```
import numpy as np
import scipy
import pandas as pd
import math
import random
import sklearn
from nltk.corpus import stopwords
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
import matplotlib.pyplot as plt
```

Capstone Project For Recommendation System. By Anshika dixit

```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving shared_articles.csv to shared_articles.csv

```
import io
articles_df = pd.read_csv(io.StringIO(uploaded['shared_articles.csv'].decode('utf-8')))
articles_df = articles_df[articles_df['eventType'] == 'CONTENT SHARED']
articles_df.head(2)
```

	timestamp	eventType	contentId	authorPersonId	authorSessi
1	1459193988	CONTENT SHARED	-4110354420726924665	4340306774493623681	894034120520623
2	1459194146	CONTENT SHARED	-7292285110016212249	4340306774493623681	894034120520623

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.

```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving users_interactions.csv to users_interactions.csv
```

```
interactions_df = pd.read_csv(io.StringIO(uploaded['users_interactions.csv'].decode('utf-8
interactions_df.head(2)
```

	timestamp	eventType	contentId	personId	sess
0	1465413032	VIEW	-3499919498720038879	-8845298781299428018	12641967703399
1	1465412560	VIEW	8890720798209849691	-1032019229384696495	36217376435875

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.

```
#data munging
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_
```

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.

```
users_interactions_count_df = interactions_df.groupby(['personId', 'contentId']).size().gr
print('# users: %d' % len(users_interactions_count_df))
users_with_enough_interactions_df = users_interactions_count_df[users_interactions_count_d
print('# users with at least 5 interactions: %d' % len(users_with_enough_interactions_df))
```

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

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.

```
print('# of interactions: %d' % len(interactions_df))
interactions_from_selected_users_df = interactions_df.merge(users_with_enough_interactions_
               how = 'right',
               left_on = 'personId',
               right_on = 'personId')
print('# of interactions from users with at least 5 interactions: %d' % len(interactions_f
```

```
# of interactions: 72312
# of interactions from users with at least 5 interactions: 69868
```

```
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

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.

```
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
```

Evaluation 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.

```
#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_it
```

```
#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['contentI
        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
                                                                                   intera
                                               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
                                                                              seed=item_id%(2*

            #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
            valid_recs_df = person_recs_df[person_recs_df['contentId'].isin(items_to_filte
            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 recomm
```

```
        #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().v
            #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_re
        global_recall_at_10 = detailed_results_df['hits@10_count'].sum() / float(detailed_

        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()
```

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

```
#popularity model
#Computes the most popular items
item_popularity_df = interactions_full_df.groupby('contentId')['eventStrength'].sum().sort
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, verbose=False):
        # Recommend the more popular items that the user hasn't seen yet.
        recommendations_df = self.popularity_df[~self.popularity_df['contentId'].isin(item
                               .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')[['eventS


        return recommendations_df
```

```
popularity_model = PopularityRecommender(item_popularity_df, articles_df)
```

```
print('Evaluating Popularity recommendation model...')
pop_global_metrics, pop_detailed_results_df = model_evaluator.evaluate_model(popularity_mo
print('\nGlobal 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': 0.3729225262
```

	_person_id	hits@10_count	hits@5_count	interacted_count	recall@10
76	3609194402293569455	50	28	192	0.260417
17	-2626634673110551643	25	12	134	0.186567
16	-1032019229384696495	23	13	130	0.176923
10	-1443636648652872475	9	5	117	0.076923
82	-2979881261169775358	40	25	88	0.454545
161	-3596626804281480007	18	12	80	0.225000
65	1116121227607581999	33	20	73	0.452055
81	692689608292948411	23	17	69	0.333333
106	-9016528795238256703	18	14	69	0.260870
52	3636910968448833585	28	21	68	0.411765

```
import nltk
nltk.download('stopwords')
#content based filtering
stopwords_list = stopwords.words('english') + stopwords.words('portuguese')

#Trains a model whose vectors size is 5000, composed by the main unigrams and bigrams foun
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 /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
<3047x5000 sparse matrix of type '<class 'numpy.float64'>'
        with 638928 stored elements in Compressed Sparse Row format>
```

```
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_streng
    user_profile_norm = sklearn.preprocessing.normalize(user_item_strengths_weighted_avg)
    return user_profile_norm

def build_users_profiles():
    interactions_indexed_df = interactions_full_df[interactions_full_df['contentId'] \
                                               .isin(articles_df['contentId'])].set_in
    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
```

```
myprofile = user_profiles[-1479311724257856983]
print(myprofile.shape)
pd.DataFrame(sorted(zip(tfidf_feature_names,
                        user_profiles[-1479311724257856983].flatten().tolist()), key=lambd
             columns=['token', 'relevance'])
```

```
(1, 5000)
```

	token	relevance
0	learning	0.305655
1	machine learning	0.255557
2	machine	0.246095
3	google	0.208590
4	data	0.172509
5	ai	0.136818
6	algorithms	0.102396
7	graph	0.098438
8	like	0.096970
9	language	0.083993
10	people	0.077122
11	use	0.073203
12	models	0.073168
13	deep	0.072377

```
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_i
        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, simila

        recommendations_df = pd.DataFrame(similar_items_filtered, columns=['contentId', 'r
                                      .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')[['recStr


        return recommendations_df

content_based_recommender_model = ContentBasedRecommender(articles_df)


print('Evaluating Content-Based Filtering model...')
cb_global_metrics, cb_detailed_results_df = model_evaluator.evaluate_model(content_based_r
print('\nGlobal 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': 0.41459984658655075, 'recall@10': 0.52416
```

	_person_id	hits@10_count	hits@5_count	interacted_count	recall@10
76	3609194402293569455	26	16	192	0.135417
17	-2626634673110551643	35	21	134	0.261194
16	-1032019229384696495	34	22	130	0.261538
10	-1443636648652872475	54	34	117	0.461538
82	-2979881261169775358	15	8	88	0.170455
161	-3596626804281480007	23	14	80	0.287500
65	1116121227607581999	15	10	73	0.205479
81	692689608292948411	20	11	69	0.289855
106	-9016528795238256703	10	5	69	0.144928
52	3636910968448833585	11	4	68	0.161765

```
#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	-91945728800522001
personId			
-9223121837663643404	0.0	0.0	C
-9212075797126931087	0.0	0.0	C
-9207251133131336884	0.0	2.0	C
-9199575329909162940	0.0	0.0	C
-9196668942822132778	0.0	0.0	C
-9188188261933657343	0.0	0.0	C
-9172914609055320039	0.0	0.0	C
-9156344805277471150	0.0	0.0	C
-9120685872592674274	0.0	0.0	C

```
users_items_pivot_matrix = users_items_pivot_matrix_df.as_matrix()
users_items_pivot_matrix[:10]
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning: Methoc
  """Entry point for launching an IPython kernel.
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]
```

```
#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.shape
```

```
(1140, 15)
```

```
Vt.shape
```

```
(15, 2926)
```

```
sigma = np.diag(sigma)
sigma.shape
```

```
(15, 15)
```

```
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]])
```

```
cf_preds_df = pd.DataFrame(all_user_predicted_ratings, columns = users_items_pivot_matrix_
cf_preds_df.head(10)
```

-9223121837663643404	-9212075797126931087	-9207251133131336 8

```
len(cf_preds_df.columns)
```

1140

9216926795620865886	0.000819	0.000313	0.0065

```
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=Fa
                                    .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']
                               .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')[['recStr


        return recommendations_df

cf_recommender_model = CFRecommender(cf_preds_df, articles_df)


print('Evaluating Collaborative Filtering (SVD Matrix Factorization) model...')
cf_global_metrics, cf_detailed_results_df = model_evaluator.evaluate_model(cf_recommender_
print('\nGlobal 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
```

	_person_id	hits@10_count	hits@5_count	interacted_count	recall@10
76	3609194402293569455	45	21	192	0.234375
17	-2626634673110551643	56	30	134	0.417910
16	-1032019229384696495	34	16	130	0.261538
10	-1443636648652872475	51	38	117	0.435897
82	-2979881261169775358	48	39	88	0.545455
161	-3596626804281480007	34	22	80	0.425000

```
class HybridRecommender:

    MODEL_NAME = 'Hybrid'

    def __init__(self, cb_rec_model, cf_rec_model, items_df):
        self.cb_rec_model = cb_rec_model
        self.cf_rec_model = cf_rec_model
        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_i
                                                       topn=1000).rename(columns={'rec

        #Getting the top-1000 Collaborative filtering recommendations
        cf_recs_df = self.cf_rec_model.recommend_items(user_id, items_to_ignore=items_to_i
                                                       topn=1000).rename(columns={'rec

        #Combining the results by contentId
        recs_df = cb_recs_df.merge(cf_recs_df,
                                   how = 'inner',
                                   left_on = 'contentId',
                                   right_on = 'contentId')

        #Computing a hybrid recommendation score based on CF and CB scores
        recs_df['recStrengthHybrid'] = recs_df['recStrengthCB'] * recs_df['recStrengthCF']

        #Sorting recommendations by hybrid score
        recommendations_df = recs_df.sort_values('recStrengthHybrid', ascending=False).hea

        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')[['recStr


        return recommendations_df

hybrid_recommender_model = HybridRecommender(content_based_recommender_model, cf_recommend


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

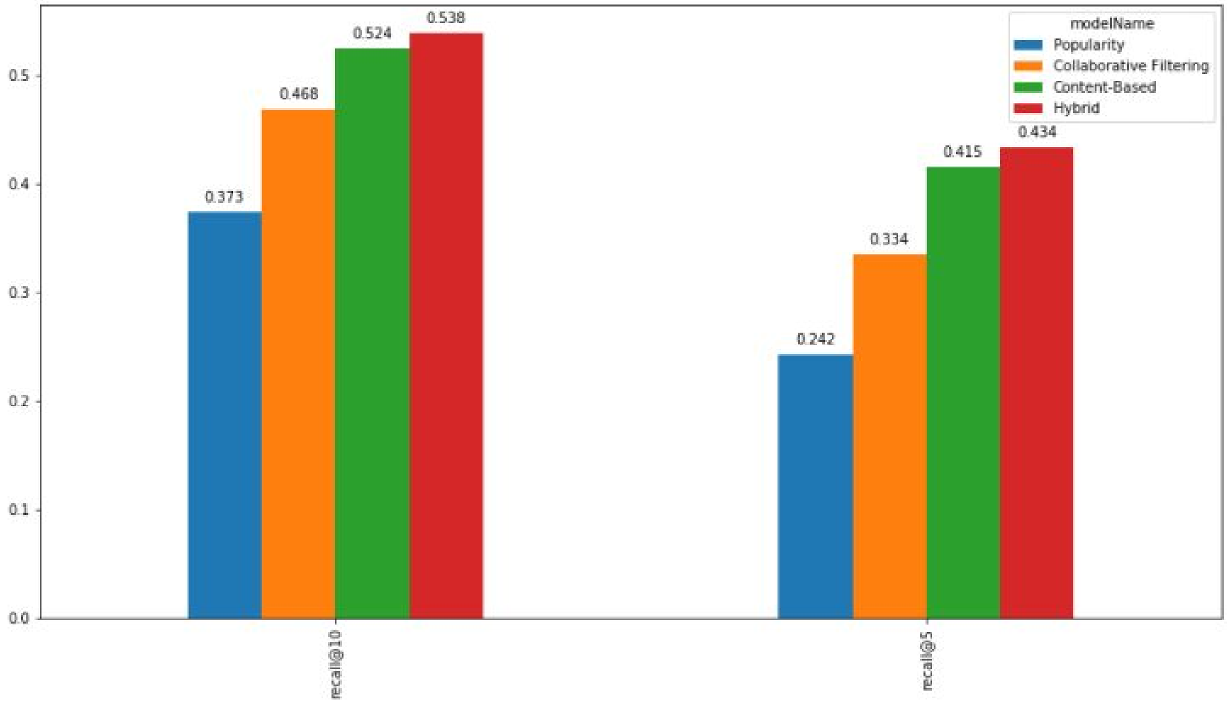
```
Evaluating Hybrid model...
1139 users processed

Global metrics:
{'modelName': 'Hybrid', 'recall@5': 0.4337765277422654, 'recall@10': 0.53796982868831
```

	_person_id	hits@10_count	hits@5_count	interacted_count	recall@10
76	3609194402293569455	40	27	192	0.208333
17	-2626634673110551643	56	38	134	0.417910
16	-1032019229384696495	35	27	130	0.269231
10	-1443636648652872475	52	37	117	0.444444
82	-2979881261169775358	31	26	88	0.352273
161	-3596626804281480007	28	20	80	0.350000
65	1116121227607581999	21	16	73	0.287671
81	692689608292948411	23	14	69	0.333333
106	-9016528795238256703	19	14	69	0.275362
52	3636910968448833585	19	16	68	0.279412

```
global_metrics_df = pd.DataFrame([pop_global_metrics, cf_global_metrics, cb_global_metrics
                        .set_index('modelName')
global_metrics_df
```

modelName	recall@10	recall@5
Popularity	0.372923	0.241754
Collaborative Filtering	0.468167	0.334058
Content-Based	0.524163	0.414600
Hybrid	0.537970	0.433777

```
%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()),
```




```
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',

inspect_interactions(-1479311724257856983, test_set=False).head(20)
```

	eventStrength	contentId	title	
115	4.285402	7342707578347442862	At eBay, Machine Learning is Driving Innovativ...	https://www.ebayinc.com/storie:
38	4.129283	621816023396605502	AI Is Here to Help You Write Emails People Wil...	http://www.wired.com/2016/08/bo
8	4.044394	-4460374799273064357	Deep Learning for Chatbots, Part 1 - Introduction	http://www.wildml.com/2016/04/de
116	3.954196	-7959318068735027467	Auto-scaling scikit-learn with Spark	https://databricks.com/blog/2016/
10	3.906891	2589533162305407436	6 reasons why I like KeystoneML	http://radar.oreilly.com/2015/07/6
28	3.700440	5258604889412591249	Machine Learning Is No Longer Just for Experts	https://hbr.org/2016/10/machine-
			10 Stats About	

```
hybrid_recommender_model.recommend_items(-1479311724257856983, topn=20, verbose=True)
```

	recStrengthHybrid	contentId	title	
0	0.484696	3269302169678465882	The barbell effect of machine learning.	http://techcrunch.com/2016/0
1	0.428711	5092635400707338872	Power to the People: How One Unknown Group of ...	https://medium.com/@atdusl
2	0.411263	5258604889412591249	No Longer Just for Experts	https://hbr.org/2016/10/mach
3	0.358686	-9033211547111606164	Google's Cloud Machine Learning service is now...	https://techcrunch.com/2016/0
4	0.335053	5250363310227021277	How Google is Remaking Itself as a "Machine Le...	https://backchannel.com/how-g
5	0.316371	-7126520323752764957	How Google is Remaking Itself as a "Machine Le...	https://backchannel.com/how-g
			The AI	