Data Analysis Lab

Recommendation Systems
User-based Systems
Content-based systems

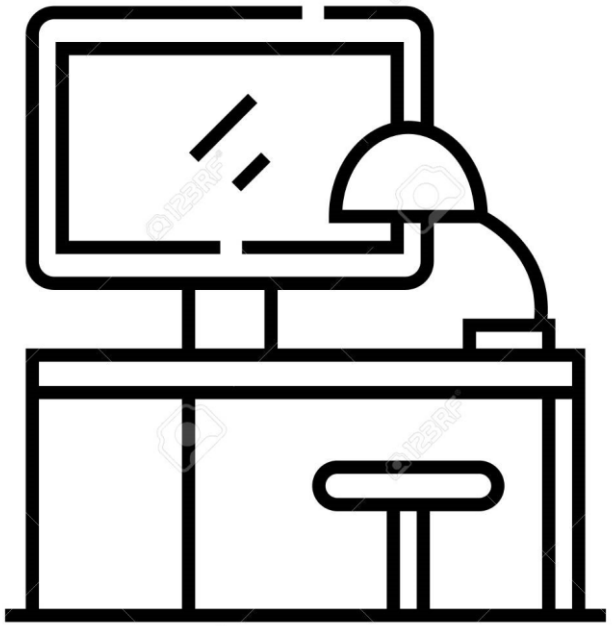
Fátima Leal

DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UPT UNIVERSIDADE PORTUCALENSE

Previous Lesson

- SVM and k-NN



Outline

- Introduction to Recommendation systems
- Naïve User-based Systems
- Content-based Systems
- Exercises

Introduction to Recommendation Systems

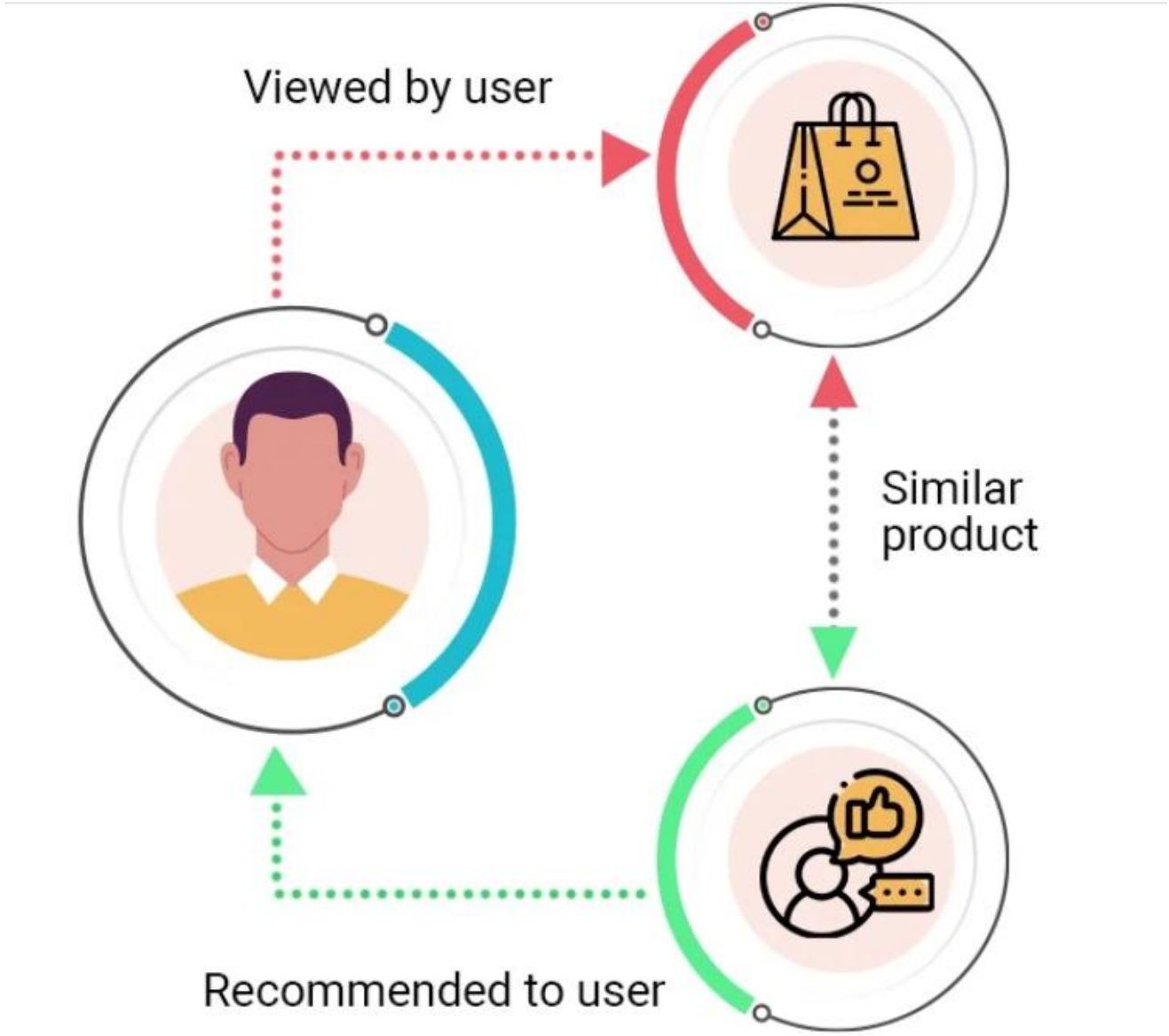
- The basic concepts are users, items, and ratings (or an implicit feedback about the products, like the fact of having bought them).
- Every model must work with known data (like in a supervised scenario), to be able to suggest the most suitable items or to predict the ratings for all the items not evaluated yet.
- We have two kinds of strategies: User/item or content-based, or collaborative filtering

Introduction to Recommendation Systems

- **Content-based filtering** is based on the information we have about users or products and its target is to associate a new user with an existing group of peers to suggest all the items positively rated by the other members, or to cluster the products according to their features and propose a subset of items **similar to the one taken into account.**

- **Collaborative-based filtering** is a little bit more sophisticated, works with explicit ratings and its purpose is **to predict this value for every item and every user**.

Content-based Recommendation Systems


Viewed by user
Similar product
Recommended to user

User-based Systems

- We assume that we have a set of users represented by feature vectors: $U = \{\bar{u}_1, \bar{u}_2, ..., \bar{u}_n\}$, where $\bar{u}_n \in \mathbf{R}^n$. Typical features are age, gender, interests, and so on.
- We have a set of items: $I = \{i_1, i_2, ..., i_m\}$
- Let's assume also that there is a relation which associates each user with a subset of items (bought or positively reviewed), items for which an explicit action or feedback has been performed:$g(\bar{u}) = \{i_1, i_2, ..., i_m\}$ where $k \in (0, m)$

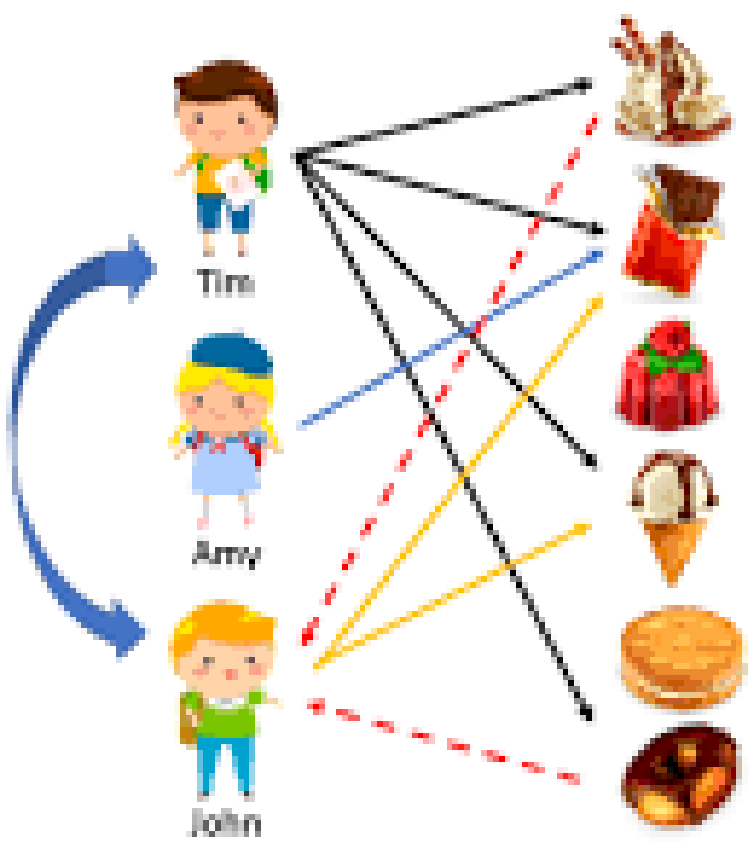
User-based Systems

- In a user-based system, the users are periodically clustered (normally using a k-nearest neighbors approach), and therefore, considering a generic user u (also new), we can immediately determine the ball containing all the users who are similar (therefore neighbors) to our sample: $B_R(\bar{u} = \{\bar{u}_1, \bar{u}_2, ..., \bar{u}_n\}$
- At this point, we can create the set of suggested items using the relation previously introduced:
  $I_{Suggested}(\bar{u} = \{\bigcup g(\bar{u}_i) where \bar{u}_i \in B_R(\bar{u})\}$
- In other words, the set contains all the unique products positively rated or bought by the neighborhood.

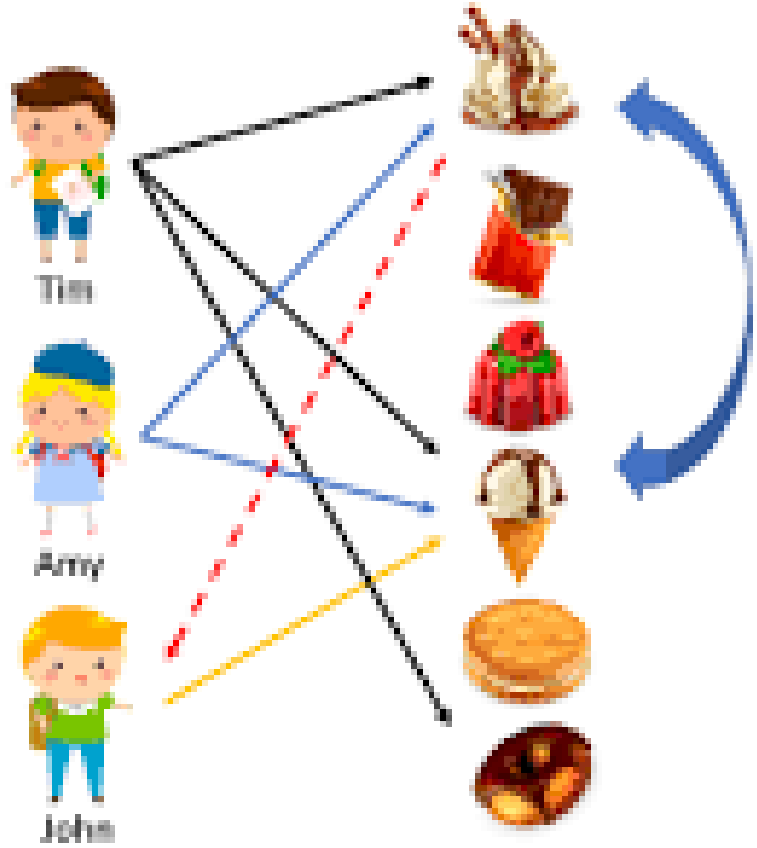
Item-based Systems

- This is probably the simplest method and it's based only on the products, modeled as feature vectors: $I = \{\bar{i}_1, \bar{i}_2, ..., \bar{i}_n\}$, where $\bar{ui}_n \in \mathbf{R}^n$.
- Just like the users, the features can also be categorical, for example, the genre of a book or a movie, and they can be used together with numerical values after encoding them.
- Then a clustering strategy is adopted, even if the most used is k-nearest neighbors as it allows controlling the size of each neighborhood to determine, given a sample product, the quality and the number of suggestions.

Content-based Systems



(a) User-based filtering



(b) Item-based filtering

Content-based example

```
nn = NearestNeighbors (n_neighbors =20 , radius =5.0 , metric
='jaccard')
```

Distance metrics

Profiling

```
import numpy as np
from sklearn . neighbors import NearestNeighbors
nb_users = 1000
users = np. zeros ( shape =( nb_users , 4))
for i in range (nb_users ):
    users [i, 0] = np. random.randint (0, 4)
    users [i, 1] = np. random.randint (0, 2)
    users [i, 2] = np. random.randint (0, 5)
    users [i, 3] = np. random.randint (0, 5)

print(users)
nb_product = 20
user_products = np.random.randint (0, nb_product , size =(nb_users , 5))
print(user_products)
```

Recommendations

```
nn = NearestNeighbors (n_neighbors =20 , radius =2.0)
nn.fit(users)
test_user = np. array ([2 , 0, 3, 2])
d, neighbors = nn.kneighbors (test_user.reshape (1, -1))
print (neighbors)
suggested_products = []
for n in neighbors :
    for products in user_products [n]:
        for product in products :
            if product != 0 and product not in suggested_products :
                suggested_products.append (product)
print (suggested_products)
```

Content-based - example

- Download Spotify Dataset
- Download the script python to generate contente-based recommendations
- Is it item or user-based?

Apply other similiraty metrics

- Euclidean distances
- Manhanttan
- Cosine
- Sigmoid_kernel
- Etc.

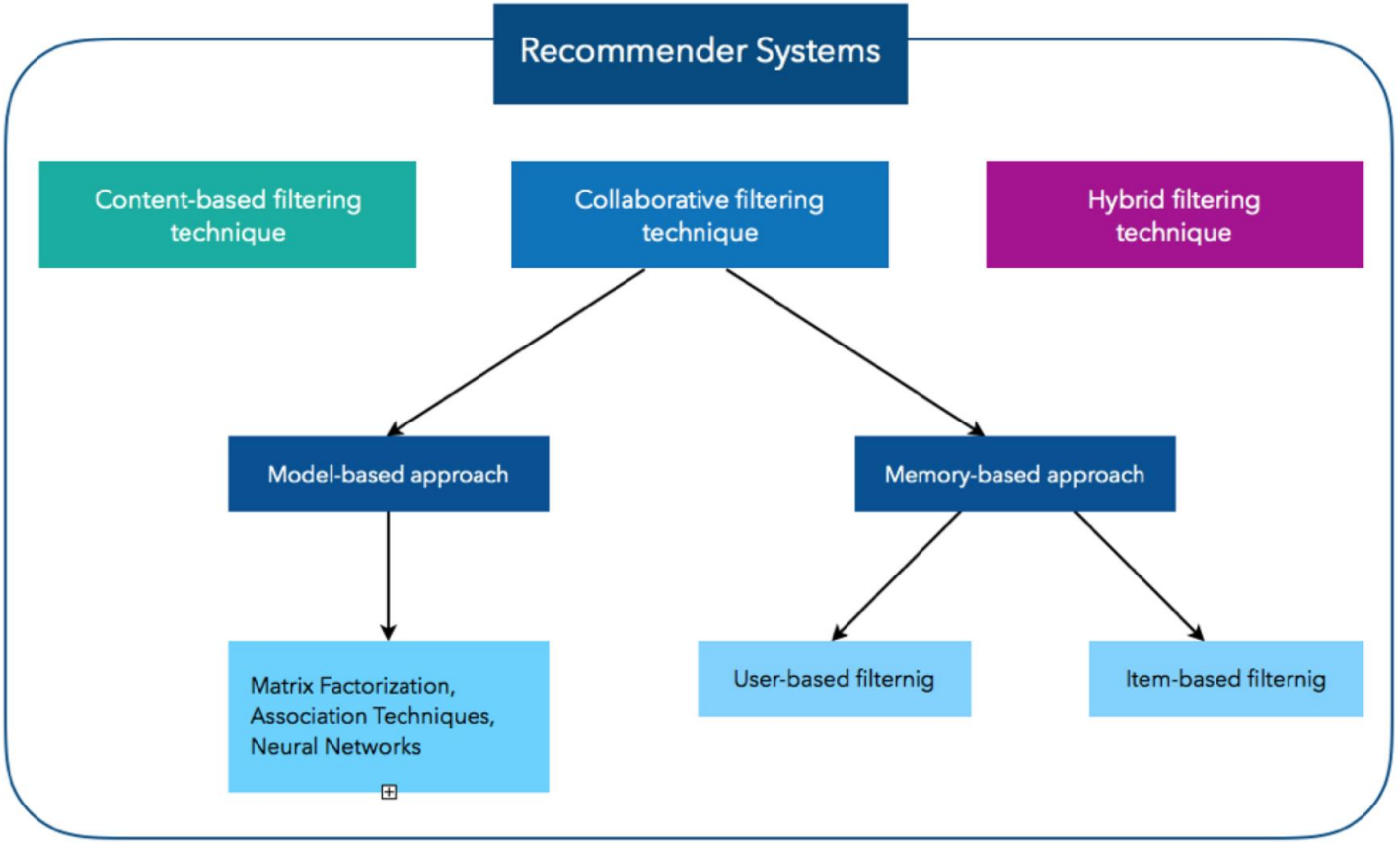
- Implement the k-nn for this dataset

Collaborative Filtering

https://surpriselib.com/

Collaborative Filtering

- Memory-based
- Model-based



Memory-based Collaborative Filtering

- As with the user-based approach, let's consider having two sets of elements: users and items. However, in this case, we don't assume that they have explicit features.
- We try to model a user-item matrix based on the preferences of each user (rows) for each item (columns).

$$M_{UxI} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{pmatrix}$$

Memory-based Collaborative Filtering

- The ratings are bounded between 1 and 5 (0 means no rating), and our goal is to cluster the users according to their rating vector.
- This allows producing recommendations even when there are no explicit pieces of information about the user.
- However, it has a drawback, called cold-startup, which means that when a new user has no ratings, it's impossible to find the right neighborhood, because he/she can belong to virtually any cluster.
- Once the clustering is done, it's easy to check which products (not rated yet) have the higher rating for a given user and therefore are more likely to be bought.

Memory-based Collaborative Filtering

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	0	3	0	3	4
User 2	4	0	0	2	0
User 3	0	0	3	0	0
User 4	3	0	4	0	3
User 5	4	3	0	4	4

Memory-based Collaborative Filtering

```
from surprise import KNNBasic
from surprise import Dataset
from surprise.model_selection import train_test_split,
cross_validate

data = Dataset.load_builtin('ml-100k')
print(data)
trainset = data.build_full_trainset()
algo = KNNBasic ()
algo.fit( trainset )

results=cross_validate(algo, data , measures =[ 'RMSE', 'MAE'], cv
=5, verbose = True )
print(results)
```

Model-based Collaborative Filtering

Model-based Collaborative Filtering

- This is currently one of the most advanced approaches and is an extension of what was already seen in the previous section. The starting point is always a rating-based user-item matrix
- However, in this case, we assume the presence of latent factors for both the users and the items. In other words, we define a generic user as: $\bar{p}_i = (p_{i1}, p_{i2}, ..., p_{ik}$ where $p_{ij} \in \mathbb{R}$
- A generic item is defined as: $\bar{q}_j = (q_{j1}, q_{j2}, ..., q_{jk}$ where $q_{jt} \in R$
- We don't know the value of each vector component (for this reason they are called latent), but we assume that a ranking is obtained as: $r_{ij} = \bar{p}_i . \bar{q}_j^T$

Model-based Collaborative Filtering

- So we can say that a ranking is obtained from a latent space of rank k, where k is the number of latent variables we want to consider in our model.
- However, there's still a big problem to solve: finding the latent variables.
- The Singular Value Decomposition (SVD) of the user-item matrix. This technique allows transforming a matrix through a low-rank factorization and can also be used in an incremental way $M_{UxI} = U \sum V^T$ where $U \in \mathbb{R}^{mxt}, \sum \in \mathbb{R}^t xt$ and $V \in \mathbb{R}^{nxm}$

Model-based Collaborative Filtering: example

```
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split, cross_validate

data = Dataset.load_builtin ('ml-100k')
trainset, testset = train_test_split (data , test_size =.25)
algo = SVD ()
algo.fit( trainset )
predictions = algo.test ( testset )
accuracy.rmse ( predictions )
cross_validate (algo , data , measures =[ 'RMSE', 'MAE'], cv =5, verbose
= True )
```

Exercises

- Use the datasets available on Moodle "MovieLens1M and Books"
- Do a exploratory/statistical analysis of the dataset
- Divide the dataset in train and test
- Apply the memory or model based collaborative filtering
- Obtain the predictions and scores

References

- Bonaccorso, G. (2020). Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work. Packt Publishing Ltd.
- Bonaccorso, G. (2018). Machine Learning Algorithms: Popular algorithms for data science and machine learning. Packt Publishing Ltd.
- Lee, W. M. (2019). Python machine learning. John Wiley & Sons.
- Hauck, T. (2014). scikit-learn Cookbook. Packt Publishing Ltd.
- Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). Introduction to linear regression analysis. John Wiley & Sons.
- Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.