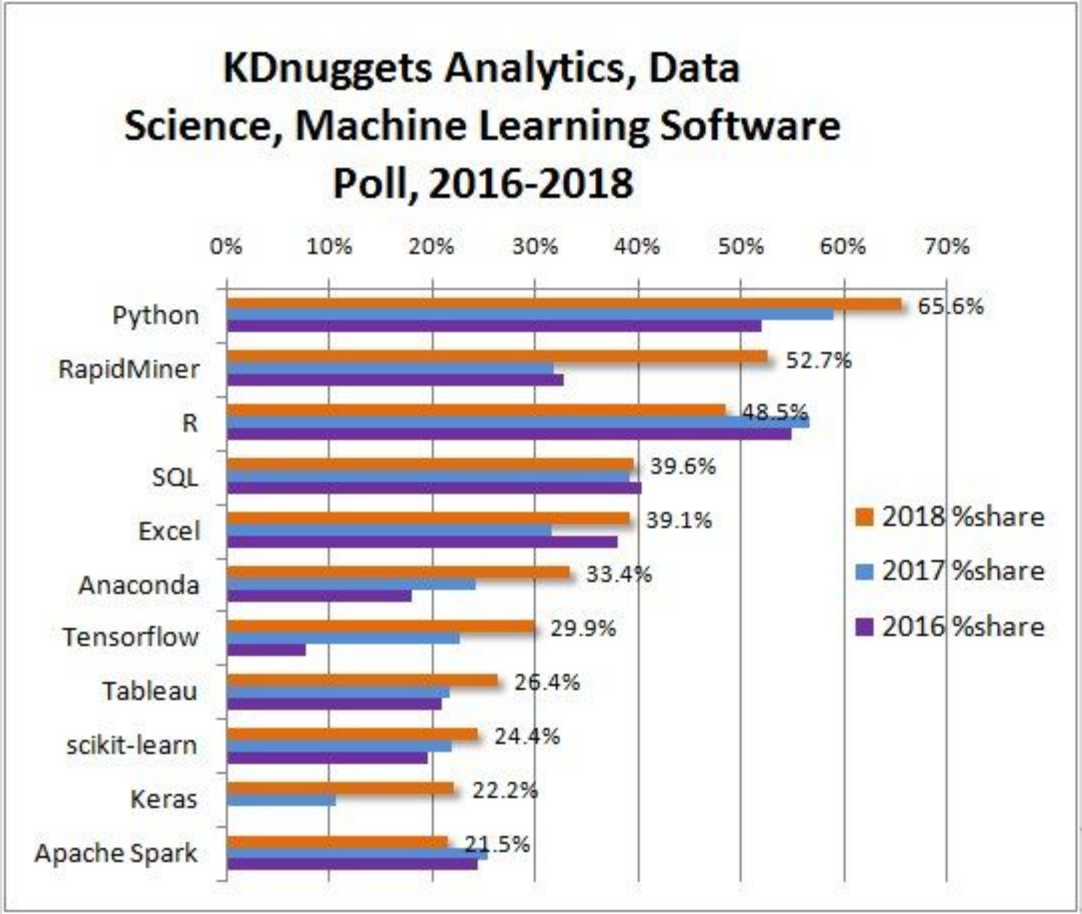
Recommender Systems

Collaborative filtering and dimensionality reduction

Mladen Jovanovic
Data Scientist // gmladen@gmail.com

BRAVO SYSTEMS



KDnuggets Analytics, Data Science, Machine Learning Software Poll, 2016-2018
0%
10%
20%
30%
40%
50%
60%
70%
Python
RapidMiner
R
SQL
Excel
Anaconda
Tensorflow
Tableau
scikit-learn
Keras
Apache Spark
65.6%
52.7%
48.5%
39.6%
39.1%
33.4%
29.9%
26.4%
24.4%
22.2%
21.5%
2018 %share
2017 %share
2016 %share

Recommender systems

- Recommender systems are considered as an **inevitable part** of any larger system that is offering some kind of **products/services** to the final user
- Recommender systems are **beneficial** to both **service providers** and **users**
- They provide **personalized experience**

Industry

Industry

- **35%** of what consumers purchase on **Amazon**
- **75%** of what they watch on **Netflix**

comes from **product recommendations**

Rating systems

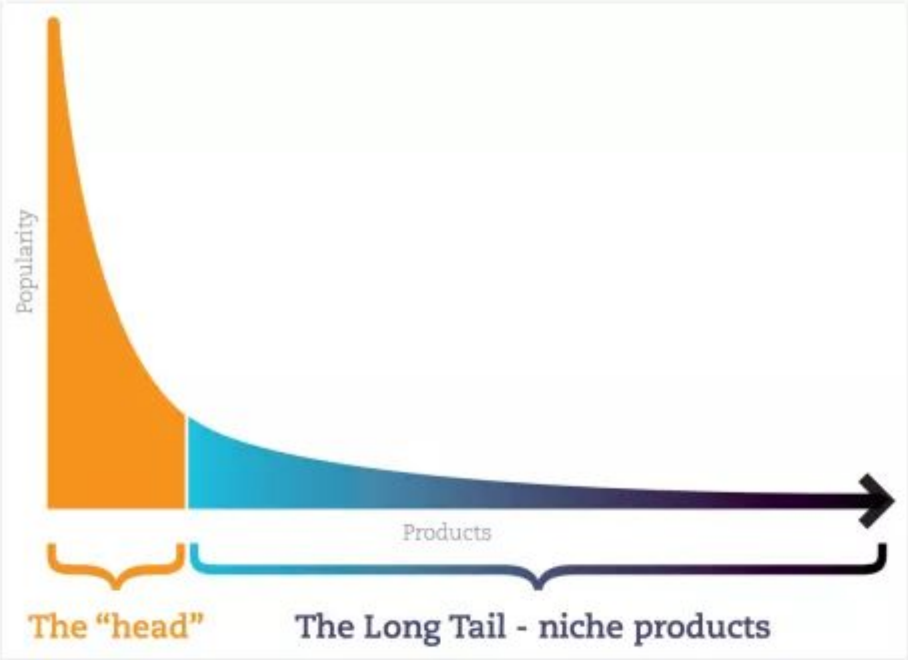
Various rating systems

- **continuous** and take any values on a specific interval (5.2 on a scale from 0 to 10)
- **discrete** (stars/grades system)
- **ordinal** (e.g. “new” Facebook like, love, sad, angry etc.)
- **binary** (like or dislike, e.g. voting news articles/comments)
- **unary** (eg. “old” like on Facebook)

Implicit / explicit feedback

- **Explicit feedback** from the users -> **ratings**
- Various **implicit feedbacks** gets collected during usage of the system (viewed items, mouse hovers etc.)

Ratings distribution

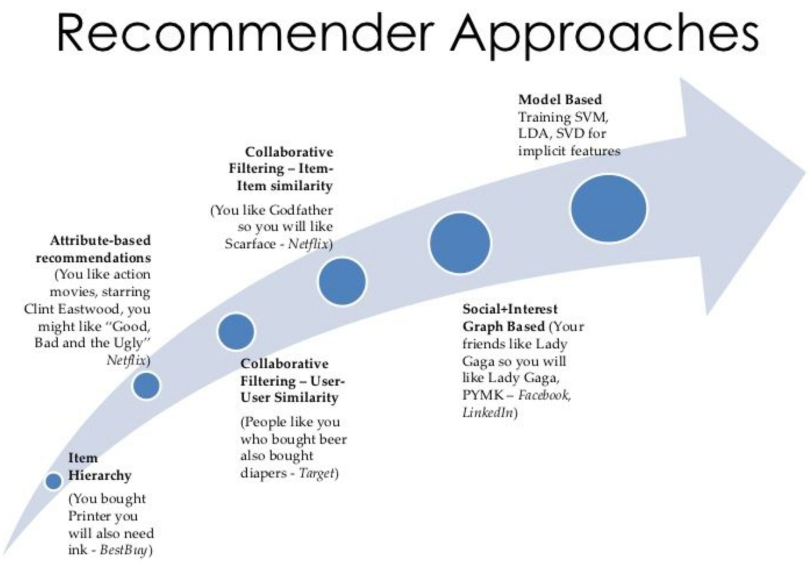


- Pareto principle
- **80%** of the ratings come from **20%** of the items

Types of recommender systems

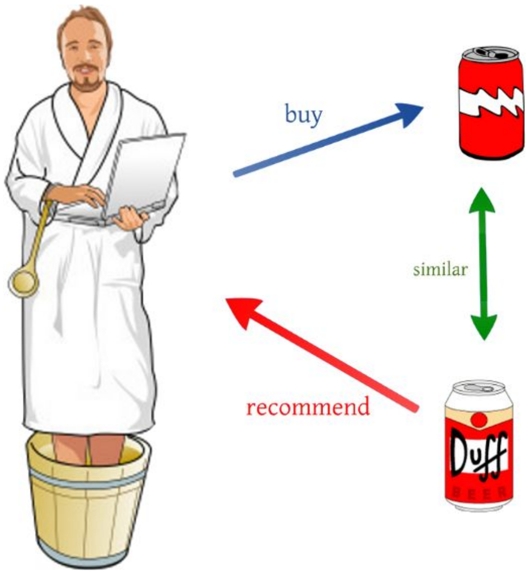
Types of recommender systems

- **Content-based**
- **Collaborative**
  - User/Item
  - Neighbourhood/Latent vector
- **Hybrid** models



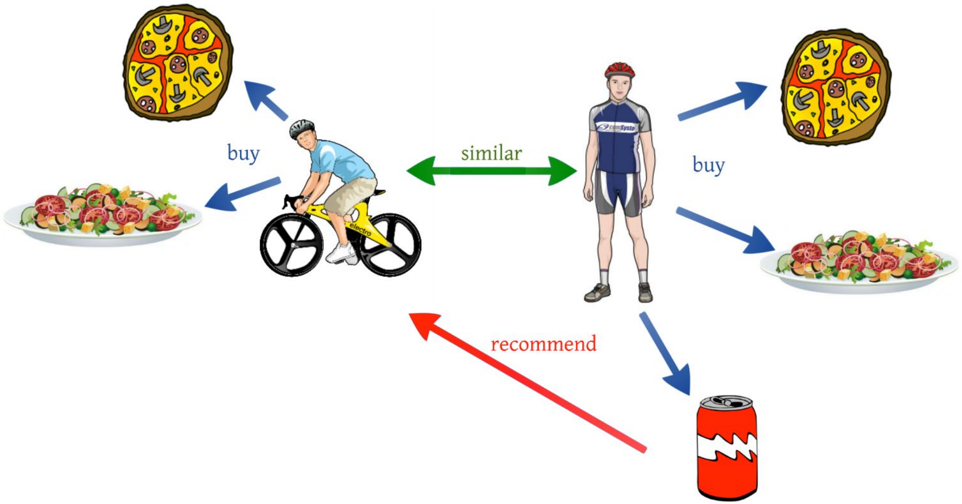
Content-based recommenders

- **dependent on properties** of an item *(categorization, description etc.)*
- **independent** of other **users' behavior**
- **no cold-start** problem
- **cannot recommend** anything genuinely "**new**"

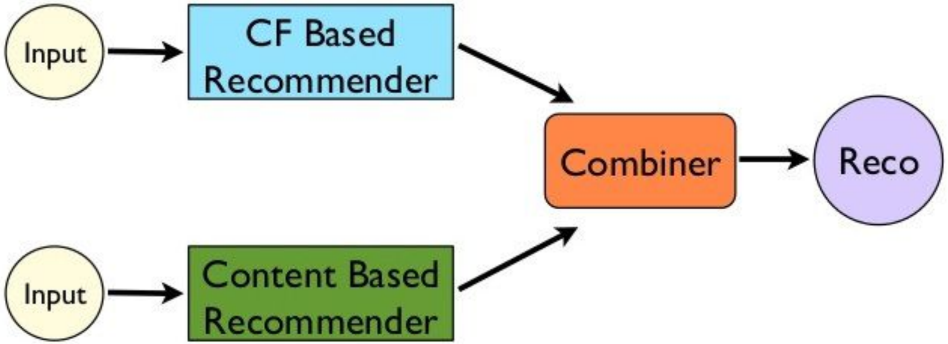


Collaborative recommenders

- **better results** than content-based
- domain **free**
- suffer from
  cold-start problem
- **sparsity** of information



Hybrid recommenders



Few words about tools/libraries

SciPy stack

- **NumPy** // Provides an abundance of useful features for **operations on N-dimensional arrays** in Python. The library provides **vectorization** of mathematical operations on the NumPy array type, which **boosts performance** and **speeds up the execution**

- **Pandas** // Pandas is a Python package for **data wrangling**. It designed for quick and easy data manipulation, aggregation and visualization

Machine learning

- **Scikit-learn**

  Scikits are additional packages of SciPy Stack designed for specific functionalities like image processing and **machine learning** facilitation.

  The library combines quality code and good documentation, ease of use and high performance and is de-facto **industry standard for machine learning with Python**.

Jupyter notebooks

- The Jupyter Notebook is an **open-source web application** that allows you to create and share documents that contain **live code, equations, visualizations and narrative text**
- **Indispensable tool for exploratory data analysis in python**

Dataset

Dataset

Movielens https://grouplens.org/datasets/movielens/

This dataset describes 5-star rating and free-text tagging activity from MovieLens, a movie recommender service.

grouplens

UNIVERSITY OF MINNESOTA

movielens

Dataset // movies table

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

Dataset // ratings table

- Another representation of the following table can be in a form of a **sparse matrix**
- **Sparse matrix** -> most elements are **zero**
- Movie recommendations -> ~99% missing entries

	userId	movieId	rating	timestamp
17208	111	3608	3.5	1098374299
15886	102	3061	4.0	957980939
6527	36	32	5.0	847056901
15282	100	88	2.0	854194208
542	7	780	3.0	851866703
85159	572	52722	3.5	1436779789
11100	73	6953	3.5	1255588164
3241	19	441	3.0	855192455
23348	165	1923	3.5	1111482007
81851	558	4024	4.0	992788160

Dataset // ratings table

	31	1029	1061	1129	1172	1263	1287	1293	1339	1343	...	134528	134783	137595	138204	60832	64997	72380	129	4736	6425
1	2.5	3.0	3.0	2.0	4.0	2.0	2.0	2.0	3.5	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	3.0	0.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Simplified dataset

- rows -> users
- columns -> movies
- **matrix R** is **sparse** (~99% missing entries)

$$R = \begin{array}{cc} \begin{pmatrix} 1 & ? & 2 & ? & ? \\ ? & ? & ? & ? & 4 \\ 2 & ? & 4 & 5 & ? \\ ? & ? & 3 & ? & ? \\ ? & 1 & ? & 3 & ? \\ 5 & ? & ? & ? & 2 \end{pmatrix} & \begin{array}{c} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \\ \text{Daniel} \\ \text{Eric} \\ \text{Zoe} \end{array} \\ \begin{array}{ccccc} \text{Titanic} & \text{ToyStory} & & & \text{Fargo} \end{array} & \end{array}$$

Content-based

Recommender Systems

Collaborative-based

Neighbourhood methods

- User/item-based similarity (similarity between vectors)

Latent factor models

- Model users in reduced multi-dimensional space

Content-based

Content-based

- New user is registered into the system
- The user **makes a selection** of **categories** of interest

[**Action**, **Comedy**, **Crime**, **Documentary**, **Drama**, **Romance**, **Thriller**]

- **Filter** movies from the **desired categories**
- Calculate **weighted ratings** and vote **counts**
- Offer **best rated items** from the chosen **categories**

Weighted rating

An item with **rating 8** with **2 votes** cannot be considered better than an item with **rating 7.9** but with **200 votes**

$$WR = (v/(v+m)) * R + (m/(v+m)) * C$$

v - number of votes for the item

m - minimum votes required to be listed

R - average rating of an item

C - mean vote across the whole report

python code

```
In [56]: # choose only movies that are in at least one of the categories (but it can be in all)
         dmr = dmr[dmr[user_cats].sum(axis=1) == len(user_cats)]
         dmr = dmr[['movieId','title'] + user_cats]
```

```
In [57]: # calculate average rating and number of ratings for each of the movies
         dmr['rating'] = dmr['movieId'].apply(lambda x: weighted_rating(x))
         dmr['count'] = dmr['movieId'].apply(lambda x: dr[dr['movieId'] == x]['rating'].count())
```

```
In [60]: # filter and sort movie list
         dmr[dmr['count'] > MIN_REVIEW].sort_values('rating', ascending=False).head(5)
```

Out[60]:

	movieId	title	Action	Drama	Thriller	rating	count
4432	6016	City of God (Cidade de Deus) (2002)	1	1	1	4.246190	69
2374	2959	Fight Club (1999)	1	1	1	4.162889	202
1018	1264	Diva (1981)	1	1	1	4.090423	14
263	293	Léon: The Professional (a.k.a. The Professiona...	1	1	1	4.052686	132
7253	69481	Hurt Locker, The (2008)	1	1	1	4.039292	26

Content-based

- Later, categories can be **derived** from the **user behavior**
- We can use all sorts of item's additional **metadata** to make the recommendation list (*e.g actors, directors*)
- Using **NLP** (natural language processing) on the movie synopsis
- Using **timestamp** of the ratings, following monthly/yearly trends etc.

Collaborative filtering

Neighbourhood methods

(user/item similarity)

Recommender Systems

Collaborative-based

Neighbourhood methods

- User/item-based similarity (similarity between vectors)

Latent factor models

- Model users in reduced multi-dimensional space

Content-based

Neighbourhood methods

- **user similarity**
  items that are recommended to a user are based on an evaluation of items by users of the same neighborhood
- **item similarity**
  similar items build neighborhoods based on appreciations of users

$$R = \begin{array}{cl} \left(\begin{array}{ccccc} 1 & ? & 2 & ? & ? \\ ? & ? & ? & ? & 4 \\ 2 & ? & 4 & 5 & ? \\ ? & ? & 3 & ? & ? \\ ? & 1 & ? & 3 & ? \\ 5 & ? & ? & ? & 2 \end{array}\right) & \begin{array}{r} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \\ \text{Daniel} \\ \text{Eric} \\ \text{Zoe} \end{array} \\ \begin{array}{ccccc} \text{Titanic} & \text{ToyStory} & & & \text{Fargo} \end{array} & \end{array}$$

Similarity measures

Plainly speaking, we're talking about similarity between two vectors

- Number of common rated items
- Manhattan distance
- Pearson correlation coefficient
- Cosine angle between vectors
- Adjusted cosine

$$R = \begin{array}{cc} \begin{pmatrix} 1 & ? & 2 & ? & ? \\ ? & ? & ? & ? & 4 \\ 2 & ? & 4 & 5 & ? \\ ? & ? & 3 & ? & ? \\ ? & 1 & ? & 3 & ? \\ 5 & ? & ? & ? & 2 \end{pmatrix} & \begin{array}{r} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \\ \text{Daniel} \\ \text{Eric} \\ \text{Zoe} \end{array} \\ \begin{array}{ccccc} \text{Titanic} & \text{ToyStory} & & & \text{Fargo} \end{array} & \end{array}$$

Neighbourhood methods

- len(**users**) **>>** len(**items**)
- **Amazon** // users: **~310 MM**, items: **12 MM**

Users will have very few mutually rated items,
while items will have a larger number of users who have co-rated them

user-based similarity

In [42]:
```
# populate data from file
rating_matrix = pd.read_csv('cf_matrix.csv',index_col=0)
```

In [63]:
```
rating_matrix.head(10)
```

Out[63]:

	31	1029	1061	1129	1172	1263	1287	1293	1339	1343	...	134528	134783	137595	138204	60832	64997	72380	129	4736	6425
1	2.5	3.0	3.0	2.0	4.0	2.0	2.0	2.0	3.5	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	3.0	0.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

10 rows × 9066 columns

user-based similarity

In [44]:
```
# calculate similarities between all users and selected user
# here we've used cosine similarity, but that's open for debate of the particular problem

sim = []
for i in range(len(rating_matrix)):
    sim.append(1 - spatial.distance.cosine(rating_matrix.iloc[USER_ID-1,:], rating_matrix.iloc[i,:]))
```

In [45]:
```
# put similarities into a dataframe for easier manipulation later on

df_sim = pd.Series(sim,index=rating_matrix.index)
df_sim.sort_values(ascending=False, inplace=True)
df_sim.head()
```

Out[45]:
```
19     1.000000
514    0.610957
537    0.522613
21     0.494830
344    0.468342
dtype: float64
```

In [46]:
```
# top rated user by defined similarity
top_sim_user = df_sim.index[1]
top_sim_user
```

Out[46]:
```
514
```

user-based similarity

In [64]:
```
# quick look at two vectors (ratings from both of the users)
cf_sim = rating_matrix.loc[[top_sim_user,USER_ID]].T
cf_sim.columns = ['top_sim_user','user']
cf_sim.head(10)
```

Out[64]:

	top_sim_user	user
31	0.0	0.0
1029	3.0	5.0
1061	0.0	3.0
1129	3.0	3.0
1172	0.0	0.0
1263	0.0	4.0
1287	0.0	4.0
1293	5.0	0.0
1339	3.0	3.0
1343	4.0	4.0

user-based similarity

In [70]:
```
# finding average rating of the top_similar_user (the threshold for the movies which we'll recommend)
# since we don't want to recommend all movies rated from a particular user

top_sim_user_avg = cf_sim[cf_sim['top_sim_user'] > 0]['top_sim_user'].mean()

# now the key is to find movies to recommend that are not rated by the user but are rated by the top_similar_user,
# since those are the films that are worth recommending

cf_sim['recommend'] = (cf_sim['top_sim_user'] > top_sim_user_avg) & (cf_sim['user'] == 0)
```

In [71]:
```
cf_sim[cf_sim['recommend']].sort_values('top_sim_user', ascending=False).head()
```

Out[71]:

	top_sim_user	user	recommend
175	5.0	0.0	True
1289	5.0	0.0	True
909	5.0	0.0	True
233	5.0	0.0	True
1120	5.0	0.0	True

user-based similarity

In [72]:
```
# getting movie ids
movie_ids = list(cf_sim[cf_sim['recommend']].sort_values('top_sim_user', ascending=False).index)
```

In [74]:
```
res = dm[dm['movieId'].isin(movie_ids)].copy()

# append corresponding ratings
res['rating'] = dr[(dr['movieId'].isin(movie_ids)) & (dr['userId'] == top_sim_user)]['rating'].values
res.sort_values('rating', ascending=False).head(10)
```

Out[74]:

	movieId	title	genres	rating
3918	5060	M*A*S*H (a.k.a. MASH) (1970)	[Comedy, Drama, War]	5.0
395	446	Farewell My Concubine (Ba wang bie ji) (1993)	[Drama, Romance]	5.0
899	1120	People vs. Larry Flynt, The (1996)	[Comedy, Drama]	5.0
730	909	Apartment, The (1960)	[Comedy, Drama, Romance]	5.0
895	1111	Microcosmos (Microcosmos: Le peuple de l'herbe...	[Documentary]	5.0
997	1243	Rosencrantz and Guildenstern Are Dead (1990)	[Comedy, Drama]	5.0
285	319	Shallow Grave (1994)	[Comedy, Drama, Thriller]	5.0
607	720	Wallace & Gromit: The Best of Aardman Animatio...	[Adventure, Animation, Comedy]	5.0
244	272	Madness of King George, The (1994)	[Comedy, Drama]	5.0
205	233	Exotica (1994)	[Drama]	5.0

User-based vs item-based

- Two users will generally have very few mutually rated items, while two **items will have a larger number of users who have co-rated them -> more stable**
- **User-based** approach requires frequent updates of the similarity matrix -> **very costly**
- **Item-based** methods recommend obvious items and items that are not novel from previous experience

Collaborative filtering

Latent Factor Models

SVD // Singular Value Decomposition

Recommender Systems

Collaborative-based

Neighbourhood methods

- User/item-based similarity (similarity between vectors)

Latent factor models

- Model users in reduced multi-dimensional space

Content-based

Dimensionality reduction

PCA // Principal Component Analysis

PCA

- **Principal component analysis (PCA)** is a mathematical procedure that **transforms** a number of (possibly) correlated variables **into a (smaller) number** of uncorrelated variables called principal components.
- The **first principal component** accounts for **as much of the variability** in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible

PCA

1. Compute the mean feature vector

$$\mu = \frac{1}{p}\sum_{k=1}^{p} x_k$$, where, x_k is a pattern ($k = 1$ to p), p = number of patterns, x is the feature matrix

2. Find the covariance matrix

$$C = \frac{1}{p}\sum_{k=1}^{p}\{x_k - \mu\}\{x_k - \mu\}^T$$ where, T represents matrix transposition

3. Compute Eigen values λ_i and Eigen vectors v_i of covariance matrix

   $Cv_i = \lambda_i v_i$ $\quad$ ($i = 1, 2, 3, \ldots q$), q = number of features

4. Estimating high-valued Eigen vectors
   - (i) Arrange all the Eigen values (λ_i) in descending order
   - (ii) Choose a threshold value, θ
   - (iii) Number of high-valued λ_i can be chosen so as to satisfy the relationship

     $$\left(\sum_{i=1}^{s}\lambda_i\right)\left(\sum_{i=1}^{q}\lambda_i\right)^{-1} \geq \theta$$, where, s = number of high valued λ_i chosen

   - (iv) Select Eigen vectors corresponding to selected high valued λ_i

5. Extract low dimensional feature vectors (principal components) from raw feature matrix.

   $P = V^T x$, where, V is the matrix of principal components and x is the feature matrix

PCA

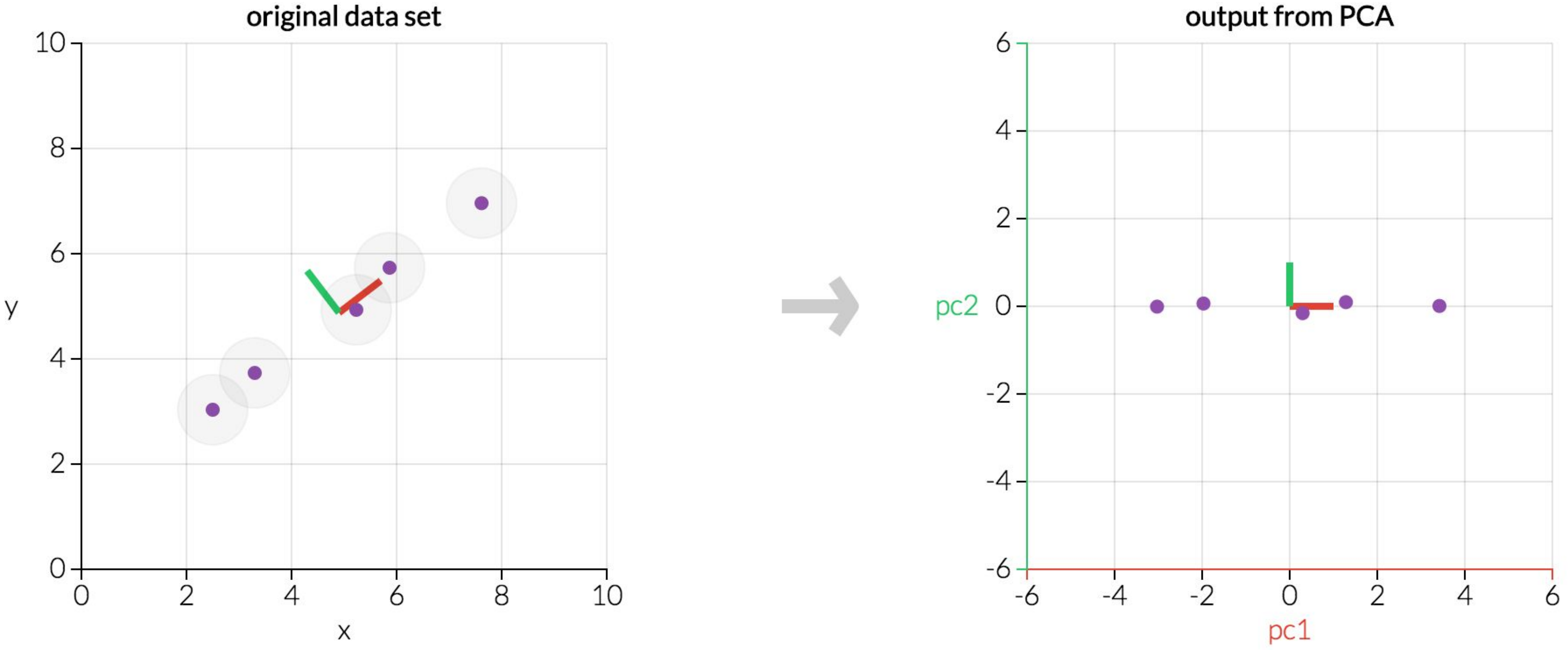
Original dataset

x	2.5	3.2	5.2	5.9	7.8
y	3.1	3.8	5	5.8	6.9

Output from PCA

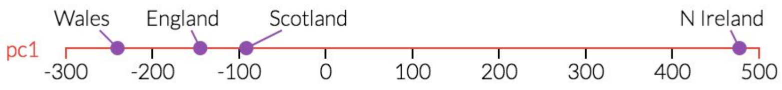
pc1	-3	-1.9	0.2	1.2	3.5

PCA // simple example (2-dim data)



PCA // 17-dim example

	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175



PCA

In a nutshell, this is what PCA is all about:

- Finding the directions of **maximum variance** in **high-dimensional data** and project it onto a **smaller dimensional subspace** while retaining most of the information
- Var(pc1) > Var(pc2) > Var(pc3) ...

Recommender Systems

Collaborative-based

Neighbourhood methods

- User/item-based similarity (similarity between vectors)

Latent factor models

- Model users in reduced multi-dimensional space

Content-based

Collaborative filtering // Netflix

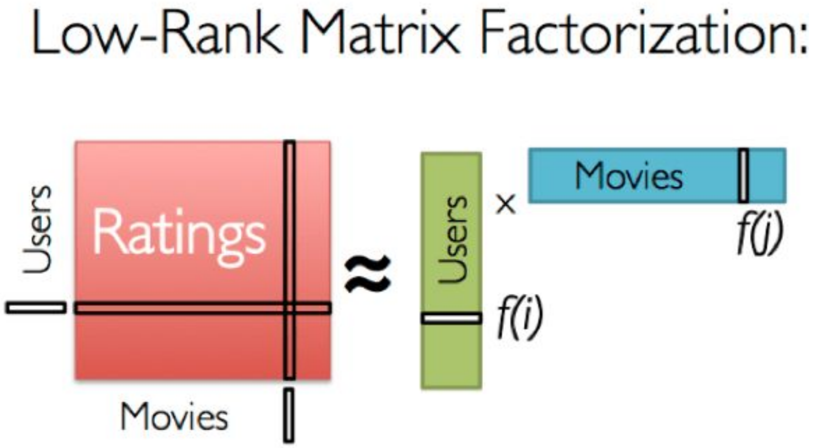
- **Open competition** to build a collaborative filtering algorithm
- **1,000,000 $ prize**
- Winner beat Netflix's accuracy by **10.06%**
- **Over 500** different approaches used

Latent Factor Models

- **Latent => Hidden**
- **Better performance** than neighbourhood (similarity-based) methods
- The factors (*components*) are called **latent** because they are there in our data but are not really discovered until you run the low-rank **matrix factorization**, then the factors emerge and hence the "latency"

SVD // Singular Value Decomposition

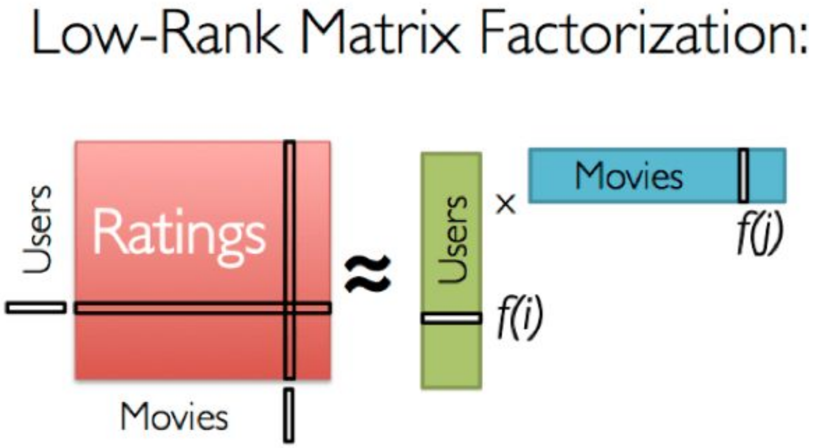
- **Matrix factorization** is breaking down of one matrix into a **product** of multiple matrices
- $\mathbf{R = MU^T}$
- **c** - number of latents factors



shape(R) = (u,m)	shape(U) = (u,**c**)	shape(M) = (**c**,m)

SVD // Singular Value Decomposition

- PCA on R gives you the **typical users U**
- PCA on R^T gives you the **typical movies M**
- SVD gives you **both in one shot**
- **$R = MU^T$**



shape(R) = (u,m)	shape(U) = (u,**c**)	shape(M) = (**c**,m)

SVD // Singular Value Decomposition

- Each principal component captures a specific latent factor
- Some of the factors can have semantics behind them

$$R = \begin{array}{cc} \begin{pmatrix} 1 & ? & 2 & ? & ? \\ ? & ? & ? & ? & 4 \\ 2 & ? & 4 & 5 & ? \\ ? & ? & 3 & ? & ? \\ ? & 1 & ? & 3 & ? \\ 5 & ? & ? & ? & 2 \end{pmatrix} & \begin{array}{l} \text{Alice} \\ \text{Bob} \\ \text{Charlie} \\ \text{Daniel} \\ \text{Eric} \\ \text{Zoe} \end{array} \\ \begin{array}{ccccc} \text{Titanic} & \text{ToyStory} & & & \text{Fargo} \end{array} & \end{array}$$

Alice	= 10% Action fan	+10% Comedy fan	+50% Romance fan	$+\cdots$
Bob	= 50% Action fan	+30% Comedy fan	+10% Romance fan	$+\cdots$
Titanic	= 20% Action	+00% Comedy	+70% Romance	$+\cdots$
Toy Story	= 30% Action	+60% Comedy	+00% Romance	$+\cdots$

SVD // Singular Value Decomposition

The value of $\mathbf{r_{ui}}$ is the result of a **dot product** between two vectors:

Vector $\mathbf{p_u}$ which is a row of **M** and which is specific to the user **u**
and **vector $\mathbf{q_i}$** which is a column of $\mathbf{U^T}$ and which is specific to the item **i**

$$\boxed{R = MU^T}$$

$$\begin{pmatrix} & & \\ & r_{ui} & \\ & & \end{pmatrix} = \begin{pmatrix} & & \\ — & p_u & — \\ & & \end{pmatrix} \begin{pmatrix} & \vert & \\ & q_i & \\ & \vert & \end{pmatrix}$$

$$r_{ui} = p_u \cdot q_i$$

$$r_{ui} = \sum_{c \in \text{concepts}} \text{affinity of } u \text{ for } c \times \text{affinity of } i \text{ for } c$$

SVD

- The SVD of sparse matrix R is not defined
- If R was dense, we could compute M and U easily
- A first option that was used for some time is to fill the missing entries of R with some simple heuristic
- Find an approximate solution to the SVD problem using Stochastic Gradient Descent

SVD - using surprise library

In [75]:
```
from surprise import Reader, Dataset, SVD, evaluate

reader = Reader()
dr.head()
```

Out[75]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

In [76]:
```
data = Dataset.load_from_df(dr[['userId', 'movieId', 'rating']], reader)
data.split(n_folds=5)
```

In [ ]:
```
svd = SVD()
evaluate(svd, data, measures=['RMSE'])
```

In [213]:
```
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Out[213]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f4c58c7ad68>
```

SVD - using surprise library

In [82]:
```
svd.predict(19, 2)
```

Out[82]:
```
Prediction(uid=19, iid=2, r_ui=None, est=3.120983027200581, details={'was_impossible': False})
```

In [83]:
```
dr[dr['userId'] == 19].head(10)
```

Out[83]:

	userId	movieId	rating	timestamp
3105	19	1	3.0	855190091
3106	19	2	3.0	855194773
3107	19	3	3.0	855194718
3108	19	4	3.0	855192868
3109	19	6	3.0	855190128
3110	19	7	3.0	855190128
3111	19	9	3.0	855190232
3112	19	10	3.0	855192496
3113	19	11	3.0	855192773
3114	19	14	5.0	855190167

Closing remarks

Closing remarks

- Recommender systems are not reserved only for the product/item recommendation
- It's a great tool to model any kind of interaction between items/services

Presentation material

- https://github.com/mladenj/recommender-systems
- Both presentation and corresponding jupyter notebook

Additional links

Additional links

- http://setosa.io/ev/principal-component-analysis/
- https://www.youtube.com/watch?v=giIXNoiqO_U
- https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf
- https://www.coursera.org/specializations/recommender-systems
- https://www.quora.com/What-is-the-difference-between-content-based-filtering-and-collaborative-filtering
- https://www.youtube.com/watch?v=z0dx-YckFko
- https://grouplens.org/blog/similarity-functions-for-user-user-collaborative-filtering/

Additional links

- https://www.youtube.com/watch?v=z0dx-YckFko
- http://thmsrey-ds.com/2017/06/27/recommender-systems-introduction-user-based-collaborative-filtering/
- https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues
- https://arxiv.org/pdf/1404.1100.pdf?utm_content=bufferb37df&utm_medium=social&utm_source=facebook.com&utm_campaign=buffer
- https://buildingrecommenders.wordpress.com/2015/11/10/the-components-of-a-recommender-system/

Questions ?

Mladen Jovanovic
Data Scientist // gmladen@gmail.com

BRAVO SYSTEMS