

*Submitted by:-*

1. *Anshika Dixit(215/UCF/056)*
2. *Isha Chauhan(215/UCF/047)*

**CERTIFICATE**

This is to certify that the declaration statement made by this group of students is correct to the best of my knowledge and belief. We have completed this Capstone Project under my guidance and supervision. The present work is the result of their original investigation, effort and study. The Capstone Project is fit for the submission and partial fulfilment of the conditions

**ACKNOWLEDGEMENT**

Completing a task is never a one-man effort. It is the results of valuable contribution of a number of individuals in a direct or indirect manner that helps you shape and achieve an objective. We profusely thank them for the support provided to us. We also express a deep sense of gratitude for providing us the opportunity and trusting us for their project.

Table of Contents

1. Introduction
2. Recommender Systems in Python
3. Loading data:- CI&T Deskdrop dataset
4. Evalution
5. Popularity model
6. Content – Based Filtering Model
7. Collaborative Filtering
8. Testing
9. Conclusion

**Introduction**

Recommender systems aim to predict users' interests and recommend product items that quite likely are interesting for them. They are among the most powerful machine learning systems that online retailers implement in order to drive sales.

Data required for recommender systems stems from explicit user ratings after watching a movie or listening to a song, from implicit search engine queries and purchase histories, or from other knowledge about the users/items themselves.

Sites like Spotify, YouTube or Netflix use that data in order to suggest playlists, so-called [Daily mixes](https://support.spotify.com/us/using_spotify/features/daily-mix/), or to make [video recommendations](https://help.netflix.com/en/node/9898), respectively.

* 1. *Need of Recommenders Systems*

Companies using recommender systems focus on increasing sales as a result of very personalized offers and an enhanced customer experience.

Recommendations typically speed up searches and make it easier for users to access content they’re interested in, and surprise them with offers they would have never searched for. What is more, companies are able to gain and retain customers by sending out emails with links to new offers that meet the recipients' interests, or suggestions of films and TV shows that suit their profiles.

The user starts to feel known and understood and is more likely to buy additional products or consume more content. By knowing what a user wants, the company gains competitive advantage and the threat of losing a customer to a competitor decrease. Providing that added value to users by including recommendations in systems and products is appealing.

* 1. Recommender system work

Recommender systems function with two kinds of information:

* Characteristic information. This is information about items (keywords, categories, etc.) and users (preferences, profiles, etc.).
* User-item interactions. This is information such as ratings, number of purchases, likes, etc.

Based on this, we can distinguish between three algorithms used in recommender systems:

* Content-based systems, which use characteristic information.
* Collaborative filtering systems, which are based on user-item interactions.
* Hybrid systems, which combine both types of information with the aim of avoiding problems that are generated when working with just one kind.

Next, we will dig a little deeper into content-based and collaborative filtering systems and see how they are different.

* [**Collaborative Filtering**](https://en.wikipedia.org/wiki/Collaborative_filtering): This method makes automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on a set of items, A is more likely to have B's opinion for a given item than that of a randomly chosen person.
* **Content-Based Filtering:-**This method uses only information about the description and attributes of the items users has previously consumed to model user's preferences. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.
* **Hybrid Methods:-**: Recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective than pure approaches in some cases. These methods can also be used to overcome some of the common problems in recommender systems such as cold start and the sparsity problem.

We will demonstrate how to implement **Collaborative Filtering**, **Content-Based Filtering** and **Hybrid methods** in Python, for the task of providing personalized recommendations to the users.

import numpy as np

import scipy

import pandas as pd

import math

import random

import sklearn

from nltk.corpus import stopwords

from scipy.sparse import csr\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from scipy.sparse.linalg import svds

from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt

Loading data: CI&T Deskdrop dataset

In this section, we load the [Deskdrop dataset](https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop), which contains a real sample of 12 months logs from CI&T's Internal Communication platform (DeskDrop). It contains users interactions articles shared in the platform. It is composed of two CSV files:

* **shared\_articles.csv**
* **users\_interactions.csv**

Take a look in this kernels for a better picture of the dataset:

* Deskdrop datasets EDA
* DeskDrop Articles Topic Modeling

## shared\_articles.csv

Contains information about the articles shared in the platform. Each article has its sharing date (timestamp), the original url, title, content in plain text, the article' lang (Portuguese: pt or English: en) and information about the user who shared the article (author).

There are two possible event types at a given timestamp:

* CONTENT SHARED: The article was shared in the platform and is available for users.
* CONTENT REMOVED: The article was removed from the platform and not available for further recommendation.

For the sake of simplicity, we only consider here the "CONTENT SHARED" event type, assuming (naively) that all articles were available during the whole one year period. For a more precise evaluation (and higher accuracy), only articles that were available at a given time should be recommended, but we let this exercise for you.

articles\_df = pd.read\_csv('../input/shared\_articles.csv')

articles\_df = articles\_df[articles\_df['eventType'] == 'CONTENT SHARED']

articles\_df.head(5)

| timestamp | eventType | contentId | authorPersonId | authorSessionId | authorUserAgent | authorRegion | authorCountry | contentType | url | title | text | lang |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1459193988 | CONTENT SHARED | -4110354420726924665 | 4340306774493623681 | 8940341205206233829 | NaN | NaN | NaN | HTML | http://www.nytimes.com/2016/03/28/business/dea... | Ethereum, a Virtual Currency, Enables Transact... | All of this work is still very early. The firs... | en |
| 2 | 1459194146 | CONTENT SHARED | -7292285110016212249 | 4340306774493623681 | 8940341205206233829 | NaN | NaN | NaN | HTML | http://cointelegraph.com/news/bitcoin-future-w... | Bitcoin Future: When GBPcoin of Branson Wins O... | The alarm clock wakes me at 8:00 with stream o... | en |
| 3 | 1459194474 | CONTENT SHARED | -6151852268067518688 | 3891637997717104548 | -1457532940883382585 | NaN | NaN | NaN | HTML | https://cloudplatform.googleblog.com/2016/03/G... | Google Data Center 360° Tour | We're excited to share the Google Data Center ... | en |
| 4 | 1459194497 | CONTENT SHARED | 2448026894306402386 | 4340306774493623681 | 8940341205206233829 | NaN | NaN | NaN | HTML | https://bitcoinmagazine.com/articles/ibm-wants... | IBM Wants to "Evolve the Internet" With Blockc... | The Aite Group projects the blockchain market ... | en |
| 5 | 1459194522 | CONTENT SHARED | -2826566343807132236 | 4340306774493623681 | 8940341205206233829 | NaN | NaN | NaN | HTML | http://www.coindesk.com/ieee-blockchain-oxford... | IEEE to Talk Blockchain at Cloud Computing Oxf... | One of the largest and oldest organizations fo... | en |

## users\_interactions.csv

Contains logs of user interactions on shared articles. It can be joined to **articles\_shared.csv** by **contentId** column.

The eventType values are:

* **VIEW**: The user has opened the article.
* **LIKE**: The user has liked the article.
* **COMMENT CREATED**: The user created a comment in the article.
* **FOLLOW**: The user chose to be notified on any new comment in the article.
* **BOOKMARK**: The user has bookmarked the article for easy return in the future.

interactions\_df = pd.read\_csv('../input/users\_interactions.csv')

interactions\_df.head(10)

| timestamp | eventType | contentId | personId | sessionId | userAgent | userRegion | userCountry |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1465413032 | VIEW | -3499919498720038879 | -8845298781299428018 | 1264196770339959068 | NaN | NaN | NaN |
| 1 | 1465412560 | VIEW | 8890720798209849691 | -1032019229384696495 | 3621737643587579081 | Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_2... | NY | US |
| 2 | 1465416190 | VIEW | 310515487419366995 | -1130272294246983140 | 2631864456530402479 | NaN | NaN | NaN |
| 3 | 1465413895 | FOLLOW | 310515487419366995 | 344280948527967603 | -3167637573980064150 | NaN | NaN | NaN |
| 4 | 1465412290 | VIEW | -7820640624231356730 | -445337111692715325 | 5611481178424124714 | NaN | NaN | NaN |
| 5 | 1465413742 | VIEW | 310515487419366995 | -8763398617720485024 | 1395789369402380392 | Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebK... | MG | BR |
| 6 | 1465415950 | VIEW | -8864073373672512525 | 3609194402293569455 | 1143207167886864524 | NaN | NaN | NaN |
| 7 | 1465415066 | VIEW | -1492913151930215984 | 4254153380739593270 | 8743229464706506141 | Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/53... | SP | BR |
| 8 | 1465413762 | VIEW | 310515487419366995 | 344280948527967603 | -3167637573980064150 | NaN | NaN | NaN |
| 9 | 1465413771 | VIEW | 3064370296170038610 | 3609194402293569455 | 1143207167886864524 | NaN | NaN | NaN |

## Data munging

As there are different interactions types, we associate them with a weight or strength, assuming that, for example, a comment in an article indicates a higher interest of the user on the item than a like, or than a simple view.

event\_type\_strength = {

'VIEW': 1.0,

'LIKE': 2.0,

'BOOKMARK': 2.5,

'FOLLOW': 3.0,

'COMMENT CREATED': 4.0,

}

interactions\_df['eventStrength'] = interactions\_df['eventType'].apply(lambda x: event\_type\_strength[x])

Recommender systems have a problem known as **user cold-start**, in which is hard do provide personalized recommendations for users with none or a very few number of consumed items, due to the lack of information to model their preferences.  
For this reason, we are keeping in the dataset only users with at leas 5 interactions.

users\_interactions\_count\_df = interactions\_df.groupby(['personId', 'contentId']).size().groupby('personId').size()

print('# users: **%d**' % len(users\_interactions\_count\_df))

users\_with\_enough\_interactions\_df = users\_interactions\_count\_df[users\_interactions\_count\_df >= 5].reset\_index()[['personId']]

print('# users with at least 5 interactions: **%d**' % len(users\_with\_enough\_interactions\_df))

# users: 1895

# users with at least 5 interactions: 1140

print('# of interactions: **%d**' % len(interactions\_df))

interactions\_from\_selected\_users\_df = interactions\_df.merge(users\_with\_enough\_interactions\_df,

how = 'right',

left\_on = 'personId',

right\_on = 'personId')

print('# of interactions from users with at least 5 interactions: **%d**' % len(interactions\_from\_selected\_users\_df))

# of interactions: 72312

# of interactions from users with at least 5 interactions: 69868

In Deskdrop, users are allowed to view an article many times, and interact with them in different ways (eg. like or comment). Thus, to model the user interest on a given article, we aggregate all the interactions the user has performed in an item by a weighted sum of interaction type strength and apply a log transformation to smooth the distribution.

def smooth\_user\_preference(x):

return math.log(1+x, 2)

interactions\_full\_df = interactions\_from\_selected\_users\_df \

.groupby(['personId', 'contentId'])['eventStrength'].sum() \

.apply(smooth\_user\_preference).reset\_index()

print('# of unique user/item interactions: **%d**' % len(interactions\_full\_df))

interactions\_full\_df.head(10)

# of unique user/item interactions: 39106

| personId | contentId | eventStrength |
| --- | --- | --- |
| 0 | -9223121837663643404 | -8949113594875411859 | 1.000000 |
| 1 | -9223121837663643404 | -8377626164558006982 | 1.000000 |
| 2 | -9223121837663643404 | -8208801367848627943 | 1.000000 |
| 3 | -9223121837663643404 | -8187220755213888616 | 1.000000 |
| 4 | -9223121837663643404 | -7423191370472335463 | 3.169925 |
| 5 | -9223121837663643404 | -7331393944609614247 | 1.000000 |
| 6 | -9223121837663643404 | -6872546942144599345 | 1.000000 |
| 7 | -9223121837663643404 | -6728844082024523434 | 1.000000 |
| 8 | -9223121837663643404 | -6590819806697898649 | 1.000000 |
| 9 | -9223121837663643404 | -6558712014192834002 | 1.584963 |

# Evaluation

Evaluation is important for machine learning projects, because it allows to compare objectivelly different algorithms and hyperparameter choices for models.  
One key aspect of evaluation is to ensure that the trained model generalizes for data it was not trained on, using **Cross-validation** techniques. We are using here a simple cross-validation approach named **holdout**, in which a random data sample (20% in this case) are kept aside in the training process, and exclusively used for evaluation. All evaluation metrics reported here are computed using the **test set**.

Ps. A more robust evaluation approach could be to split train and test sets by a reference date, where the train set is composed by all interactions before that date, and the test set are interactions after that date. For the sake of simplicity, we chose the first random approach for this notebook, but you may want to try the second approach to better simulate how the recsys would perform in production predicting "future" users interactions.

interactions\_train\_df, interactions\_test\_df = train\_test\_split(interactions\_full\_df,

stratify=interactions\_full\_df['personId'],

test\_size=0.20,

random\_state=42)

print('# interactions on Train set: **%d**' % len(interactions\_train\_df))

print('# interactions on Test set: **%d**' % len(interactions\_test\_df))

# interactions on Train set: 31284

# interactions on Test set: 7822

In Recommender Systems, there are a set metrics commonly used for evaluation. We chose to work with **Top-N accuracy metrics**, which evaluates the accuracy of the top recommendations provided to a user, comparing to the items the user has actually interacted in test set.  
This evaluation method works as follows:

* For each user
  + For each item the user has interacted in test set
    - Sample 100 other items the user has never interacted.  
      Ps. Here we naively assume those non interacted items are not relevant to the user, which might not be true, as the user may simply not be aware of those not interacted items. But let's keep this assumption.
    - Ask the recommender model to produce a ranked list of recommended items, from a set composed one interacted item and the 100 non-interacted ("non-relevant!) items
    - Compute the Top-N accuracy metrics for this user and interacted item from the recommendations ranked list
* Aggregate the global Top-N accuracy metrics

The Top-N accuracy metric choosen was **Recall@N** which evaluates whether the interacted item is among the top N items (hit) in the ranked list of 101 recommendations for a user.  
Ps. Other popular ranking metrics are **NDCG@N** and **MAP@N**, whose score calculation takes into account the position of the relevant item in the ranked list (max. value if relevant item is in the first position). You can find a reference to implement this metrics in this [post](http://fastml.com/evaluating-recommender-systems/).

*Indexing by personId to speed up the searches during evaluation*

interactions\_full\_indexed\_df = interactions\_full\_df.set\_index('personId')

interactions\_train\_indexed\_df = interactions\_train\_df.set\_index('personId')

interactions\_test\_indexed\_df = interactions\_test\_df.set\_index('personId')

def get\_items\_interacted(person\_id, interactions\_df):

*# Get the user's data and merge in the movie information.*

interacted\_items = interactions\_df.loc[person\_id]['contentId']

return set(interacted\_items if type(interacted\_items) == pd.Series else [interacted\_items])

*#Top-N accuracy metrics consts*

EVAL\_RANDOM\_SAMPLE\_NON\_INTERACTED\_ITEMS = 100

class **ModelEvaluator**:

def get\_not\_interacted\_items\_sample(self, person\_id, sample\_size, seed=42):

interacted\_items = get\_items\_interacted(person\_id, interactions\_full\_indexed\_df)

all\_items = set(articles\_df['contentId'])

non\_interacted\_items = all\_items - interacted\_items

random.seed(seed)

non\_interacted\_items\_sample = random.sample(non\_interacted\_items, sample\_size)

return set(non\_interacted\_items\_sample)

def \_verify\_hit\_top\_n(self, item\_id, recommended\_items, topn):

try:

index = next(i for i, c **in** enumerate(recommended\_items) if c == item\_id)

except:

index = -1

hit = int(index **in** range(0, topn))

return hit, index

def evaluate\_model\_for\_user(self, model, person\_id):

*#Getting the items in test set*

interacted\_values\_testset = interactions\_test\_indexed\_df.loc[person\_id]

if type(interacted\_values\_testset['contentId']) == pd.Series:

person\_interacted\_items\_testset = set(interacted\_values\_testset['contentId'])

else:

person\_interacted\_items\_testset = set([int(interacted\_values\_testset['contentId'])])

interacted\_items\_count\_testset = len(person\_interacted\_items\_testset)

*#Getting a ranked recommendation list from a model for a given user*

person\_recs\_df = model.recommend\_items(person\_id,

items\_to\_ignore=get\_items\_interacted(person\_id,

interactions\_train\_indexed\_df),

topn=10000000000)

hits\_at\_5\_count = 0

hits\_at\_10\_count = 0

*#For each item the user has interacted in test set*

for item\_id **in** person\_interacted\_items\_testset:

*#Getting a random sample (100) items the user has not interacted*

*#(to represent items that are assumed to be no relevant to the user)*

non\_interacted\_items\_sample = self.get\_not\_interacted\_items\_sample(person\_id,

sample\_size=EVAL\_RANDOM\_SAMPLE\_NON\_INTERACTED\_ITEMS,

seed=item\_id%(2\*\*32))

*#Combining the current interacted item with the 100 random items*

items\_to\_filter\_recs = non\_interacted\_items\_sample.union(set([item\_id]))

*#Filtering only recommendations that are either the interacted item or from a random sample of 100 non-interacted items*

valid\_recs\_df = person\_recs\_df[person\_recs\_df['contentId'].isin(items\_to\_filter\_recs)]

valid\_recs = valid\_recs\_df['contentId'].values

*#Verifying if the current interacted item is among the Top-N recommended items*

hit\_at\_5, index\_at\_5 = self.\_verify\_hit\_top\_n(item\_id, valid\_recs, 5)

hits\_at\_5\_count += hit\_at\_5

hit\_at\_10, index\_at\_10 = self.\_verify\_hit\_top\_n(item\_id, valid\_recs, 10)

hits\_at\_10\_count += hit\_at\_10

*#Recall is the rate of the interacted items that are ranked among the Top-N recommended items,*

*#when mixed with a set of non-relevant items*

recall\_at\_5 = hits\_at\_5\_count / float(interacted\_items\_count\_testset)

recall\_at\_10 = hits\_at\_10\_count / float(interacted\_items\_count\_testset)

person\_metrics = {'hits@5\_count':hits\_at\_5\_count,

'hits@10\_count':hits\_at\_10\_count,

'interacted\_count': interacted\_items\_count\_testset,

'recall@5': recall\_at\_5,

'recall@10': recall\_at\_10}

return person\_metrics

def evaluate\_model(self, model):

*#print('Running evaluation for users')*

people\_metrics = []

for idx, person\_id **in** enumerate(list(interactions\_test\_indexed\_df.index.unique().values)):

*#if idx % 100 == 0 and idx > 0:*

*# print('%d users processed' % idx)*

person\_metrics = self.evaluate\_model\_for\_user(model, person\_id)

person\_metrics['\_person\_id'] = person\_id

people\_metrics.append(person\_metrics)

print('**%d** users processed' % idx)

detailed\_results\_df = pd.DataFrame(people\_metrics) \

.sort\_values('interacted\_count', ascending=False)

global\_recall\_at\_5 = detailed\_results\_df['hits@5\_count'].sum() / float(detailed\_results\_df['interacted\_count'].sum())

global\_recall\_at\_10 = detailed\_results\_df['hits@10\_count'].sum() / float(detailed\_results\_df['interacted\_count'].sum())

global\_metrics = {'modelName': model.get\_model\_name(),

'recall@5': global\_recall\_at\_5,

'recall@10': global\_recall\_at\_10}

return global\_metrics, detailed\_results\_df

model\_evaluator = ModelEvaluator()

Popularity model

A common (and usually hard-to-beat) baseline approach is the Popularity model. This model is not actually personalized - it simply recommends to a user the most popular items that the user has not previously consumed. As the popularity accounts for the "wisdom of the crowds", it usually provides good recommendations, generally interesting for most people.  
Ps. The main objective of a recommender system is to leverage the long-tail items to the users with very specific interests, which goes far beyond this simple technique.

*#Computes the most popular items*

item\_popularity\_df = interactions\_full\_df.groupby('contentId')['eventStrength'].sum().sort\_values(ascending=False).reset\_index()

item\_popularity\_df.head(10)

| contentId | eventStrength |
| --- | --- |
| 0 | -4029704725707465084 | 307.733799 |
| 1 | -6783772548752091658 | 233.762157 |
| 2 | -133139342397538859 | 228.024567 |
| 3 | -8208801367848627943 | 197.107608 |
| 4 | -6843047699859121724 | 193.825208 |
| 5 | 8224860111193157980 | 189.044680 |
| 6 | -2358756719610361882 | 183.110951 |
| 7 | 2581138407738454418 | 180.282876 |
| 8 | 7507067965574797372 | 179.094002 |
| 9 | 1469580151036142903 | 170.548969 |

class **PopularityRecommender**:

MODEL\_NAME = 'Popularity'

def \_\_init\_\_(self, popularity\_df, items\_df=None):

self.popularity\_df = popularity\_df

self.items\_df = items\_df

def get\_model\_name(self):

return self.MODEL\_NAME

def recommend\_items(self, user\_id, items\_to\_ignore=[], topn=10, ver

bose=False):

*# Recommend the more popular items that the user hasn't seen yet.*

recommendations\_df = self.popularity\_df[~self.popularity\_df['contentId'].isin(items\_to\_ignore)] \

.sort\_values('eventStrength', ascending = False) \

.head(topn)

if verbose:

if self.items\_df **is** None:

raise **Exception**('"items\_df" is required in verbose mode')

recommendations\_df = recommendations\_df.merge(self.items\_df, how = 'left',

left\_on = 'contentId',

right\_on = 'contentId')[['eventStrength', 'contentId', 'title', 'url', 'lang']]

return recommendations\_df

popularity\_model = PopularityRecommender(item\_popularity\_df, articles\_df)

Here we perform the evaluation of the Popularity model, according to the method described above.  
It achieved the **Recall@5** of **0.2417**, which means that about **24%** of interacted items in test set were ranked by Popularity model among the top-5 items (from lists with 100 random items). And **Recall@10** was even higher (**37%**), as expected.  
It might be surprising to you that usually Popularity models could perform so well!

print('Evaluating Popularity recommendation model...')

pop\_global\_metrics, pop\_detailed\_results\_df = model\_evaluator.evaluate\_model(popularity\_model)

print('**\n**Global metrics:**\n%s**' % pop\_global\_metrics)

pop\_detailed\_results\_df.head(10)

Evaluating Popularity recommendation model..

1139 users processed

Global metrics:

{'modelName': 'Popularity', 'recall@5': 0.2417540271030427, 'recall@10': 0.37292252620813093}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | hits@5\_count | hits@10\_count | Interacted\_count | recall@5 | recall@10 | \_person\_id |
| 76 | 28 | 50 | 192 | 0.145833 | 0.260417 | 3609194402293569455 |
| 17 | 12 | 25 | 134 | 0.089552 | 0.186567 | -2626634673110551643 |
| 16 | 13 | 23 | 130 | 0.100000 | 0.176923 | -1032019229384696495 |
| 10 | 5 | 9 | 117 | 0.042735 | 0.076923 | -1443636648652872475 |
| 82 | 25 | 40 | 88 | 0.284091 | 0.454545 | -2979881261169775358 |
| 161 | 12 | 18 | 80 | 0.150000 | 0.225000 | -3596626804281480007 |
| 65 | 20 | 33 | 73 | 0.273973 | 0.452055 | 1116121227607581999 |
| 81 | 17 | 23 | 69 | 0.246377 | 0.333333 | 692689608292948411 |
| 106 | 14 | 18 | 69 | 0.202899 | 0.260870 | -9016528795238256703 |
| 52 | 21 | 28 | 68 | 0.308824 | 0.411765 | 3636910968448833585 |

# Content-Based Filtering model

Content-based filtering approaches leverage description or attributes from items the user has interacted to recommend similar items. It depends only on the user previous choices, making this method robust to avoid the cold-start problem. For textual items, like articles, news and books, it is simple to use the raw text to build item profiles and user profiles.  
Here we are using a very popular technique in information retrieval (search engines) named [TF-IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf). This technique converts unstructured text into a vector structure, where each word is represented by a position in the vector, and the value measures how relevant a given word is for an article. As all items will be represented in the same [Vector Space Model](https://en.wikipedia.org/wiki/Vector_space_model), it is to compute similarity between articles.  
See this [presentation](https://www.slideshare.net/gabrielspmoreira/discovering-users-topics-of-interest-in-recommender-systems-tdc-sp-2016) (from slide 30) for more information on TF-IDF and Cosine similarity.

*#Ignoring stopwords (words with no semantics) from English and Portuguese (as we have a corpus with mixed languages)*

stopwords\_list = stopwords.words('english') + stopwords.words('portuguese')

*#Trains a model whose vectors size is 5000, composed by the main unigrams and bigrams found in the corpus, ignoring stopwords*

vectorizer = TfidfVectorizer(analyzer='word',

ngram\_range=(1, 2),

min\_df=0.003,

max\_df=0.5,

max\_features=5000,

stop\_words=stopwords\_list)

item\_ids = articles\_df['contentId'].tolist()

tfidf\_matrix = vectorizer.fit\_transform(articles\_df['title'] + "" + articles\_df['text'])

tfidf\_feature\_names = vectorizer.get\_feature\_names()

tfidf\_matrix

<3047x5000 sparse matrix of type '<class 'numpy.float64'>'

with 638928 stored elements in Compressed Sparse Row format>

To model the user profile, we take all the item profiles the user has interacted and average them. The average is weighted by the interaction strength, in other words, the articles the user has interacted the most (eg. liked or commented) will have a higher strength in the final user profile.

def get\_item\_profile(item\_id):

idx = item\_ids.index(item\_id)

item\_profile = tfidf\_matrix[idx:idx+1]

return item\_profile

def get\_item\_profiles(ids):

item\_profiles\_list = [get\_item\_profile(x) for x **in** ids]

item\_profiles = scipy.sparse.vstack(item\_profiles\_list)

return item\_profiles

def build\_users\_profile(person\_id, interactions\_indexed\_df):

interactions\_person\_df = interactions\_indexed\_df.loc[person\_id]

user\_item\_profiles = get\_item\_profiles(interactions\_person\_df['contentId'])

user\_item\_strengths = np.array(interactions\_person\_df['eventStrength']).reshape(-1,1)

*#Weighted average of item profiles by the interactions strength*

user\_item\_strengths\_weighted\_avg = np.sum(user\_item\_profiles.multiply(user\_item\_strengths), axis=0) / np.sum(user\_item\_strengths)

user\_profile\_norm = sklearn.preprocessing.normalize(user\_item\_strengths\_weighted\_avg)

return user\_profile\_norm

def build\_users\_profiles():

interactions\_indexed\_df = interactions\_train\_df[interactions\_train\_df['contentId'] \

.isin(articles\_df['contentId'])].set\_index('personId')

user\_profiles = {}

for person\_id **in** interactions\_indexed\_df.index.unique():

user\_profiles[person\_id] = build\_users\_profile(person\_id, interactions\_indexed\_df)

return user\_profiles

user\_profiles = build\_users\_profiles()

len(user\_profiles)

1140

Let's take a look in the profile. It is a [unit vector](https://en.wikipedia.org/wiki/Unit_vector) of 5000 length. The value in each position represents how relevant is a token (unigram or bigram) for me.  
Looking my profile, it appears that the top relevant tokens really represent my professional interests in **machine learning**, **deep learning**, **artificial intelligence** and **google cloud platform**! So we might expect good recommendations here!

myprofile = user\_profiles[-1479311724257856983]

print(myprofile.shape)

pd.DataFrame(sorted(zip(tfidf\_feature\_names,

user\_profiles[-1479311724257856983].flatten().tolist()), key=lambda x: -x[1])[:20],

columns=['token', 'relevance'])

(1, 5000)

|  |  |  |
| --- | --- | --- |
|  | token | relevance |
| 0 | learning | 0.298732 |
| 1 | machine learning | 0.245992 |
| 2 | machine | 0.237843 |
| 3 | google | 0.202839 |
| 4 | data | 0.169776 |
| 5 | ai | 0.156203 |
| 6 | algorithms | 0.115666 |
| 7 | like | 0.097744 |
| 8 | language | 0.087609 |
| 9 | people | 0.082024 |
| 10 | deep | 0.081542 |
| 11 | deep learning | 0.080979 |
| 12 | research | 0.076020 |
| 13 | algorithm | 0.074905 |
| 14 | apple | 0.074050 |
| 15 | intelligence | 0.072663 |
| 16 | use | 0.072597 |
| 17 | human | 0.072494 |
| 18 | models | 0.072388 |
| 19 | artificial | 0.072062 |

class **ContentBasedRecommender**:

MODEL\_NAME = 'Content-Based'

def \_\_init\_\_(self, items\_df=None):

self.item\_ids = item\_ids

self.items\_df = items\_df

def get\_model\_name(self):

return self.MODEL\_NAME

def \_get\_similar\_items\_to\_user\_profile(self, person\_id, topn=1000):

*#Computes the cosine similarity between the user profile and all item profiles*

cosine\_similarities = cosine\_similarity(user\_profiles[person\_id], tfidf\_matrix)

*#Gets the top similar items*

similar\_indices = cosine\_similarities.argsort().flatten()[-topn:]

*#Sort the similar items by similarity*

similar\_items = sorted([(item\_ids[i], cosine\_similarities[0,i]) for i **in** similar\_indices], key=lambda x: -x[1])

return similar\_items

def recommend\_items(self, user\_id, items\_to\_ignore=[], topn=10, verbose=False):

similar\_items = self.\_get\_similar\_items\_to\_user\_profile(user\_id)

*#Ignores items the user has already interacted*

similar\_items\_filtered = list(filter(lambda x: x[0] **not** **in** items\_to\_ignore, similar\_items))

recommendations\_df = pd.DataFrame(similar\_items\_filtered, columns=['contentId', 'recStrength']) \

.head(topn)

if verbose:

if self.items\_df **is** None:

raise **Exception**('"items\_df" is required in verbose mode')

recommendations\_df = recommendations\_df.merge(self.items\_df, how = 'left',

left\_on = 'contentId',

right\_on = 'contentId')[['recStrength', 'contentId', 'title', 'url', 'lang']]

return recommendations\_df

content\_based\_recommender\_model = ContentBasedRecommender(articles\_df)

With personalized recommendations of content-based filtering model, we have a **Recall@5** to about **0.162**, which means that about **16%** of interacted items in test set were ranked by this model among the top-5 items (from lists with 100 random items). And **Recall@10** was **0.261 (52%)**. The lower performance of the Content-Based model compared to the Popularity model may indicate that users are not that fixed in content very similar to their previous reads.

print('Evaluating Content-Based Filtering model...')

cb\_global\_metrics, cb\_detailed\_results\_df = model\_evaluator.evaluate\_model(content\_based\_recommender\_model)

print('**\n**Global metrics:**\n%s**' % cb\_global\_metrics)

cb\_detailed\_results\_df.head(10)

Evaluating Content-Based Filtering model...

1139 users processed

Global metrics:

{'modelName': 'Content-Based', 'recall@5': 0.16287394528253643, 'recall@10': 0.2614420864229097}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | hits@5\_count | hits@10\_count | Interacted\_count | recall@5 | recall@5 | \_person\_id |
| 17 | 18 | 29 | 134 | 0.134328 | 0.216418 | -2626634673110551643 |
| 16 | 20 | 33 | 130 | 0.153846 | 0.253846 | -1032019229384696495 |
| 10 | 32 | 47 | 117 | 0.273504 | 0.401709 | -1443636648652872475 |
| 82 | 6 | 15 | 88 | 0.068182 | 0.170455 | -2979881261169775358 |
| 161 | 11 | 23 | 80 | 0.137500 | 0.287500 | -3596626804281480007 |
| 65 | 8 | 13 | 73 | 0.109589 | 0.178082 | 1116121227607581999 |
| 81 | 8 | 19 | 69 | 0.115942 | 0.275362 | 692689608292948411 |
| 106 | 3 | 9 | 69 | 0.043478 | 0.130435 | -9016528795238256703 |
| 52 | 3 | 8 | 68 | 0.044118 | 0.117647 | 3636910968448833585 |

# Collaborative Filtering model

Collaborative Filtering (CF) has two main implementation strategies:

* **Memory-based**: This approach uses the memory of previous users interactions to compute users similarities based on items they've interacted (user-based approach) or compute items similarities based on the users that have interacted with them (item-based approach).  
  A typical example of this approach is User Neighbourhood-based CF, in which the top-N similar users (usually computed using Pearson correlation) for a user are selected and used to recommend items those similar users liked, but the current user have not interacted yet. This approach is very simple to implement, but usually do not scale well for many users. A nice Python implementation of this approach in available in [Crab](http://muricoca.github.io/crab/).

* **Model-based**: This approach, models are developed using different machine learning algorithms to recommend items to users. There are many model-based CF algorithms, like neural networks, bayesian networks, clustering models, and latent factor models such as Singular Value Decomposition (SVD) and, probabilistic latent semantic analysis.

## Matrix Factorization

Latent factor models compress user-item matrix into a low-dimensional representation in terms of latent factors. One advantage of using this approach is that instead of having a high dimensional matrix containing abundant number of missing values we will be dealing with a much smaller matrix in lower-dimensional space.  
A reduced presentation could be utilized for either user-based or item-based neighborhood algorithms that are presented in the previous section. There are several advantages with this paradigm. It handles the sparsity of the original matrix better than memory based ones. Also comparing similarity on the resulting matrix is much more scalable especially in dealing with large sparse datasets.

Here we a use popular latent factor model named [Singular Value Decomposition (SVD)](https://en.wikipedia.org/wiki/Singular_value_decomposition). There are other matrix factorization frameworks more specific to CF you might try, like [surprise](https://github.com/NicolasHug/Surprise), [mrec](https://github.com/Mendeley/mrec) or [python-recsys](https://github.com/ocelma/python-recsys). We chose a [SciPy](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svds.html) implemenation of SVD because it is available on Kaggle kernels. P.s. See an example of SVD on a movies dataset in this [blog post](https://beckernick.github.io/matrix-factorization-recommender/).

An important decision is the number of factors to factor the user-item matrix. The higher the number of factors, the more precise is the factorization in the original matrix reconstructions. Therefore, if the model is allowed to memorize too much details of the original matrix, it may not generalize well for data it was not trained on. Reducing the number of factors increases the model generalization.

*#Creating a sparse pivot table with users in rows and items in columns*

users\_items\_pivot\_matrix\_df = interactions\_train\_df.pivot(index='personId',

columns='contentId',

values='eventStrength').fillna(0)

users\_items\_pivot\_matrix\_df.head(10)

| contentId | -9222795471790223670 | -9216926795620865886 | -9194572880052200111 | -9192549002213406534 | -9190737901804729417 | -9189659052158407108 | -9176143510534135851 | -9172673334835262304 | -9171475473795142532 | -9166778629773133902 | ... | 9191014301634017491 | 9207286802575546269 | 9208127165664287660 | 9209629151177723638 | 9209886322932807692 | 9213260650272029784 | 9215261273565326920 | 9217155070834564627 | 9220445660318725468 | 9222265156747237864 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| personId |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -9223121837663643404 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9212075797126931087 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9207251133131336884 | 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 |
| -9199575329909162940 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9196668942822132778 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9188188261933657343 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9172914609055320039 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9156344805277471150 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9120685872592674274 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| -9109785559521267180 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

users\_items\_pivot\_matrix = users\_items\_pivot\_matrix\_df.as\_matrix()

users\_items\_pivot\_matrix[:10]

/opt/conda/lib/python3.6/site-packages/ipykernel\_launcher.py:1: FutureWarning: Method .as\_matrix will be removed in a future version. Use .values instead.

"""Entry point for launching an IPython kernel.

Out[22]:

array([[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.]])

users\_ids = list(users\_items\_pivot\_matrix\_df.index)

users\_ids[:10]

[-9223121837663643404,

-9212075797126931087,

-9207251133131336884,

-9199575329909162940,

-9196668942822132778,

-9188188261933657343,

-9172914609055320039,

-9156344805277471150,

-9120685872592674274,

-9109785559521267180]

In [24]:

users\_items\_pivot\_sparse\_

users\_items\_pivot\_sparse\_matrix = csr\_matrix(users\_items\_pivot\_matrix)

users\_items\_pivot\_sparse\_matrix

<1140x2926 sparse matrix of type '<class 'numpy.float64'>'

with 31284 stored elements in Compressed Sparse Row format>

*#The number of factors to factor the user-item matrix.*

NUMBER\_OF\_FACTORS\_MF = 15

*#Performs matrix factorization of the original user item matrix*

*#U, sigma, Vt = svds(users\_items\_pivot\_matrix, k = NUMBER\_OF\_FACTORS\_MF)*

U, sigma, Vt = svds(users\_items\_pivot\_sparse\_matrix, k = NUMBER\_OF\_FACTORS\_MF)

U.shape

(1140, 15)

Vt.shape

(15, 2926)

sigma = np.diag(sigma)

sigma.shape

(15, 15)

After the factorization, we try to to reconstruct the original matrix by multiplying its factors. The resulting matrix is not sparse any more. It was generated predictions for items the user have not yet interaction, which we will exploit for recommendations.

all\_user\_predicted\_ratings = np.dot(np.dot(U, sigma), Vt)

all\_user\_predicted\_ratings

array([[ 0.01039915, 0.00081872, -0.01725263, ..., 0.00140708,

0.0110647 , 0.00226063],

[-0.00019285, -0.00031318, -0.00264624, ..., 0.00251658,

0.00017609, -0.00189488],

[-0.01254721, 0.0065947 , -0.00590676, ..., 0.00698975,

-0.01015696, 0.01154572],

...,

[-0.02995379, 0.00805715, -0.01846307, ..., -0.01083078,

-0.00118591, 0.0096798 ],

[-0.01845505, 0.00467019, 0.01219602, ..., 0.00409507,

0.00019482, -0.00752562],

[-0.01506374, 0.00327732, 0.13391269, ..., -0.01191815,

0.06422074, 0.01303244]])

all\_user\_predicted\_ratings\_norm = (all\_user\_predicted\_ratings - all\_user\_predicted\_ratings.min()) / (all\_user\_predicted\_ratings.max() - all\_user\_predicted\_ratings.min())

*#Converting the reconstructed matrix back to a Pandas dataframe*

cf\_preds\_df = pd.DataFrame(all\_user\_predicted\_ratings\_norm, columns = users\_items\_pivot\_matrix\_df.columns, index=users\_ids).transpose()

cf\_preds\_df.head(10)

| -9223121837663643404 | -9212075797126931087 | -9207251133131336884 | -9199575329909162940 | -9196668942822132778 | -9188188261933657343 | -9172914609055320039 | -9156344805277471150 | -9120685872592674274 | -9109785559521267180 | ... | 9105269044962898535 | 9109075639526981934 | 9135582630122950040 | 9137372837662939523 | 9148269800512008413 | 9165571805999894845 | 9187866633451383747 | 9191849144618614467 | 9199170757466086545 | 9210530975708218054 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| contentId |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -9222795471790223670 | 0.139129 | 0.137930 | 0.136531 | 0.143948 | 0.136815 | 0.137339 | 0.137508 | 0.143534 | 0.136428 | 0.135681 | ... | 0.137351 | 0.127822 | 0.137946 | 0.139653 | 0.140324 | 0.136888 | 0.135787 | 0.134560 | 0.135862 | 0.136246 |
| -9216926795620865886 | 0.138044 | 0.137916 | 0.138698 | 0.137878 | 0.137969 | 0.137990 | 0.137974 | 0.138049 | 0.138217 | 0.138151 | ... | 0.137962 | 0.139527 | 0.138009 | 0.138117 | 0.139634 | 0.138058 | 0.138222 | 0.138864 | 0.138480 | 0.138323 |
| -9194572880052200111 | 0.135998 | 0.137652 | 0.137283 | 0.137536 | 0.140363 | 0.137807 | 0.141246 | 0.136284 | 0.135301 | 0.138512 | ... | 0.139257 | 0.143161 | 0.139139 | 0.140077 | 0.154976 | 0.140109 | 0.140654 | 0.135861 | 0.139332 | 0.153114 |
| -9192549002213406534 | 0.141924 | 0.137996 | 0.134663 | 0.137080 | 0.139946 | 0.138574 | 0.139473 | 0.144469 | 0.143333 | 0.138428 | ... | 0.140233 | 0.167426 | 0.138849 | 0.137037 | 0.141820 | 0.139260 | 0.139513 | 0.136804 | 0.140862 | 0.148793 |
| -9190737901804729417 | 0.140209 | 0.137408 | 0.138708 | 0.138672 | 0.137725 | 0.138218 | 0.138390 | 0.138418 | 0.134883 | 0.140193 | ... | 0.138373 | 0.138459 | 0.138169 | 0.137990 | 0.134041 | 0.137820 | 0.138100 | 0.138286 | 0.138630 | 0.136178 |
| -9189659052158407108 | 0.138932 | 0.138699 | 0.138117 | 0.137621 | 0.138920 | 0.137766 | 0.138568 | 0.138200 | 0.140572 | 0.140365 | ... | 0.140725 | 0.148152 | 0.137645 | 0.138165 | 0.149152 | 0.138912 | 0.139595 | 0.139807 | 0.140419 | 0.145698 |
| -9176143510534135851 | 0.143208 | 0.138673 | 0.139514 | 0.139114 | 0.137664 | 0.137447 | 0.139833 | 0.140564 | 0.144698 | 0.144440 | ... | 0.138367 | 0.146220 | 0.136204 | 0.138087 | 0.137317 | 0.137917 | 0.138546 | 0.142601 | 0.141431 | 0.142154 |
| -9172673334835262304 | 0.138527 | 0.138021 | 0.138274 | 0.137827 | 0.137997 | 0.138037 | 0.138104 | 0.138259 | 0.137633 | 0.138397 | ... | 0.138588 | 0.140146 | 0.138013 | 0.137839 | 0.137033 | 0.137969 | 0.138337 | 0.138361 | 0.138813 | 0.137538 |
| -9171475473795142532 | 0.140720 | 0.137865 | 0.138061 | 0.137633 | 0.138231 | 0.138089 | 0.139009 | 0.137552 | 0.137143 | 0.140581 | ... | 0.139046 | 0.139895 | 0.138000 | 0.137958 | 0.136061 | 0.138183 | 0.138817 | 0.138060 | 0.139205 | 0.137198 |
| -9166778629773133902 | 0.138989 | 0.137725 | 0.136520 | 0.137723 | 0.138559 | 0.137951 | 0.138189 | 0.138496 | 0.139470 | 0.137546 | ... | 0.138233 | 0.144002 | 0.138050 | 0.137533 | 0.139036 | 0.138399 | 0.138330 | 0.137148 | 0.138027 | 0.140283 |

len(cf\_preds\_df.columns)

1140

class **CFRecommender**:

MODEL\_NAME = 'Collaborative Filtering'

def \_\_init\_\_(self, cf\_predictions\_df, items\_df=None):

self.cf\_predictions\_df = cf\_predictions\_df

self.items\_df = items\_df

def get\_model\_name(self):

return self.MODEL\_NAME

def recommend\_items(self, user\_id, items\_to\_ignore=[], topn=10, verbose=False):

*# Get and sort the user's predictions*

sorted\_user\_predictions = self.cf\_predictions\_df[user\_id].sort\_values(ascending=False) \

.reset\_index().rename(columns={user\_id: 'recStrength'})

*# Recommend the highest predicted rating movies that the user hasn't seen yet.*

recommendations\_df = sorted\_user\_predictions[~sorted\_user\_predictions['contentId'].isin(items\_to\_ignore)] \

.sort\_values('recStrength', ascending = False) \

.head(topn)

if verbose:

if self.items\_df **is** None:

raise **Exception**('"items\_df" is required in verbose mode')

recommendations\_df = recommendations\_df.merge(self.items\_df, how = 'left',

left\_on = 'contentId',

right\_on = 'contentId')[['recStrength', 'contentId', 'title', 'url', 'lang']]

return recommendations\_df

cf\_recommender\_model = CFRecommender(cf\_preds\_df, articles\_df)

Evaluating the Collaborative Filtering model (SVD matrix factorization), we observe that we got **Recall@5 (33%)** and **Recall@10 (46%)** values, much higher than Popularity model and Content-Based model.

print('Evaluating Collaborative Filtering (SVD Matrix Factorization) model...')

cf\_global\_metrics, cf\_detailed\_results\_df = model\_evaluator.evaluate\_model(cf\_recommender\_model)

print('**\n**Global metrics:**\n%s**' % cf\_global\_metrics)

cf\_detailed\_results\_df.head(10)

Evaluating Collaborative Filtering (SVD Matrix Factorization) model...

1139 users processed

Global metrics:

{'modelName': 'Collaborative Filtering', 'recall@5': 0.33405778573254924, 'recall@10': 0.46816670928151366}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | hits@5\_count | hits@10\_count | Interacted\_count | recall@5 | recall@10 | \_person\_id |
| 76 | 21 | 45 | 192 | 0.109375 | 0.234375 | 3609194402293569455 |
| 17 | 30 | 56 | 134 | 0.223881 | 0.417910 | -2626634673110551643 |
| 16 | 16 | 34 | 130 | 0.123077 | 0.261538 | -1032019229384696495 |
| 10 | 38 | 51 | 117 | 0.324786 | 0.435897 | -1443636648652872475 |
| 82 | 39 | 48 | 88 | 0.443182 | 0.545455 | -2979881261169775358 |
| 161 | 22 | 34 | 80 | 0.275000 | 0.425000 | -3596626804281480007 |
| 65 | 24 | 32 | 73 | 0.328767 | 0.438356 | 1116121227607581999 |
| 81 | 16 | 21 | 69 | 0.231884 | 0.304348 | 692689608292948411 |
| 106 | 20 | 28 | 69 | 0.289855 | 0.405797 | -9016528795238256703 |
| 52 | 23 | 30 | 68 | 0.338235 | 0.441176 | 3636910968448833585 |

## Hybrid Recommender

What if we combine Collaborative Filtering and Content-Based Filtering approaches?  
Would that provide us with more accurate recommendations?  
In fact, hybrid methods have performed better than individual approaches in many studies and have being extensively used by researchers and practioners.  
Let's build a simple hybridization method, as an ensemble that takes the weighted average of the normalized CF scores with the Content-Based scores, and ranking by resulting score. In this case, as the CF model is much more accurate than the CB model, the weights for the CF and CB models are 100.0 and 1.0, respectivelly.

class **HybridRecommender**:

MODEL\_NAME = 'Hybrid'

def \_\_init\_\_(self, cb\_rec\_model, cf\_rec\_model, items\_df, cb\_ensemble\_weight=1.0, cf\_ensemble\_weight=1.0):

self.cb\_rec\_model = cb\_rec\_model

self.cf\_rec\_model = cf\_rec\_model

self.cb\_ensemble\_weight = cb\_ensemble\_weight

self.cf\_ensemble\_weight = cf\_ensemble\_weight

self.items\_df = items\_df

def get\_model\_name(self):

return self.MODEL\_NAME

def recommend\_items(self, user\_id, items\_to\_ignore=[], topn=10, verbose=False):

*#Getting the top-1000 Content-based filtering recommendations*

cb\_recs\_df = self.cb\_rec\_model.recommend\_items(user\_id, items\_to\_ignore=items\_to\_ignore, verbose=verbose,

topn=1000).rename(columns={'recStrength': 'recStrengthCB'})

*#Getting the top-1000 Collaborative filtering recommendations*

cf\_recs\_df = self.cf\_rec\_model.recommend\_items(user\_id, items\_to\_ignore=items\_to\_ignore, verbose=verbose,

topn=1000).rename(columns={'recStrength': 'recStrengthCF'})

*#Combining the results by contentId*

recs\_df = cb\_recs\_df.merge(cf\_recs\_df,

how = 'outer',

left\_on = 'contentId',

right\_on = 'contentId').fillna(0.0)

*#Computing a hybrid recommendation score based on CF and CB scores*

*#recs\_df['recStrengthHybrid'] = recs\_df['recStrengthCB'] \* recs\_df['recStrengthCF']*

recs\_df['recStrengthHybrid'] = (recs\_df['recStrengthCB'] \* self.cb\_ensemble\_weight) \

+ (recs\_df['recStrengthCF'] \* self.cf\_ensemble\_weight)

*#Sorting recommendations by hybrid score*

recommendations\_df = recs\_df.sort\_values('recStrengthHybrid', ascending=False).head(topn)

if verbose:

if self.items\_df **is** None:

raise **Exception**('"items\_df" is required in verbose mode')

recommendations\_df = recommendations\_df.merge(self.items\_df, how = 'left',

left\_on = 'contentId',

right\_on = 'contentId')[['recStrengthHybrid', 'contentId', 'title', 'url', 'lang']]

return recommendations\_df

hybrid\_recommender\_model = HybridRecommender(content\_based\_recommender\_model, cf\_recommender\_model, articles\_df,

cb\_ensemble\_weight=1.0, cf\_ensemble\_weight=100.0)

Our simple hybrid approach surpasses Content-Based filtering with its combination with Collaborative Filtering. Now we have a **Recall@5** of **34.2%** and **Recall@10** of **47.9%**

print('Evaluating Hybrid model...')

hybrid\_global\_metrics, hybrid\_detailed\_results\_df = model\_evaluator.evaluate\_model(hybrid\_recommender\_model)

print('**\n**Global metrics:**\n%s**' % hybrid\_global\_metrics)

hybrid\_detailed\_results\_df.head(10)

Evaluating Hybrid model...

1139 users processed

Global metrics:

{'modelName': 'Hybrid', 'recall@5': 0.34275121452313984, 'recall@10': 0.4796727179749425}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | hits@5\_count | hits@10\_count | interacted\_count | recall@5 | recall@10 | \_person\_id |
| 76 | 22 | 45 | 192 | 0.114583 | 0.234375 | 3609194402293569455 |
| 17 | 31 | 58 | 134 | 0.231343 | 0.432836 | -2626634673110551643 |
| 16 | 21 | 37 | 130 | 0.161538 | 0.284615 | -1032019229384696495 |
| 10 | 40 | 51 | 117 | 0.341880 | 0.435897 | -1443636648652872475 |
| 82 | 38 | 50 | 88 | 0.431818 | 0.568182 | -2979881261169775358 |
| 161 | 23 | 35 | 80 | 0.287500 | 0.437500 | -3596626804281480007 |
| 65 | 23 | 32 | 73 | 0.315068 | 0.438356 | 1116121227607581999 |
| 81 | 16 | 21 | 69 | 0.231884 | 0.304348 | 692689608292948411 |
| 106 | 20 | 27 | 69 | 0.289855 | 0.391304 | -9016528795238256703 |
| 52 | 22 | 29 | 68 | 0.323529 | 0.426471 | 3636910968448833585 |

## Comparing the methods

global\_metrics\_df = pd.DataFrame([cb\_global\_metrics, pop\_global\_metrics, cf\_global\_metrics, hybrid\_global\_metrics]) \

.set\_index('modelName')

global\_metrics\_df

|  |  |  |
| --- | --- | --- |
|  | recall@5 | recall@10 |
| Model Name |  |  |
| Content-Based | 0.162874 | 0.261442 |
| Popularity | 0.241754 | 0.372923 |
| Collaborative Filtering | 0.334058 | 0.468167 |
| Hybrid | 0.342751 | 0.479673 |

%matplotlib inline

ax = global\_metrics\_df.transpose().plot(kind='bar', figsize=(15,8))

for p **in** ax.patches:

ax.annotate("**%.3f**" % p.get\_height(), (p.get\_x() + p.get\_width() / 2., p.get\_height()), ha='center', va='center', xytext=(0, 10), textcoords='offset points')

Testing

Let's test the best model (Hybrid) for my user.

def inspect\_interactions(person\_id, test\_set=True):

if test\_set:

interactions\_df = interactions\_test\_indexed\_df

else:

interactions\_df = interactions\_train\_indexed\_df

return interactions\_df.loc[person\_id].merge(articles\_df, how = 'left',

left\_on = 'contentId',

right\_on = 'contentId') \

.sort\_values('eventStrength', ascending = False)[['eventStrength',

'contentId',

'title', 'url', 'lang']]

Here we see some articles I interacted in Deskdrop from train set. It can be easily observed that among my main interests are **machine learning**, **deep learning**, **artificial intelligence**, and **google cloud platform**.

inspect\_interactions(-1479311724257856983, test\_set=False).head(20)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15 | 4.285402 | 7342707578347442862 | At eBay, Machine Learning is Driving Innovativ... | https://www.ebayinc.com/stories/news/at-ebay-m... | en |
| 38 | 4.129283 | 621816023396605502 | AI Is Here to Help You Write Emails People Wil... | http://www.wired.com/2016/08/boomerang-using-a... | en |
| 8 | 4.044394 | -4460374799273064357 | Deep Learning for Chatbots, Part 1 - Introduction | http://www.wildml.com/2016/04/deep-learning-fo... | en |
| 116 | 3.954196 | -7959318068735027467 | Auto-scaling scikit-learn with Spark | https://databricks.com/blog/2016/02/08/auto-sc... | en |
| 10 | 3.906891 | 2589533162305407436 | 6 reasons why I like KeystoneML | http://radar.oreilly.com/2015/07/6-reasons-why... | en |
| 28 | 3.700440 | 5258604889412591249 | Machine Learning Is No Longer Just for Experts | https://hbr.org/2016/10/machine-learning-is-no... | en |
| 6 | 3.700440 | -398780385766545248 | 10 Stats About Artificial Intelligence That Wi... | http://www.fool.com/investing/2016/06/19/10-st... | en |
| 113 | 3.643856 | -6467708104873171151 | 5 reasons your employees aren't sharing their ... | http://justcuriousblog.com/2016/04/5-reasons-y... | en |
| 42 | 3.523562 | -4944551138301474550 | Algorithms and architecture for job recommenda... | https://www.oreilly.com/ideas/algorithms-and-a... | en |
| 43 | 3.459432 | -8377626164558006982 | Bad Writing Is Destroying Your Company's Produ... | https://hbr.org/2016/09/bad-writing-is-destroy... | en |
| 41 | 3.459432 | 444378495316508239 | How to choose algorithms for Microsoft Azure M... | https://azure.microsoft.com/en-us/documentatio... | en |
| 3 | 3.321928 | 2468005329717107277 | How Netflix does A/B Testing - uxdesign.cc - U... | https://uxdesign.cc/how-netflix-does-a-b-testi... | en |
| 101 | 3.321928 | -8085935119790093311 | Graph Capabilities with the Elastic Stack | https://www.elastic.co/webinars/sneak-peek-of-... | en |
| 107 | 3.169925 | -1429167743746492970 | Building with Watson Technical Web Series | https://www-304.ibm.com/partnerworld/wps/servl... | pt |
| 16 | 3.169925 | 6340108943344143104 | Text summarization with TensorFlow | https://research.googleblog.com/2016/08/text-s... | en |
| 49 | 3.169925 | 1525777409079968377 | Probabilistic Programming | http://probabilistic-programming.org/wiki/Home | en |
| 44 | 3.169925 | -5756697018315640725 | Being A Developer After 40 - Free Code Camp | https://medium.freecodecamp.com/being-a-develo... | en |
| 97 | 3.087463 | 2623290164732957912 | Creative Applications of Deep Learning with Te... | https://www.kadenze.com/courses/creative-appli... | en |
| 32 | 3.000000 | 279771472506428952 | 5 Unique Features Of Google Compute Engine Tha... | http://www.forbes.com/sites/janakirammsv/2016/... | en |
| 78 | 2.906891 | -3920124114454832425 | Worldwide Ops in Minutes with DataStax & Cloud | http://www.datastax.com/2016/01/datastax-enter... | en |

**The recommendations really matches my interests, as I would read all of them!**

hybrid\_recommender\_model.recommend\_items(-1479311724257856983, topn=20, verbose=True)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 25.436876 | 3269302169678465882 | The barbell effect of machine learning. | http://techcrunch.com/2016/06/02/the-barbell-e... | en |
| 1 | 25.369932 | -8085935119790093311 | Graph Capabilities with the Elastic Stack | https://www.elastic.co/webinars/sneak-peek-of-... | en |
| 2 | 24.493428 | 1005751836898964351 | Seria Stranger Things uma obra de arte do algo... | https://www.linkedin.com/pulse/seria-stranger-... | pt |
| 3 | 24.382997 | -8377626164558006982 | Bad Writing Is Destroying Your Company's Produ... | https://hbr.org/2016/09/bad-writing-is-destroy... | en |
| 4 | 24.362064 | -6727357771678896471 | This Super Accurate Portrait Selection Tech Us... | http://petapixel.com/2016/06/29/super-accurate... | en |
| 5 | 24.190327 | -8190931845319543363 | Machine Learning Is At The Very Peak Of Its Hy... | https://arc.applause.com/2016/08/17/gartner-hy... | en |
| 6 | 24.172285 | 7395435905985567130 | The AI business landscape | https://www.oreilly.com/ideas/the-ai-business-... | en |
| 7 | 23.932289 | 5092635400707338872 | Power to the People: How One Unknown Group of ... | https://medium.com/@atduskgreg/power-to-the-pe... | en |
| 8 | 23.865716 | -5253644367331262405 | Hello, TensorFlow! | https://www.oreilly.com/learning/hello-tensorflow | en |
| 9 | 23.811519 | 1549650080907932816 | Spark comparison: AWS vs. GCP | https://www.oreilly.com/ideas/spark-comparison... | en |
| 10 | 23.537832 | 621816023396605502 | AI Is Here to Help You Write Emails People Wil... | http://www.wired.com/2016/08/boomerang-using-a... | en |
| 11 | 23.195716 | -1901742495252324928 | Designing smart notifications | https://medium.com/@intercom/designing-smart-n... | en |
| 12 | 23.101347 | 882422233694040097 | Infográfico: Algoritmos para Aprendizado de Má... | https://www.infoq.com/br/news/2016/07/infograf... | pt |
| 13 | 22.725769 | 2468005329717107277 | How Netflix does A/B Testing - uxdesign.cc - U... | https://uxdesign.cc/how-netflix-does-a-b-testi... | en |
| 14 | 22.561032 | -5756697018315640725 | Being A Developer After 40 - Free Code Camp | https://medium.freecodecamp.com/being-a-develo... | en |
| 15 | 22.448418 | -4944551138301474550 | Algorithms and architecture for job recommenda... | https://www.oreilly.com/ideas/algorithms-and-a... | en |
| 16 | 22.342822 | 1415230502586719648 | Machine Learning Is Redefining The Enterprise ... | http://www.forbes.com/sites/louiscolumbus/2016... | en |
| 17 | 22.311658 | -8771338872124599367 | Funcionários do mês no CoolHow: os Slackbots -... | https://medium.com/coolhow-creative-lab/funcio... | pt |
| 18 | 22.278853 | 5258604889412591249 | Machine Learning Is No Longer Just for Experts | https://hbr.org/2016/10/machine-learning-is-no... | en |
| 19 | 22.239822 | -5027816744653977347 | Apple acquires Turi, a machine learning company | https://techcrunch.com/2016/08/05/apple-acquir... | en |

# Conclusion

It could be observed that for articles recommendation, content-based filtering and a hybrid method performed better than Collaborative Filtering alone.

There is large room for improvements of the results. Here are some tips:

* In this example, we've completely ignored the time, considering that all articles were available to be recommended to users at any time. A better approach would be to filter only articles that were available for users at a given time.
* You could leverage the available contextual information to model users preferences across time (period of day, day of week, month), location (country and state/district) and devices (browser, mobile native app).  
  This contextual information can be easily incorporated in [Learn-to-Rank](https://en.wikipedia.org/wiki/Learning_to_rank) models (like XGBoost Gradient Boosting Decision Trees with ranking objective), Logistic models (with categorical features [One-Hot encoded](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) or [Feature Hashed](https://en.wikipedia.org/wiki/Feature_hashing)), and [Wide & Deep models](https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html), which is implemented in [TensorFlow](https://docs.w3cub.com/tensorflow~guide/tutorials/wide_and_deep/). Take a look in the summary my solution shared for [Outbrain Click Prediction](https://www.kaggle.com/c/outbrain-click-prediction/discussion/27897#157215) competition.
* Those basic techniques were used for didactic purposes. There are more advanced techniques in RecSys research community, specially advanced Matrix Factorization and Deep Learning models.

You can know more about state-of-the-art methods published in Recommender Systems on [ACM RecSys conference](https://recsys.acm.org/).  
If you are more like practioner than researcher, you might try some Collaborative Filtering frameworks in this dataset, like [surprise](https://github.com/NicolasHug/Surprise), [mrec](https://github.com/Mendeley/mrec), [python-recsys](https://github.com/ocelma/python-recsys) and [Spark ALS Matrix Factorization](https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html) (distributed implementation for large datasets).  
Take a look in this [presentation](https://www.slideshare.net/gabrielspmoreira/discovering-users-topics-of-interest-in-recommender-systems-tdc-sp-2016) where I describe a production recommender system, focused on Content-Based Filtering and Topic Modeling techniques.

The

End