



# *Recommendation System Using Matrix Factorization*

# Content

- Recommender System
- Matrix Factorization
- Gradient Approaches
- Result
- Conclusion
- References

# Goal -



To recommend the movies that user might want to watch.

NETFLIX

Browse ▾

DVD

Search



Joshua ▾

## Top Picks for Joshua



## Trending Now



## Because you watched Narcos



## New Releases



# What are recommender systems?

- Recommender systems aim to predict users' interests and recommend product items that quite likely are interesting for them. They are among the most powerful machine learning systems that online retailers implement in order to drive sales.
- Data required for recommender systems stems from explicit user ratings after watching a movie or listening to a song, from implicit search engine queries and purchase histories, or from other knowledge about the users/items themselves.
- Examples of Recommendation systems are Netflix or YouTube that suggest playlists or make video recommendations

# Why do we need recommender systems?

- Companies using recommender systems focus on increasing sales as a result of very personalized offers and an enhanced customer experience.
- Recommendations typically speed up searches and make it easier for users to access content they're interested in, and surprise them with offers they would have never searched for.
- The user starts to feel known and understood and is more likely to buy additional products or consume more content. By knowing what a user wants, the company gains competitive advantage and the threat of losing a customer to a competitor decreases.

## Types of recommender systems:

- ***Content-based*** systems, which use characteristic information.
- ***Collaborative filtering*** systems, which are based on user-item interactions.
- ***Hybrid systems***, which combine both types of information with the aim of avoiding problems that are generated when working with just one kind.

# Dataset

Movie lens Dataset consists of :

- 100000 ratings (1 lakh)
- 600 users

A	B	C	D
userId	movieId	rating	timestamp
1	1	4	964982703
1	3	4	964981247
1	6	4	964982224
1	47	5	964983815
1	50	5	964982931
1	70	3	964982400
1	101	5	964980868
1	110	4	964982176
1	151	5	964984041
1	157	5	964984100
1	163	5	964983650
1	216	5	964981208
1	223	3	964980985
1	231	5	964981179
1	235	4	964980908
1	260	5	964981680
1	296	3	964982967
1	316	3	964982310
1	333	5	964981179
1	349	4	964982563
1	356	4	964980962
1	362	5	964982588
1	367	4	964981710
1	423	3	964982363
1	441	4	964980868
1	457	5	964981909
1	480	4	964982346
1	500	3	964981208



# Matrix factorization



- Matrix factorization comes in limelight after Netflix competition (2006) when Netflix announced a prize money of \$1 million to those who will improve its root mean square performance by 10%.















# Matrix :

The diagram illustrates the matrix factorization equation  $R \approx U V^T$ . Matrix  $R$  is a square matrix with 'Users' on the vertical axis and 'Movies' on the horizontal axis. It is labeled 'Sparse' and contains a pattern of small gray squares representing non-zero entries. Matrix  $U$  is a vertical rectangle with 'Users' on the vertical axis and dimension  $d$  on the horizontal axis. Matrix  $V^T$  is a horizontal rectangle with dimension  $d$  on the vertical axis and 'Movies' on the horizontal axis. The matrices are connected by an approximation symbol  $\approx$  and a multiplication symbol  $\times$ .

- $R$  (users,movies)
- $U$  (users,d)
- $V^T$  (d,movies)
- here  $d$  is number of latent features

# Matrix Factorization

	M1	M2	M3	M4	M5
 Comedy	3	1	1	3	1
 Action	1	2	4	1	3

	 Comedy	 Action
 A		
 B		
 C		
 D		

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

## Matrix factorization (contd):

Approximate user's  $u$  rating of item  $i$  :





$$r_{ui} = p_u q_i$$





Here  $r$  is the rating defined for user  $u$  and item  $i$ ,  $p$  is row of  $U$  for user and  $q$  is the column of transpose( $V$ ) for a specific item  $i$ .

# Cost Function :

	M1	M2	M3	M4	M5
F1	1.2	3.1	0.3	2.5	0.2
F2	2.4	1.5	4.4	0.4	1.1

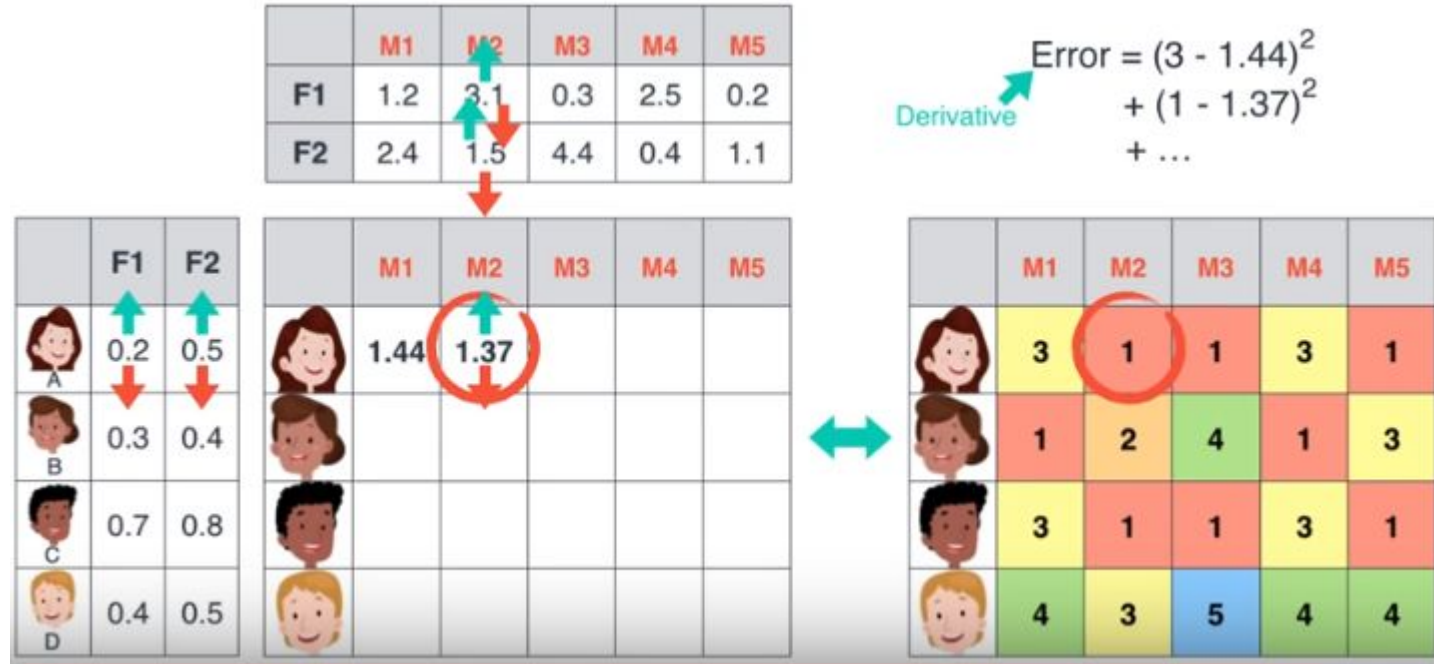
$$1.2 \times 0.2 + 2.4 \times 0.5 = 1.44$$

	F1	F2
	0.2	0.5
	0.3	0.4
	0.7	0.8
	0.4	0.5

	M1	M2	M3	M4	M5
	1.44	1.37	2.26	0.7	0.59
	1.32	1.53	1.85	0.91	0.5
	2.76	3.37	3.73	2.07	1.02
	1.68	1.99	2.32	1.2	0.63

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

# Cost Function (contd) :



## Cost Function (contd) :

$$\text{MSE} = (Y - \tilde{Y})^2 = (r_{ui} - \sum_a^b p_u q_i)^2$$

As our main task is to minimize the error or cost function i.e. we have to find in which direction we have to go for that we have to differentiate it, hence new equation will become

$$\frac{\partial \text{MSE}}{\partial q_i} = -2(Y - \tilde{Y}) * p_u$$

$$\frac{\partial \text{MSE}}{\partial p_u} = -2(Y - \tilde{Y}) * q_i$$

## Cost Function (contd) :

Hence the updated value for p and u will be now become:

$$\tilde{p}_u = p_u + \alpha \frac{\partial MSE}{\partial p_u}$$

$$\tilde{q}_i = q_i + \alpha \frac{\partial MSE}{\partial q_i}$$

Where  $\alpha$  is learning rate generally its value is very small as higher  $\alpha$  can overshoot the minimum value



Simple model will be a very poor generalization of data.

Complex model may not perform well in test data due to over fitting.

There is need to choose the right model in between simple and complex model.

Regularization helps to choose preferred model complexity, so that model is better at predicting.

## What is Regularization?

Regularization is a technique to discourage the complexity of the model. It does this by penalizing the loss function. This helps to solve the overfitting problem.

Let's understand how penalizing the loss function helps simplify the model

$$L(x, y) = \sum_{i=1}^n (y_i - f(x_i))^2$$
$$f(x_i) = h_{\theta}x = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

-As the degree of the input features increases the model becomes complex and tries to fit all the data points.

## Assumption of Regularization-

Regularization works on assumption that smaller weights generate simpler model and thus helps avoid overfitting.

$$f(x_i) = h_{\theta}x = \theta_0 + \theta_1x_1 + \theta_2x_2^2 + \theta_3x_3^3 + \theta_4x_4^4$$

$$f(x_i) = h_{\theta}x = \theta_0 + \theta_1x_1 + \theta_2x_2^2$$

## **What if the input variables have an impact on the output?**

-Penalize all the weights by making them small.Regularization term keeps the weights small making the model simpler and avoiding overfitting.

$$L(x, y) = \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2$$

$$\text{where } h_{\theta}x_i = \theta_0 + \theta_1x_1 + \theta_2x_2^2 + \theta_3x_3^3 + \theta_4x_4^4$$

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$

$\lambda$  = penalty term or regularization parameter which determines how much to penalizes the weights.

## **Regularization approach which can be used to remove the overfitting.**

L1 regularization add linear magnitude of coefficient as penalty term while in L2 it add square magnitude of coefficient to the loss function/error function (as discussed above). L1 return sparse matrices while L2 does not.

In recommendation dataset also, there are high chances of overfitting of data. So, to remove the overfitting problem we can add L2 regularization (as matrix is already sparse we do not need L1 regularization here).

### **Regularised cost function-**

$$\text{RSE} = \sum_{u,i \in T} (r_{ui} - p_u^T q_i) + \lambda(\|p_u\|^2 + \|q_i\|^2),$$

Regularized squared error

## Regularised Gradient Descent

$$(P, Q) = \arg \min_{p, q} \text{RSE} \Rightarrow \begin{cases} \frac{\partial}{\partial p_u} \text{RSE} = -2(r_{ui} - p_u q_i)q_i + 2\lambda p_u \\ \frac{\partial}{\partial q_i} \text{RSE} = -2(r_{ui} - p_u q_i)p_u + 2\lambda q_i \end{cases}$$
$$\Rightarrow \begin{cases} p_u \leftarrow p_u + \eta((r_{ui} - p_u q_i)q_i - \lambda p_u) \\ q_i \leftarrow q_i + \eta((r_{ui} - p_u q_i)p_u - \lambda q_i) \end{cases},$$

## Sliding Window method In Gradient Descent

- In gradient we have compared the old and new error and break the condition if old error is less than new error and this is slow optimisation technique.
- To overcome this problem of too many local minima to reach at global minima sliding window is helpful.
- To speed up gradient descent algorithm we have used sliding window method in gradient descent.



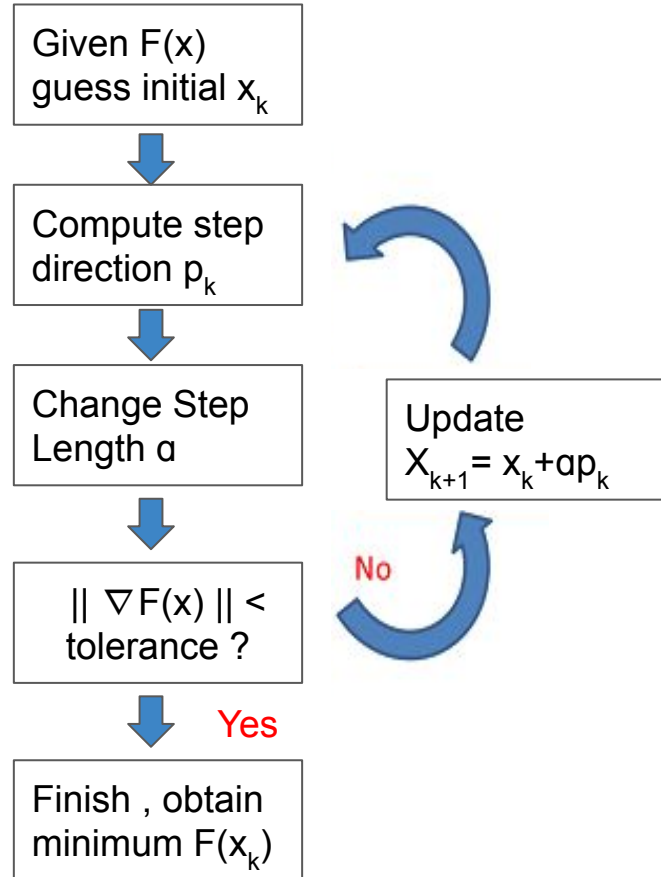


## Applying sliding window technique-

- In this method, we have taken mean of errors upto 'k' errors, where k is window size.
- Then window moves to one step ahead and delete the error at first position and again take mean of errors upto 'k' errors.
- **Stopping condition-** if old mean of error is less than new mean of error or less than some threshold.



# Line Search method In Gradient Descent



# P.S.O



Particle Swarm Optimization (PSO) is a swarm intelligence technique where a population of particles moves in a hyper-dimensional solution space to find the global optimum.

PSO is a metaheuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions.

# Updation rule :

The velocity and position of each particle is updated as follows :

$$v_i^{t+1} = v_i^t + c_1 * r_1 * (pb_i^t - x_i^t) + c_2 * r_2 * (lb_i^t - x_i^t)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

- $c_1, c_2$  are acceleration coefficient
- $r_1, r_2 \in [0, 1]$
- $p_i$  particle , position vector  $x_i$ , velocity vector  $v_i$ , personal best vector  $pb_i$  , neighbourhood best  $lb_i$

Finally the position of particle p is updated as :

$$U_p^{t+1} = U_p^t - c(\delta \frac{\partial J}{\partial U_{ij}} - (1 - \delta)(U_{best}^t - U_p^t))$$

$$V_p^{t+1} = V_p^t - c(\delta \frac{\partial J}{\partial V_{ij}} - (1 - \delta)(V_{best}^t - V_p^t))$$

## Results (Rmse) :

Technique	Test	Train
Gradient Descent	1.806	1.601
Regularised	1.90	1.80
Sliding Window	1.632	1.501
Line Search	1.832	1.33
PSO	1.23	0.87

# Results (Recommendations) :

## M.F using Gradient Descent

9102	Slow Learners (2015)
8446	The Fault in Our Stars (2014)
8418	Mulan II (2004)
4583	Tightrope (1984)
4496	Attack of the Puppet People (1958)
4182	Dogfight (1991)

## M.F using sliding window

9102	Slow Learners (2015)
8446	The Fault in Our Stars (2014)
8418	Mulan II (2004)
4583	Tightrope (1984)
4496	Attack of the Puppet People (1958)
4182	Dogfight (1991)

## M.F using L.S

9102	Slow Learners (2015)
8446	The Fault in Our Stars (2014)
8418	Mulan II (2004)
4583	Tightrope (1984)
4496	Attack of the Puppet People (1958)
4182	Dogfight (1991)

## M.F using Regularised

9102	Slow Learners (2015)
8418	Mulan II (2004)
4182	Dogfight (1991)
1813	Clue (1985)
695	Maltese Falcon, The (1941)
4496	Attack of the Puppet People (1958)

## M.F using P.S.O

8251	Rage of Honor (1987)
4693	Last Boy Scout, The (1991)
4073	Femme Fatale (2002)
9163	Applesauce (2015)
5927	3 Extremes (Three... Extremes) (Saam gaang yi) (2004)
3578	Spy Game (2001)



# Conclusion

In this project we are trying to find the movies that we can recommend to user as per their interest. By using Matrix Factorization and some of the variate of Gradient Descent we able to do so.

In our project Swarm optimisation works so well that it minimises the error to 0.87.

# References-

- [Combining Swarm with Gradient Search for Maximum Margin Matrix Factorization](#)
- <https://www.youtube.com/watch?v=ZspR5PZemcs>
- <https://towardsdatascience.com/paper-summary-matrix-factorization-techniques-for-recommender-systems-82d1a7ace74>