Building SMART Recommendation Systems in Python (Using Intellikit)

Arthur Kakande, MSc.
AI & Intelligent Systems
https://github.com/ArthurKakande/recommender_systems

Data Fest Africa 2024 - Nairobi

POLLICY

slides

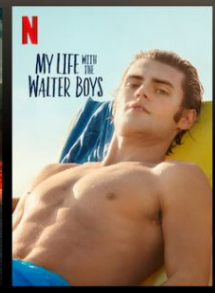
Outline

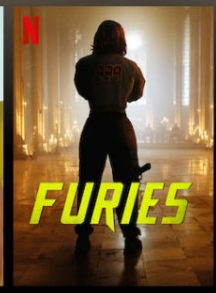
- What? Why? Who?
- Collaborative Filtering
- Content based Approach
- Knowledge based
- Intellikit Library
- Coding Demo

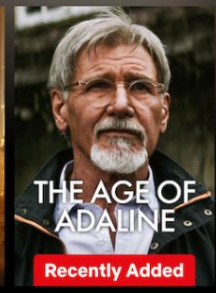
Netflix

NETFLIX

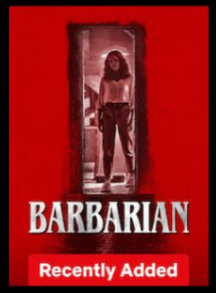
Your Next Watch

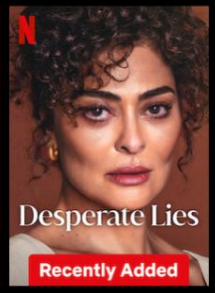


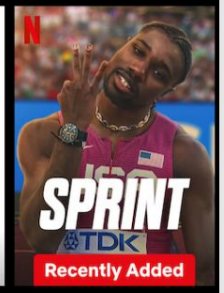




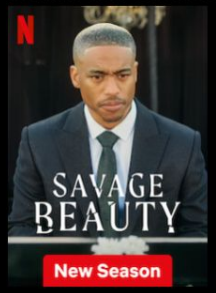
New on Netflix



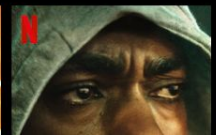


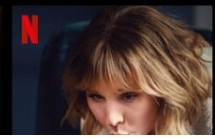


We Think You'll Love These



My List





youtube.com

YouTube DE

Search

Home
Shorts
Subscriptions

You

History
Playlists
Watch later
Liked videos

Subscriptions

Natnael Lecture Hub
freeCodeCamp.o...
Open Geospatial ...
Publications Office ...
20th Century Studios
Giraffe Academy
Cat Burns
Show more

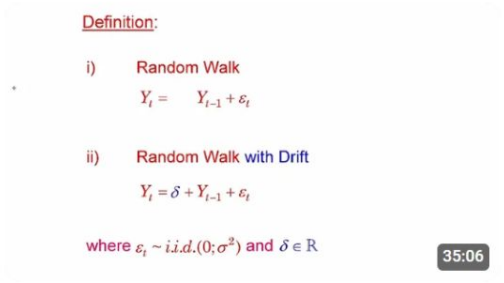
Explore

All | Trailers | Gaming | Music | Smosh | House Targaryen | News | Mixes | Sketch comedy | APIs | Databases | Acrobatics | Li

Gkbarry on Turning down the Sidemen, Secrets of Brand Trips and THAT TikTok...
The Useless Hotline Podcast
14K views • 7 hours ago

SIDEMEN ULTIMATE HIDE & SEEK ON AN ISLAND vs 40 YOUTUBERS
Sidemen
195K views • 34 minutes ago

Guessing Movies From One Frame
Smosh Games
509K views • 4 days ago



The Random Walk | Time Series Lecture 18
cserbal
29 views • 2 days ago

Das Excelseminar für Studenten
2h Seminar + Excel-Zertifikat = gewappnet für Studium und Beruf!
Sponsored · Hochschulinitiative

Get Ready For The New Labour Government | The Russell Howard Hour...
Russell Howard
294K views • 2 days ago

beforward.jp

We use cookies to improve your experience on our website. By continuing to use our website, you agree to our use of cookies. Accept

Shop By Make

- TOYOTA (62,731)
- NISSAN (32,265)
- HONDA (28,043)
- MAZDA (10,527)
- MITSUBISHI (10,359)
- SUBARU (9,328)
- SUZUKI (33,595)
- ISUZU (4,067)
- DAIHATSU (28,432)
- HINO (2,423)
- LEXUS (5,321)
- MERCEDES-BENZ (16,423)
- BMW (16,522)
- VOLKSWAGEN (4,388)
- AUDI (5,388)
- PEUGEOT (995)
- FORD (2,050)
- VOLVO (2,572)
- LAND ROVER (6,890)

SEARCH FOR CARS

Deals TODAY'S OFFER !! prime seller

Make Select | Model Seleccione | Price Select

Type Select | Steering Select | Year Select ~ Select

SEARCH

401,402 items match

Use BUY NOW to Get Special Perks!

Check Out Our New Videos!

How To Buy

Today Special Offer

Earn up to $300pts with Every Purchase

Vehicles with Points Reward

TOYOTA
LAND CRUISER
PRADO
Vehicle Price:
$32,480

TOYOTA
HILUX
Vehicle Price:
$17,620

TOYOTA
HILUX
Vehicle Price:
$15,190

TOYOTA
HILUX
Vehicle Price:
$15,790

TOYOTA
HILUX
Vehicle Price:
$12,560


TOYOTA
HILUX
Vehicle Price:
$11,330

NOT REGISTERED YET?
It only takes 10 seconds!

see more ...

New Arrivals | Clearance | Premium Class | 3rd Party Seller

View vehicles **shipping from:**

All | Japan | Singapore | UK | UAE | Thailand | Korea | China


NISSAN
CONDOR


DAIHATSU
HIJET TRUCK


MERCEDES-BENZ
B-CLASS


LAND ROVER
RANGE ROVER


MERCEDES-BENZ
G-CLASS


TOYOTA
PROBOX VAN

Create account

Sign up & enjoy these features

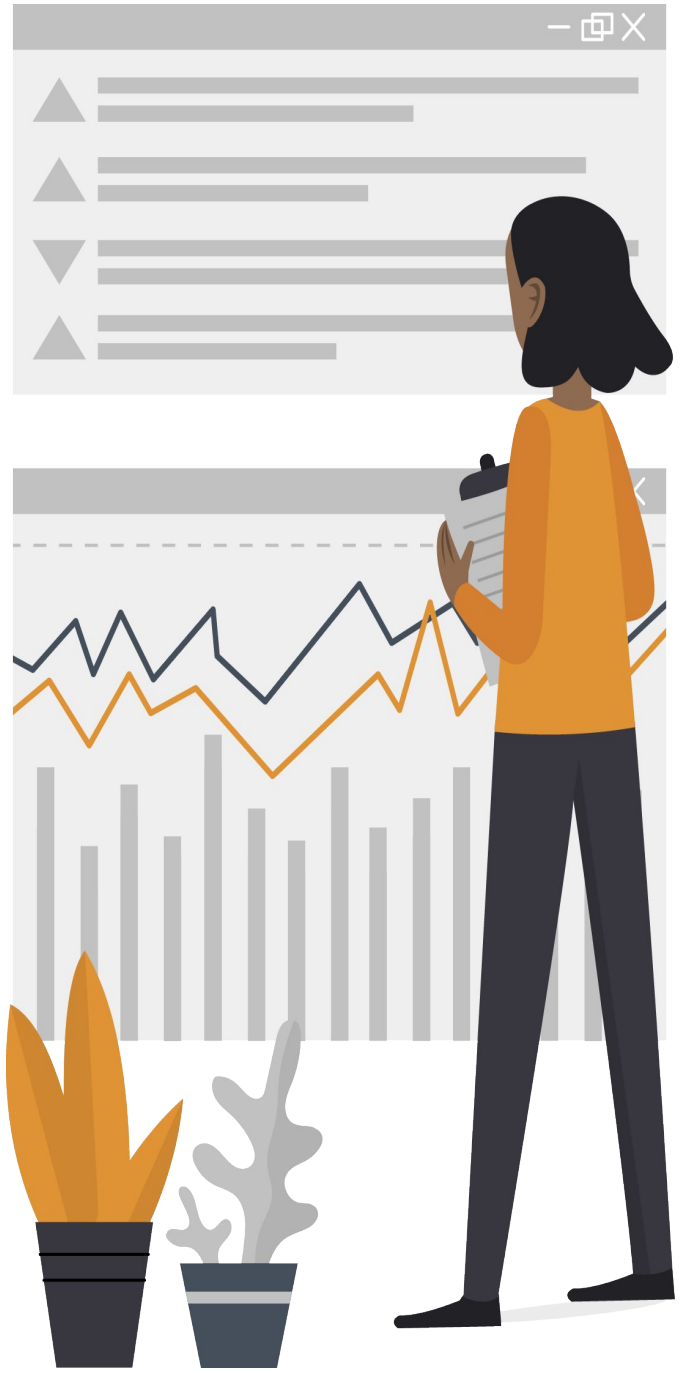
Earn Points | Favorites | Notify Me

Save Search | Easy Inquiry | Buy Now

CREATE ACCOUNT

Local Services
Getting Closer To You

- BF Tanzania
- BF Zambia
- BF Malawi
- BF Mozambique



What?

Recommender systems are software tools and techniques providing suggestions for items to be of use to a user.

POLLICY

Why?

- Users rely on recommendations
- Users lack sufficient personal expertise
- Number of items is very large e.g. around 1010 books in Amazon
- Recommendations need to be personalized

POLLICY

Why?

- Sell more diverse items
- Increase user satisfaction (Users enjoy the recommendations)
- Increase user fidelity (Users feel recognized (but not creeped out))
- Understand users

POLLICY

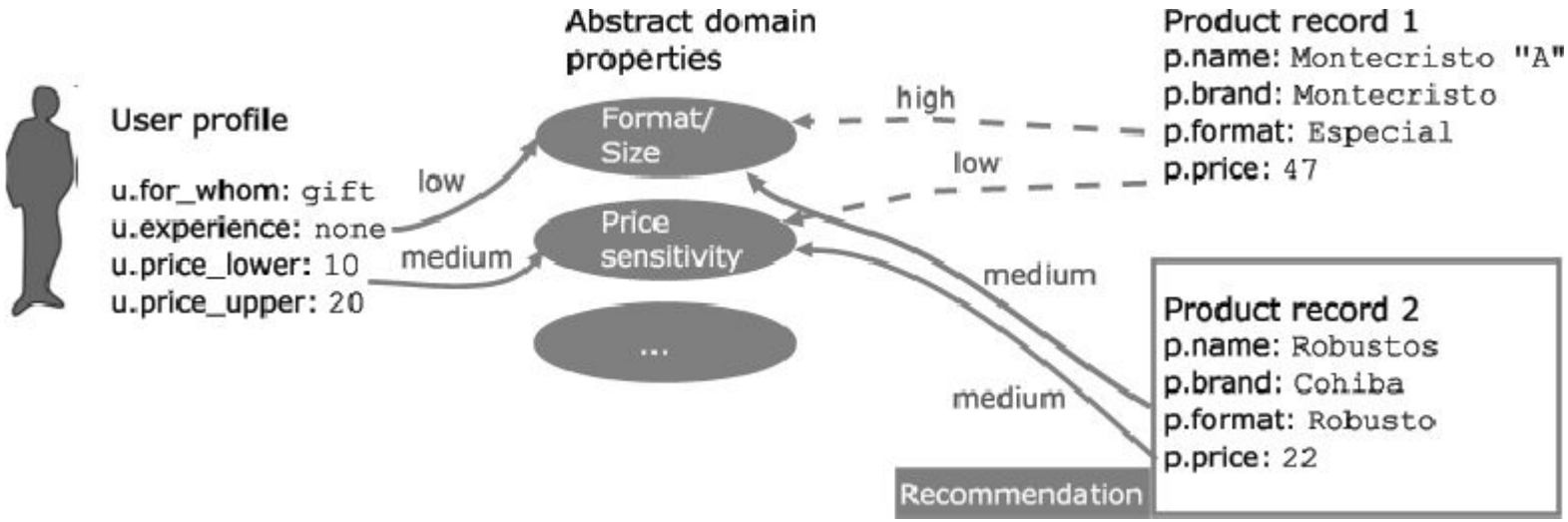
Who?

- Retailers and e-commerce in general
- Amazon, Netflix, etc.
- Service sites, e.g. travel sites
- Media organizations
- Dating apps

POLLICY

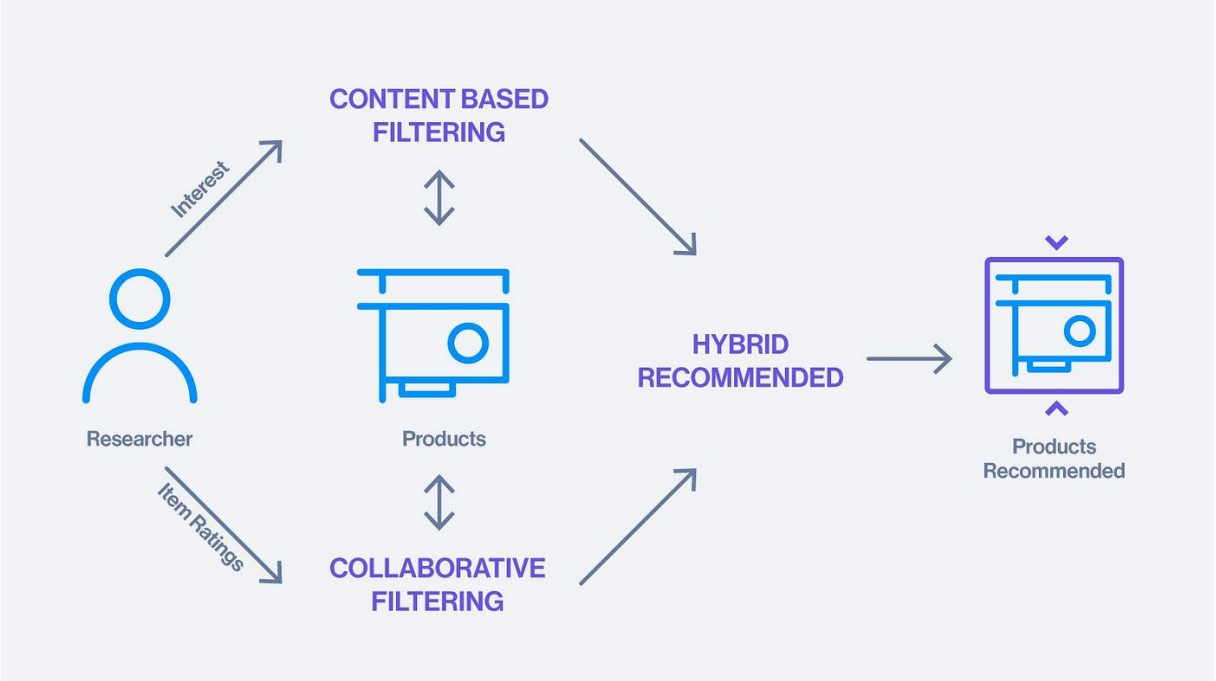
The recommender system problem

Estimate the utility for a user of an item for which the user has not expressed utility



Approaches

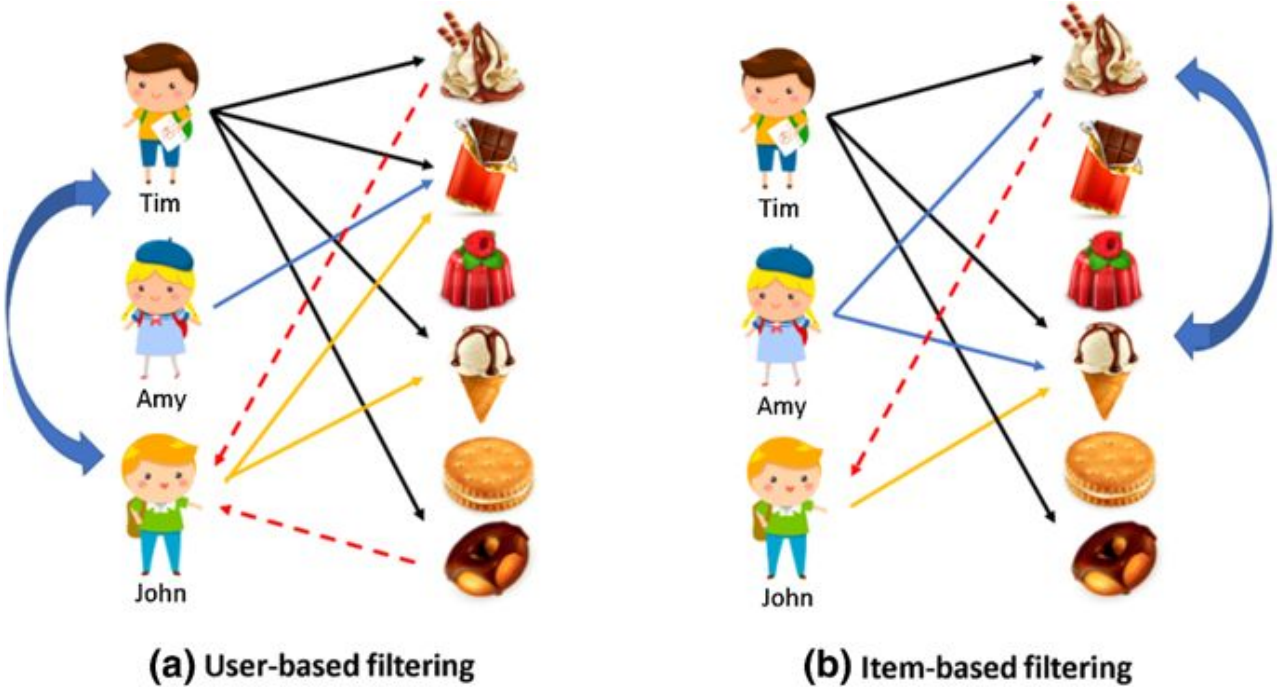
- Collaborative filtering
- Content-based (item features)
- Knowledge-based (expert system)
- Hybrid
- Others



Collaborative Filtering

How does collaborative filtering work?

- User has seen/liked certain items
- Community has seen/liked certain items
- Recommend to users items similar to the ones they have seen/liked
- Based on finding similar users
- Based on finding similar items



(a) User-based filtering

(b) Item-based filtering

Ratings data

ITEMS 

USERS 

	1	2	3	4	5	6	7	8	9	10	11	12
1	1			5					2		3	
2	5		1		5		4			5	3	2
3		5	5			1		3			2	4
4		3			5	4				5		1
5				2	5	4			4	5		
6	4		3			1			4		2	
7		4	1		4		5				4	1

ITEMS

USERS

	1	2	3	4	5	6	7	8	9	10	11	12
1	1			5					2		3	
2	5		1		5		4			5	3	2
3		5	5			1		3			2	4
4		3			5	4				5		1
5				2	5	4			4	5		
6	4		3			1			4		2	
7		4	1		4		5				4	1

$$\text{dist}(i,j) = \frac{\sum_{\alpha \in U_i \cap U_j} |r_i(\alpha) - r_j(\alpha)|}{|U_i \cap U_j|}$$

Compute user intersection size

ITEMS

USERS

	1	2	3	4	5	6	7	8	9	10	11	12	$\lvert U_i \cap U_j \rvert$	dis(i,j)
1	1			5					2		3		0	
2	5		1		5		4			5	3	2	3	
3		5	5			1		3			2	4	3	
4		3			5	4				5		1		
5				2	5	4			4	5			3	
6	4		3			1			4		2		1	
7		4	1		4		5				4	1	3	

Compute user distance

ITEMS

USERS

	1	2	3	4	5	6	7	8	9	10	11	12
1	1			5					2		3	
2	5		1		5		4			5	3	2
3		5	5			1		3			2	4
4		3			5	4				5		1
5				2	5	4			4	5		
6	4		3			1			4		2	
7		4	1		4		5				4	1

$\lvert U_i \cap U_j \rvert$	$\mathbf{dis}(i,j)$
0	
3	0.33
3	2.67
3	0.00
1	3.00
3	0.67

Pick top-3 most similar

ITEMS

USERS

	1	2	3	4	5	6	7	8	9	10	11	12	$\lvert U_i \cap U_j \rvert$	dis (i, j)
2	5		1		5		4			5	3	2	3	0.33
4		3			5	4				5		1		
5				2	5	4			4	5			3	0.00
7		4	1		4		5				4	1	3	0.67

Estimate unrated items

ITEMS

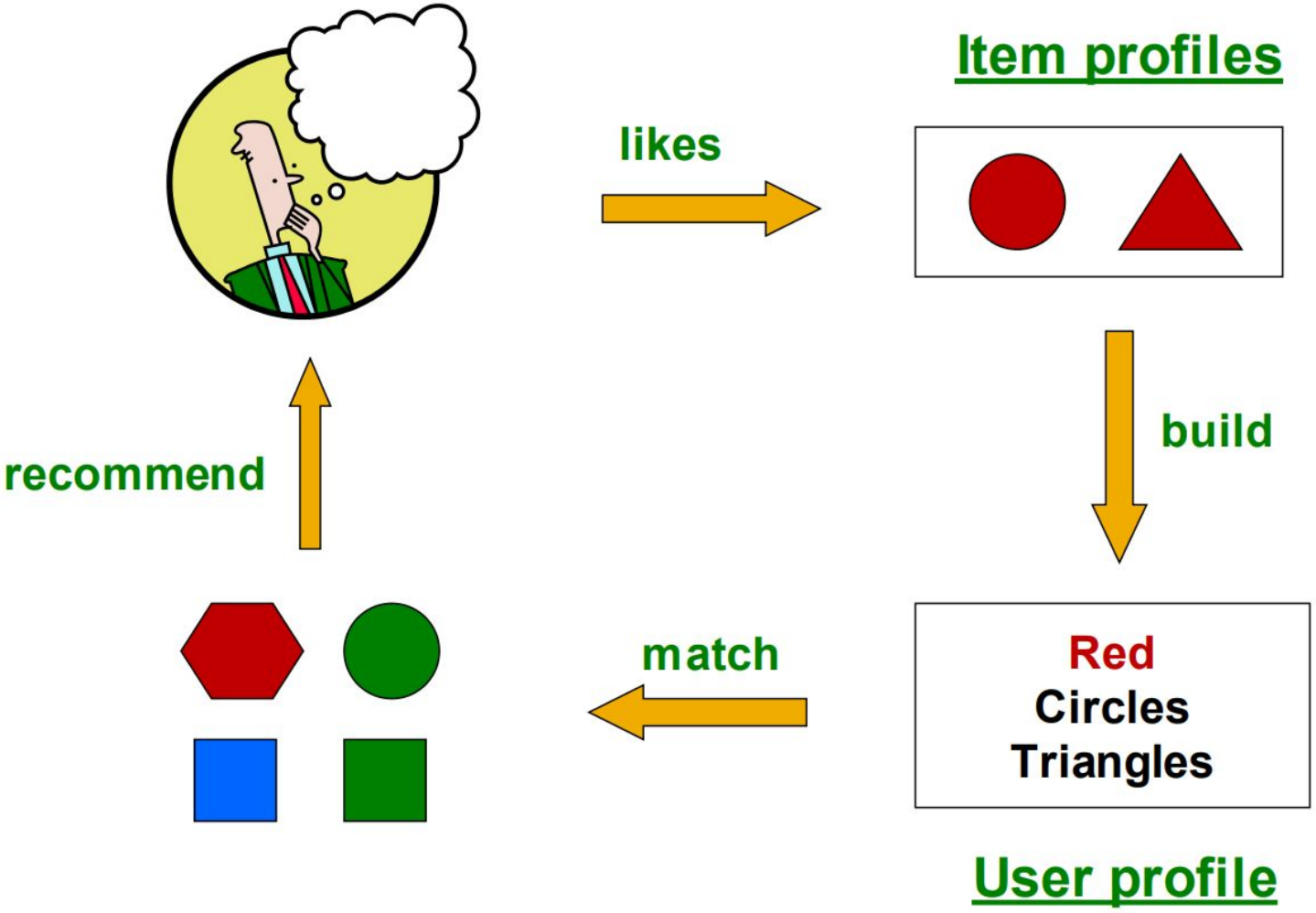
USERS

	1	2	3	4	5	6	7	8	9	10	11	12
2	5		1		5		4			5	3	2
4	5.0	3	1.0	2.0	5	4	4.5	-	4.0	5	3.5	1
5				2	5	4			4	5		
7		4	1		4		5				4	1

$\lvert U_i \cap U_j \rvert$	dis (i, j)
3	0.33
3	0.00
3	0.67

Content-Based Approach

Plan of action



How does the content-based approach work?

For each item, we create an item profile. A profile is a set (vector) of features. e.g. Movies: author, title, actor, director,... Text/Blogs: Set of “important” words in document

We can pick important features using; Usual heuristic from text mining is TF-IDF (Term frequency * Inverse Doc Frequency)

- Term ... Feature
- Document ... Item

	Melissa McCarthy	Actor A	Actor B	...	Johnny Depp	Comic Genre	Spy Genre	Pirate Genre
Movie X	0	1	1	0	1	1	0	1
Movie Y	1	1	0	1	0	1	1	0

Cosine Similarity

Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Cosine Similarity Example

	idf	Doc1			Doc2			Doc3		
Term		tf	tf•idf	tf•idf (norm.)	tf	tf•idf	tf•idf (norm.)	tf	tf•idf	tf•idf (norm.)
car	1,65	27,00	44,55	0,8974	4,00	6,60	0,0756	24,00	39,60	0,5953
auto	2,08	3,00	6,24	0,1257	33,00	68,64	0,7867	0,00	0,00	0,0000
insurance	1,62	0,00	0,00	0,0000	33,00	53,46	0,6127	29,00	46,98	0,7062
best	1,5	14,00	21,00	0,4230	0,00	0,00	0,0000	17,00	25,50	0,3833
Eu.-Length			49,65	1,0000		87,25	1,0000		66,52	1,0000

The pairwise cosine similarities of the documents are:

sim(Doc1,Doc2)

= 0.8974·0.0756 + 0.1257·0.7867 + 0.0000·0.6127 + 0.4230·0.0000

= 01668

sim(Doc1,Doc3)

= 0.8974·0.5953 + 0.1257·0.0000 + 0.0000·0.7062 + 0.4230·0.3833

= 0.6963

sim(Doc2,Doc3)

= 0.0756·0.5953 + 0.7867·0.0000 + 0.6127·0.7062 + 0.0000·0.3833

= 0.4777

Content Based Filtering in intellikit

Here's how a movie recommender system can be implemented;

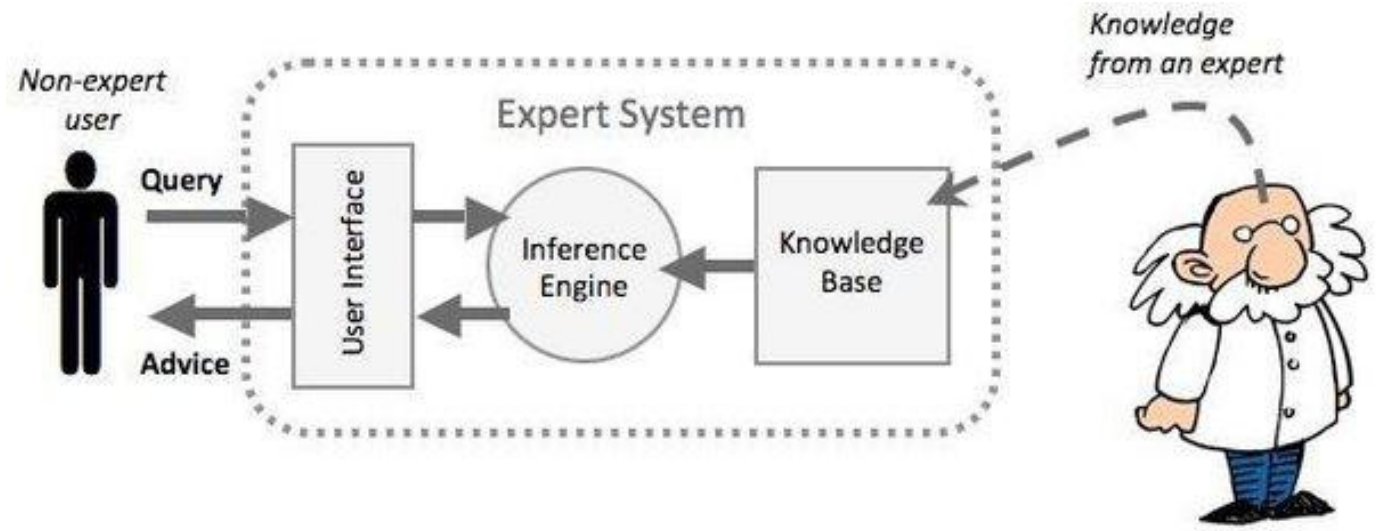
```
# Load the movies dataset

movies = pd.read_csv("/content/imdb_top_1000.csv")

# Select the columns to focus on and merge them into a single column

movies["document"] = movies[["Series_Title", "Genre", "Overview", "Director",
"Released_Year", "Star1", "Star2", "Star3", "Star4"]].apply(lambda x: " ".join(x), axis=1)

# Set the series_title column as the id column

movies = movies.set_index("Series_Title")

# Get the user's query

query = input("2012, Drama, A movie about a boy who gets lost and is adopted by and
aussie family")

top_similar_docs_vsm = ik.vector_space_model(query, movies["document"], k=5)
```

Knowledge-Based Recommenders

How knowledge based systems work

KBS (also known as Expert systems) solve problems by automatically combining explicitly represented knowledge in a knowledge base

The primary inference mechanisms used is called Case-Based Reasoning (CBR)

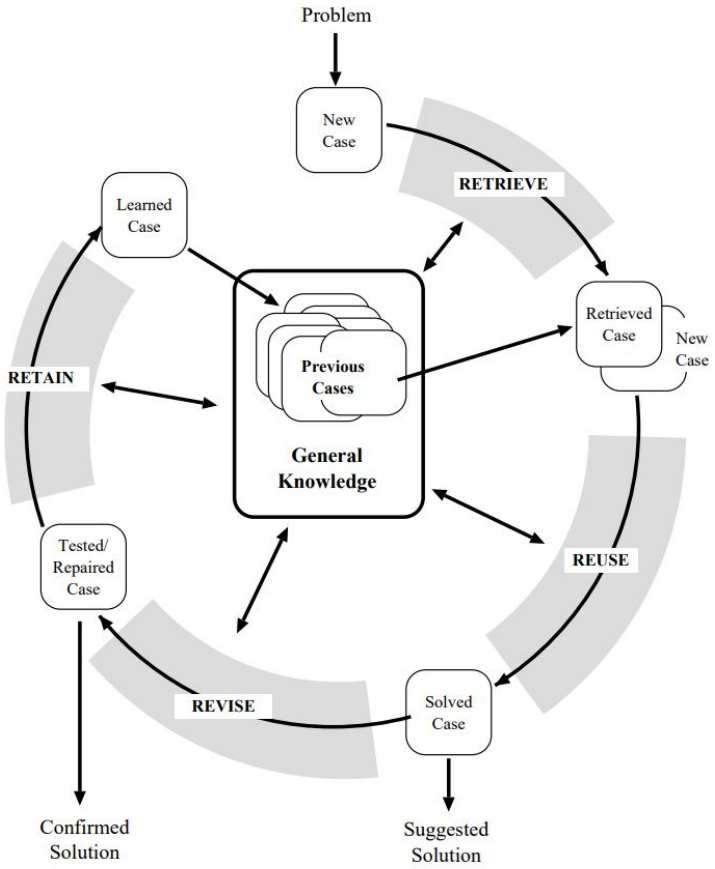


Case Based Reasoning and Similarity

The main principle of CBR is: similar problems have similar solutions

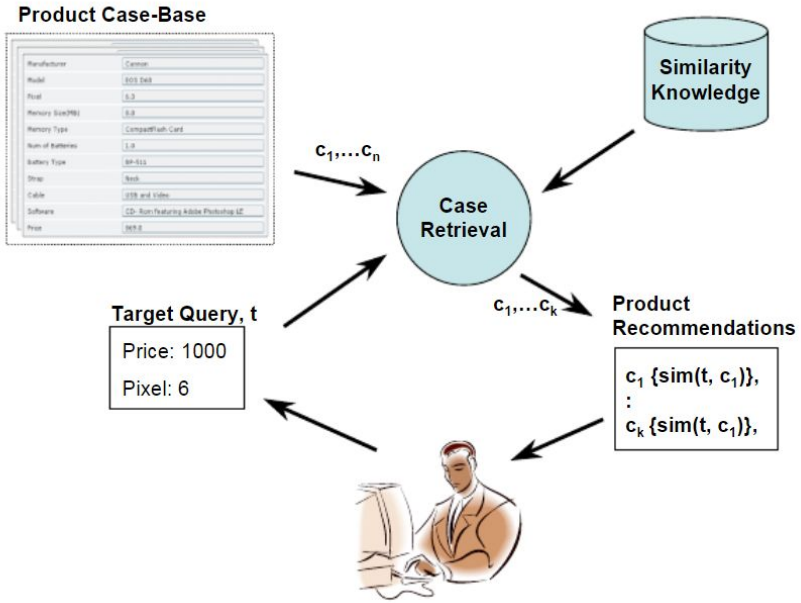
Cases = Experience = Problem/Solution pair

To solve a problem we retrieve case with a similar problem from the case base



Case Based Recommenders

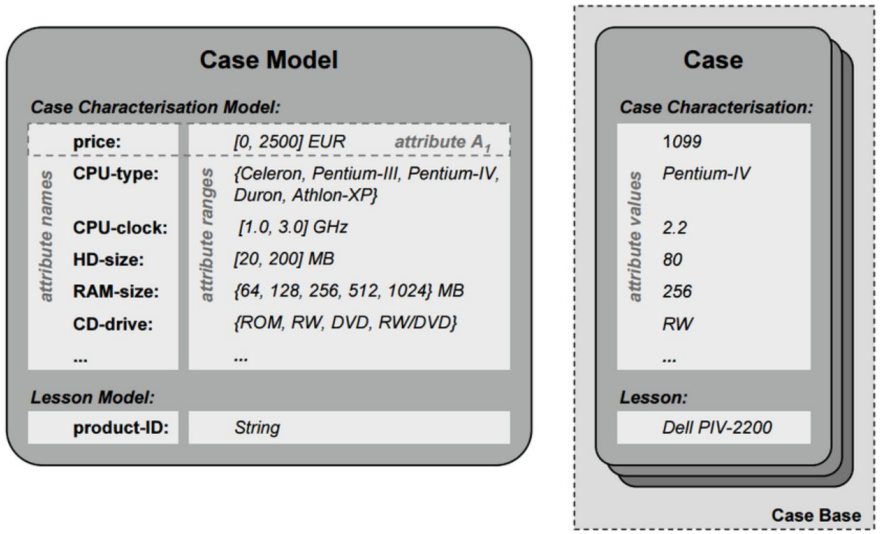
In its simplest form a case-based recommendation system will retrieve and rank product suggestions by comparing the **user's target query** to the **descriptions of products stored** in its casebase using **similarity knowledge** to identify products that are close matches to the target query.



Case Representation

Content/items are represented in an unstructured or semi-structured manner.

- Attribute-value case representation
- Object-oriented case representation



Product Recommendation

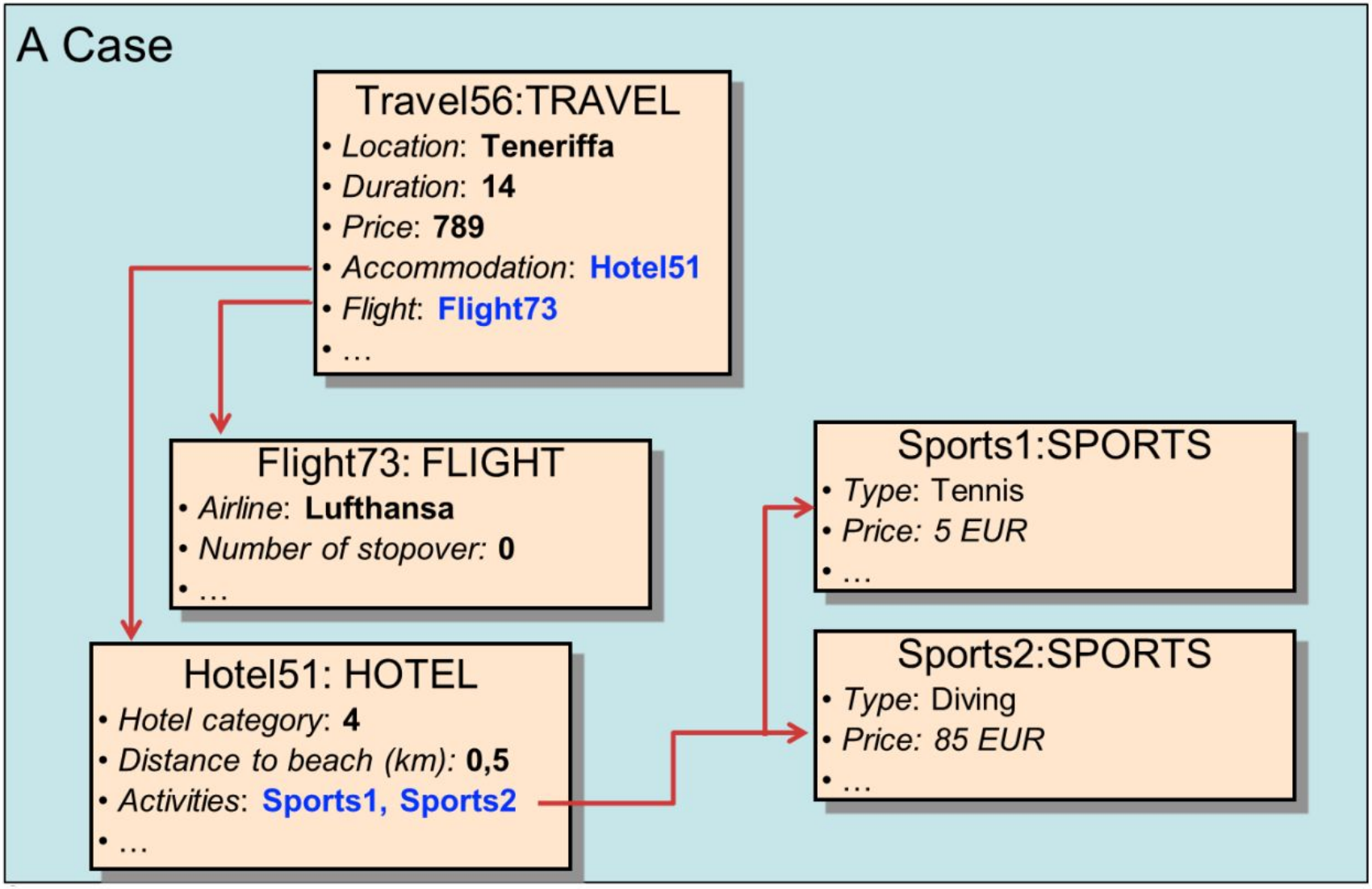
→ Representation of a „travel case“

Characterization

- *Country*: SYMBOL {Teneriffa, Lanzarote, Mallorca, Ibiza…}
- *Accommodation*: SYMBOL {Flat, Hotel, Bungalow, Camping, …}
- *Accommodation-Name:* STRING
- *Price:* REAL [100,10000]
- *Sport offers:* SYMBOL SET {Diving, Sailing, Golf, Tennis, …}
- …

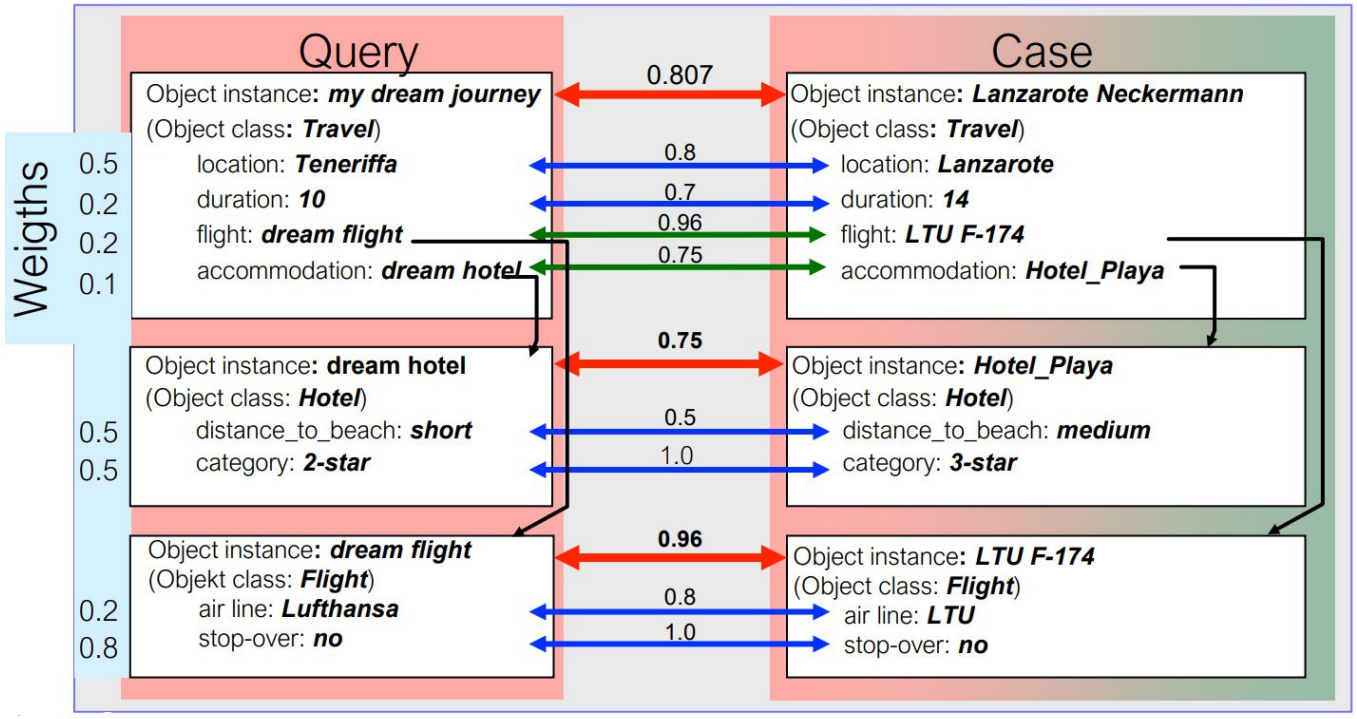
“Solution information”

- *Product description: STRING*


A Case
Travel56:TRAVEL
• Location: Teneriffa
• Duration: 14
• Price: 789
• Accommodation: Hotel51
• Flight: Flight73
• …
Flight73: FLIGHT
• Airline: Lufthansa
• Number of stopover: 0
• …
Hotel51: HOTEL
• Hotel category: 4
• Distance to beach (km): 0,5
• Activities: Sports1, Sports2
• …
Sports1:SPORTS
• Type: Tennis
• Price: 5 EUR
• …
Sports2:SPORTS
• Type: Diving
• Price: 85 EUR
• …

Similarity Assessment

The second important distinguishing feature of case-based recommender systems relates to their use of various sophisticated approaches to similarity assessment when it comes to judging which cases to retrieve in response to some user query.



The “**intellikit**” Python library

Pros

- Intellikit is a flexible library for CBR and IR tasks.
- let’s quickly use inbuilt similarity and distance measures on your task
- has inbuilt retrievers (linear retriever, MACFAC)
- Is open source and expandable

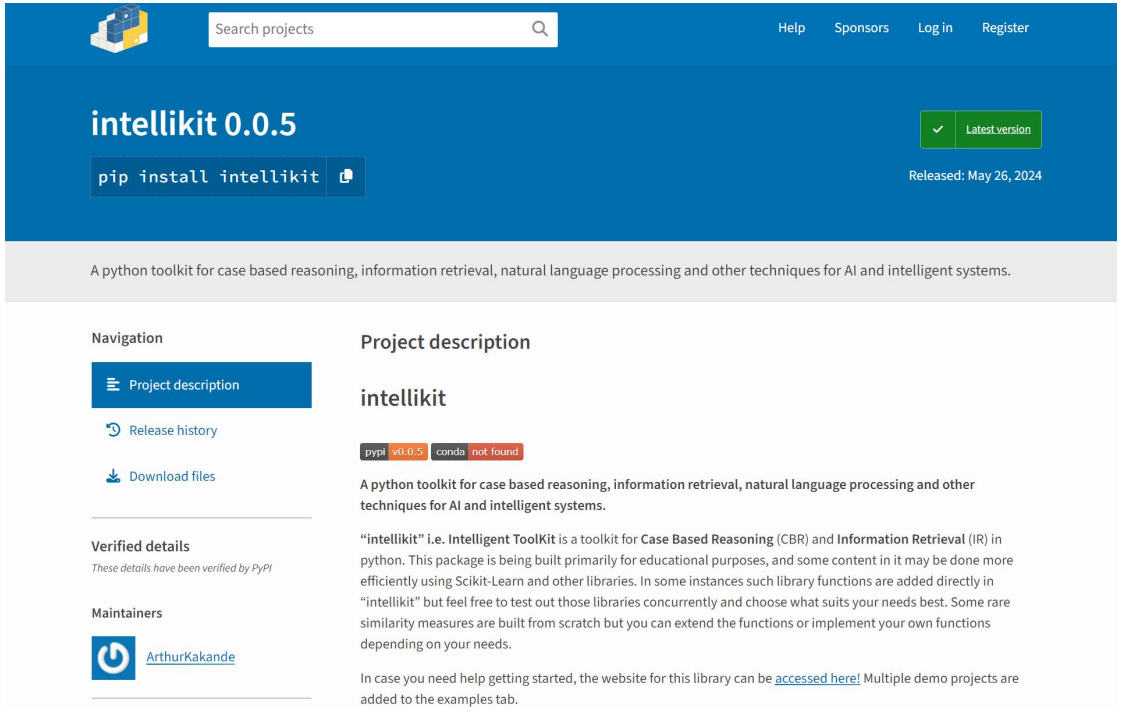
Cons

- Hasn’t been tested on very large datasets
- does not include very complex similarity and distance measures
- No API connection so you may have to create your own
- Add more +++

Installation

https://arthurkakande.github.io/intellikit/installation/

- PyPI: pip install intellikit
- Importing: import intellikit as ik



Adding a case base

intellikit allows you to use a pandas dataframe as a case base

intellikit currently takes in attribute-value case representation and object-oriented case representation

Building your first recommendation system

1. Specify the similarity measures
2. Specify the query and select relevant features
3. Assign weights
4. Choose the number of top results to see
5. Choose a retriever and give it all the above specifications
6. (You may need to do some data preprocessing based on your dataframe)

Similarity

Similarity Measures - Hamming Distance

Hamming Distance: Looks at the number of positions at which the corresponding symbols are different. The symbols may be strings, numbers, etc, e.g.

- "karolin" and "kathrin" is 3.
- "karolin" and "kerstin" is 3.
- "kathrin" and "kerstin" is 4.
- 0000 and 1111 is 4.
- 2173896 and 2233796 is 3.

In intellikit this distance is also normalised to give you hamming similarity

`ik.sim_hamming()`

`ik.dis_hamming()`

Similarity Measures - Strings

Levenshtein (Edit distance): Minimal number of insert, delete, or replace operations of single characters to transform the strings into one another. e.g;

d(„car“, „cars“) = 1

d(„car“, „vehicle“) = 7

In intellikit we have both the Levenshtein distance and the normalised levenshtein distance which serves as similarity.

`ik.sim_levenshtein()`

`ik.dis_levenshtein()`

Similarity Measures - Strings

N-Grams:

Splitting of both strings into all possible substrings of length N

Distance defined based on number of identical sub-strings

e.g.: q="CAKE", c="ProCAKE", n=3

- • = {"CAK","AKE"}, = {"Pro","roC","oCA","CAK","AKE"}
- • $ssssss\ qq, cc$ =
- 2/5 = 0.4

This distance is also normalised to a similarity measure

ik.sim_ngram()

Similarity Measures - Strings

Exact Match

While the previous measures return some sort of value ranging from 0 to 1 on how similar the strings are, the exact match returns 0 or 1. It's much suitable for systems where the strings (categories are already listed - and have less chances of errors) e.g.:

- q=“cake”, c=“cake” is 1.
- "karolin" and "kathrin" is 0.
- "karolin" and "kerstin" is 0.
- "kathrin" and "kerstin" is 0.

`ik.sim_stringEM()`

Similarity Measures - Numerical Attributes

City block metric (the sum of absolute differences between points across all the dimensions) `ik.sim_cityblock()`

Euclidean distance (the distance between two points) `ik.sim_euclidean()`

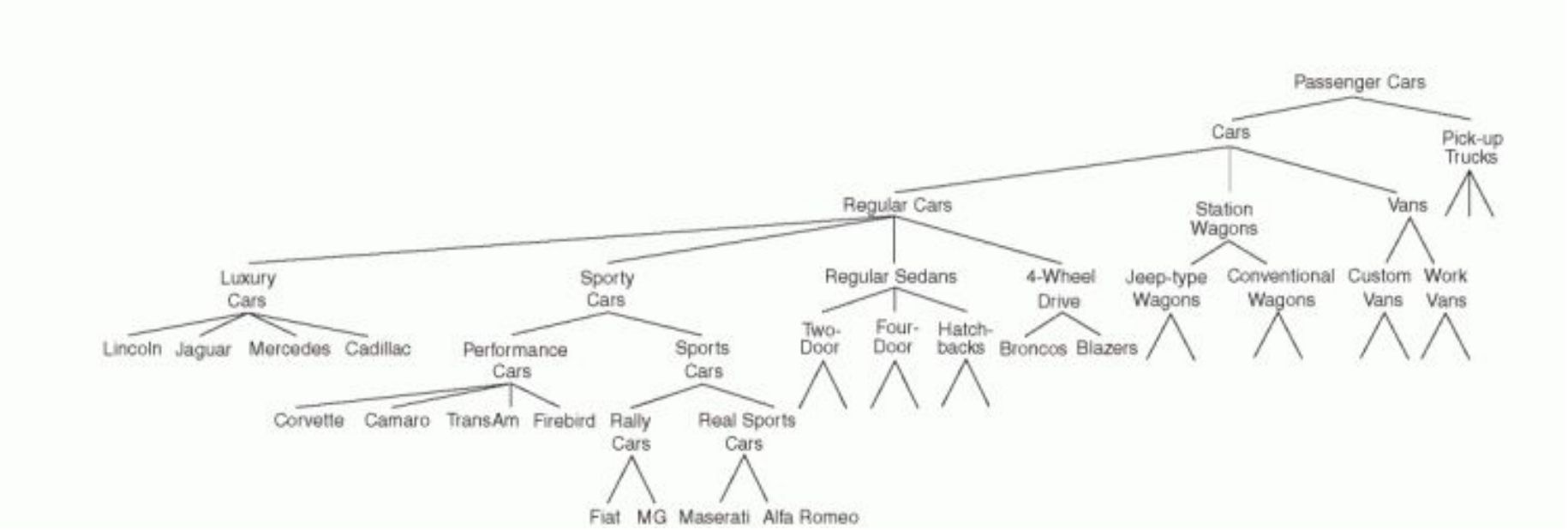
Weighted Euclidean distance (Same as above but weighted) `ik.sim_weighted()`

Exact Match (returns 0 if the number isn't an exact much and 1 if it is) `ik.sim_numEM()`

Log similarity (converts the two figures to log 10 and obtain the difference, this figure is similar to a normalised city block metric above) `ik.sim_logDifference()`

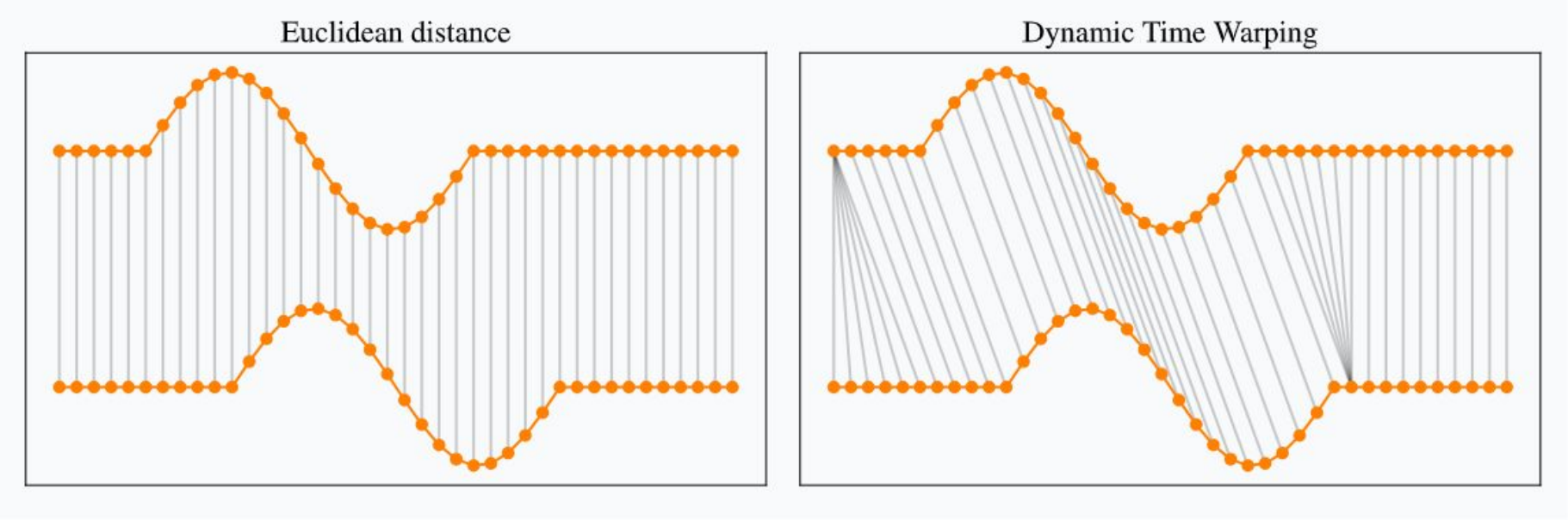
Advanced Measures - Taxonomies

Taxonomies can often be used to represent items that are closer based on categories they fall in e.g. items that share a manufacture, or colours like brown and yellow might be best structured closer to each other using a taxonomy.



Advanced Measures - Time Series

Time series cases if they match can be easily compared the same way numerical features are compared. When not, a complex method called Dynamic Time Warping can be used. Dynamic time warping (DTW) is a way of comparing two, temporal sequences that don't perfectly sync up through mathematics



Attribute weights

Intellikit also takes in a weight for each attribute, where no weight is specified, a default weight of 1 for each feature is assumed. However users are encourage to create a weight variable.

```
feature_weights = {
   'Colour': 0.2,
   'Price': 0.5,
   'Manufacturer': 0.3,
}
```

Retrievers

Linear Search

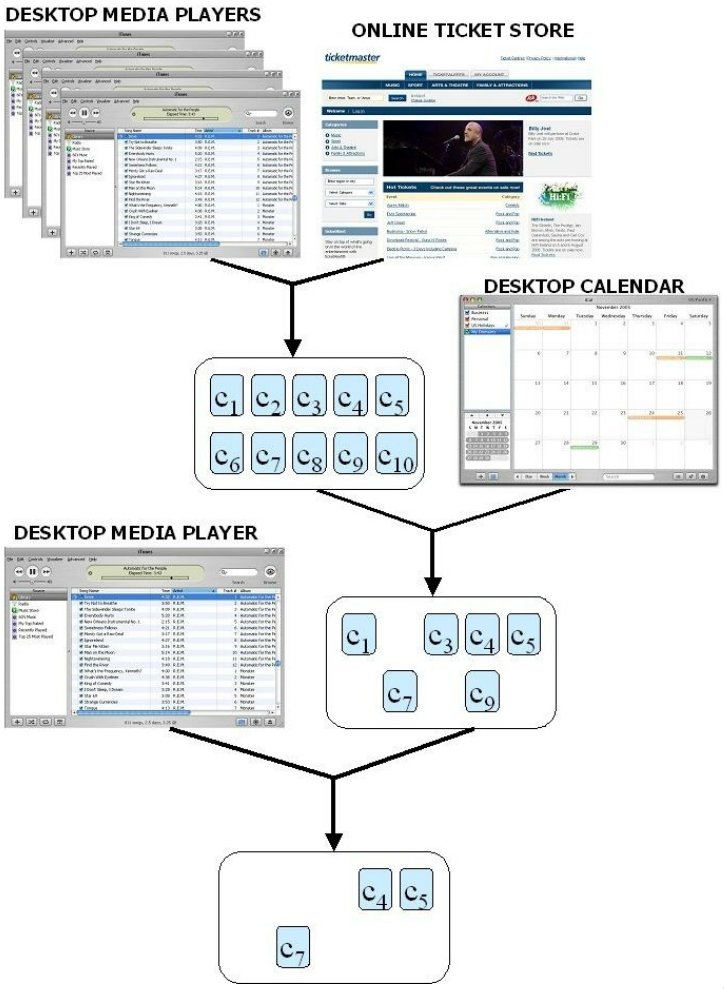
The linear retriever performs a K-NN search by computing all similarities between the query and each case one by one sequentially

```
linearRetriever(df, query, similarity_functions, feature_weights, top_n=1)
```

MACFAC Retriever

MAC = Many Are Called, FAC = Few Are Chosen. This performs a two-staged retrieval where the first phase (MAC) uses a lightweight similarity to remove irrelevant cases for the second phase (FAC) where the final similarity for the filtered cases is evaluated.

```
macfacRetriever(df, query, mac_features,
fac_features, similarity_functions,
feature_weights, top_n_mac=2,
top_n_fac=1)
```



Demo Time

Rashid Kisejjere, Data Fellow

POLLICY

QUESTIONS & ANSWERS