INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Introduction to recommendation system with collaborative filtering

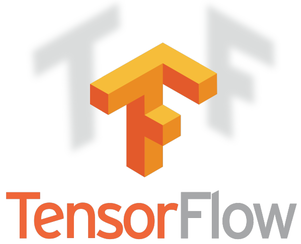
IA Frameworks

Brendan Guillouet - 14/12/2020

Institut National des Sciences Appliquées de Toulouse

Tools

ML Python Libraries



Python Environment

Viz' Python Libraries

Framework & Tool

Table of contents

Introduction

E-commerce and recommandation system

Context

Customer relationship management

Appetence score

Product recommendation

Recommendation System

Content Base. (User and Item metadata).

Méthode de bandit (Appetence score)

Collaborative Filtering

Based on existing *product X customer* relationship

E-commerce and recommandation system

Context

Customer relationship management

Appetence score

Product recommendation

Recommendation System

Content Base. (User and Item metadata).

Méthode de bandit (Appetence score)

Collaborative Filtering

Based on existing *product X customer* relationship

Collaborative Filtering

2 types of solutions

Based on neighbourhood.

Based on latent vector.

Matrix factorisation.

Matrix completion.

Neural Network.

Difficulties

How to evaluate the recommendation?

Number of parameters to estimate.

Cold start, number of latent factors, etc.

Neighbourhood-based methods

Introduction

Two methods

User-User Filter

Choose a similarity metric between users.

For a user u, find the set of k closest users S_u^k.

Predict a rate $\hat{r}_{u,i}$ for an item i as a linear combination of know rates $r_{u',i}$ for $u' \in S_u^k$.

Item-Item Filter

Choose a similarity metric between items.

For an item i, find the set of k closest users S_i^k.

Predict a rate $\hat{r}_{u,i}$ for a user i as a linear combination of know rates $r_{u,i'}$ for $i' \in S_i^k$.

Similarity metric

For two users u and v *(or for two items i and j):*

Pearson correlation coefficient.

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u) \cdot (r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \cdot \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}}$$

I_u is the set of items with known rate from user *u.*

$r_{u,i}$ rate of item i given by user $u.$

$\bar{r}_u$ mean rate given by user $u.$

Similarity metric

For two users u and v *(or for two items i and j):*

Spearman Rank Correlation Coefficient

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v} (\tilde{r_{u,i}} - \bar{\tilde{r}}_u) \cdot (\tilde{r_{v,i}} - \bar{\tilde{r}}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (\tilde{r_{u,i}} - \bar{\tilde{r}}_u)^2} \cdot \sqrt{\sum_{i \in I_u \cap I_v} (\tilde{r_{v,i}} - \bar{\tilde{r}}_v)^2}}$$

I_u is the set of items with known rate from user *u.*

$\tilde{r}_{u,i}$ is the rank of product i within the items ranked by user u.

$\bar{\tilde{r}}_u$ mean rank given by user u.

Similarity metric

For two users u and v *(or for two items i and j):*

Cosine

$$s(u,v) = cos(u,v) = \frac{< u, v >}{||u|| \cdot ||v||}$$

Based on vectorial space structure.

Similarity metric

For two items i and j

Conditional Probability

$$s(i,j) = P[j \in B \mid i \in B]$$

B is the purchase history.

User-User Filters

Main assumption : customers with a similar profile will have similar tastes.

Generic Formula

$$\hat{r}_{u,i} = \frac{\sum_{u' \in S_u^k} s(u,u') \cdot r_{u',i}}{\sum_{u' \in S_u^k} |s(u,u')|}$$

$r_{u,i}$ rate of item i given by user u.

s is the similarity metric between u and u'.

S_u^k is the set of k closest neighbours of user u.

User-User Filters

Main assumption : customers with a similar profile will have similar tastes.

Generic Formula with mean

$$\hat{r}_{u,j} = \bar{r}_{u,.} + \frac{\sum_{u' \in S_u^k} s(u,u') \cdot (r_{u',i} - \bar{r}_{u',.})}{\sum_{u' \in S_u^k} |s(u,u')|}$$

$r_{u,i}$ rate of item i given by user u.

s is the similarity metric between u and u'.

S_u^k is the set of k closest neighbours of user u.

$\bar{r}_{u,.}$ mean rate given by user u.

User-User Filters

Main assumption : customers with a similar profile will have similar tastes.

Generic Formula with Z-score

$$\hat{r}_{u,j} = \bar{r}_{u,.} + \sigma_{u,.} \cdot \frac{\sum_{u' \in S_u^k} s(u,u') \cdot \frac{(r_{u',i} - \bar{r}_{u',.})}{\sigma_{u',.}}}{\sum_{u' \in S_u^k} |s(u,u')|}$$

$r_{u,i}$ rate of item i given by user u.

s is the similarity metric between u and u'.

S_u^k is the set of k closest neighbours of user u.

$\bar{r}_{u,.}$ mean rate given by user u.

$\sigma_{u,.}$ is standard deviation of rate given by user u.

Item-Item Filters

Main assumption : customers will prefer products similar to those he appreciate.

Generic Formula with Z-score

$$\hat{r}_{u,i} = \bar{r}_{.,i} + \sigma_{.,i} \cdot \frac{\sum_{i' \in S_i^k} s(i,i') \cdot \frac{(r_{u,i'} - \bar{r}_{.,i'})}{\sigma_{.,i'}}}{\sum_{i' \in S_i^k} |s(i,i')|}$$

$r_{u,i}$ rate of item i given by user u.

s is the similarity metric between i and i'.

S_i^k is the set of k closest neighbours of user i.

$\bar{r}_{.,i}$ mean rate given by item i.

$\sigma_{.,i}$ is standard deviation of rate given by user i.

Remarks

Easy to interpret results.

k can be chosen with cross validation.

Good performance on small dataset..

.. but suffer from scalability problem as user base grows.

Use with Surprise python library.

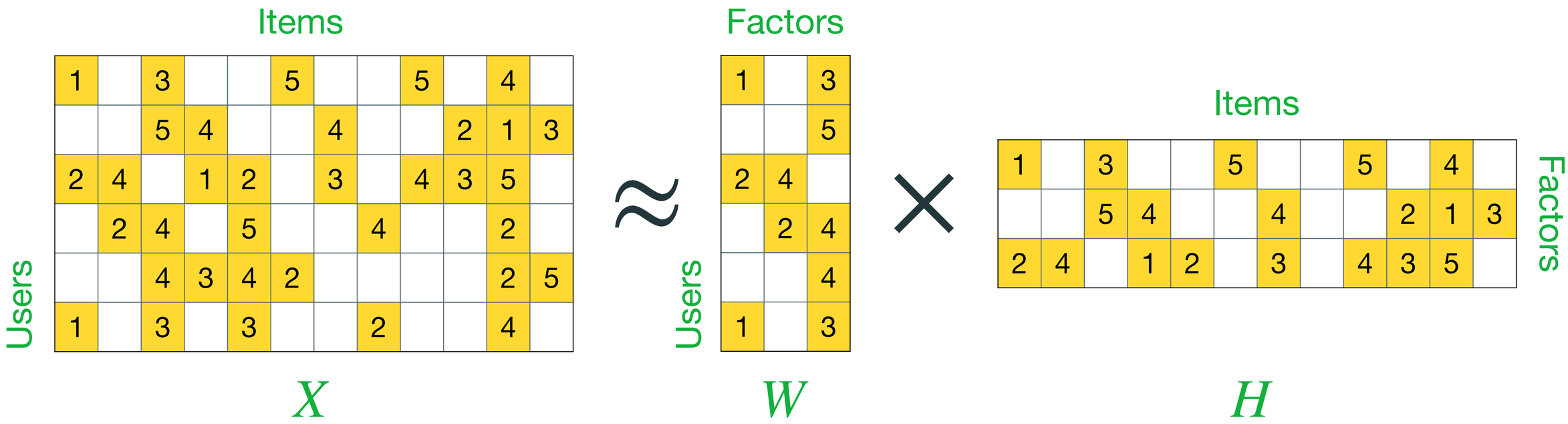
Latent Vector Based Methods

Factorisation

Matrix Factorisation

Let $X \in \mathcal{M}_{N \times M}$ be the matrice of user/item notations
where N is the number of users and M the number of items.

Main idea: Approximate X as a product of two matrices $W \in \mathcal{M}_{N \times K}$ and $H \in \mathcal{M}_{K \times M}$



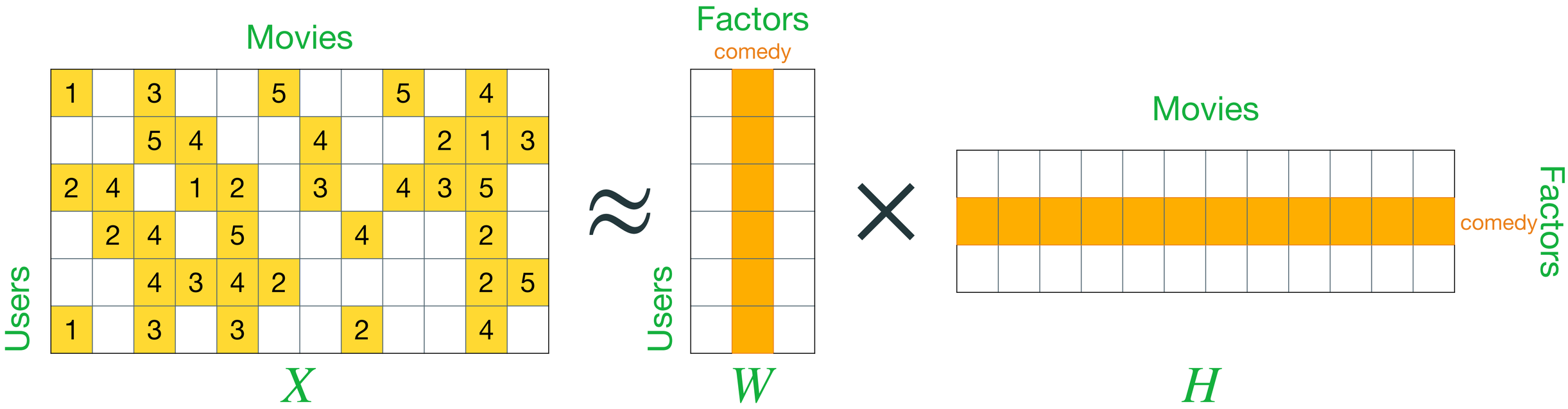
Each entry of X can be approximated with the following formula.

$$X_{i,j} = W_{i,.}^T \cdot H_{.j} = \sum_{k \in K} w_{i,k} \cdot h_{k,j}$$

Matrix Factorisation

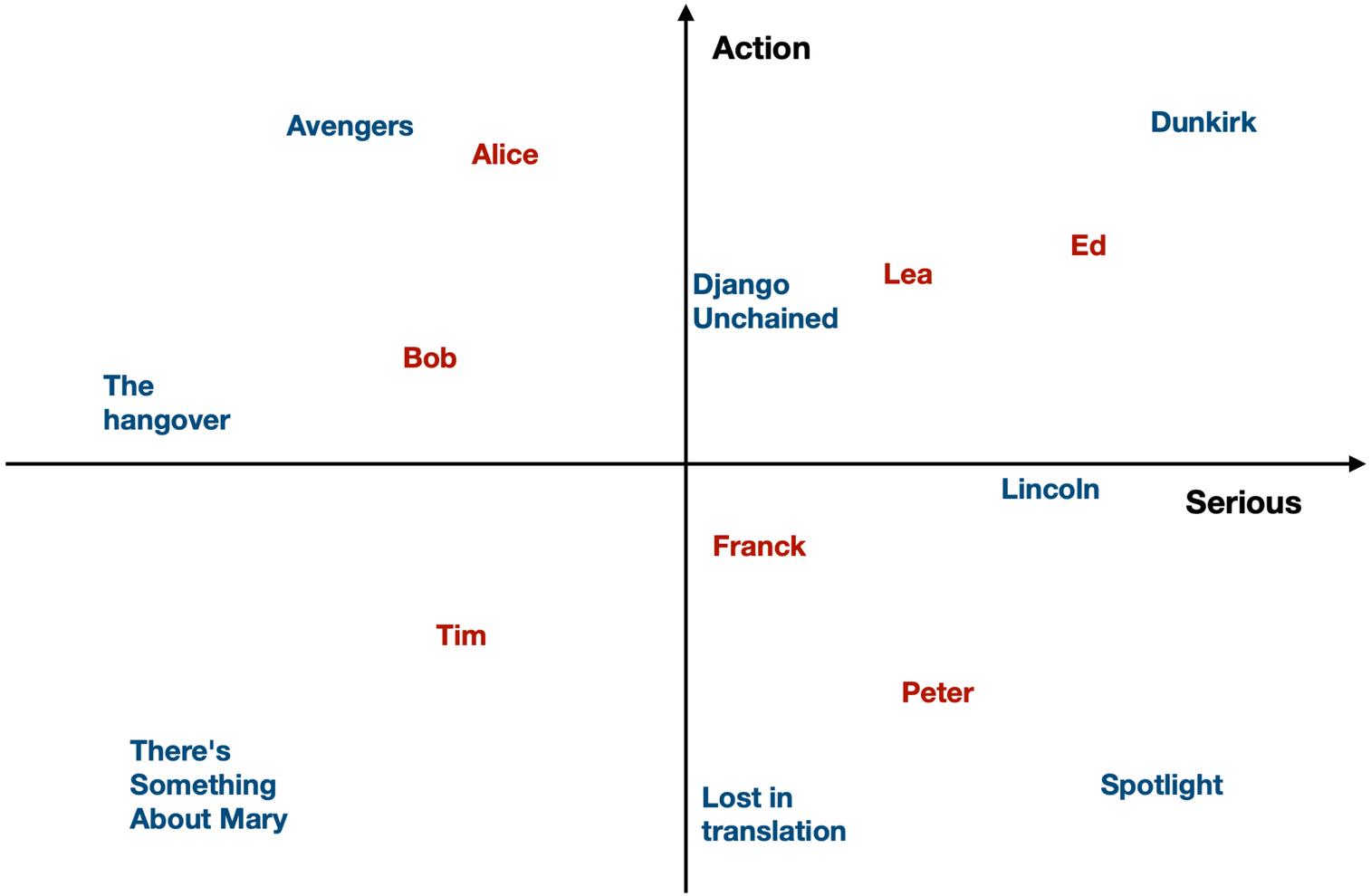
The latent vectors can be interpreted as embedding.

Example: Movie recommendation



Latent factors can be interpreted as the genre of the movie.

Interpretation



Similar users/movies get embedded nearby in the embedding space

Non negative matrix factorisation

To recover W and H the objective is to solve this minimisation problem:

$$\min_{W,H \geq 0, rk(W)=r, rk(H)=r} L(X, WH) + \lambda_w P(W) + \lambda_h P(H)$$

L is the loss function (measure accuracy of prediction).

λ_w, λ_h are penalisation parameters.

P is the penalty function.

Constraint of positivity on W and H ⟶ **N**on Negative **M**atrix **F**actorisation

NMF - ALS ALGORITHMS

EXAMPLE: with L_2 regularisation and euclidean loss function (a non convex problem).

$$\min_{W,H\geq 0, rk(W)=r, rk(H)=r} \sum_{i,j} (x_{i,j} - w_i^T h_j)^2 + \sum_{i=1}^{N} \lambda_w ||w_i||^2 + \sum_{j=1}^{M} \lambda_h ||h_j||^2$$

SOLUTION: Alternate **L**east **S**quare**.**

$\forall i$, fix all variables except w_i and solve:

$$argmin_{w_i} \sum_{j} (x_{i,j} - w_i^T h_j)^2 + \lambda_w ||w_i||^2$$

$\forall j$, fix all variables except h_i and solve:

$$argmin_{h_j} \sum_{i} (x_{i,j} - w_i^T h_j)^2 + \lambda_h ||h_j||^2$$

NMF - ALS ALGORITHMS

$$argmin_{w_i} \sum_j (x_{i,j} - w_i^T h_j)^2 + \lambda_w ||w_i||^2$$

This is a regularised least squares regression problem which has the following solution:

$$w_i = (\sum_j h_j h_j^T + \lambda_w I_k)^{-1} (\sum_j x_{i,j} v_j)$$

Hence NMF problem can be solved by updating these values iteratively.

$$w_i \leftarrow (\sum_j h_j h_j^T + \lambda_w I_k)^{-1} (\sum_j x_{i,j} h_j)$$

$$h_j \leftarrow (\sum_i w_i w_i^T + \lambda_h I_k)^{-1} (\sum_i x_{i,j} w_j)$$

NMF Algorithm

Lee and seung [1999] Euclidean loss, no regularisation.

$$H_{k,j} \leftarrow H_{k,j} \frac{(W^T X)_{k,j}}{(W^T W H)_{k,j}}, \quad W_{i,k} \leftarrow W_{i,k} \frac{(X H^T)_{i,k}}{(V H H^T)_{i,k}},$$

Brunet and al. [1999] Kullback-Leibler loss, no regularisation.

$$H_{k,j} \leftarrow H_{k,j} \frac{\sum_l \frac{W_{l,k} X_{l,j}}{(WH)_{l,j}}}{\sum_l W_{l,k}}, \quad W_{i,k} \leftarrow W_{i,k} \frac{\sum_l \frac{H_{k,l} X_{i,l}}{(WH)_{i,l}}}{\sum_l H_{k,l}}$$

Implementation

These algorithms has been widely studied and a lot of different implementation exist.

NMF R package. (9 implementation among *Kim an park [2007], Lee and Seung [1999], Brunet al[2004])*.

A lot of interpretation tools such that consensus matrix, dendrogram, etc.

MlLib Spark. (*Koren et al[2009])*.

Implementation that handle missing value.

Surprise Python. (*Zhang et al. [2006], Luo et al. [2014])*.

Scikit-Learn Python. (*Lee and Seung [1999], Hoyer [2004])*.

Remarks

SVD decomposition can also provide a solution for recommendation

$$X = UDV*$$

Good property as unity and optimal solution guaranteed for fixed rank.

Optimise the rank.

NMF minimisation problem does not always lead to well posed problem.

Cold start problem is a major research point.

Most implementations of these algorithm treat missing rates as zero.

Latent Vector Based Methods

Completion

SoftImpute - Mazumder and al.[2010]

Mazumder et al. [2010] re-write the minimisation problem as

$$\min_{M} \frac{1}{2} ||P_{\Omega}(X) - P_{\Omega}(M)||_F^2 + \lambda ||M||_*$$

where:

Ω contains the pairs of indices (i, j) where X is observed.

$P_{\Omega}(X)_{i,j} = X_{i,j}$ if $X_{i,j}$ is known, 0 otherwise.

$||M||_*$ is the nuclear norm of M.

SoftImputE

If $\hat{M}$ solves the problem, then it satisfies the following stationary condition

$$\hat{M} = S_\lambda(Z)$$

where:

$$Z = P_\Omega(X) + P_{\Omega^\perp}(\hat{M})$$

and

$$S_\lambda(Z) = UD_\lambda V', \quad D_\lambda = diag[(d_1 - \lambda)_+, \ldots, (d_r - \lambda)_+]$$

Reconstruct $S_\lambda(Z) = UD_\lambda V^T$ from SVD decomposition is called “soft-thresholded SVD”.

For sufficiently large λ, D_λ will be rank-reduced, and hence so will be $UD_\lambda V^T$.

SoftImpute Algorithm - Pseudo Code

Initialise $\hat{M}$

Iterate over the following steps:

- compute $Z = P_{\Omega}(X) + P_{\Omega^{\perp}}(\hat{M})$
- compute $\hat{M} = S_{\lambda}(Z)$

We only use known values of X.

Much more faster than NM.

Use within soft impute R package

Latent Vector Based Methods

Neural network

Neural Recommander system

As for the NMF the main idea is to minimise the objective function

$$\sum_{i,j \in \Omega} (r_{i,j} - w_i^T h_j)_2^2 + \lambda(||W||_2^2 + ||H||_2^2||)$$

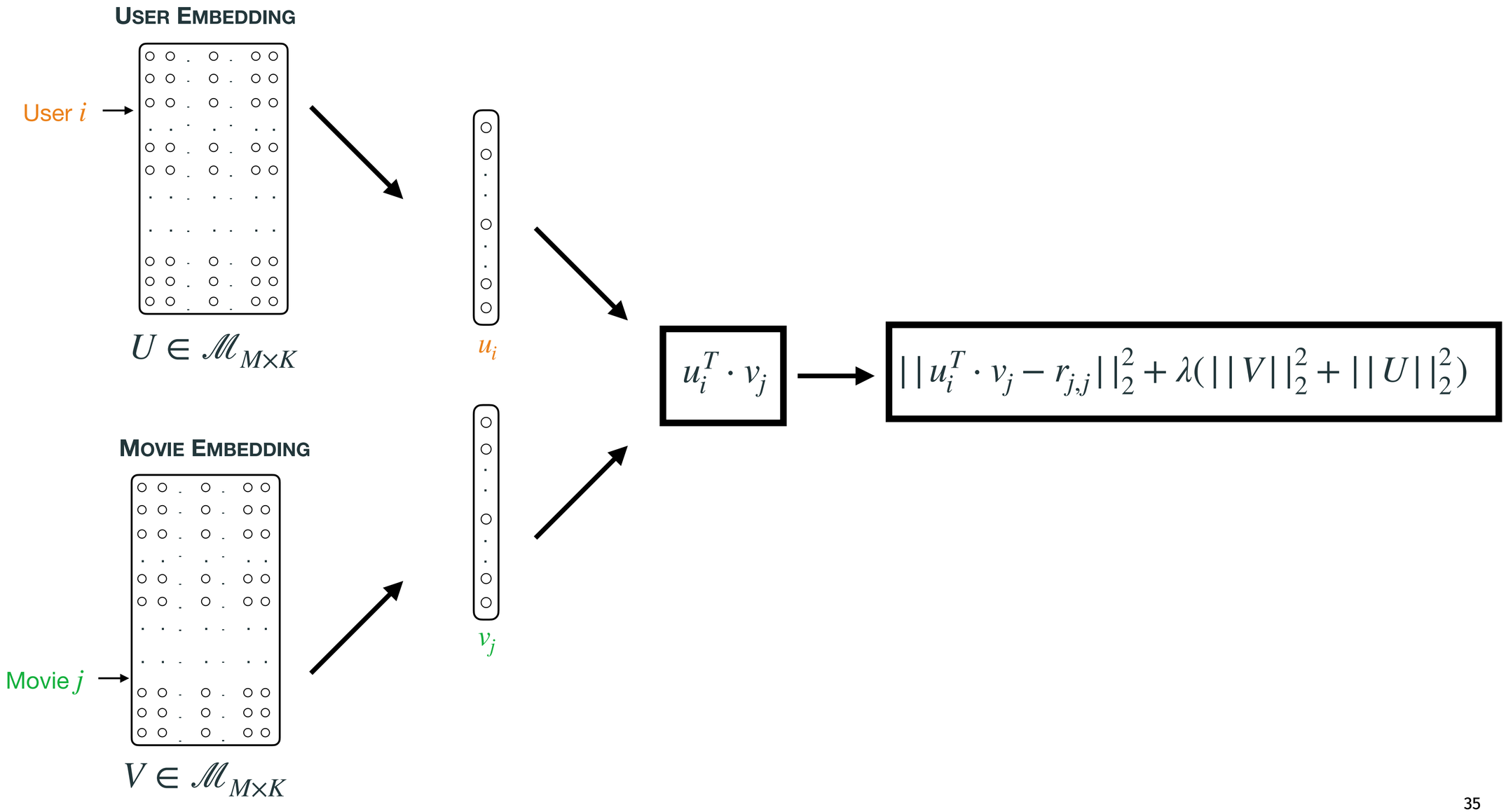
Where

The embedding w_i of a movie i in an embedding space of size k.

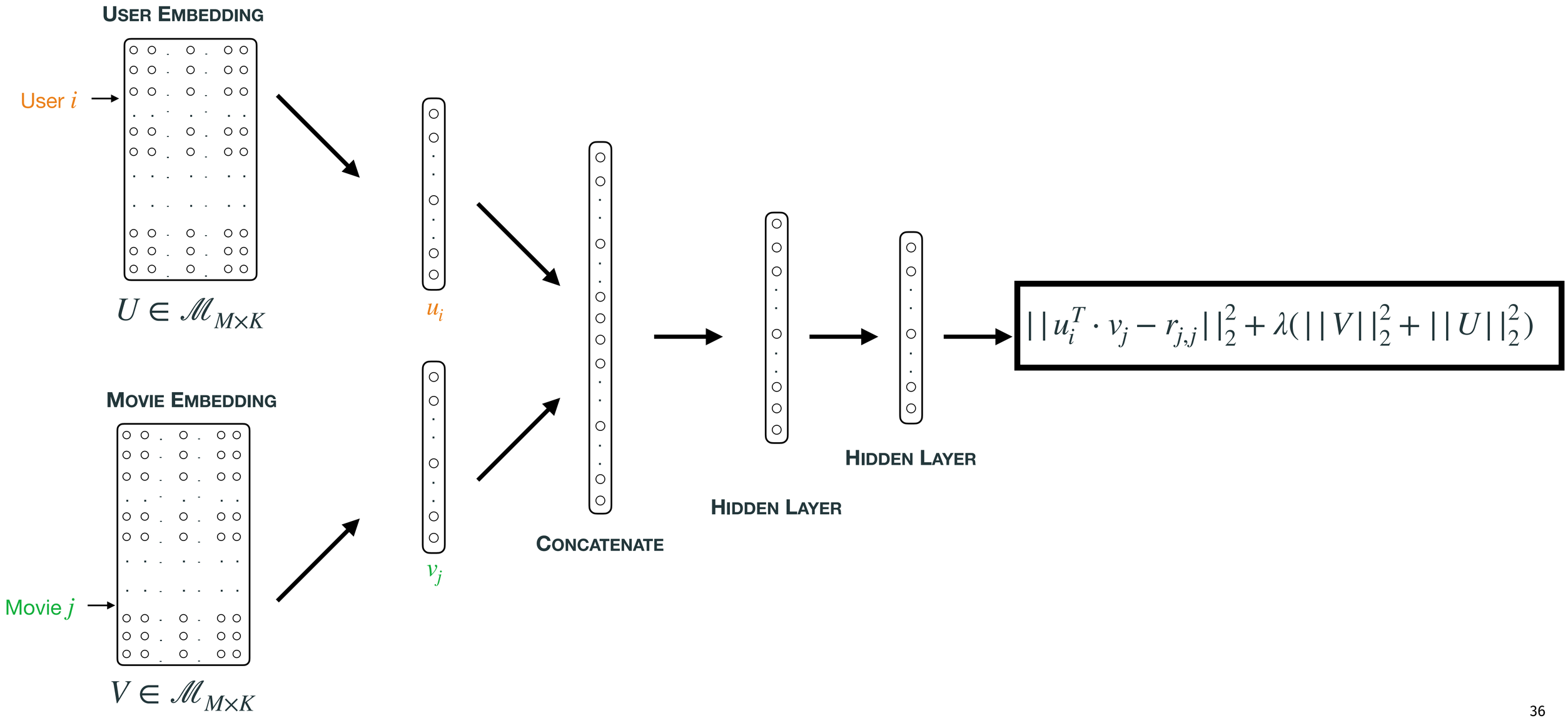
The embedding h_j of a movie j in an embedding space of size k.

Solution: Use a neural network architecture and framework to solve it.

Architecture



Deep architecture



Embedding with keras

```
import tensorflow.keras.layer as kl
user_id_input = kl.Input(shape=[1], name='user')
item_id_input = kl.Input(shape=[1], name='item')

user_embedding = kl.Embedding(output_dim=embedding_size, input_dim=max_user_id + 1,
                              input_length=1, name='user_embedding')(user_id_input)
item_embedding = kl.Embedding(output_dim=embedding_size, input_dim=max_item_id + 1,
                              input_length=1, name='item_embedding')(item_id_input)

user_vecs = kl.Flatten()(user_embedding)
item_vecs = kl.Flatten()(item_embedding)

y = kl.Dot(axes=1)([user_vecs, item_vecs])
```

embedding_size = *K*, size of latent vector.

input_dim = *M (N)*, Size of users (items).

input_length = *1*, (*i.e.* the indice).

All weights of user_embedding are trainable and tuned during backprob.

Remarks

Handle missing data (as it just optimise over existing user item couple).

Good performance comparing to other methods (especially if you have GPU).

Hard to interpret the function you designed to compare the vector…

…but you can still interpret the embedding vectors.

TP

MovieLens Dataset

MovieLens dataset will be used during TP

- It’s produce by the groupLens company (https://grouplens.org/).
- Various size of datasets
  - Stable one. 25 Millions of rate. Use for challenge and benchmark
  - Small one. 100k of rates. Use for education of research (not stable).
- Contains various metadata that we won’t use in this TP.
  - Genre, tag, Age of user, etc…

TP

surprise.ipynb. Use Surprise's Python library to generate:

- Neighbourhood-based methods.
- Factorisation latent vector based methods.

tensorflow.ipynb. Use Tensorflow's Python library to generate:

- Neural network latent vector based methods.

FORMER TP

R.ipynb. Use SoftImpute's R library.

pyspark.ipynb. Use ml's Pyspark library.

References

http://wikistat.fr/pdf/st-m-datSc3-colFil.pdf h

ttps://cse.iitk.ac.in/users/piyush/courses/ml_autumn16/771A_lec14_slides.pdf

https://m2dsupsdlclass.github.io/lectures- labs/slides/03_recommender_systems/index.html

References

Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the national academy of sciences*, 101(12) :4164–4169, 2004.

Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov) :1457–1469, 2004.

Hyunsoo Kim and Haesun Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12) :1495–1502, 2007.

Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 549–553. SIAM, 2006.

References

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8) :30–37, 2009.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755) :788, 1999.

Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2) :1273–1284, 2014.

Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research*, 11(Aug) :2287–2322, 2010.