CDS513 - Predictive Business Analytics

Recommender Systems (Hybrid)

This notebook is a practical introduction to the main [Recommender System (https://en.wikipedia.org/wiki/Recommender_system)](https://en.wikipedia.org/wiki/Recommender_system) (RecSys) techniques. The objective of a RecSys is to recommend relevant items for users, based on their preference. Preference and relevance are subjective, and they are generally inferred by items users have consumed previously.
The main families of methods for RecSys are:

- **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** (http://recommender-systems.org/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.

In this notebook, we use a dataset we've shared on Kaggle Datasets: Articles Sharing and Reading from CI&T Deskdrop (https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop). 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.

In [1]:
```
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)](https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop), which contains a real sample of 12 months logs (Mar. 2016 - Feb. 2017) from CI&T's Internal Communication platform (DeskDrop). It contains about 73k logged users interactions on more than 3k public 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 exercice for you.

In [2]:
```
articles_df = pd.read_csv('C:/Users/USER/Desktop/shared_articles.csv/shared_articles.csv')
articles_df = articles_df[articles_df['eventType'] == 'CONTENT SHARED']
articles_df.head(5)
```

Out[2]:

	timestamp	eventType	contentId	authorPersonId	authorSessionId	authorUserAgent	authorRegion	authorCountry	conte
1	1459193988	CONTENT SHARED	-4110354420726924665	4340306774493623681	8940341205206233829	NaN	NaN	NaN	
2	1459194146	CONTENT SHARED	-7292285110016212249	4340306774493623681	8940341205206233829	NaN	NaN	NaN	
3	1459194474	CONTENT SHARED	-6151852268067518688	3891637997717104548	-1457532940883382585	NaN	NaN	NaN	
4	1459194497	CONTENT SHARED	2448026894306402386	4340306774493623681	8940341205206233829	NaN	NaN	NaN	
5	1459194522	CONTENT SHARED	-2826566343807132236	4340306774493623681	8940341205206233829	NaN	NaN	NaN	

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.

In [3]:
```
interactions_df = pd.read_csv('C:/Users/USER/Desktop/users_interactions.csv/users_interactions.csv')
interactions_df.head(10)
```

Out[3]:

	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.

In [4]:
```
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.

In [5]:
```
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()[['person
print('# users with at least 5 interactions: %d' % len(users_with_enough_interactions_df))
```

```
# users: 1895
# users with at least 5 interactions: 1140
```

In [6]:
```
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.

In [7]:
```
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
```

Out[7]:

	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.

In [8]:
```
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/)](http://fastml.com/evaluating-recommender-systems/).

In [9]:
```
#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')
```

In [10]:
```
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])
```

In [11]:
```
#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_I
                                                                              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-intera
            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'].su
        global_recall_at_10 = detailed_results_df['hits@10_count'].sum() / float(detailed_results_df['interacted_count']

        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()
```

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)](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)](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)](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.

In [12]:
```
import nltk
nltk.download('stopwords')
#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 stopw
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
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\USER\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
C:\Users\USER\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is
deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instea
d.
  warnings.warn(msg, category=FutureWarning)
```

Out[12]:
```
<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.

In [13]:
```
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_it
    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
```

In [14]:
```
import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=DeprecationWarning)

user_profiles = build_users_profiles()
len(user_profiles)
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\utils\validation.py:593: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html (https://numpy.org/doc/stable/reference/generated/numpy.matrix.html)
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\utils\validation.py:593: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html (https://numpy.org/doc/stable/reference/generated/numpy.matrix.html)
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\utils\validation.py:593: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html (https://numpy.org/doc/stable/reference/generated/numpy.matrix.html)
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\utils\validation.py:593: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html (https://numpy.org/doc/stable/reference/generated/numpy.matrix.html)
```

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!

In [15]:
```
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)
```

Out[15]:

	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

In [16]:
```
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',


        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.

In [17]:
```
import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=DeprecationWarning)

print('Evaluating Content-Based Filtering model...')
cb_global_metrics, cb_detailed_results_df = model_evaluator.evaluate_model(content_based_recommender_model)
print('\nGlobal metrics:\n%s' % cb_global_metrics)
cb_detailed_results_df.head(10)
```

```
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
```

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/)](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)](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)](https://github.com/NicolasHug/Surprise), [mrec (https://github.com/Mendeley/mrec)](https://github.com/Mendeley/mrec) or [python-recsys (https://github.com/ocelma/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)](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/)](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.

In [18]:
```
#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)
```

Out[18]:

contentId personId	-9222795471790223670	-9216926795620865886	-9194572880052200111	-9192549002213406534	-9190737901804729417	-918965905
-9223121837663643404	0.0	0.0	0.0	0.0	0.0	
-9212075797126931087	0.0	0.0	0.0	0.0	0.0	
-9207251133131336884	0.0	2.0	0.0	0.0	0.0	
-9199575329909162940	0.0	0.0	0.0	0.0	0.0	
-9196668942822132778	0.0	0.0	0.0	0.0	0.0	
-9188188261933657343	0.0	0.0	0.0	0.0	0.0	
-9172914609055320039	0.0	0.0	0.0	0.0	0.0	
-9156344805277471150	0.0	0.0	0.0	0.0	0.0	
-9120685872592674274	0.0	0.0	0.0	0.0	0.0	
-9109785559521267180	0.0	0.0	0.0	0.0	0.0	

10 rows × 2926 columns

In [19]:
```
users_items_pivot_matrix = users_items_pivot_matrix_df.values
users_items_pivot_matrix[:10]
```

Out[19]:
```
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.]])
```

In [20]:
```
users_ids = list(users_items_pivot_matrix_df.index)
users_ids[:10]
```

Out[20]:
```
[-9223121837663643404,
 -9212075797126931087,
 -9207251133131336884,
 -9199575329909162940,
 -9196668942822132778,
 -9188188261933657343,
 -9172914609055320039,
 -9156344805277471150,
 -9120685872592674274,
 -9109785559521267180]
```

In [21]:
```
users_items_pivot_sparse_matrix = csr_matrix(users_items_pivot_matrix)
users_items_pivot_sparse_matrix
```

Out[21]:
```
<1140x2926 sparse matrix of type '<class 'numpy.float64'>'
	with 31284 stored elements in Compressed Sparse Row format>
```

In [22]:
```
#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)
```

In [23]:
```
U.shape
```
Out[23]:
```
(1140, 15)
```

In [24]:
```
Vt.shape
```
Out[24]:
```
(15, 2926)
```

In [25]:
```
sigma = np.diag(sigma)
sigma.shape
```
Out[25]:
```
(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.

In [26]:
```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt)
all_user_predicted_ratings
```
Out[26]:
```
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]])
```

In [27]:
```
all_user_predicted_ratings_norm = (all_user_predicted_ratings - all_user_predicted_ratings.min()) / (all_user_predicted_
```

In [28]:
```
#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_i
cf_preds_df.head(10)
```

Out[28]:

contentId	-9223121837663643404	-9212075797126931087	-9207251133131336884	-9199575329909162940	-9196668942822132778	-918818826
-9222795471790223670	0.139129	0.137930	0.136531	0.143948	0.136815	
-9216926795620865886	0.138044	0.137916	0.138698	0.137878	0.137969	
-9194572880052200111	0.135998	0.137652	0.137283	0.137536	0.140363	
-9192549002213406534	0.141924	0.137996	0.134663	0.137080	0.139946	
-9190737901804729417	0.140209	0.137408	0.138708	0.138672	0.137725	
-9189659052158407108	0.138932	0.138699	0.138117	0.137621	0.138920	
-9176143510534135851	0.143208	0.138673	0.139514	0.139114	0.137664	
-9172673334835262304	0.138527	0.138021	0.138274	0.137827	0.137997	
-9171475473795142532	0.140720	0.137865	0.138061	0.137633	0.138231	
-9166778629773133902	0.138989	0.137725	0.136520	0.137723	0.138559	

10 rows × 1140 columns

In [29]:
```
len(cf_preds_df.columns)
```

Out[29]: 1140

In [30]:
```
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',


        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.

In [31]:
```
print('Evaluating Collaborative Filtering (SVD Matrix Factorization) model...')
cf_global_metrics, cf_detailed_results_df = model_evaluator.evaluate_model(cf_recommender_model)
print('\nGlobal metrics:\n%s' % cf_global_metrics)
cf_detailed_results_df.head(10)
```

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

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.

In [32]:
```
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', 'ti


        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)
```

We have a new champion!
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%**

In [33]:
```
import warnings

with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=DeprecationWarning)

print('Evaluating Hybrid model...')
hybrid_global_metrics, hybrid_detailed_results_df = model_evaluator.evaluate_model(hybrid_recommender_model)
print('\nGlobal metrics:\n%s' % hybrid_global_metrics)
hybrid_detailed_results_df.head(10)
```

```
C:\Users\USER\AppData\Local\Temp/ipykernel_1660/3102219382.py:13: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  non_interacted_items_sample = random.sample(non_interacted_items, sample_size)

Evaluating Hybrid model...
```

Comparing the methods

In [34]:
```
global_metrics_df = pd.DataFrame([cb_global_metrics, cf_global_metrics, hybrid_global_metrics]) \
                        .set_index('modelName')
global_metrics_df
```

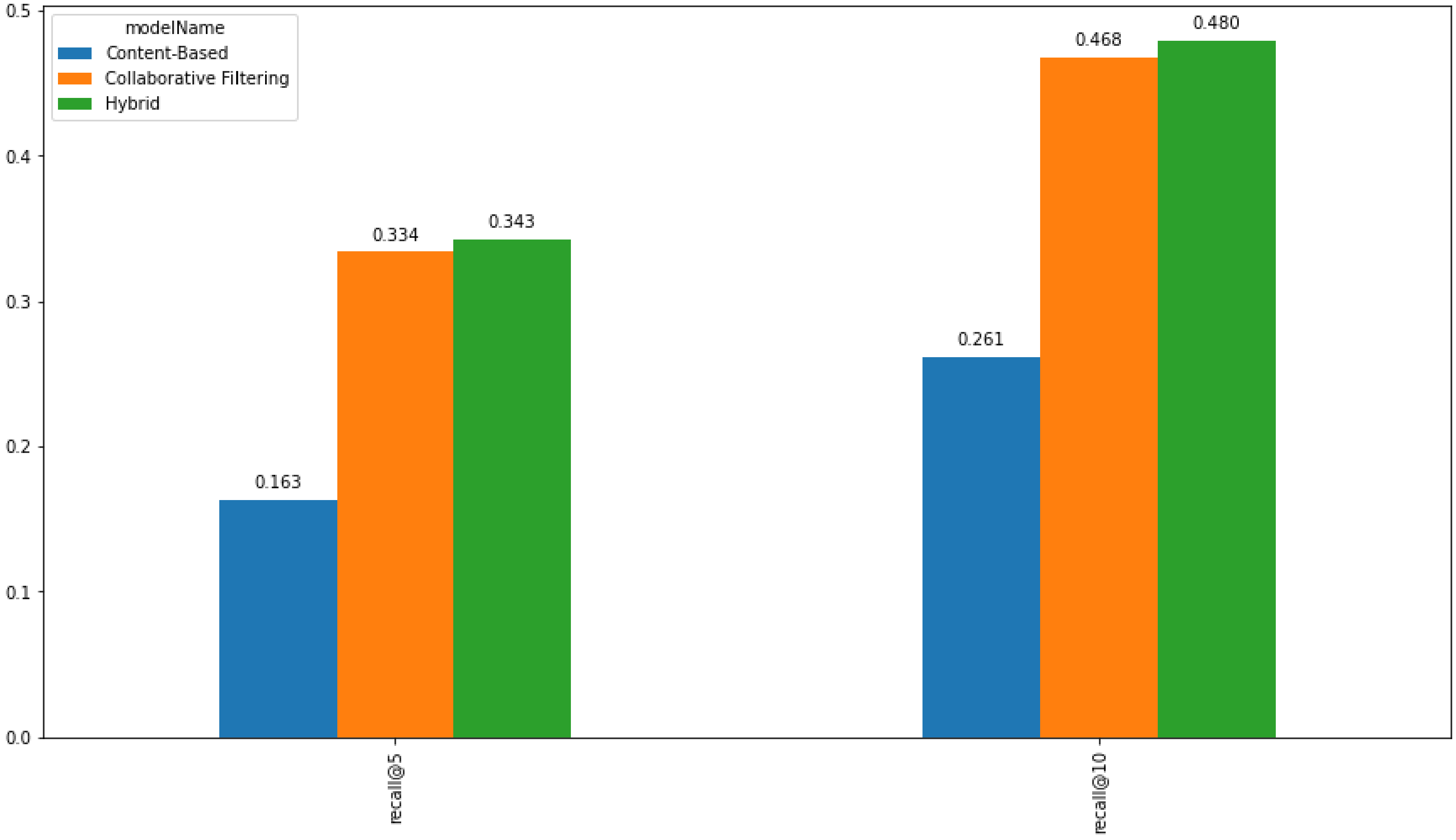
Out[34]:

modelName	recall@5	recall@10
Content-Based	0.162874	0.261442
Collaborative Filtering	0.333930	0.468039
Hybrid	0.342623	0.479673

In [35]:

```
e().plot(kind='bar', figsize=(15,8))

height(), (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='center', xytext=(0, 10), textcoords='offset 
```



Testing

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

In [36]:
```
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**.

In [37]:
```
inspect_interactions(-1479311724257856983, test_set=False).head(20)
```

Out[37]:

	eventStrength	contentId	title	url	lang
115	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!

In [38]:
```
hybrid_recommender_model.recommend_items(-1479311724257856983, topn=20, verbose=True)
```

Out[38]:

	recStrengthHybrid	contentId	title	url	lang
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

In this notebook, we've explored and compared the main Recommender Systems techniques on [CI&T Deskdrop (https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop)](https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop) dataset. 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)](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)](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) or [Feature Hashed (https://en.wikipedia.org/wiki/Feature_hashing)](https://en.wikipedia.org/wiki/Feature_hashing)), and [Wide & Deep models (https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html)](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/)](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)](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/)](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)](https://github.com/NicolasHug/Surprise), [mrec (https://github.com/Mendeley/mrec)](https://github.com/Mendeley/mrec), [python-recsys (https://github.com/ocelma/python-recsys)](https://github.com/ocelma/python-recsys) and [Spark ALS Matrix Factorization (https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html)](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)](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.

In [ ]:

```

```