collaborative filtering algorithm

⇒ Optimization algorithm

Given $\theta^1, \ldots, \theta^{n_u}$ [i.e. given the parameter vectors for each users' preferences]
to learn x^i :

$$\min_{x^i} \frac{1}{2} \sum_{j:r(i,j)=1} [(\theta^j)^T x^i - y^{(i,j)}]^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^i)^2$$

* Basically means, that out of all the ~~ratings~~ polls given by users $(\theta^1, \ldots, \theta^{n_u})$, select the feature vector x^i which minimizes the mean square error.

→ Since, we want to learn all the features for all the films, so add all the square errors to make a combined optimization objective:

$$\min_{x^1, x^2, \ldots, x^{m_m}} \frac{1}{2} \sum_{i=1}^{m_m} \sum_{j:r(i,j)=1} [(\theta^j)^T x^i - y^{(i,j)}]^2 + \frac{\lambda}{2} \sum_{i=1}^{m_m} \sum_{k=1}^{n} (x_k^i)^2$$

⇒ How does collaborative filtering fare with content based recommendation systems?

→ Collaborative filtering : Given $\theta^1, \ldots, \theta^{n_u}$ (movie ratings)
estimate $x^1, \ldots, x^{n_m}$

→ Content based recommendation : Given $x^1, \ldots, x^{n_m}$ (movie ratings)
estimate $\theta^1, \ldots, \theta^{n_u}$

→ Which one to choose first is kind of like the chicken-egg problem.

→ Ideal way :

* Randomly initialize θ
* Generate x with collaborative fittering
* Then use content based recommendation to improve θ
* Then use that ($\uparrow$) to improve x
* ..... so on

This works really well practically!

* The algo is termed collaborative filtering because in this all the users come together & collaborate by rating the movies & filling the poll, they're effectively helping the algorithm in churning out better recommendation for everybody!

⇒ Collaborative filtering Algorithm

→ We discussed earlier about the ideal way of doing collaborative filtering with content based model where we go back & forth between the 2 models. Calculate $\theta \rightarrow x \rightarrow \theta \rightarrow x \ldots$

→ There's a more efficient algorithm which can solve for θ & x simultaneously:

⇒ Optimization Objective

→ Minimizing $x^1, \ldots, x^{n_m}$ & $\theta^1, \ldots, \theta^{n_u}$ simultaneously:

$$J(x^1, \ldots, x^{n_m}, \theta^1, \ldots, \theta^{n_u}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right]^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^j)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^i)^2$$

→ Objective:

$$\min_{x^1, \ldots, x^{n_m}, \theta^1, \ldots, \theta^{n_u}} J(x^1, \ldots, x^{n_m}, \theta^1, \ldots, \theta^{n_u})$$

→ The term $\sum_{(i,j):r(i,j)=1} [(\theta^{(j)})^T x^{(i)} - y^{(i,j)}]^2$:

* Sums over all pairs (i,j) for $r(i,j)=1$
* This has a really interesting property
  - Keep x constant : Becomes similar to

$$\left.\sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} [(\theta^{(j)})^T x^{(i)} - y^{(i,j)}]^2\right\} \text{Given } x, \text{ learn } \theta$$

  - Keep θ constant : Becomes similar to

$$\left.\sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} [(\theta^{(j)})^T x^{(i)} - y^{(i,j)}]^2\right\} \text{Given } \theta, \text{ learn } x$$

* In order to come up with just one optimization function, we treat J as a func^n of both x & θ.
* We've dropped the $x_0 = 1$ term. [?] $\therefore x, \theta \in \mathbb{R}^n$

As we are now learning all the features so if the system needs a feature always $=1$, it can learn that by itself.

⇒ Full Collaborative Filtering Algorithm

1) Initialize $x^1, \ldots, x^{n_m}, \theta^1, \ldots, \theta^{n_u}$ to small random values.

* This serves to break symmetry & ensures that the algorithm learns features that are different from each other.

2) Minimize $J(x^1, \ldots, x^{n_m}, \theta^1, \ldots, \theta^{n_u})$ using gradient descent (or an advanced optimization algo).

$$x_k^i := x_k^i - \alpha \Bigg(\underbrace{\sum_{j:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right] \theta_k^j}_{\frac{\partial}{\partial x_k^i}} + \lambda x_k^i \Bigg)$$

$$\theta_k^j := \theta_k^j - \alpha \Bigg(\underbrace{\sum_{i:r(i,j)=1} \left[(\theta^j)^T x^i - y^{(i,j)} \right] x_k^i + \lambda \theta_k^j}_{\frac{\partial}{\partial \theta_k^j}} \Bigg)$$

3) For a user with parameters θ & a movie with (learned) features x, predict the rating by $\theta^T x$.

⇒ Vectorization : Low Rank matrix factorization

→ Suppr. we have a dataset:

Movie	User1	User2	User3	User4
M1	5	5	0	0
M2	5	?	?	0
M3	?	4	0	?
M4	0	0	5	4
M5	0	0	5	?

→ We can also vectorize the above dataset in Y, s.t.

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & ? \end{bmatrix} \equiv y^{(i,j)}$$ for each user j & movie i

→ Vectorizing the predicted rating:

$$\begin{bmatrix} (\theta^1)^T x^1 & (\theta^2)^T x^1 & \cdots & (\theta^{n_u})^T x^1 \\ (\theta^1)^T x^2 & (\theta^2)^T x^2 & \cdots & (\theta^{n_u})^T x^2 \\ \vdots & \vdots & \ddots & \vdots \\ (\theta^1)^T x^{n_m} & (\theta^2)^T x^{n_m} & \cdots & (\theta^{n_u})^T x^{n_m} \end{bmatrix}$$

→ We can define 2 main matrices:

$$X = \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^{n_m})^T \end{bmatrix}$$ → Each row containing features of a particular movie

& Θ = $\begin{bmatrix} (\theta^1)^T \\ (\theta^2)^T \\ \vdots \\ (\theta^{n_u})^T \end{bmatrix}$ → Each row containing weights for the features of a given user

* Given X & Θ, we can get the full matrix Y' of all predicted ratings of all movies of all users by: $Y' = X\Theta^T$

* This algorithm is called low rank matrix factorization.

This comes from the property that $X \cdot \Theta^T$ creates a low rank matrix

* Now we can apply gradient descent to:

$$\min (Y - Y') = \min (Y - X\Theta^T)$$

⇒ Finding related movies

→ For each product i, we learn a feature vector $x^i \in \mathbb{R}^n$. eg: x_1 = romance, x_2 = action, x_3 = Comedy, ...

* After learning, it might difficult to characterize feature into human understandable form.

→ How to find movies j related to movie i?

If we have 2 matrices x^i & x^j; we want to minimize $\|x^i - x^j\|$ → dist. b/w those 2 movies

eg: 5 most similar movies to movie i

Find the 5 movies j with smallest $\|x^i - x^j\|$

⇒ Implementation detail: Mean normalization

→ Say we have a dataset where a user hasn't rated any movie (U_5):

Movie	U_1	U_2	U_3	U_4	U_5
M_1	5	5	0	0	?
M_2	5	?	?	0	?
M_3	?	4	0	?	?
M_4	0	0	5	4	?
M_5	0	0	5	?	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & ? & ? \end{bmatrix}$$

→ Say $\theta \in \mathbb{R}^2$ as $n = 2$

Want to learn θ^5

→ Let's analyze the optimization objective

$$\min_{x^1,\dots,x^{n_m},\ \theta^1,\dots,\theta^{n_u}} \left\{ \underbrace{\frac{1}{2} \sum_{(i,j):r(i,j)=1} \left[(\theta^{(j)})^T x^{(i)} - y^{(i,j)}\right]^2}_{\text{Not relevant as these are not filled for } 5^{th} \text{ user whose } r(i,5)=1} + \underbrace{\frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2}_{\text{Not relevant while minimizing } \theta^5} + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \right\}$$

* So, we're only minimizing $\boxed{\frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2}$,

which for our example can be simplified to $\frac{\lambda}{2} \left[(\theta_1^{(5)})^2 + (\theta_2^{(5)})^2\right]$

* Of course, if the goal is to minimize this term then $\boxed{\theta^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}}$. As there's no data, the algo will minimize it to the minimum possible value.

* As, this user will have indirectly rated all movies to 0, so we won't be able to recommend anything to this user.

→ This is where mean normalization comes in!

⇒ Mean normalization

→ We can normalize the data relative to the mean.

→ Firstly, we'll use a Y matrix to store the data from previous ratings:

* i^{th} row : Rating for i^{th} movie

* j^{th} column : Ratings given by j^{th} user

→ We now a n_m dimensional column vector u:

$$u = \begin{bmatrix} u_1 \\ \vdots \\ u_{n_m} \end{bmatrix}$$

where

$$\mu_i = \frac{\sum_{j:r(i,j)=1} Y_{i,j}}{\sum_j r(i,j)}$$

$= \frac{\text{Sum of ratings for } i^{th} \text{ movie whichever user 'j' has rated it}}{\text{No. of user 'j' who rated the movie i}}$

* μ_i = Mean of the previous ratings for the i^{th} movie.

→ We can normalize the data by subtracting μ, the mean rating, from the actual ratings for each user

→ eg: for our previous example

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} ; \quad \mu = \begin{bmatrix} (5+5+0+0)/4 \\ (5+0)/2 \\ (4+0)/2 \\ (5+4+0+0)/4 \\ (0+0+5+0)/4 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

$$Y := Y - \mu = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

→ By subtracting the mean of a movie from its ratings, we've essentially normalized each film to have an average rating of 0!

* Now, we can use this new Y matrix to learn θ^j & x^i.

* Slight modification to linear regression prediction

Include the mean normalization term:

$$(\theta^j)^T x^i + \mu_i$$

→ Now, for a new user the initial predicted value will be equal to the μ term instead of simply being initialized to 0, which makes for more sense intuitively. (As we're predicting the average rating, which is the best we can do)

eg: For Eve, $\theta^{Eve} = 0$

→ Movies with no ratings: Can apply mean-normalization to rows!

* But not relevant, as you shouldn't recommend an unrated movie.