Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model

Matrix Factorization Techniques For Recommender Systems

Yehuda Koren, Robert Bell and Chris Volinsky

안중찬

2023. 01. 10.

CONTENTS

01 INTRODUCTION

02 PRELIMINARIES

03 A NEIGHBORHOOD MODEL

04 LATENT FACTOR MODEL

05 AN INTEGRATED MODEL

06 EVALUATION THROUGH A TOP-K RECOMMENDER

07 MORE ABOUT MATRIX FACTORIZATION

08 IMPLEMENTATION

01 INTRODUCTION

Recommender System?

- Matching consumers with most appropriate products
- Analyzing patterns of user interest in products to provide personalized recommendation that suits a user's taste

⇨ **A key to enhancing user satisfaction and loyalty !**

Broadly, recommender systems are based on one of **TWO** strategies..

1. Content Filtering : Creates a profile for each user or product to characterize its nature

e.g.) Movie profile → genre, the participating actors, its box office popularity, etc..

(-) require gathering external information that might not be available or easy to collect

2. Collaborative Filtering : Relies only on past user behavior, does not require the creation of explicit profiles

e.g.) User's previous transactions or product ratings

(-) Cold Start, inability to address the system's new products of new users

01 INTRODUCTION – Collaborative Filtering(CF)

Collaborative Filtering:
relationships between users and interdependencies among products to identify new user-item associations.

CF techniques

- require no domain knowledge
- avoid the need for extensive data collection
- uncover complex and unexpected patterns that would be difficult or impossible to profile using known data attributes

CF systems need to compare fundamentally different object: **Items VS. Users.**

TWO main disciplines of CF

1. **Neighborhood approaches**
2. **Latent factor Models**

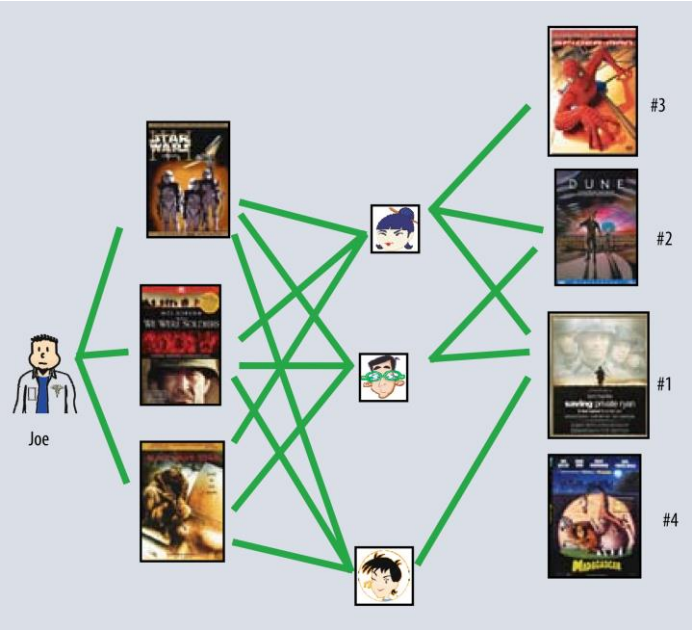
01 INTRODUCTION – CF(1) Neighborhood approach

Neighborhood method:

Computing relationships between items or, alternatively, between users

Item-oriented approaches / User-oriented approaches

- **Item-oriented approach** based on ratings of similar items by the same user
- **User-oriented approach** based on ratings of similar users by the same item



ex. User-oriented approach

-most effective at detecting **very localized relationships**

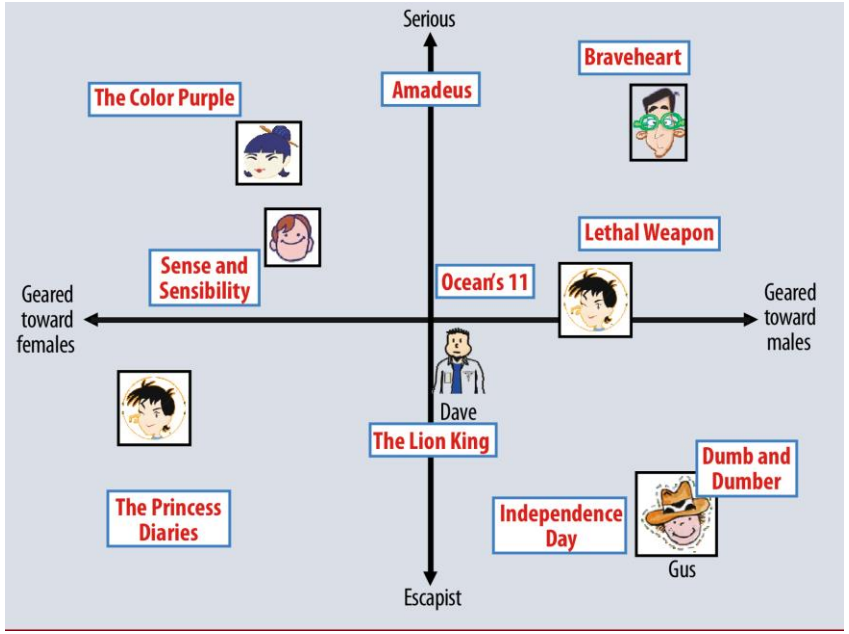
-unable to capture **the totality of weak signals** encompassed in all of a user's ratings.

01 INTRODUCTION – CF(2) Latent factor model

Latent Factor Model:

Transforming both items and users to the same latent factor space

→ try to explain the ratings by characterizing both items and user on, say, 20~100 factors inferred from the ratings pattern



ex. simplified illustration of the latent factor approach

-effective at estimating **overall structure** that relate **simultaneously** to most or all items

-poor at detecting **strong associations among a small set** of closely related items

01 INTRODUCTION – Types of input

· **Explicit feedback:** **explicit input by users regarding their interest in products**

e.g.) Star ratings – Netflix, Thumbs-up/down – TiVo

NOT always available

· **Implicit feedback:** **indirectly reflect opinion through observing user behavior**

e.g.) purchase history, search patterns, mouse movement

- Our model:

main focus

· Explicit feedback

+

· Implicit feedback

+

· Neighborhood method

+

· Latent Factor model

02 PRELIMINARIES – (1)Baseline estimates

- BASIC TERM: $for\ users;\ u, v$
$for\ items;\ i, j$
$r_{ui} = preference\ by\ user\ u\ of\ item\ i$
$\hat{r}_{ui} = the\ predicted\ value\ of\ r_{ui}$
$set\ K = \{(u,i)|r_{ui}\ is\ known\}$
$\lambda_i = Regularization\ factor$

· **Baseline estimate for an unknow rating r_{ui} :**

$$b_{ui} = \mu + b_u + b_i$$

μ : $overall\ average\ rating$
b_u: $observed\ deviation\ of\ user\ u$
b_i : $observed\ deviation\ of\ item\ i$

· **Objective :** $\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$

Regularizing term
; to avoid overfitting by penalizing the magnitude of the parameters

02 PRELIMINARIES – (2)Neighborhood models(CorNgbr)

· **Item-oriented method:**
- Better scalability, improved accuracy
- More amenable to explaining the reasoning behind predictions

↓

Our main focus

· **Similarity measure s_{ij} :** $s_{ij} \overset{\text{def}}{=} \frac{n_{ij}}{n_{ij} + \lambda_2} \rho_{ij}$

n_{ij} : *the number of users that rated both i and j*
ρ_{ij} : *pearson correlation coefficient*
λ_2 : *typically*, 100

· **Predicted vale of r_{ui} is taken as a weighted average of the ratings of neighboring items**:

(denoted as CorNgbr)

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}\left(r_{uj} - b_{uj}\right)}{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}}$$

$S^k(i;u) = set\ of\ k\ neighbors\ (for\ item\ i, user\ u)$

02 PRELIMINARIES – (2)Neighborhood models(WgtNgbr)

Some concerns

- Suitability of a similarity measure that isolates the relations b/w two items,
  without analyzing the interactions within the full set of neighbors
- Interpolation weights fully rely on the neighbors even in cases where neighborhood information is absent

· **More accurate neighborhood model**: (denoted as WgtNgbr)

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in S^k(i;u)} \theta_{ij}^u (r_{uj} - b_{uj})$$

$S^k(i;u) = set\ of\ k\ neighbors\ (for\ item\ i, user\ u)$
$\theta_{ij}^u = interpolation\ weights\ \{\theta_{ij}^u | j \in S^k(i;u)\}$

- Derivation of the interpolation weights can be done efficiently by estimating all inner products b/w item ratings

R. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", IEEE International Conference on Data Mining (ICDM'07), pp. 43–52, 2007.

02 PRELIMINARIES – (3)Latent factor models

- We focus on models that induced by **SVD** on the user-item rating matrix

Typical SVD model :

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i \,, \qquad p_u \in \boldsymbol{R}^f : user - factors\ vector\ for\ each\ user\ u$$
$$q_i \in \boldsymbol{R}^f : item - factors\ vector\ for\ each\ item\ i$$

However, Conventional SVD is undefined when knowledge about the matrix is incomplete

∴ Modify model using only the observed ratings, such as :

$$\min_{p_*, q_*, b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda_3(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

+) Paterek's NSVD model :

$$p_u \rightarrow \frac{\sum_{j\in R(u)} x_j}{\sqrt{|R(u)|}} \,, \qquad \hat{r}_{ui} = b_{ui} + q_i^T \frac{\sum_{j\in R(u)} x_j}{\sqrt{|R(u)|}}$$

$$R(u) = the\ set\ of\ items\ rated\ by\ user\ u$$

models users based on the items that they rated
(avoids explicitly parameterizing each user)

02 PRELIMINARIES – (4)Data

- Dataset : **Netflix data of more than 100 million movies ratings performed by anonymous Netflix customers**

- Quality measure : **RMSE (root mean squared error)**

$$\sqrt{\sum_{(u,i)\in|TestSet|}(r_{ui}-\hat{r}_{ui})^2 \Big/ |TestSet|}$$

- **Implicit feedback**

→ **Which movies** users rate, **regardless of how** they rated these movies

Making binary matrix ; “1” for “rated”, “0” for “not rated”

To keep generality, $R(u) = the\ set\ of\ items\ for\ which\ ratings\ by\ u\ are\ available$

$N(u) = the\ set\ of\ items\ for\ which\ u\ provided\ an\ implicit\ preference$

03 A NEIGHBORHOOD MODEL

To facilitate **global optimization**, Set **global weights independent of** a specific user

· **Initial sketch:**

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij}$$

(+) Implicit feedback

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j \in \mathrm{N}(u)} c_{ij}$$

w_{ij}'s and c_{ij}'s are offsets added to baseline estimates

→ **emphasizes the influence of missing ratings!**

03 A NEIGHBORHOOD MODEL

The current scheme encourages greater deviations from baseline for users that provided many ratings or plenty of implicit feedback (high $|R(u)|$ or $|N(u)|$)

↔ **Overemphasizes the dichotomy between heavy raters and those that rarely rate**

∴ **Replacing the prediction rule:**

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij}$$

Pruning parameters

$$R^k(i;u) \stackrel{\text{def}}{=} R(u) \cap s^k(i)$$

$$N^k(i;u) \stackrel{\text{def}}{=} N(u) \cap s^k(i)$$

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

03 A NEIGHBORHOOD MODEL

· **FINAL PREDICTION RULE :**

$$\min_{b_*, w_*, c_*} \sum_{(u,i)\in\mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{N}^k(i;u)} c_{ij} - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} \right)^2 + \lambda_4 \left(b_u^2 + b_i^2 + \sum_{j\in\mathrm{R}^k(i;u)} w_{ij}^2 + \sum_{j\in\mathrm{N}^k(i;u)} c_{ij}^2 \right)$$

· **Simple gradient descent solver:**

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $\forall j \in \mathrm{R}^k(i;u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_4 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i;u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_4 \cdot c_{ij} \right)$

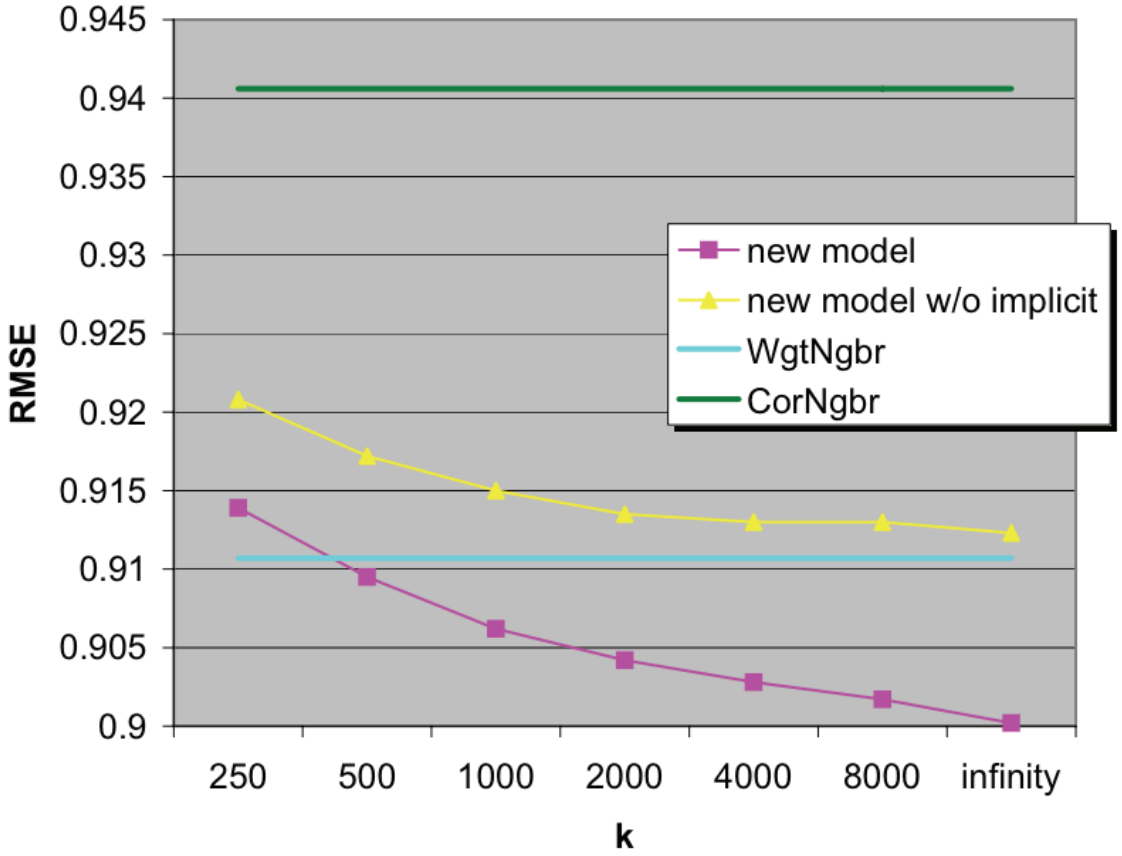
$e_{ui} \stackrel{\text{def}}{=} r_{ui} - \hat{r}_{ui}$

γ(step size) = 0.005

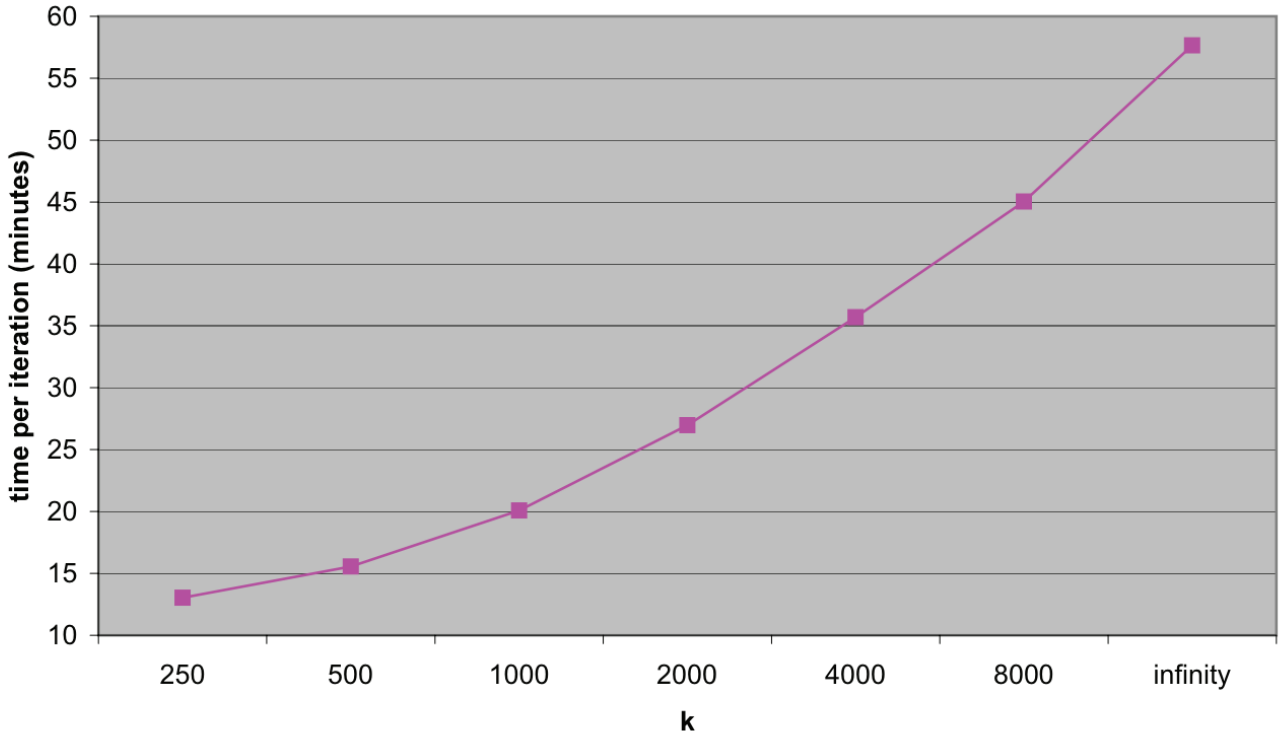
λ_4 = 0.002

by cross validation for Netflix data

03 A NEIGHBORHOOD MODEL – Experimental Results



Comparison of neighborhood-based models



Running times per iteration of the neighborhood model

04 LATENT FACTOR MODEL

· **SVD** ($\hat{r}_{ui} = b_{ui} + p_u^T q_i$) **+ Implicit feedback :**

Denoted as “Asymmetric-SVD”

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

$q_i, x_i, y_i \in \boldsymbol{R}^f$

· **Benefits of Asymmetric-SVD**

1. *Fewer parameters : the number of users is much larger than the number of items*
2. *New users : does not parametrize users*
3. *Explainability : predictions are a direct function of past user’s feedback*
4. *Efficient integration of implicit feedbacks*

04 LATENT FACTOR MODEL

Q. Predictive accuracy VS. Benefits of Asymmetric-SVD ?

We do not really have much independent implicit feedback for Netflix dataset

- **Modified model (more accurate):**

Denoted as “SVD++”

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

$p_u = a\ free\ user - factors\ vector\ learnt\ from\ the\ given\ explicit\ ratings$

- **Comparison of SVD-based models:**

Model	50 factors	100 factors	200 factors
SVD	0.9046	0.9025	0.9009
Asymmetric-SVD	0.9037	0.9013	0.9000
SVD++	0.8952	0.8924	0.8911

05 AN INTEGRATED MODEL

Latent factor models and neighborhood models nicely complement each other

- **Integrated model:**

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

This rule provides 3-tier model for recommendations:

1. General properties of the item and user
2. Interactions between the user profile and the item profile
3. Fine grained adjustments that are hard to profile

05 AN INTEGRATED MODEL

- **Solve regularized squared error function through gradient descending:**

- $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j) - \lambda_7 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
- $\forall j \in \mathrm{N}(u)$:
  $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |\mathrm{N}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
- $\forall j \in \mathrm{R}^k(i;u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i;u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij} \right)$

$e_{ui} \stackrel{\text{def}}{=} r_{ui} - \hat{r}_{ui}$

$\gamma_1, \gamma_2 = 0.007$

$\gamma_3 = 0.001$

$\lambda_6 = 0.005$

$\lambda_7, \lambda_8 = 0.015$

$k = 300$ $(neighborhood\ size)$

- **Performance of the integrated model:**

	50 factors	100 factors	200 factors
RMSE	0.8877	0.8870	0.8868
time/iteration	17min	20min	25min

06 EVALUATION THROUGH A TOP-K RECOMMENDER

- Successful improvements of recommender = enhancing user's satisfaction:

Slightly better RMSE will lead to completely different and better recommendations?

TOP-K RECOMMENDATIONS !

- for each movie i, rated 5-stars by user u + 1000 additional random movies
- order the 1001 movies based on their predicted rating in a decreasing order
- best case : movie i precede the rest 1000 random movies $\rightarrow$ appeal of a top-K recommender

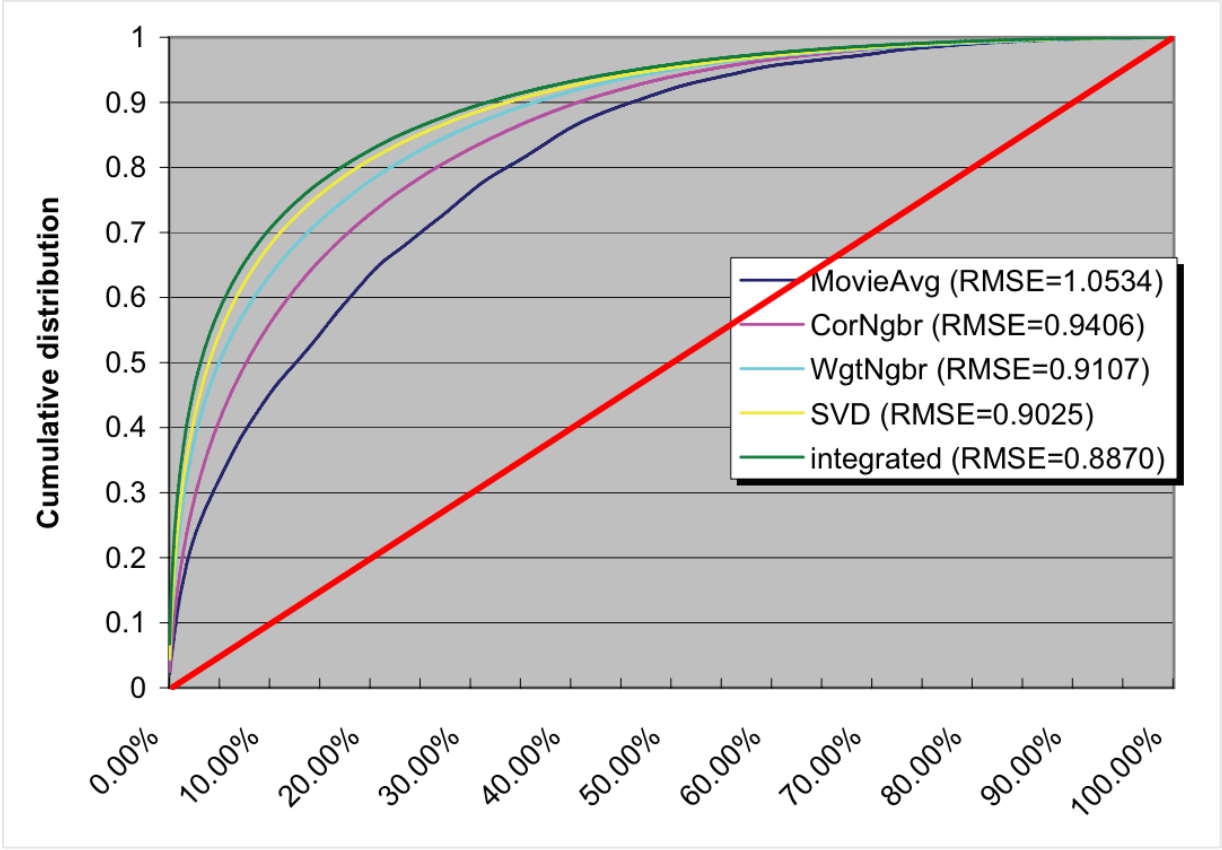
0%: none of the random movies appears before i (best case)

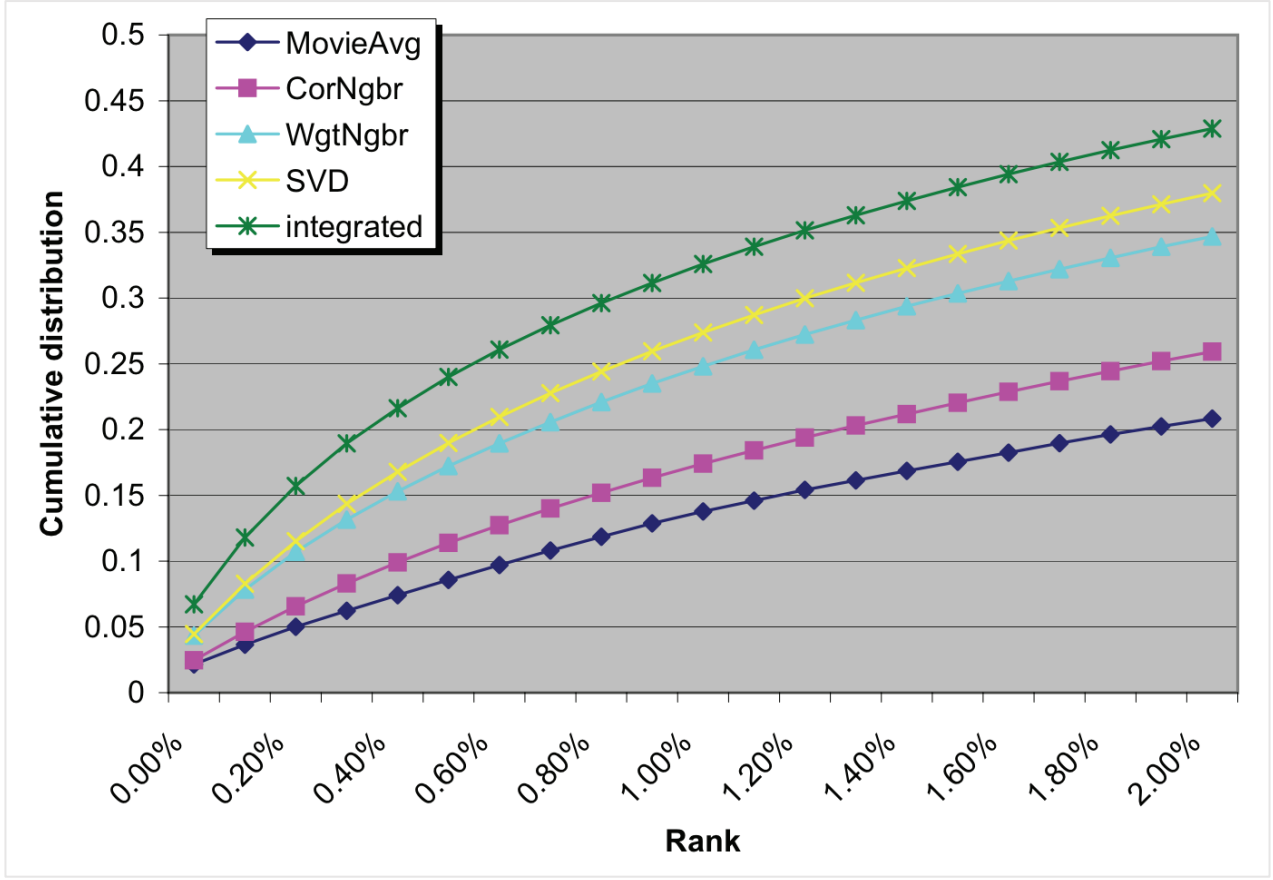
100%: all of the random movies appears before i (worst case)

Distribution of the 384,573(5-star ratings) ranks is analyzed.

06 EVALUATION THROUGH A TOP-K RECOMMENDER

- **Results:**





Zoom in on the head of x-axis

07 MORE ABOUT MATRIX FACTORIZATION

- **Basic Matrix Factorization Model:** $\hat{r}_{ui} = q_i^T p_u$

-Objective: $\min_{q^*, p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$ $(K = the\ set\ of\ the(u,i)\ pairs\ for\ which\ r_{ui}\ is\ known)$

- **Learning Algorithms:**

(1) Stochastic gradient descent

- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$

(2) Alternating least squares

; fix one of the unknows, the optimization problem becomes quadratic and can be solved optimally

- **Favorable in TWO cases** - when the system can use parallelization
  - for systems centered on implicit data

07 MORE ABOUT MATRIX FACTORIZATION

- **Additional input sources:**

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a]$$

$N(u) = the\ set\ of\ items\ for\ which\ user\ u\ expressed\ an\ implicit\ perference$

$x_i \in \boldsymbol{R}^f$

$A(u) = the\ set\ of\ another\ user\ attributes(e.g.)\ demographics, gender, age\ group..$

$y_a \in \boldsymbol{R}^f$

- **Temporal dynamics:**

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$

- **Inputs with varying confidence levels:**

$$\min_{p^*, q^*, b^*} \sum_{(u,i) \in \kappa} c_{ui}(r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2)$$

$(c_{ui} = confidence\ in\ observing\ r_{ui})$

- Not all observed ratings deserve the same weight or confidence
- Systems built around implicit feedback

07 MORE ABOUT MATRIX FACTORIZATION

- **Netflix Prize Competition:**

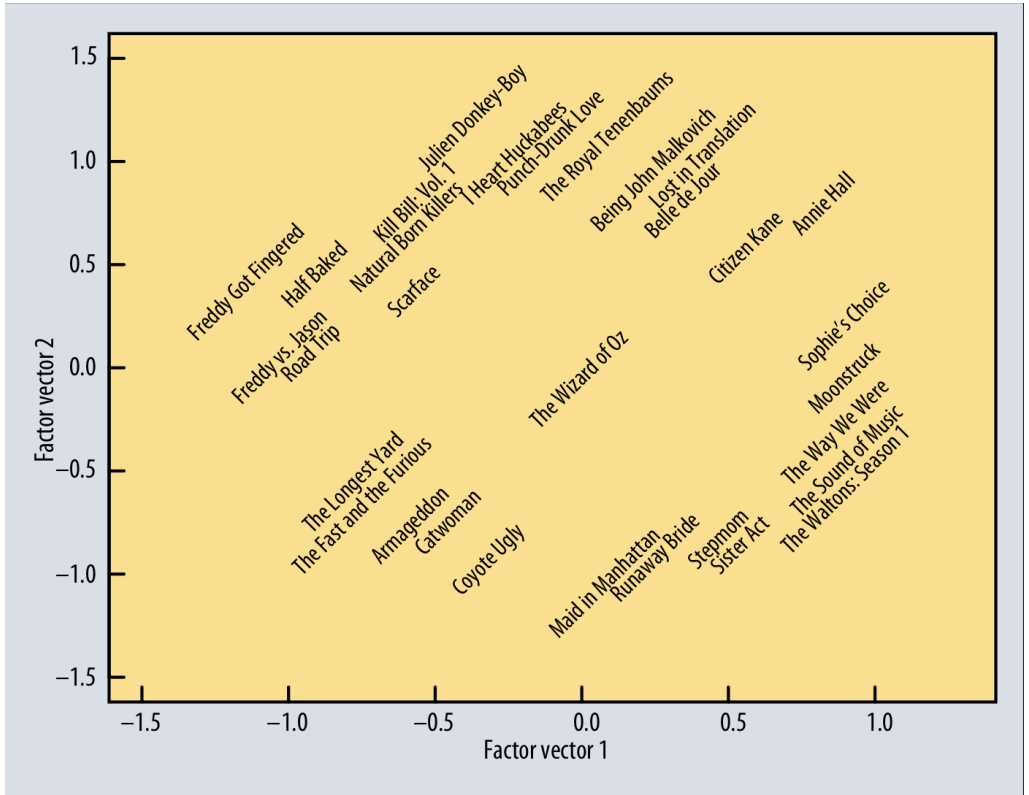


Figure 3. The first two vectors from a matrix decomposition of the Netflix Prize data. Selected movies are placed at the appropriate spot based on their factor vectors in two dimensions. The plot reveals distinct genres, including clusters of movies with strong female leads, fraternity humor, and quirky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

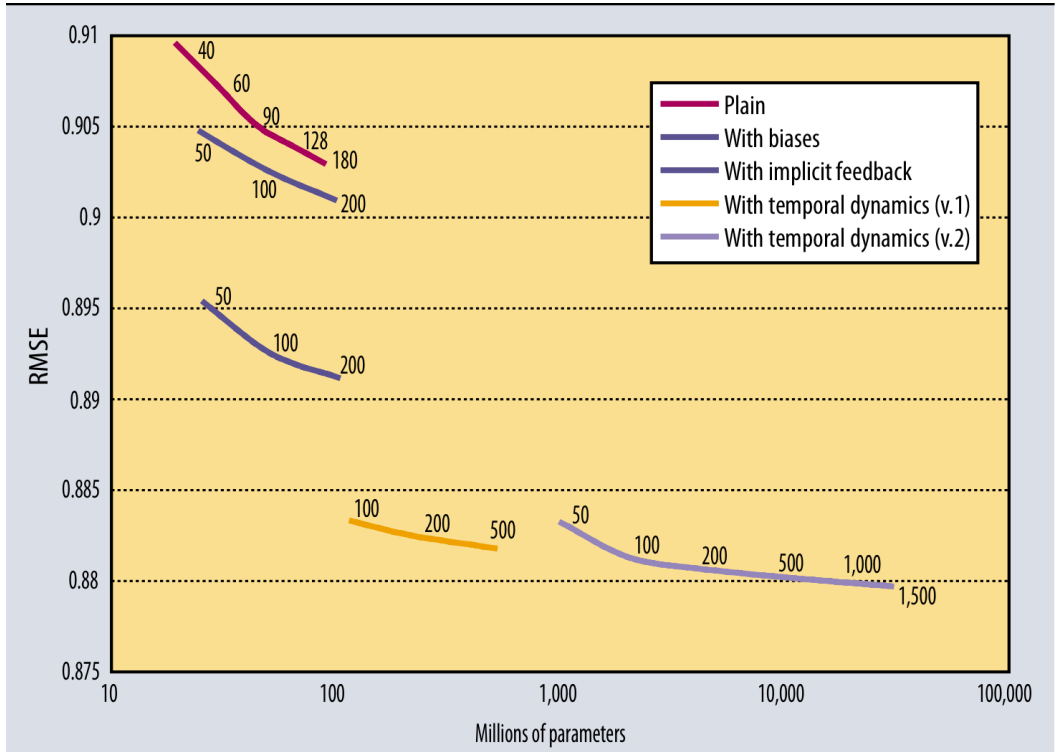
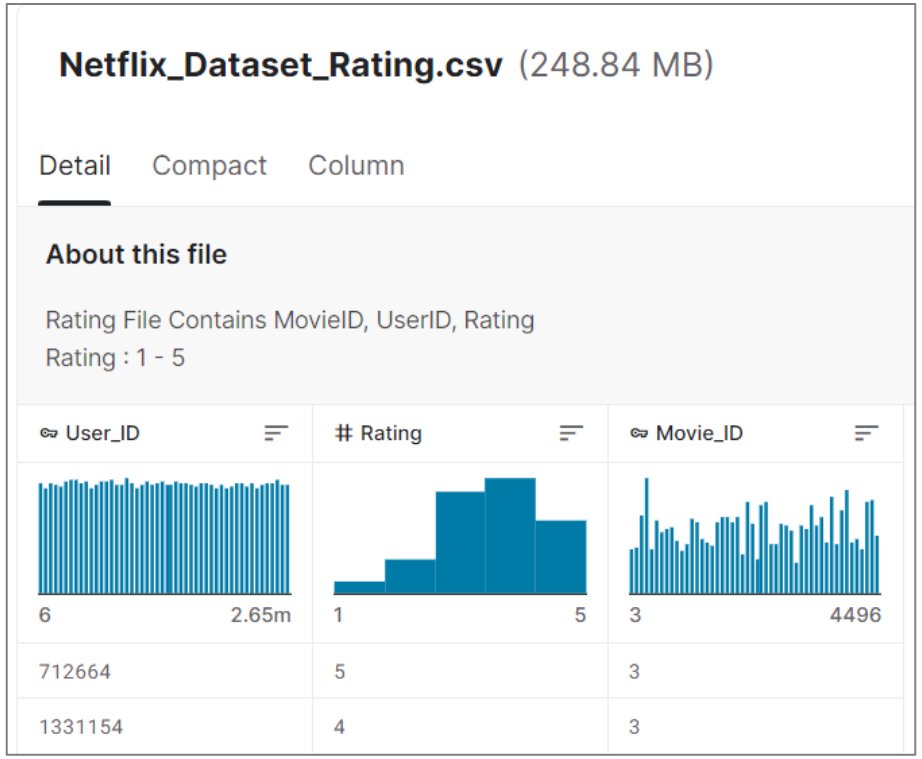


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more accurate. For comparison, the Netflix system achieves RMSE = 0.9514 on the same dataset, while the grand prize's required accuracy is RMSE = 0.8563.

08 IMPLEMENTATION

- **Dataset: Netflix Movie Ratings (27,329 users/3,292,522 ratings)**



17,337,458 ratings

1,350 movies

143,458 users

```
In [1]: import pandas as pd
        import numpy as np

In [2]: data = pd.read_csv('Netflix_Dataset_Rating/Netflix_Dataset_Rating.csv')

In [4]: data = data[data['User_ID'] < 100000]

In [7]: data.to_csv('./reduced_1.csv', index=False)
```

```
In [5]: df['Rating'].describe().astype('int')

count    644151
mean          3
std           1
min           1
25%           3
50%           4
75%           4
max           5
Name: Rating, dtype: int32

In [6]: print('Unique Values:\n',df.nunique())

Unique Values:
 User_ID     5431
Rating          5
Movie_ID     1350
dtype: int64
```

644,151 ratings

1,350 movies

5,431 users

08 IMPLEMENTATION – SVD/SVD++

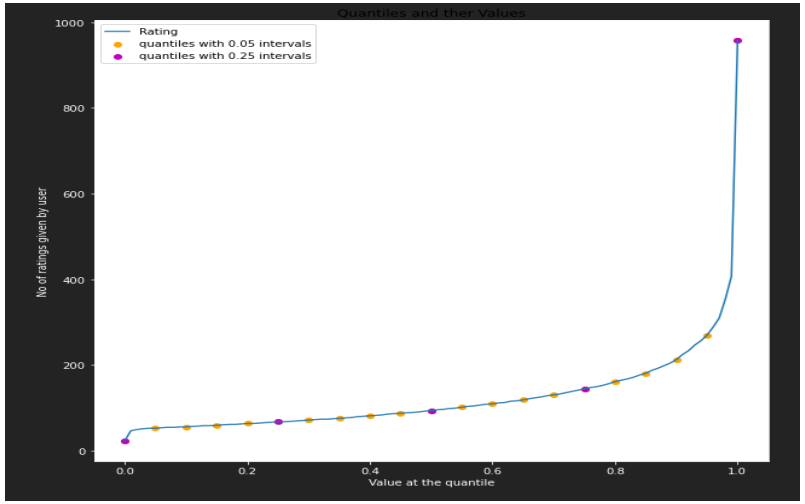
· Using Surprise/ Data visualization

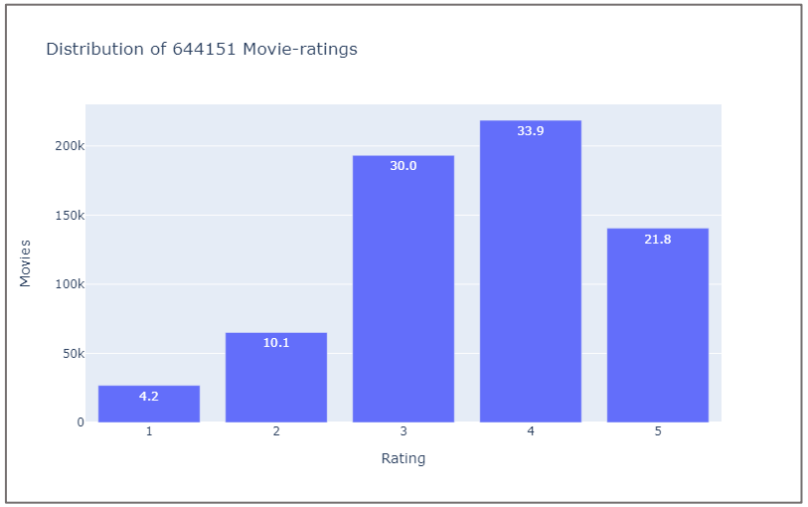
```
[20] from surprise import Reader, Dataset, SVD, SVDpp
     from surprise import accuracy
     from surprise.model_selection import train_test_split

[6]  df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/netflix/Netflix_rating_reduced.csv')

[22] data = Dataset.load_from_df(df[['User_ID','Movie_ID','Rating']],Reader())
     trainset, testset = train_test_split(data, test_size=0.3, random_state=101)
     trainset = data.build_full_trainset()

[11] ###SVD EXPERIMENT###
     algo1 = SVD(n_epochs=10, n_factors=10)
     algo2 = SVD(n_epochs=10, n_factors=30)
     algo3 = SVD(n_epochs=10, n_factors=50)

[12] ###SVD++ EXPERIMENT###
     algo1pp = SVDpp(n_epochs=10, n_factors=10, vebose=True)
     algo2pp = SVDpp(n_epochs=10, n_factors=30)
     algo3pp = SVDpp(n_epochs=10, n_factors=50)
```

- Testset : Randomly extraction of 30% of dataset

MODEL	10 factors	30 factors	50 factors
SVD	0.8727	0.8568	0.8391
SVD++	0.8336	0.8061	0.7850

(epochs = 10)





08 IMPLEMENTATION – SVD/SVD++

· Using Numpy

```
class SVD():
    def __init__(self, train, test, factor, learning_rate, reg_param, epochs, verbose = False):
        self.R = train
        self.test = test
        self.num_users, self.num_items = train.shape
        self.f = factor
        self.learning_rate = learning_rate
        self.reg_param = reg_param
        self.epochs = epochs
        self.verbose = verbose

    def fit(self):
        #initialize latent matrix(Xavier)
        self.P = np.random.normal(scale = 1./self.f, size = (self.num_users, self.f))
        self.Q = np.random.normal(scale = 1./self.f, size = (self.num_items, self.f))

        #initialize biases
        self.b_u = np.zeros(self.num_users)
        self.b_i = np.zeros(self.num_items)
        self.mu = np.mean(self.R[np.where(self.R != 0)])

        self.training_epochs = []
        start = timer()

        for epoch in range(self.epochs):

            for u in range(self.num_users):
                for i in range(self.num_items):
                    if self.R[u,i] > 0:
                        self.gradient_descent(u,i, self.R[u,i])
```

```
trainset, testset = train_test_split(df, stratify=df['User_ID'], test_size=0.3, random_state=88)
```

- Testset : Stratified extraction of 30% of dataset

```
def gradient(self, error, u, i):
    dp = (error * self.Q[i,:]) - (self.reg_param * self.P[u,:])
    dq = (error * self.P[u,:]) - (self.reg_param * self.Q[i,:])
    return dp, dq

def gradient_descent(self, u, i, rating):
    prediction = self.get_prediction(u,i)
    error = rating - prediction

    self.b_u[u] += self.learning_rate * (error - self.reg_param * self.b_u[u])
    self.b_i[i] += self.learning_rate * (error - self.reg_param * self.b_i[i])

    dp, dq = self.gradient(error, u, i)
    self.P[u,] += self.learning_rate * dp
    self.Q[i,] += self.learning_rate * dq

def get_prediction(self, u, i):
    return self.mu + self.b_u[u] + self.b_i[i] + self.P[u,].dot(self.Q[i,].T)
```

(learning rate = 0.005, regularization parameter = 0.02, epochs = 10)

MODEL	10 factors	30 factors	50 factors
SVD	0.9092	0.9102	0.9103
SVD++	0.8826	0.8818	0.8825

08 IMPLEMENTATION

· Integrated Model (incomplete)

- $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j) - \lambda_7 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
- $\forall j \in \mathrm{N}(u)$:
  $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |\mathrm{N}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
- $\forall j \in \mathrm{R}^k(i;u)$:
  $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot \left(|\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij} \right)$
- $\forall j \in \mathrm{N}^k(i;u)$:
  $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot \left(|\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij} \right)$

```python
def gradient(self, u, i, N_u, R_k_iu, N_k_iu):
    pred, bias_weight = self.prediction(u, i, N_u, R_k_iu, N_k_iu)
    error = self.R[u,i] - pred

    dbu = error - self.lambda6 * self.b_u[u]
    dbi = error - self.lambda6 * self.b_i[i]
    dq = error * (self.P[u] + (np.sum(self.y[N_u], axis = 0)/np.sqrt(len(N_u)))) - self.lambda7 * self.Q[i]
    dp = error * self.Q[i] - self.lambda7 * self.P[u]
    dy = (error * self.Q[i]/np.sqrt(len(self.N_u))).reshape(1, -1) - self.lambda7 * self.y[N_u]
    dw = error * bias_weight/np.sqrt(len(self.R_k_iu)) - self.lambda8 * self.w[i,R_k_iu]
    dc = error/np.sqrt(len(N_k_iu)) - self.lambda8 * self.c[i, N_k_iu]
    return dbu, dbi, dq, dp, dy, dw, dc, error**2

def gradient_descent(self, u, i, N_u, R_k_iu, N_k_iu):
    dbu, dbi, dq, dp, dy, dw, dc, squared_error = self.gradient(u, i, N_u, R_k_iu, N_k_iu)
    self.b_u[u] += self.gamma1 * dbu
    self.b_i[i] += self.gamma1 * dbi
    self.Q[i] += self.gamma2 * dq
    self.P[u] += self.gamma2 * dp
    self.y[N_u] += self.gamma2 * dy
    if len(R_k_iu) > 0:
        self.w[i,R_k_iu] += self.gamma3 * dw
    if len(N_k_iu) > 0:
        self.c[i,N_k_iu] += self.gamma3 * dc
    return squared_error
```

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

```python
def prediction(self, u, i, N_u, R_k_iu, N_k_iu):
    p = self.P[u] + np.sum(self.y[N_u], axis = 0)/np.sqrt(len(N_u))
    factor = np.dot(p, self.Q[i].T)
    neighbor_exp = 0
    neighbor_imp = 0
    if len(R_k_iu) > 0:
        bias_diff = self.R[u,R_k_iu] - (self.mu + self.b_u[u] + self.b_i[R_k_iu])
    else:
        bias_diff = 0
        neighbor_exp = 0
    if len(N_k_iu) > 0:
        neighbor_imp = np.sum(self.c[i,N_k_iu])/np.sqrt(len(N_k_iu))
    else:
        neighbor_imp = 0
    return self.mu + self.b_u[u] + self.b_i[i] + factor + neighbor_exp + neighbor_imp , bias_diff
```

THANK YOU –