ITP4514

Artificial Intelligence & Machine Learning

Lecture 10 – Recommender Systems

AY 2021-22

Module Plan

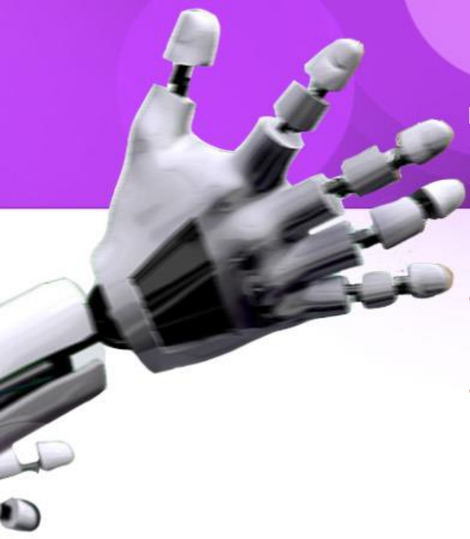
L10 – Recommender Systems

Lesson Outline

1. Introduction
2. Candidate Generation Overview
3. Types of Recommender Systems
4. Content-Based Filtering
5. Collaborative Filtering
6. Retrieval, Scoring, and Re-ranking

1. Introduction

- A system that helps users find compelling content in large corpora. (e.g., YouTube, App Store)
- Two commonly used recommendations:
  - – Homepage Recommendations
  - – Related Item Recommendations

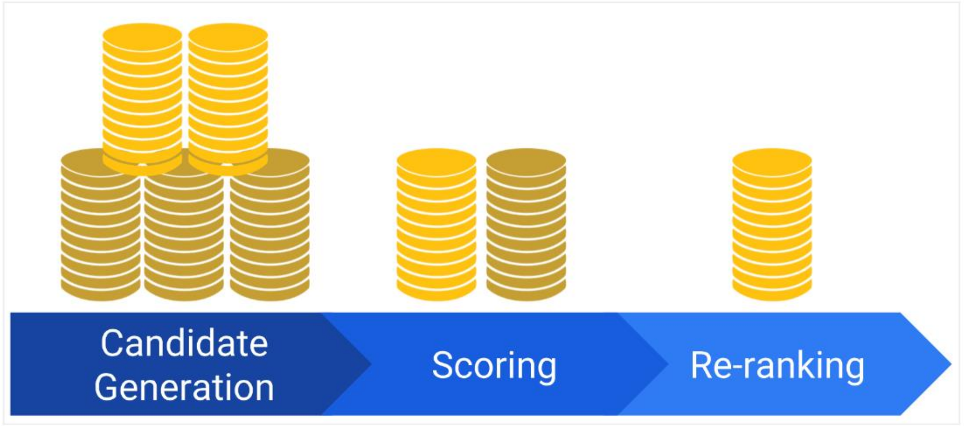


Terminology

- Items – entities a system recommends
- Query – information a system uses to make recommendations
  - User information: uid / browse record
  - Additional Content: time / user's device
- Embedding – A mapping from a discrete set

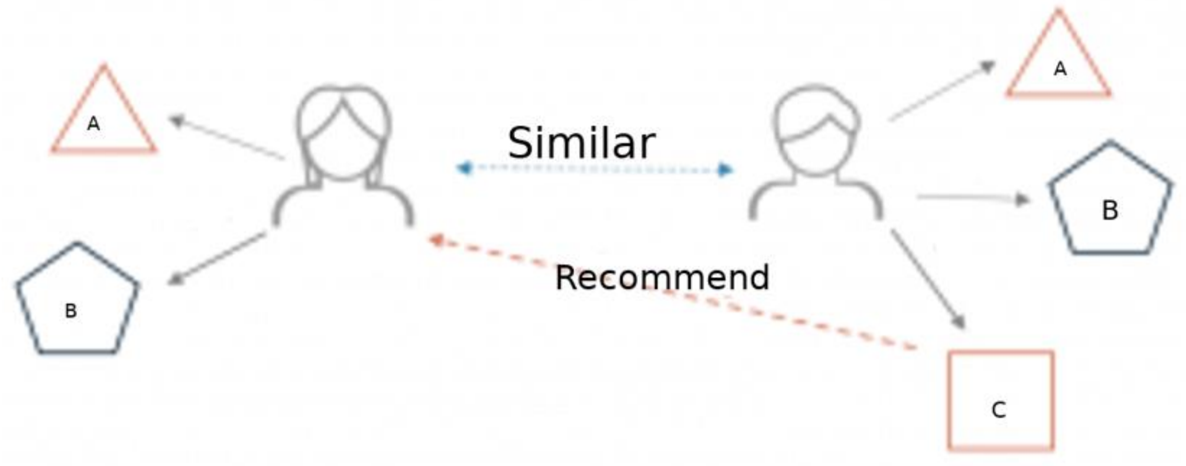
Recommender Systems

- Common architecture for recommendation system consists of the following components:
  - Candidate Generation
  - Scoring
  - Re-ranking



What is a Recommender System?

- How recommender systems work in the context of an e-commerce site:



What is a Recommender System? (2)

- Two users buy the same items A and B from an e-commerce store.
- When this happens the *similarity index* of these two users is computed.
- Depending on the score the system can recommend item C to the other user because it detects that those two users are *similar* in terms of the items they purchase.

Underlying Concepts

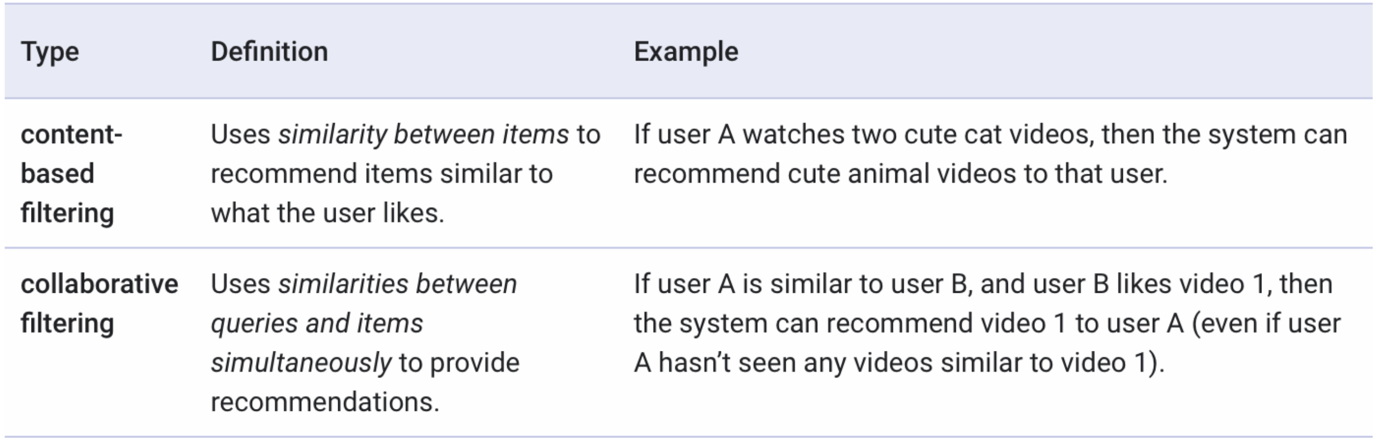
- Recommendation engines have been around for a while and there have been some key learnings to leverage:
  1. *A user’s actions are the best indicator of user intent.* Ratings and feedback tends to be very biased and lower volumes.
  2. *Past actions and purchases* drive new purchases and the overlap with other people’s purchases and actions is a fantastic predictor.

2. Candidate Generation

- Candidate generation is the first stage of recommendation. Given a query, the system generates a set of relevant candidates.

Candidate Generation

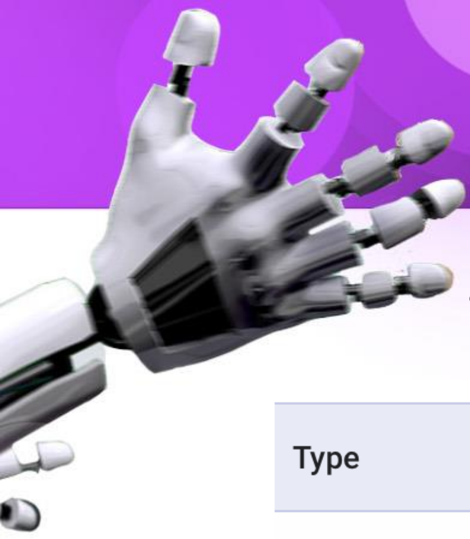
- Two common generation approaches



Type	Definition	Example
content-based filtering	Uses *similarity between items* to recommend items similar to what the user likes.	If user A watches two cute cat videos, then the system can recommend cute animal videos to that user.
collaborative filtering	Uses *similarities between queries and items simultaneously* to provide recommendations.	If user A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A (even if user A hasn't seen any videos similar to video 1).

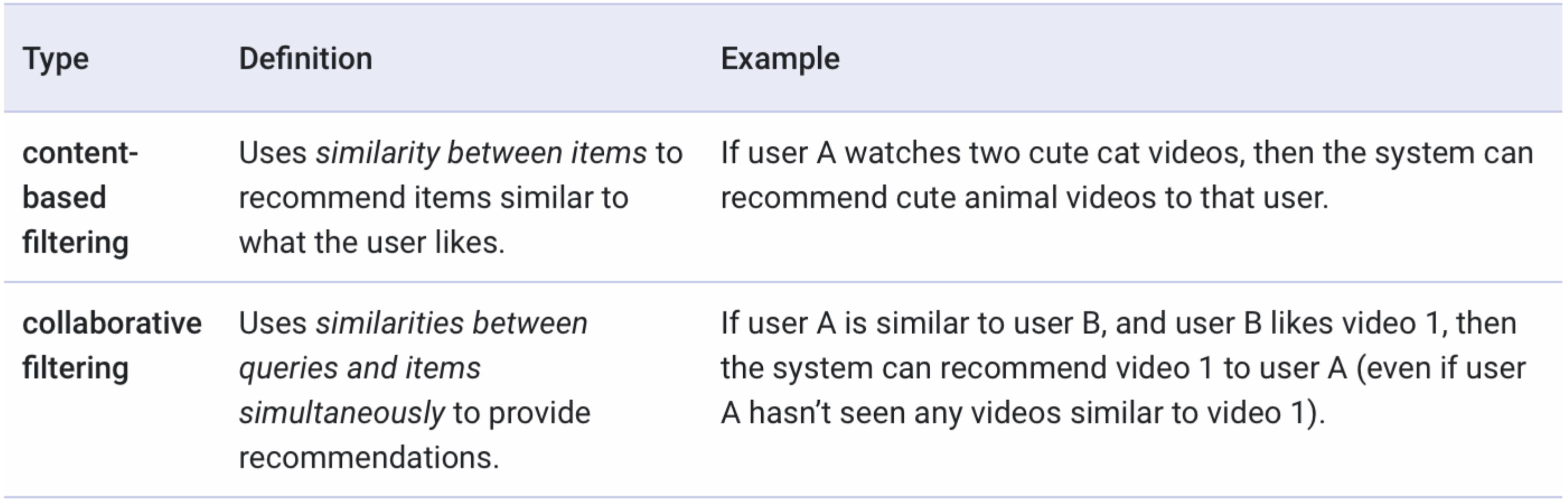
Embedding Space

- Both content-based and collaborative filtering map each item and each query (or context) to an embedding vector in a common embedding space $E = \mathbb{R}^d$.
- Typically, the embedding space is low-dimensional (that is, is much smaller than the size of the corpus), and captures some latent structure of the item or query set. Similar items, such as YouTube videos that are usually watched by the same user, end up close together in the embedding space. The notion of "closeness" is defined by a similarity measure.

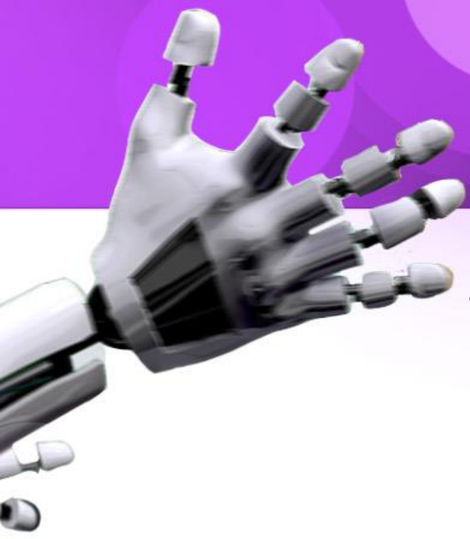


Similarity Measures

- Two common generation appraoches

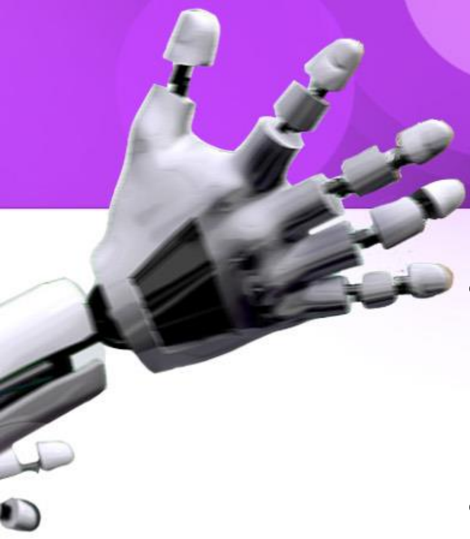


Type	Definition	Example
content-based filtering	Uses *similarity between items* to recommend items similar to what the user likes.	If user A watches two cute cat videos, then the system can recommend cute animal videos to that user.
collaborative filtering	Uses *similarities between queries and items simultaneously* to provide recommendations.	If user A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A (even if user A hasn't seen any videos similar to video 1).



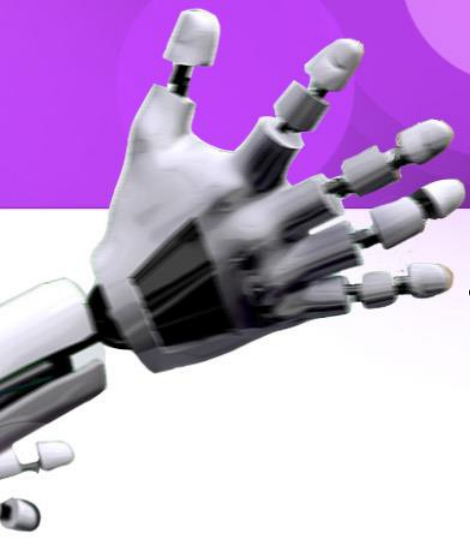
Similarity Measures

- To determine the degree of similarity, most recommendation systems rely on one or more of the following:
  - cosine
  - dot product
  - Euclidean distance



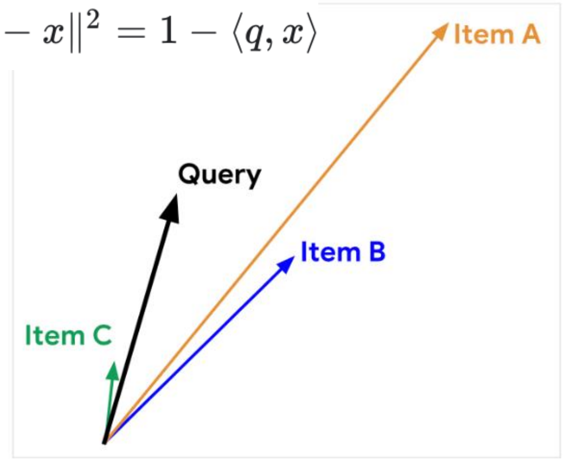
Similarity Measures

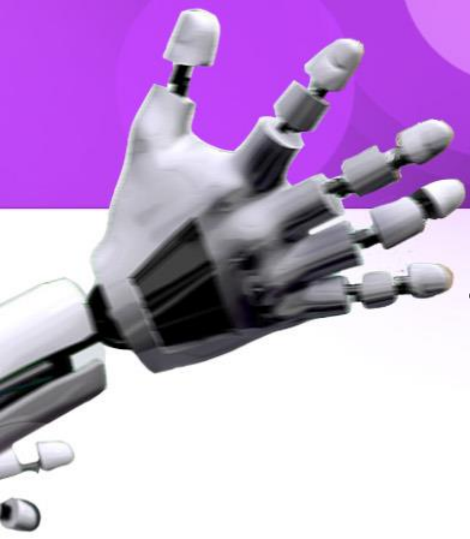
- cosine
  - This is simply the cosine of the angle between the two vectors, s(q,x) = cos(q, x)
- dot product
  - The dot product of two vectors is $s(q,x) = \langle q, x \rangle = \sum_{i=1}^{d} q_i x_i$. It is also given by $s(q,x) = \|x\| \|q\| \cos(q,x)$ (the cosine of the angle multiplied by the product of norms). Thus, if the embeddings are normalized, then dot-product and cosine coincide.



Similarity Measures

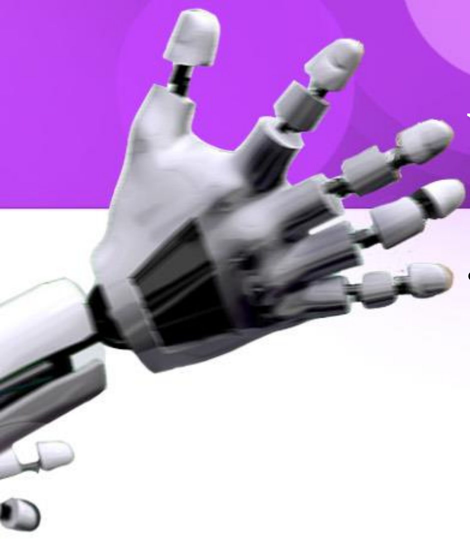
- Euclidean distance
  - This is the usual distance in Euclidean space, $s(q, x) = \|q - x\| = \left[\sum_{i=1}^{d} (q_i - x_i)^2\right]^{\frac{1}{2}}$
    A smaller distance means higher similarity. Note that when the embeddings are normalized, the squared Euclidean distance coincides with dot product (and cosine) up to a constant, since in that case $\frac{1}{2}\|q - x\|^2 = 1 - \langle q, x \rangle$





Comparing Similarity Measures

- Consider the example in the figure to the right. The black vector illustrates the query embedding. The other three embedding vectors (Item A, Item B, Item C) represent candidate items. Depending on the similarity measure used, the ranking of the items can be different.
- Using the image, try to determine the item ranking using all three of the similarity measures: cosine, dot product, and Euclidean distance.



Which Similarity Measure to Choose

- Compared to the cosine, the dot product similarity is sensitive to the norm of the embedding. That is, the larger the norm of an embedding, the higher the similarity (for items with an acute angle) and the more likely the item is to be recommended. This can affect recommendations as follows:
  - Items that appear very frequently in the training set (for example, popular YouTube videos) tend to have embeddings with large norms. If capturing popularity information is desirable, then you should prefer dot product. However, if you're not careful, the popular items may end up dominating the recommendations. In practice, you can use other variants of similarity measures that put less emphasis on the norm of the item.
  - Items that appear very rarely may not be updated frequently during training. Consequently, if they are initialized with a large norm, the system may recommend rare items over more relevant items. To avoid this problem, be careful about embedding initialization, and use appropriate regularization. We will detail this problem in the first exercise.

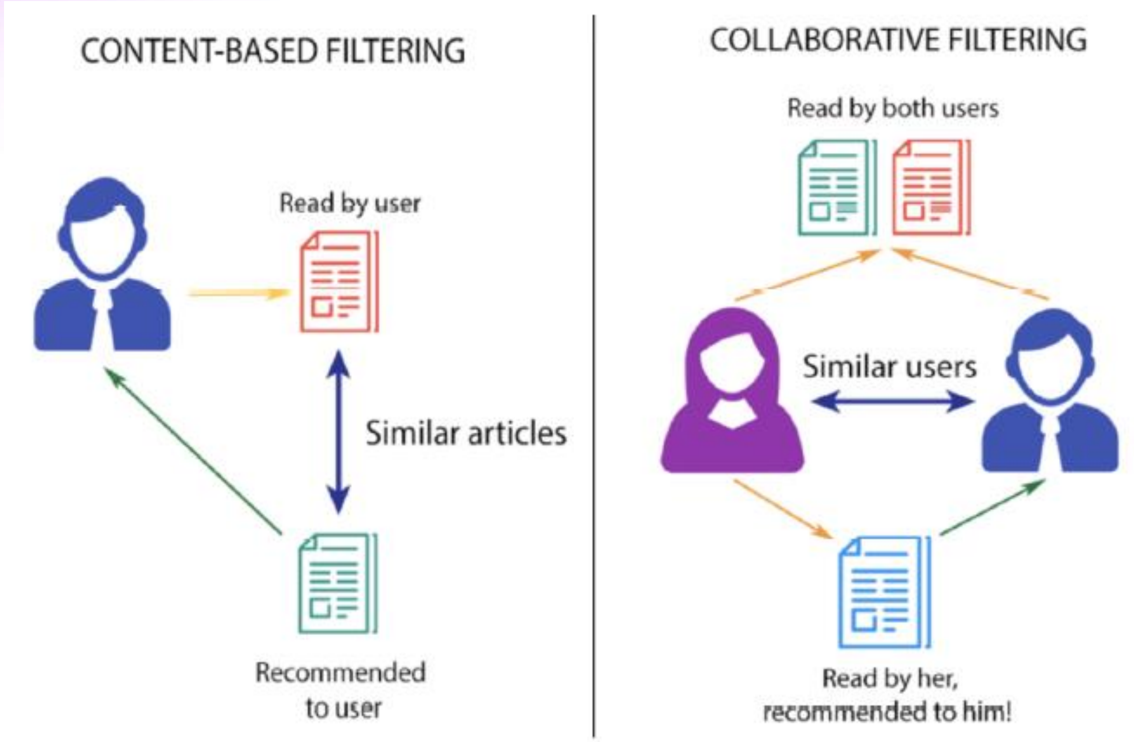
3. Types of Recommender Systems

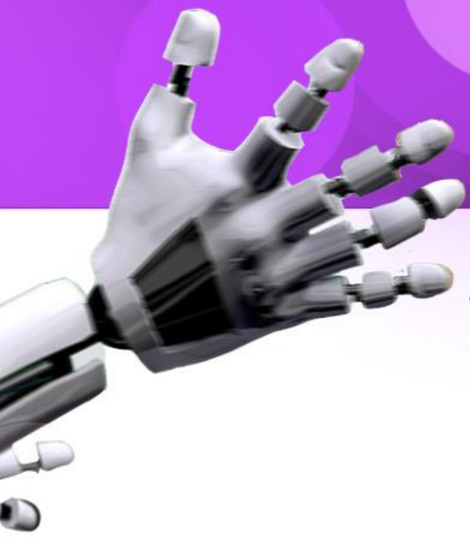
- Types of Recommender Systems
- Content-Based Filtering
- Collaborative Filtering
- Memory-Based Methods
- Model-Based Methods

Types of Recommender Systems

- There are two major approaches to build recommender systems:
  1. Content-Based Filtering
  2. Collaborative Filtering (CF)
     a. Memory-based methods (Nearest neighbor)
        i. User-based collaborative filtering
        ii. Item-based collaborative filtering
     b. Model-based methods (Matrix factorization)

Types of Recommender Systems (2)



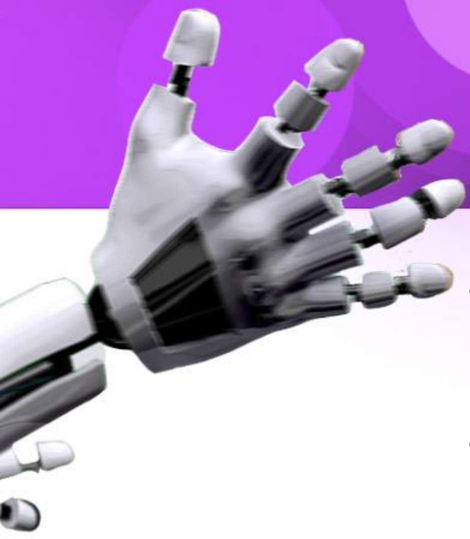


Content-Based Filtering

- In content-based filtering, the similarity between different products is calculated on the basis of the *attributes of the products*.
- For instance, in a *content-based movie recommender system*, the similarity between the movies is calculated on the basis of genres, the actors in the movie, the director of the movie, etc.

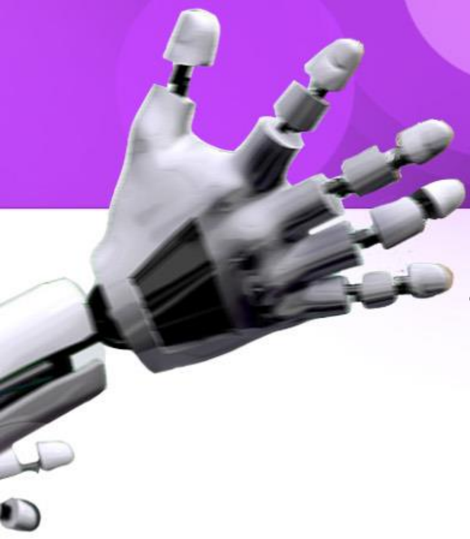
Collaborative Filtering

- *Collaborative filtering* leverages the power of the crowd.
- The intuition behind collaborative filtering is that if a user A likes products X and Y, and if another user B likes product X, there is a fair bit of chance that he will like the product Y as well.



Collaborative Filtering (2)

- Take the example of a *movie recommender system*.
- Suppose a huge number of users have assigned the same ratings to movies X and Y.
- A new user comes who has assigned the same rating to movie X but hasn't watched movie Y yet.
- Collaborative filtering system will recommend him the movie Y.



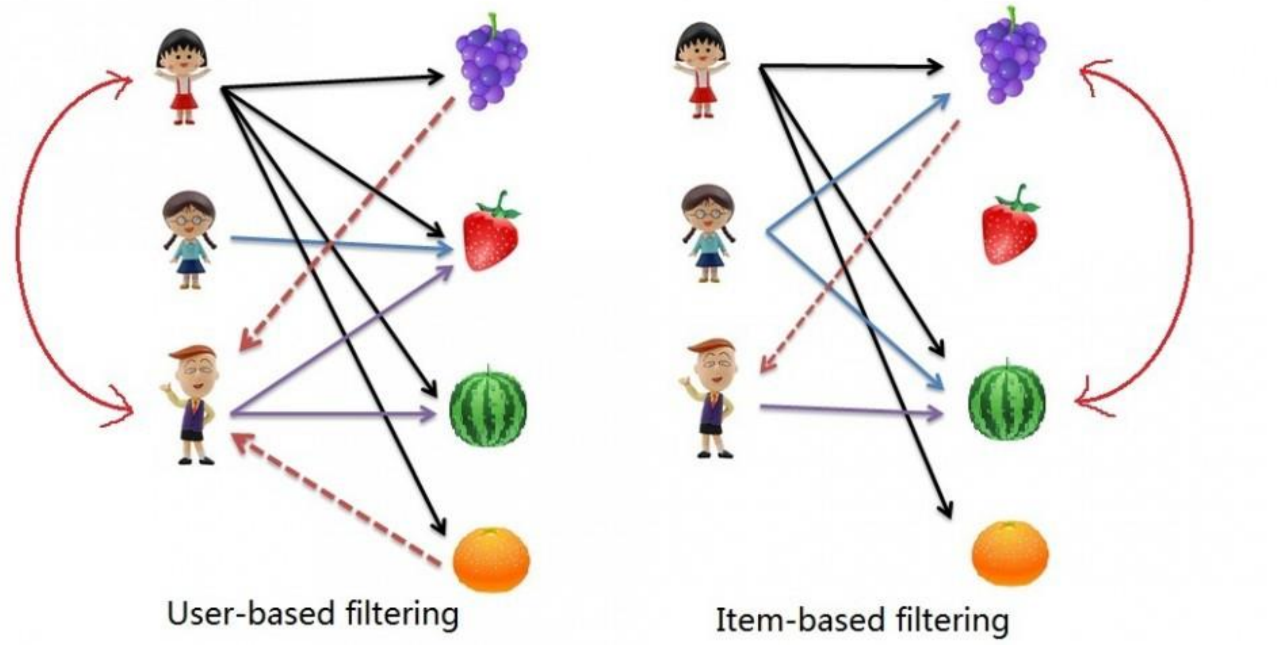
Collaborative Filtering (3)

- There are two types of collaborative models:
  1. Memory-based methods; and
  2. Model-based methods.
- The advantage of *memory-based* techniques is that they are simple to implement and the resulting recommendations are often easy to explain.

Memory-Based Methods

- *Memory-based techniques* are divided into two:
  1. User-based collaborative filtering: In this model products are recommended to a user based on the fact that the products have been liked by users similar to the user.
  2. Item-based collaborative filtering: These systems identify similar items based on users' previous ratings.

Memory-Based Methods (2)



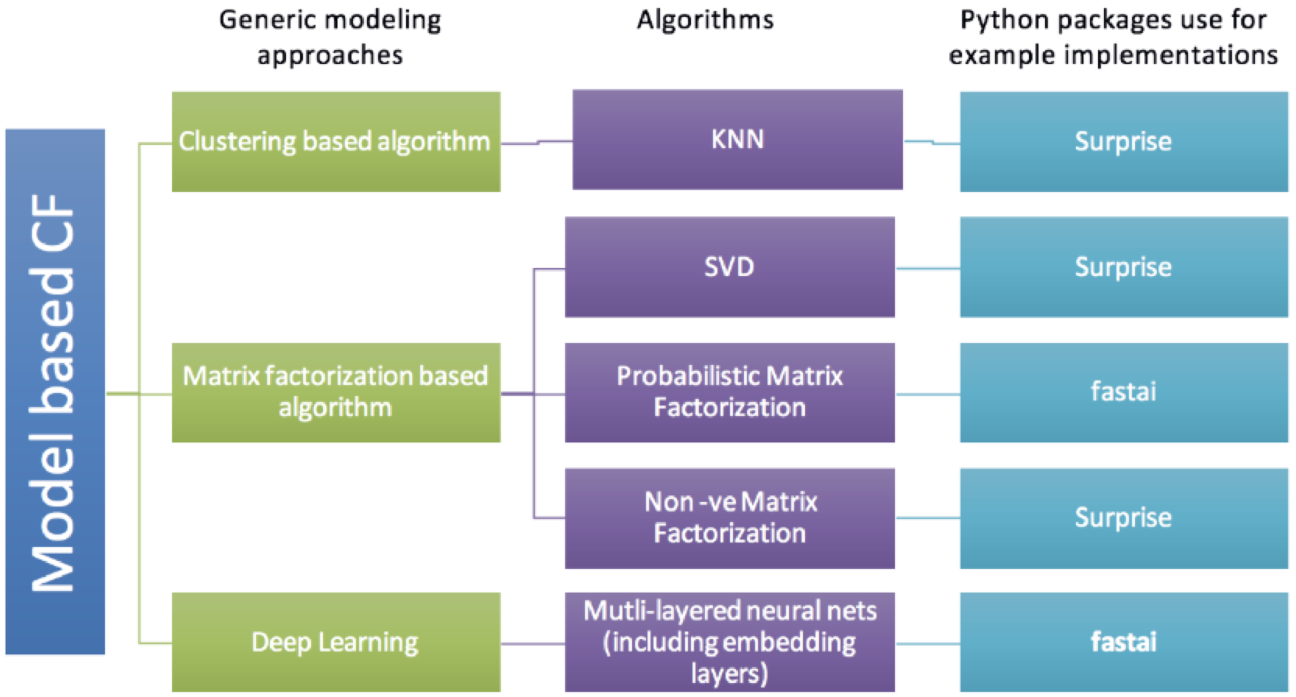
Model-Based Methods

- Model-based methods are based on *matrix factorization* and are better at dealing with sparsity.
- They are developed using data mining, machine learning algorithms to predict users' rating of unrated items.
- In this approach techniques such as *dimensionality reduction* are used to improve the accuracy.

Model-Based Methods (2)

- In this approach, CF models are developed using *machine learning algorithms* to predict a user's rating of unrated items.
- Some of these models/techniques include: k-nearest neighbors, clustering, matrix factorization, and deep learning models like *autoencoders* and using techniques like embeddings as low-dimensional hidden factors for items and users.

Model-Based Methods (3)



Generic modeling approaches
Algorithms
Python packages use for example implementations
Model based CF
Clustering based algorithm
KNN
Surprise
SVD
Surprise
Matrix factorization based algorithm
Probabilistic Matrix Factorization
fastai
Non -ve Matrix Factorization
Surprise
Deep Learning
Mutli-layered neural nets (including embedding layers)
fastai

my favorite || can add multiple layers || best accuracy || easy with fast.ai

Model-Based Methods (4)

Dimensionality Reduction

- In the *user-item matrix*, there are two dimensions:
  1. The number of users
  2. The number of items
- If the matrix is mostly empty, reducing dimensions can improve the performance of the algorithm in terms of both space and time.
- You can use various methods like *matrix factorization* or *autoencoders* to do this.

Model-Based Methods (5)

- Matrix factorization can be seen as breaking down a large matrix into *a product of smaller ones*.
- This is similar to the factorization of integers, where 12 can be written as 6×2 or 4×3.
- In the case of matrices, a matrix A with dimensions $m \times n$ can be reduced to a product of two matrices X and Y with dimensions m x p and $p \times n$ respectively.

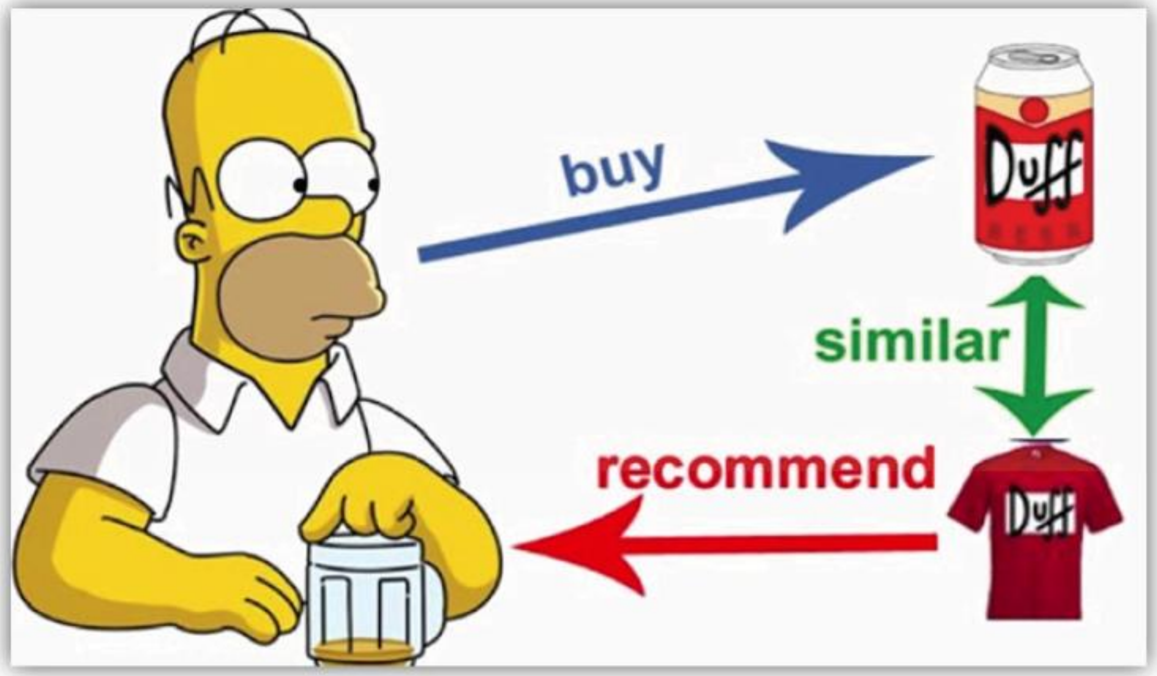
4. Content-Based Filtering

- Introduction
- How Content-Based Filtering Works?
- Implementation

Introduction

- Content-based filtering algorithms are given user preferences for items and recommend similar items based on a domain-specific notion of item content.
- This approach also extends naturally to cases where *item metadata* is available (e.g., movie stars, book authors, and music genres).

Introduction (2)



How Content-Based Filtering Works?

- A content-based recommender works with data that the user provides, either *explicitly (rating)* or *implicitly (clicking on a link)*.
- Based on that data, a user profile is generated, which is then used to make suggestions to the user.
- As the user provides more inputs or takes actions on those recommendations, the engine becomes more and more accurate.

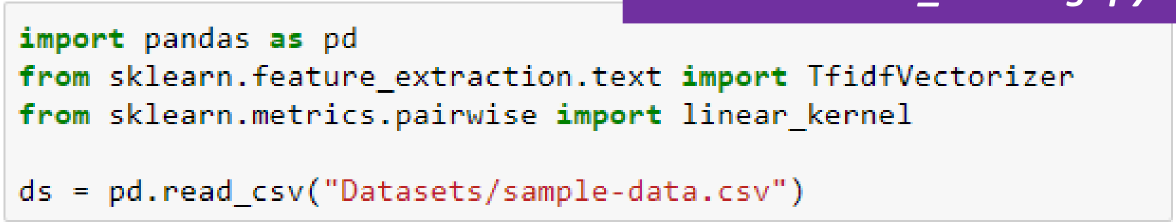
How Content-Based Filtering Works? (2)

- A recommender system has to decide between two methods for information delivery when providing the user with recommendations:
  - Exploitation. The system chooses documents similar to those for which the user has already expressed a preference.
  - Exploration. The system chooses documents where the user profile does not provide evidence to predict the user’s reaction.

Implementation

- Load the data

Content-Based_Filtering.ipynb



```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

ds = pd.read_csv("Datasets/sample-data.csv")
```

- Creating a TF-IDF Vectorizer

  The TF*IDF algorithm is used to *weigh a keyword* in any document and *assign the importance* to that keyword based on the number of times it appears in the document.

Implementation (2)

- Each word or term has its respective TF and IDF score.
- The product of the TF and IDF scores of a term is called the TF*IDF weight of that term.
- The *TF (term frequency)* of a word is the number of times it appears in a document.
- When you know it, you’re able to see if you’re using a term too often or too infrequently.

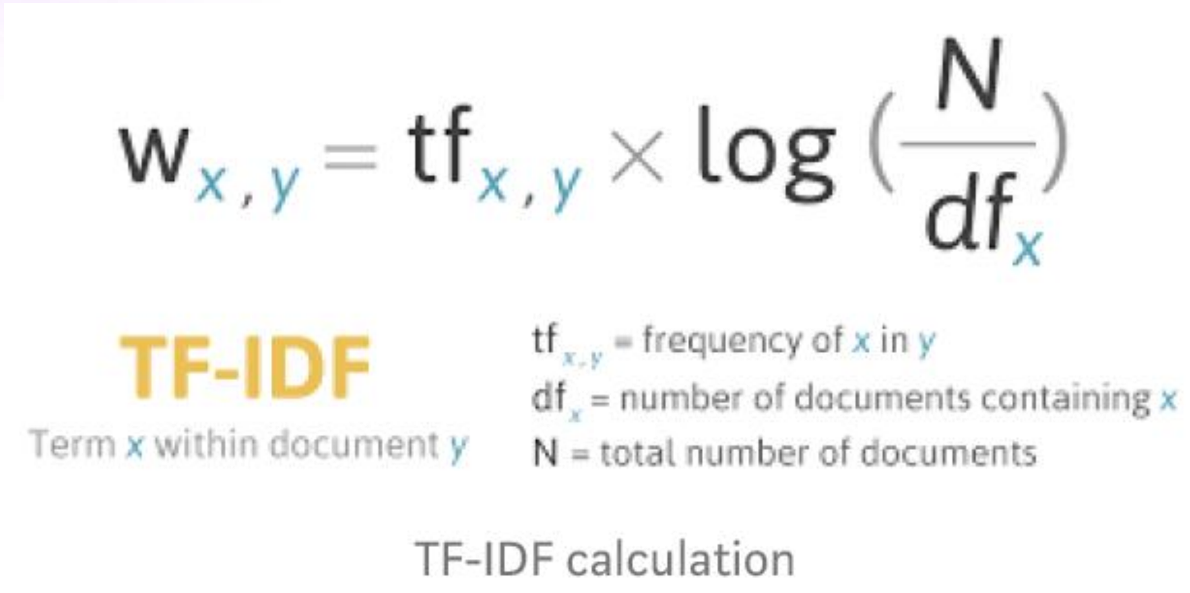
Implementation (3)

```
TF(t) = (Number of times term t appears
in a document) / (Total number of terms
in the document)
```

- The IDF (inverse document frequency) of a word is the measure of how significant that term is in the whole corpus.

```
IDF(t) = log_e(Total number of documents
/ Number of documents with term t in it)
```

Implementation (4)



$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

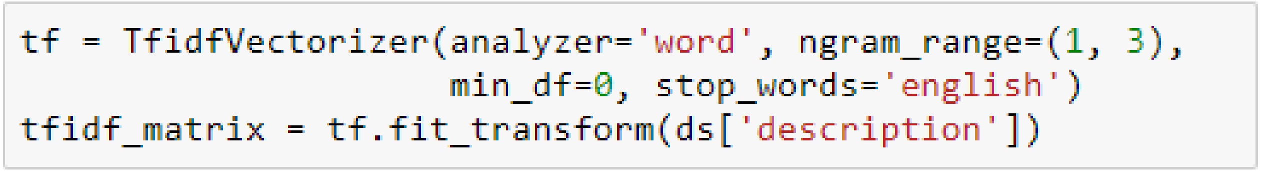
df_x = number of documents containing x

N = total number of documents

TF-IDF calculation

Implementation (5)

- In Python, scikit-learn provides you a pre-built TF-IDF vectorizer that calculates the TFIDF score for each document's description, word-by-word.



```
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3),
                     min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(ds['description'])
```

Implementation (6)

- Here, the *tfidf_matrix* is the matrix containing each word and its TF-IDF score with regard to each document, or item in this case.
- Also, *stop words* are simply words that add no significant value to our system, like ‘an’, ‘is’, ‘the’, and hence are ignored by the system.
- Now, we have a representation of every item in terms of its description.
- Next, we need to calculate the relevance or *similarity* of one document to another.

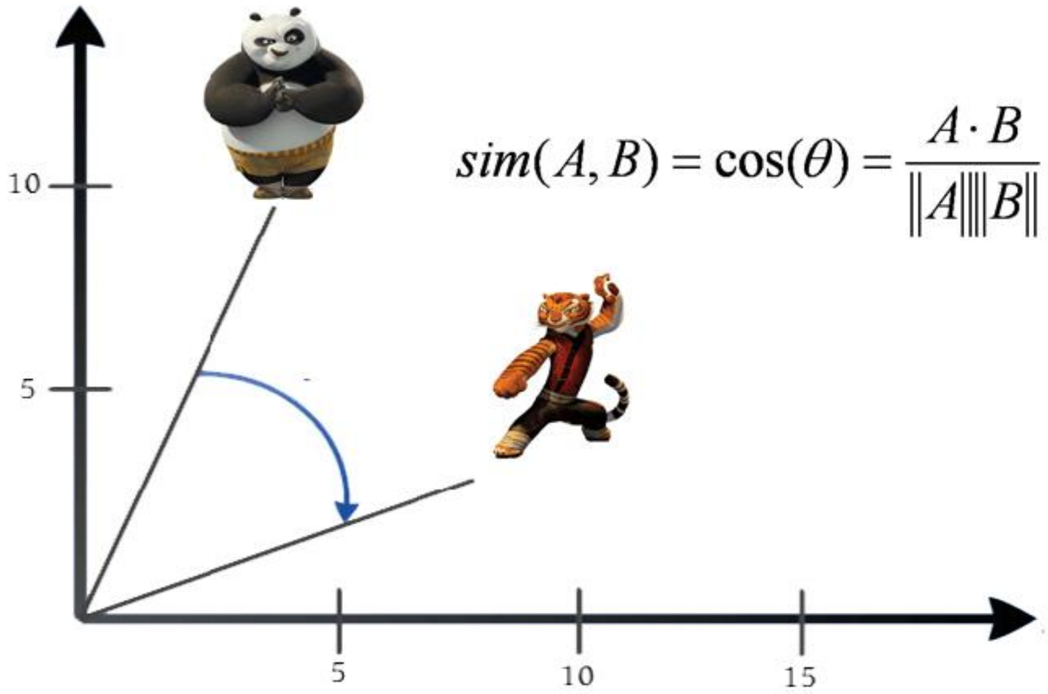
Implementation (7)

Vector Space Model

- In this model, *each item is stored as a vector* of its attributes (which are also vectors) in an n-dimensional space, and the angles between the vectors are calculated to determine the *similarity between the vectors*.
- This is *Cosine Similarity*.

Implementation (8)

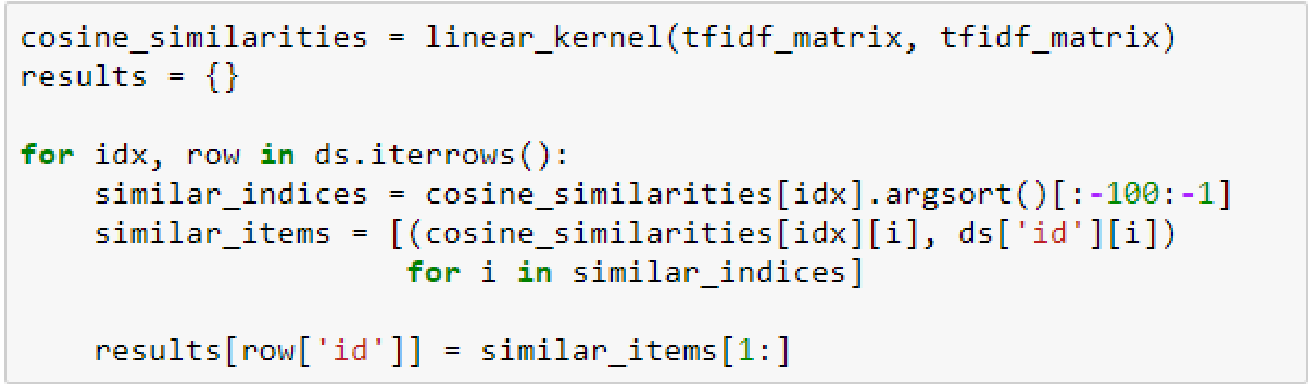
Calculating Cosine Similarity



$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Implementation (9)

- Here we've calculated the *cosine similarity* of each item with every other item in the dataset, and then arranged them according to their similarity with item i, and stored the values in *results*.



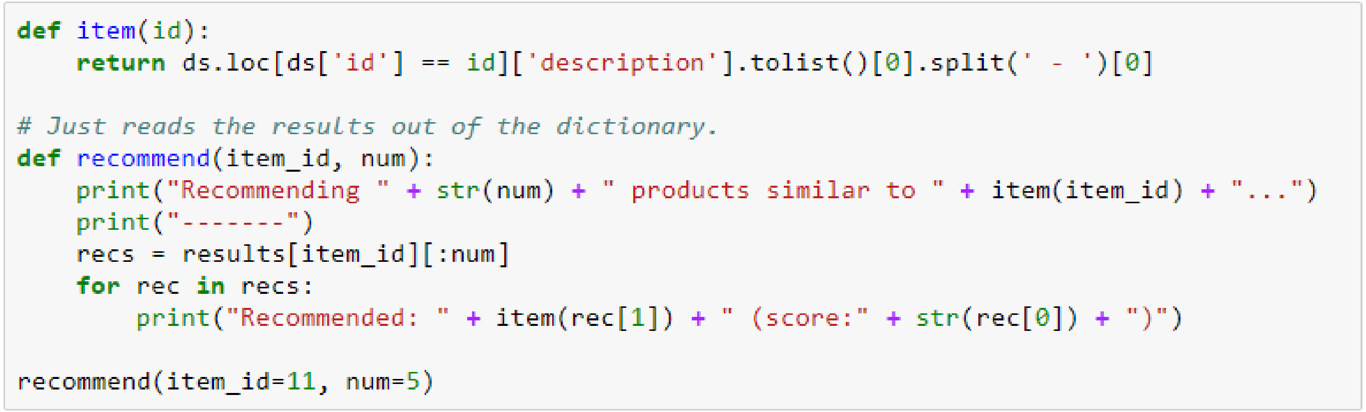
```
cosine_similarities = linear_kernel(tfidf_matrix, tfidf_matrix)
results = {}

for idx, row in ds.iterrows():
    similar_indices = cosine_similarities[idx].argsort()[:-100:-1]
    similar_items = [(cosine_similarities[idx][i], ds['id'][i])
                     for i in similar_indices]

    results[row['id']] = similar_items[1:]
```

Implementation (10)

Make a Recommendation



```
def item(id):
    return ds.loc[ds['id'] == id]['description'].tolist()[0].split(' - ')[0]

# Just reads the results out of the dictionary.
def recommend(item_id, num):
    print("Recommending " + str(num) + " products similar to " + item(item_id) + "...")
    print("-------")
    recs = results[item_id][:num]
    for rec in recs:
        print("Recommended: " + item(rec[1]) + " (score:" + str(rec[0]) + ")")

recommend(item_id=11, num=5)
```

```
Recommending 5 products similar to Baby sunshade top...
-------
Recommended: Sunshade hoody (score:0.2133029602108501)
Recommended: Baby baggies apron dress (score:0.10975311296284813)
Recommended: Runshade t-shirt (score:0.09988151262780706)
Recommended: Runshade t-shirt (score:0.09530698241688194)
Recommended: Runshade top (score:0.08510550093018401)
```

5. Collaborative Filtering

5.1 User-Based Collaborative Filtering

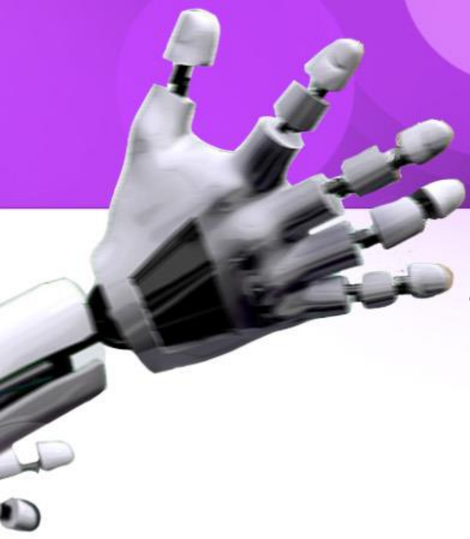
5.2 Item-Based Collaborative Filtering

5.1 User-Based Collaborative Filtering

- User-Based Collaborative Filtering (UB-CF)
- Implement UB-CF with KNN

User-Based Collaborative Filtering

- Imagine that we want to recommend a movie to our friend Stanley.
- We could assume that *similar people will have similar taste*.
- Suppose that me and Stanley have seen the same movies, and we rated them all almost identically.
- But Stanley hasn’t seen ‘The Godfather: Part II’ and I did. If I love that movie, it sounds logical to think that he will too.
- With that, we have created an artificial rating based on our similarity.



User-Based Collaborative Filtering (2)

- UB-CF uses that logic and recommends items by *finding similar users* to the active user (to whom we are trying to recommend a movie).
- A specific application of this is the user-based Nearest Neighbor algorithm.
- This algorithm needs two tasks.

User-Based Collaborative Filtering (3)

1. Find the K-nearest neighbors (KNN) to the user a, using a *similarity function w* to measure the distance between each pair of users:

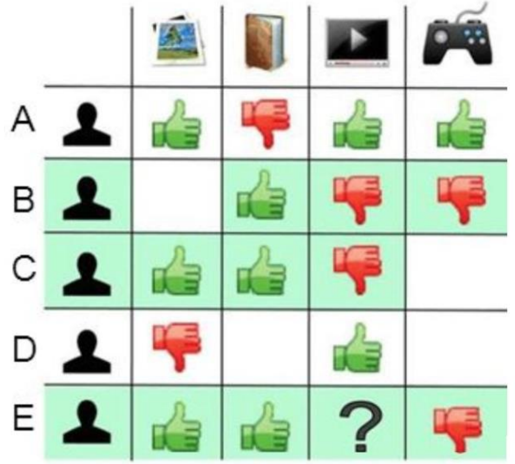
$$Similarity(a, i) = w(a, i), i \in K$$

2. Predict the rating that user *a* will give to all items the *k neighbors* have consumed but *a* has not.

   We Look for the item *j* with the best predicted rating.

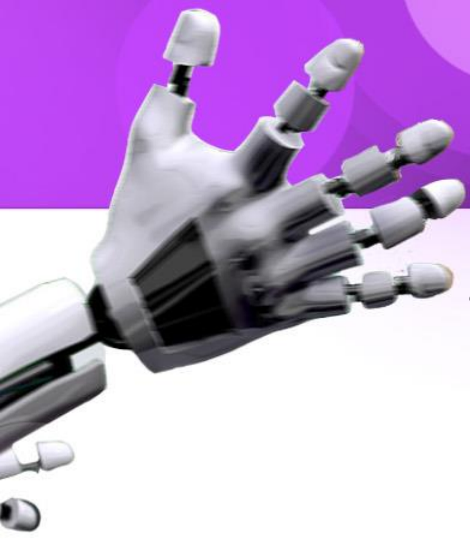
User-Based Collaborative Filtering (4)

- In other words, we are creating a *User-Item Matrix*, predicting the ratings on items the active user has not see, based on the *other similar users*.
- This technique is *memory-based*.



		🖼	📖	▶	🎮
A	👤	👍	👎	👍	👍
B	👤		👍	👎	👎
C	👤	👍	👍	👎	
D	👤	👎		👍	
E	👤	👍	👍	?	👎

User-Item Matrix



Implement UB-CF with KNN

- Here is an example of implementing UB-CF with *K-Nearest Neighbors* algorithm:

User-Based_Filtering.ipynb

5.2 Item-Based Collaborative Filtering

- Item-Based Collaborative Filtering (IB-CF)
- Implement IB-CF Using KNN

Item-Based Collaborative Filtering

- Instead of focusing on his friends, we could focus on what *items* from all the options are *more similar* to what we know he enjoys.
- This new focus is known as Item-Based Collaborative Filtering (IB-CF).

Item-Based Collaborative Filtering (2)

- We could divide IB-CF in two sub tasks:
  1. *Calculate similarity* among the items:
     - Cosine-Based Similarity
     - Correlation-Based Similarity
     - Adjusted Cosine Similarity
     - 1-Jaccard distance
  2. *Calculation of Prediction*:
     - Weighted Sum
     - Regression

Item-Based Collaborative Filtering (3)

- The difference between UB-CF and this method is that, in this case, we directly pre-calculate the similarity between the co-rated items, skipping K-neighborhood search.

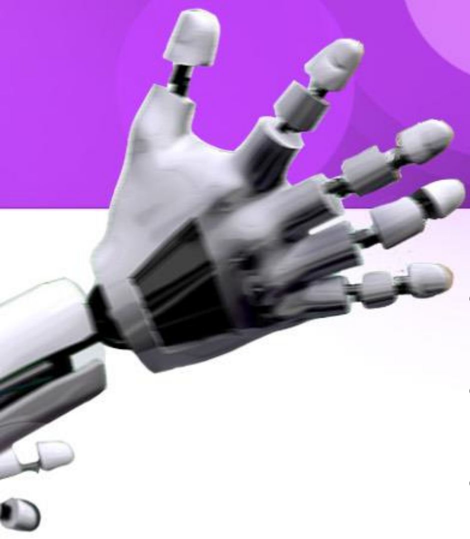
Implement IB-CF with KNN

- Here is an example of implementing IB-CF with *K-Nearest Neighbors* algorithm:

Item-Based_Filtering.ipynb

6. Retrieval, Scoring, and Re-ranking

- How would you decide which items to recommend?

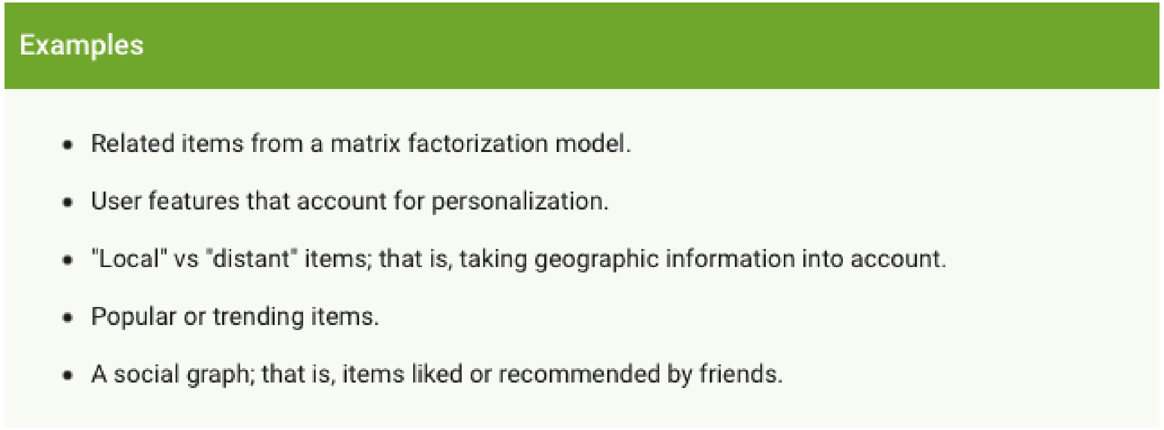


Retrieval

- [TODO]
- Make Effective Presentations
- Using Awesome Backgrounds
- Engage your Audience
- Capture Audience Attention

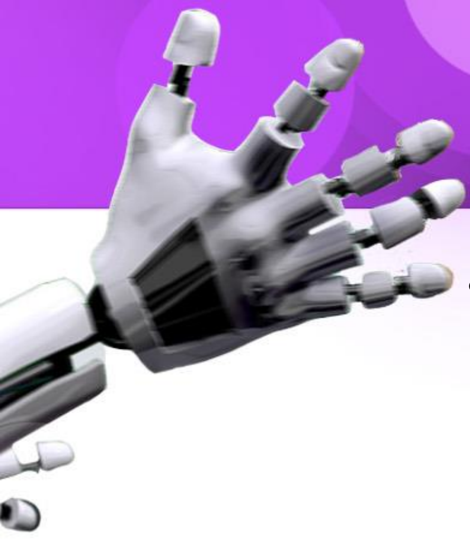
Scoring

- After candidate generation, another model scores and ranks the generated candidates to select the set of item to display



Examples

- Related items from a matrix factorization model.
- User features that account for personalization.
- "Local" vs "distant" items; that is, taking geographic information into account.
- Popular or trending items.
- A social graph; that is, items liked or recommended by friends.



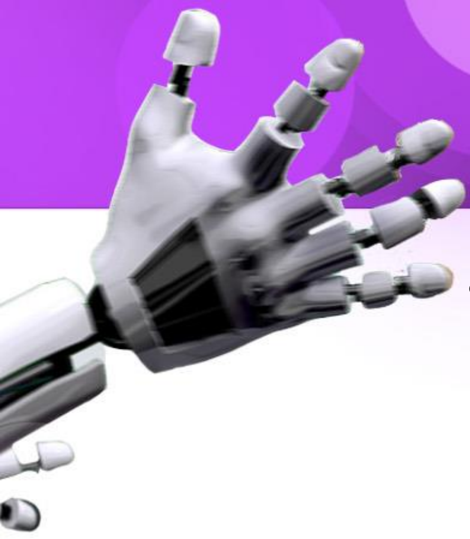
Scoring

- The system combines different sources into a common pool of candidates that are then scored by a single model and ranked according to that score. For example, the system can train a model to predict the probability of a user watching a video on YouTube given the following:
  - query features (for example, user watch history, language, country, time)
  - video features (for example, title, tags, video embedding)
- The system can then rank the videos in the pool of candidates according to the prediction of the model.

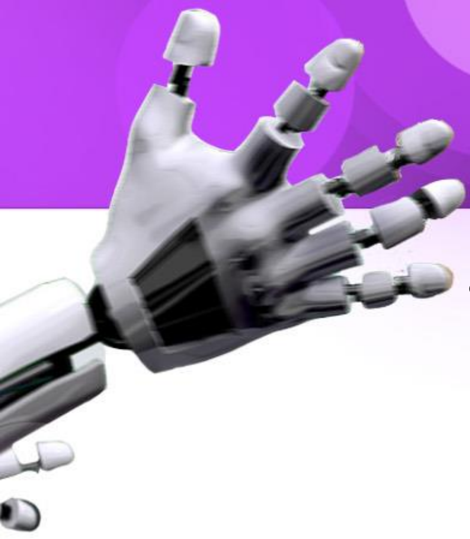
Re-ranking

- In the final stage of a recommendation system, the system can re-rank the candidates to consider additional criteria or constraints. One re-ranking approach is to use filters that remove some candidates.
- Another re-ranking approach is to manually transform the score returned by the ranker.
- **Freshness, diversity, and fairness** can help improve the recommendation system.

Re-ranking – Freshness

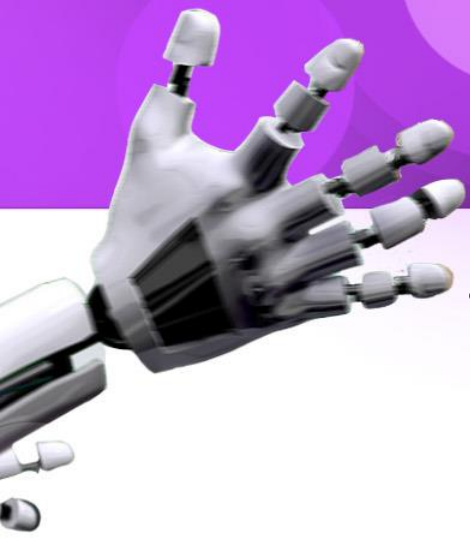


- Most recommendation systems aim to incorporate the latest usage information. Keeping the model fresh helps the model make good recommendations.
  - Re-run training as often as possible to learn on the latest training data.
  - Create an “average” user to represent new users in matrix factorization models.
  - Use a DNN such as a softmax model
  - Add document age as a feature



Re-ranking – Diversity

- If the system always recommend items that are "closest" to the query embedding, the candidates tend to be very similar to each other. This lack of diversity can cause a bad or boring user experience.
  - Train multiple candidate generators using different sources
  - Train multiple rankers using different objective functions
  - Re-rank items based on other metadata to ensure diversity



Re-ranking – Fairness

- The model should treat all users fairly.
  - Include diverse perspectives in design and development
  - Train ML models on comprehensive data sets Add auxiliary data when the data is too sparse
  - Track metrics on each demographic to watch for biases
  - Make separate models for underserved groups

Conclusion

- Purposes of recommender systems
- Types of recommender systems
- Components of a recommender system
- Use embeddings to represent items and queries
- Common techniques used in candidate generations

Self Study Guide

Self Study Guide

References

- ✓ Coelho, L. P. & Richert. W. (2015). *Building Machine Learning Systems with Python* (2nd ed.). Birmingham: Packt Publishing. (Ch. 8)
- ✓ Julian, D. (2016). *Designing Machine Learning Systems with Python: Design efficient machine learning systems that give you more accurate results* (1st ed.). Birmingham: Packt Publishing. (Ch. 9)

Useful Sites

- ✓ https://developers.google.com/machine-learning/recommendation