In [1]:
```
# EXECUTE FIRST

# computational imports
import numpy as np
import pandas as pd
from ast import literal_eval
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import nltk
from nltk.tokenize import sent_tokenize
from nltk import word_tokenize
nltk.download('averaged_perceptron_tagger')
from sklearn.feature_extraction import text
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
import string
# plotting imports
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
from scipy.spatial import distance
# for reading files from urls
import urllib.request
# display imports
from IPython.display import display, IFrame
from IPython.core.display import HTML

# EXECUTE FIRST
# computational imports
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity
from surprise import Reader, Dataset, KNNBasic, NormalPredictor,BaselineOnly,KNNWithMea
from surprise import SVD, SVDpp, NMF, SlopeOne, CoClustering
from surprise.model_selection import cross_validate
from surprise.model_selection import GridSearchCV
from surprise import accuracy
import random
from ast import literal_eval
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import itertools

# plotting imports
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
plt.style.use('ggplot')

# for reading files from urls
import urllib.request

from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet as wn
```

```
from nltk.tokenize import sent_tokenize

# display imports
from IPython.display import display, IFrame
from IPython.core.display import HTML
import surprise
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Dawit\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Load data

In [2]:

```
#read in the file
df = pd.read_csv('movies_metadata_clean.csv')
df = df.drop_duplicates(subset='title', keep="first", inplace=False)
df= df[0:3000]
print(f'The shape of the dataframe is {df.shape}')

# Convince Python that this column should be treated like a list, not a string.
df['genres'] = df['genres'].apply(literal_eval)
df[1:2]
```

```
The shape of the dataframe is (3000, 10)
```

Out[2]:

	id	title	budget	genres	overview	revenue	runtime	vote_average	vote_count	y
1	8844	Jumanji	65000000.0	[Adventure, Fantasy, Family]	When siblings Judy and Peter discover an encha...	262797249.0	104.0	6.9	0	1!

In [3]:

```
movies = pd.DataFrame({'movie_id': df["id"],
                       'title':df["title"],
                       'genres': df["genres"]
                       })
```

Generate users and ratings for all movies in the dataset

In [4]:

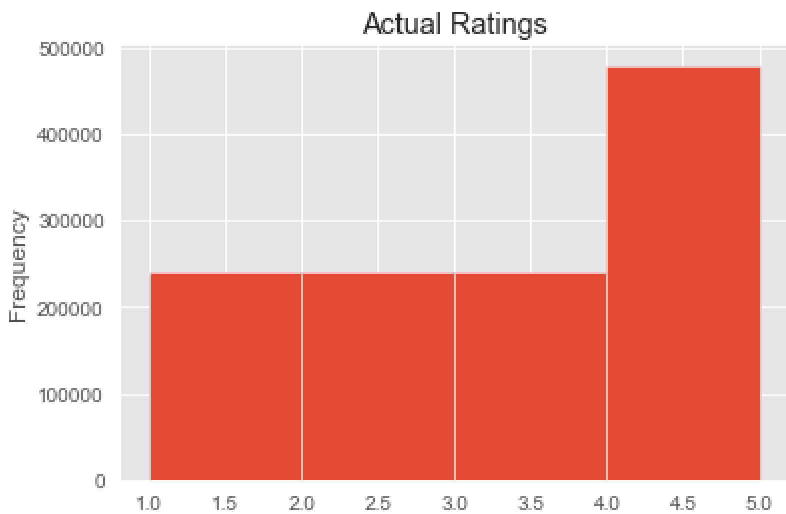
```
n_users = 401
n_movies = 3000
#generate a rating for each user/movie combination
ratings = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id
np.random.seed(12)
randratings = np.random.randint(1,6, ratings.shape[0])
ratings['rating'] = randratings
ratings["user_id"] = ratings["user_id"].astype(str)
ratings["movie_id"] = ratings["movie_id"].astype(str)
ratings.dtypes, ratings.shape
```

Out[4]:
```
(user_id     object
 movie_id    object
 rating       int32
 dtype: object,
 (1200000, 3))
```

Assess rating distribution

In [5]:
```
ratings.rating.plot(kind='hist', bins=4, title='Actual Ratings')
```

Out[5]: <AxesSubplot:title={'center':'Actual Ratings'}, ylabel='Frequency'>



In [6]:
```
import matplotlib.pyplot as plt
plt.close('all')
```

Compute Similarity among movies in the dataset

Lemma Tokenizer function to extract root words from text

In [7]:
```
def get_wordnet_pos(word, pretagged = False):
    """Map POS tag to first character lemmatize() accepts"""
    if pretagged:
        tag = word[1].upper()
    else:
        tag = nltk.pos_tag([word])[0][1][0].upper()

    tag_dict = {"J": wn.ADJ,
                "N": wn.NOUN,
                "V": wn.VERB,
                "R": wn.ADV}

    return tag_dict.get(tag, wn.NOUN)

#create a tokenizer that uses lemmatization (word shortening)
class LemmaTokenizer(object):
    def __init__(self):
        self.wnl = WordNetLemmatizer()
    def __call__(self, articles):
```

```
        #get the sentences
        sents = sent_tokenize(articles)
        #get the parts of speech for sentence tokens
        sent_pos = [nltk.pos_tag(word_tokenize(s)) for s in sents]
        #flatten the list
        pos = [item for sublist in sent_pos for item in sublist]
        #lemmatize based on POS (otherwise, all words are nouns)
        lems = [self.wnl.lemmatize(t[0], get_wordnet_pos(t, True)) for t in pos if t[0]
        #clean up in-word punctuation
        lems_clean = [''.join(c for c in s if c not in string.punctuation) for s in lem
        return lems_clean



#lemmatize the stop words
lemmatizer = WordNetLemmatizer()
lemmatized_stop_words = [lemmatizer.lemmatize(w) for w in text.ENGLISH_STOP_WORDS]
#extend the stop words with any other words you want to add, these are bits of contract
lemmatized_stop_words.extend(['ve','nt','ca','wo','ll'])
```

Obtain *TF - IDF Scores for all movies based on the* overview *column*

1. Intialize vetorizer

2. Remove stop words in the vector

3. Costruct TD-IDF matrix for all movies

In [8]:

```
df = df.copy()
tfidf = TfidfVectorizer(tokenizer=LemmaTokenizer(), lowercase=True, stop_words=lemmatiz
vec3 = CountVectorizer(tokenizer=LemmaTokenizer(), lowercase=True, stop_words=lemmatize
df['overview'] = df['overview'].fillna('')
tfidf_matrix = tfidf.fit_transform(df['overview'])
tfidf_matrix.shape
```

Out[8]:

```
(3000, 100)
```

In [9]:

```
feature_names = tfidf.get_feature_names()
corpus_index = df['title']
pd.DataFrame(tfidf_matrix.todense(), index=corpus_index, columns=feature_names)[1:5]
```

Out[9]:

	american	attempt	beautiful	begin	best	boy	brother	child	city	...	want	wa
title												
Jumanji	0.510952	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
Grumpier Old Men	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
Waiting to Exhale	0.610904	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
Father of the Bride Part II	0.504329	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0

4 rows × 100 columns

Compute Cosine Similarity using TD-IDF scores and store in a matrix

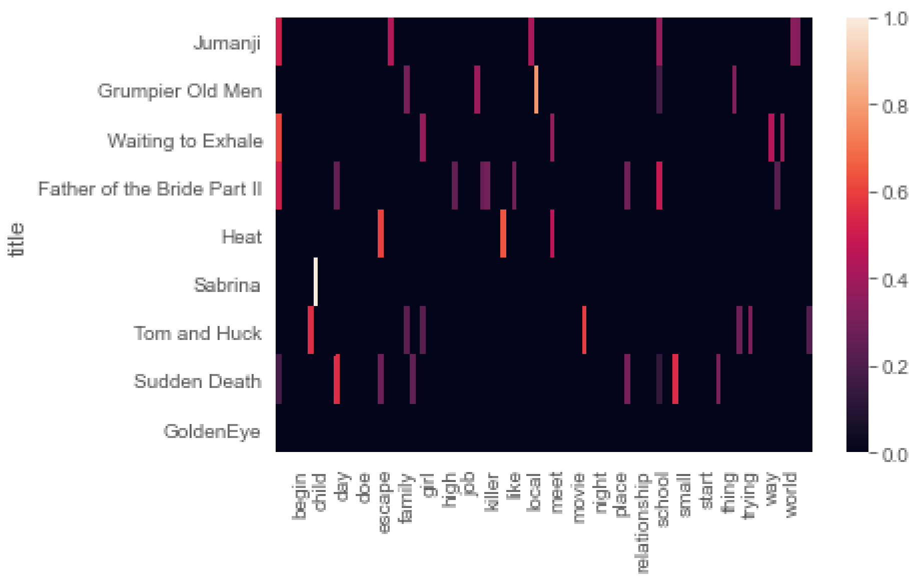
In [10]:
```
# Compute the cosine similarity matrix
sim_matrix = linear_kernel(tfidf_matrix, tfidf_matrix)
dpdf = pd.DataFrame(sim_matrix, columns=df['title'], index=df['title'])
dpdf.shape
```

Out[10]: (3000, 3000)

Content Based Recommendation

TD-IDF visulization

In [11]:
```
ax = sns.heatmap(pd.DataFrame(tfidf_matrix.todense()[1:10], index=df['title'][1:10], co
```



Cosine similairty table

In [12]:
```
sample = dpdf.iloc[1:5,1:5]
cm = sns.color_palette("Blues", as_cmap=True)
sample.style.set_caption('Dot Product with most similar movies highlighted.')\
    .background_gradient(cmap=cm)
# sample
```

Out[12]:

Dot Product with most similar movies highlighted.

title	Jumanji	Grumpier Old Men	Waiting to Exhale	Father of the Bride Part II

title	Jumanji	Grumpier Old Men	Waiting to Exhale	Father of the Bride Part II
title				
Jumanji	1.000000	0.064428	0.312143	0.440489
Grumpier Old Men	0.064428	1.000000	0.000000	0.084790
Waiting to Exhale	0.312143	0.000000	1.000000	0.308097
Father of the Bride Part II	0.440489	0.084790	0.308097	1.000000

Content- based recommendation function

In [13]:
```
def content_recommender(df, seed, seedCol, sim_matrix,  topN=2):
    #get the indices based off the seedCol
    indices = pd.Series(df.index, index=df[seedCol]).drop_duplicates()

    # Obtain the index of the item that matches our seed
    idx = indices[seed]

    # Get the pairwsie similarity scores of all items and convert to tuples
    sim_scores = list(enumerate(sim_matrix[idx]))

    #delete the item that was passed in
    del sim_scores[idx]

    # Sort the items based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the top-n most similar items.
    sim_scores = sim_scores[:topN]

    # Get the item indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the topN most similar items
    return df.iloc[movie_indices]
```

Test function

In [14]:
```
content_recommender(df, 'The Locusts', 'title', sim_matrix, 10)
# sim_matrix = linear_kernel(tfidf_matrix, tfidf_matrix)d
```

Out[14]:

	id	title	budget	genres	overview	revenue	runtime	vote_average	vote_
627	40926	Frisk	0.0	[Drama, Thriller]	A first person narrative of a gay serial kille...	0.0	88.0	3.0	
1095	11298	The Howling	1000000.0	[Drama, Horror]	After a bizarre and near fatal encounter with ...	17985893.0	91.0	6.4	

	id	title	budget	genres	overview	revenue	runtime	vote_average	vote_
21	1710	Copycat	0.0	[Drama, Thriller]	An agoraphobic psychologist and a female detec...	0.0	124.0	6.5	
476	10909	Kalifornia	9000000.0	[Thriller, Crime]	A journalist duo go on a tour of serial killer...	2395231.0	117.0	6.5	
333	9271	Virtuosity	30000000.0	[Action, Crime, Science Fiction, Thriller]	The Law Enforcement Technology Advancement Cen...	24048000.0	106.0	5.4	
46	807	Se7en	33000000.0	[Crime, Mystery, Thriller]	Two homicide detectives are on a desperate hun...	327311859.0	127.0	8.1	
371	8987	The River Wild	45000000.0	[Action, Adventure, Crime, Thriller]	While on a family vacation, rafting expert Ga...	0.0	111.0	6.1	
1780	26610	Insomnia	0.0	[Crime, Drama, Thriller]	Detectives Jonas and Erik are called to the mi...	0.0	96.0	6.6	
1289	32146	Body Parts	0.0	[Horror, Thriller]	A criminal psychologist loses his arm in a car...	0.0	88.0	5.6	
586	274	The Silence of the Lambs	19000000.0	[Crime, Drama, Thriller]	FBI trainee, Clarice Starling ventures into a ...	272742922.0	119.0	8.1	

Function for conent based recommendation for N users

In [15]:
```
# list_of_movies = df["title"]
def content_recommender2(n_users,n_items,df):
    list_of_movies = df["title"][1:n_users+1]
    list_of_recommendations=[]
    for movie in list_of_movies:
        list_of_recommendations.append(list(content_recommender(df, str(movie), 'title'
    return list_of_recommendations
```

Make recommendations for N users

In [16]:
```
content_list_of_recommendations = content_recommender2(400,10,df)
```

Collaborative - Filtering recommendation system using KNN

KNN- Model setup

In [17]:
```
our_seed = 14

#Define a Reader
reader = Reader(rating_scale=(1,5)) # defaults to (0,5)

#Create the dataset
data = Dataset.load_from_df(ratings, reader)

#Define the algorithm object
knn = KNNBasic(k= 4, n_jobs=-1, verbose=False) #the default for k is 40, we're also set

random.seed(our_seed)
np.random.seed(our_seed)

#cross validation
knn_cv = cross_validate(knn, data, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv)

#extract RMSE
knn_RMSE = np.mean(knn_cv['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5792  1.5810  1.5807  1.5799  1.5785  1.5799  0.0009
Fit time          38.81   41.77   21.38   19.10   22.36   28.68   9.58
Test time         125.70  94.71   69.75   71.24   73.90   87.06   21.33
{'test_rmse': array([1.57924483, 1.58103116, 1.58071122, 1.5799147 , 1.57852976]), 'fit_
time': (38.809500217437744, 41.77214026451111, 21.377328634262085, 19.097674131393433, 2
2.36443567276001), 'test_time': (125.70344090461731, 94.70789527893066, 69.7464795112609
9, 71.23967361450195, 73.89880084991455)}

The RMSE across five folds was 1.5798863348788867
```

Grid search for hyperparameter tuning of the KNN model

In [18]:
```
data.build_full_trainset()
```

Out[18]:
```
<surprise.trainset.Trainset at 0x197a4f89c48>
```

In [19]:
```
#Define a Reader object
```

```
#The Reader object helps in parsing the file or dataframe containing ratings
reader = Reader(rating_scale=(1,5)) # defaults to (0,5)


#Create the dataset
data = Dataset.load_from_df(ratings, reader)
raw_ratings = data.raw_ratings

# shuffle ratings
random.seed(our_seed)
np.random.seed(our_seed)
random.shuffle(raw_ratings)

#A = 90% of the data, B = 10% of the data
threshold = int(.9 * len(raw_ratings))
A_raw_ratings = raw_ratings[:threshold]
B_raw_ratings = raw_ratings[threshold:]

data.raw_ratings = A_raw_ratings  # data is now the set A

# Select your best algo
print('Grid Search...')
param_grid = {'k': [3,5], 'min_k': [1,3]} #this will all combinations of max k of 3 and
grid_search = GridSearchCV(KNNBasic, param_grid, measures=['rmse'], cv=3)
grid_search.fit(data)
knn_gs_algo = grid_search.best_estimator['rmse']

# retrain on the whole set A
trainset = data.build_full_trainset()
knn_gs_algo.fit(trainset)

# Compute biased accuracy on A
predictions = knn_gs_algo.test(trainset.build_testset())
print(f'Biased accuracy on A = {accuracy.rmse(predictions)}')

# Compute unbiased accuracy on B
testset = data.construct_testset(B_raw_ratings)  # testset is now the set B
predictions = knn_gs_algo.test(testset)
print(f'Unbiased accuracy on B = {accuracy.rmse(predictions)}')
```

```
Grid Search...
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
```

```
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.6917
Biased accuracy on A = 0.6916816445638049
RMSE: 1.5500
Unbiased accuracy on B = 1.549958370462392
```

In [20]:

```
#we can see what our best parameters were
grid_search.best_params['rmse']
```

Out[20]: `{'k': 5, 'min_k': 1}`

Retrain model using new parameters

In [21]:

```
#set our seeds again
random.seed(our_seed)
np.random.seed(our_seed)

#reset the data.raw_ratings to 100%
data.raw_ratings = raw_ratings

#trainset
trainset = data.build_full_trainset()

#build the algorithm using the best parameters
knn_gs_algo = grid_search.best_estimator['rmse']

#fit to the data
knn_gs_algo.fit(trainset)

#predict user 1, movie 11
knn_gs_algo.predict("1","9626")
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
```

Out[21]:

```
Prediction(uid='1', iid='9626', r_ui=None, est=1.787987652965251, details={'actual_k':
5, 'was_impossible': False})
```

Test recommendation for user 1

In [22]:

```
sample =  movies.copy()
sample['est_rating'] = sample.apply(lambda x: knn_gs_algo.predict("1", x['movie_id']).e
sample.sort_values('est_rating', ascending=False)[1:10]
```

Out[22]:

	movie_id	title	genres	est_rating
381	29444	S.F.W.	[Comedy, Drama]	4.887442
2387	18892	Jawbreaker	[Comedy]	4.887442
2525	15660	Mommie Dearest	[Drama]	4.887381
1390	14908	McHale's Navy	[Action, Comedy, Romance]	4.887381
385	315	Faster, Pussycat! Kill! Kill!	[Action, Crime]	4.887381
2958	11481	Repulsion	[Drama, Horror, Thriller]	4.887366

	movie_id	title	genres	est_rating
2667	28501	The Pit and the Pendulum	[Fantasy, Horror, Drama]	4.887366
1847	16619	Ordinary People	[Drama]	4.775084
197	79593	The Tie That Binds	[Thriller]	4.775084

Collaborative recommendation function

In [23]:
```
def collaborative_recommender(n_users,df):
    sample = df.copy()
    list_of_recommendations=[]
    for user in range(n_users+1):
        if user == 0:
            pass
        else:
            sample["estimated_rating"]= movies.apply(lambda x: knn_gs_algo.predict(str(
            list_of_recommendations.append(list(sample.sort_values('estimated_rating',a
    return list_of_recommendations
```

Make collaborative recommendations for N users

In [24]:
```
collaborative_list_of_recommendations = collaborative_recommender(400,movies)
```

--

Analyze recommendations from content-based and collaborative models above

Function to compute similairty, diversity and total coverage for recommendations made using the two models

In [25]:
```
def uniqueCombinations(list_elements):
    l = list(itertools.combinations(list_elements, 2))
    s = set(l)
    return list(s)
```

In [26]:
```
def compute_mean_sim_score(n_users,recommendations,sim_matrix,df):
    sim_scores_movies =[]
    score = 0
    N_unique_combinations =0
    for m in recommendations:
        for pair in uniqueCombinations(m):
            score = score + dpdf[pair[0]][pair[1]]
    #         print(dpdf[pair[0]][pair[1]])
    #         print(pair)
            N_unique_combinations += 1
        N_unique_combinations += 0
        sim_scores_movies.append(score)
        score = 0
```

```
    unique_combination_per_set =  N_unique_combinations /n_users
    mean_sim_scores_movies = [n/unique_combination_per_set for n in sim_scores_movies]
    total_recommendations = list(itertools.chain.from_iterable(recommendations))
    number_Total_recommendations = len(set(total_recommendations))
    Total_movies = len(df['movie_id'])
    coverage = number_Total_recommendations /Total_movies
    similarity =  sum(mean_sim_scores_movies)/len(mean_sim_scores_movies)
    diversity = 1- similarity

    return mean_sim_scores_movies, similarity, diversity, coverage,total_recommendation
```

Findings for content-based recommendations

Content based recommendatons - Similarity, Diversity, Coverage

In [27]:

```
content_similarity = compute_mean_sim_score(400,content_list_of_recommendations ,dpdf,m
content_diversity = compute_mean_sim_score(400,content_list_of_recommendations,dpdf,mov
content_coverage = compute_mean_sim_score(400,content_list_of_recommendations ,dpdf,mov
print("Collaborative similarity", content_similarity)
print("Collaborative  Diversity", content_diversity)
print("Collaborative  coverage", content_coverage)
```

```
Collaborative similarity 0.4648508515411847
Collaborative  Diversity 0.5351491484588153
Collaborative  coverage 0.5953333333333334
```

Findings for Collaborative filtering

Collaborative recommendatons - Similarity, Diversity, Coverage

In [28]:

```
collab_similarity = compute_mean_sim_score(400,collaborative_list_of_recommendations ,d
collab_diversity = compute_mean_sim_score(400,collaborative_list_of_recommendations,dpd
collab_coverage = compute_mean_sim_score(400,collaborative_list_of_recommendations ,dpd
print("Collaborative similarity", collab_similarity)
print("Collaborative  Diversity", collab_diversity)
print("Collaborative  coverage", collab_coverage)
```

```
Collaborative similarity 0.06625481199399376
Collaborative  Diversity 0.9337451880060063
Collaborative  coverage 0.6813333333333333
```

In [ ]:

Similarity distribution comparison, content-based Vs Collaborative

In [29]:

```
content_mean_sim_scores= compute_mean_sim_score(400,content_list_of_recommendations,dpd
collab_mean_sim_scores = compute_mean_sim_score(400,collaborative_list_of_recommendatio
```

In [30]:

```
content_collab_sim_scores = pd.DataFrame({"Content_similarity distribution":content_mea
                  "Collaborative_similarity_distribution" : collab_mean_sim_scores})
```

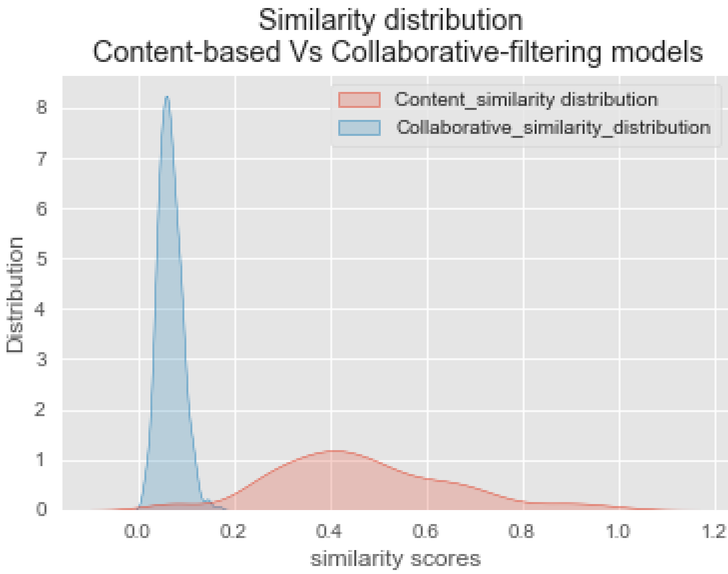
In [31]:

```
sim_dis = sns.kdeplot(data = content_collab_sim_scores, fill =True, )
```

```
sim_dis.set_xlabel("similarity scores", fontsize = 12)
sim_dis.set_ylabel("Distribution", fontsize = 12)
sim_dis.set(title='Similarity distribution \n Content-based Vs Collaborative-filtering
```

Out[31]:

```
[Text(0.5, 1.0, 'Similarity distribution \n Content-based Vs Collaborative-filtering mod
els')]
```



Popularity plot comparison, content-based Vs Collaborative

In [34]:

```
from collections import Counter
total_content_recommendations = compute_mean_sim_score(400,content_list_of_recommendati
content_based_count = [ ]
content_counter = Counter(total_content_recommendations)
for i in total_content_recommendations:
    content_based_count.append(content_counter[i])
```

In [35]:

```
total_collab_recommendations = compute_mean_sim_score(400,collaborative_list_of_recomme
collab_based_count = []
collab_counter = Counter(total_collab_recommendations)
for i in total_collab_recommendations:
    collab_based_count.append(collab_counter[i])
```

In [36]:

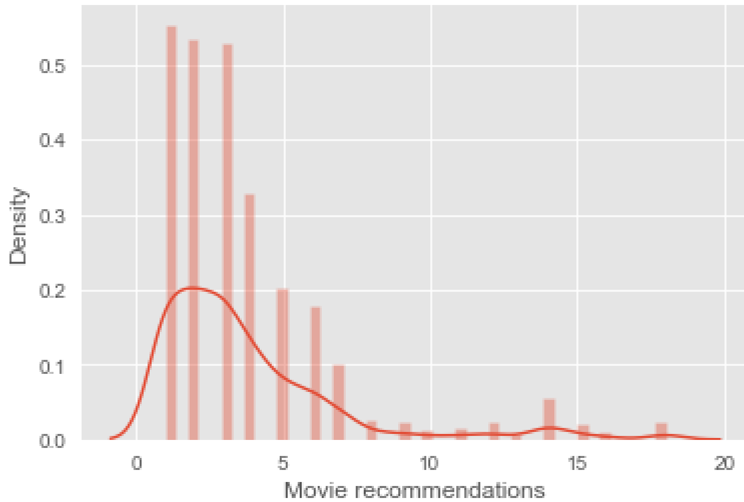
```
import warnings
warnings.filterwarnings('ignore')
```

In [37]:

```
sns.distplot(pd.Series(content_based_count, name = "Movie recommendations"))
```

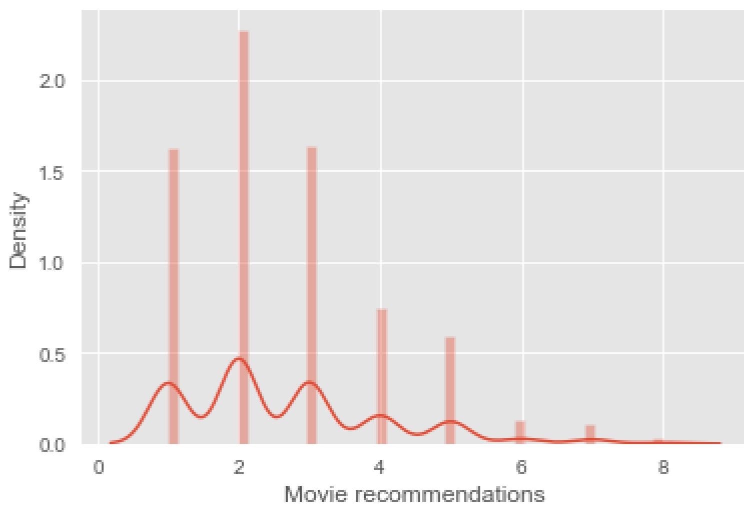
Out[37]:

```
<AxesSubplot:xlabel='Movie recommendations', ylabel='Density'>
```



In [38]:
```
sns.distplot(pd.Series(collab_based_count, name = "Movie recommendations"))
```

Out[38]: <AxesSubplot:xlabel='Movie recommendations', ylabel='Density'>



Hybrid Recommendation

Fetch highest rated movie for each user in the dataset to use as historical data for each user's content-based recommendation

In [39]:
```
highest_rated = ratings.groupby(['user_id'])['rating'].transform(max) == ratings['ratin
highest_rated_df = ratings[highest_rated]
total_highest_rated_movies = highest_rated_df.groupby(['user_id','movie_id'])['rating']
highest_rated_movies = total_highest_rated_movies.groupby('user_id').apply(lambda x: x.
highest_rated_movies['user_id'] = highest_rated_movies['user_id'].astype(int)
highest_rated_movies = highest_rated_movies.sort_values("user_id")
```

Function to get user ID and the title of the movie they rated highly

In [40]:
```
def extract_users_movie(df, rating):
    dic ={}
    user = 1
    for movieid in rating['movie_id']:
        dic[user] = list(df.loc[df['movie_id'] == movieid, 'title'])[0]
        user = user +1
#     dic = {k: list(v[1]) for k,v in dic.items()}
    return dic
```

In [41]:
```
users_history = extract_users_movie(movies, highest_rated_movies)
```

Modify content recommender, make new set of recommendation to take a dictionary

In [42]:
```
df.index = range(len(df))
movies.index =range(len(df))
```

In [43]:
```
def modified_content_recommender(dic,n_items,df):
    recommendation_list = []
    for movie in dic.values():
        recommendation_list.append(list(content_recommender(df, str(movie), 'title', si
    return recommendation_list
```

Build a hybrid recommender

New content-based recommender + the collaborative model from above

Function will also return purely content and purely collaborative recommendaitons for comparison

In [44]:
```
def hybrid_recommender(df,n_users,n_items, history, top_n_conent, top_n_coll, sim_m):
    content_based = modified_content_recommender(history,n_items, df)
    collaborative_filter = collaborative_recommender(n_users,df)
    total_recommendations = []
    for i in range(len(content_based )):
        total_recommendations.append(list(set(random.sample(content_based [i],top_n_con
    return total_recommendations, content_based, collaborative_filter
```

Make all recommendations

In [46]:
```
result = hybrid_recommender(df,400,10,users_history,2,8,dpdf)
```

Findings from all 3 models

Content based recommendations analysis

In [47]:
```
print("content based similarity", compute_mean_sim_score(400,result[1],dpdf, movies)[1]
print("content based Diversity",compute_mean_sim_score(400,result[1],dpdf, movies)[2])
print("content based coverage", compute_mean_sim_score(400,result[1],dpdf, movies)[3])
```

```
content based similarity 0.46492154055899637
content based Diversity 0.5350784594410036
content based coverage 0.581
```

Collaborative filtering recommendations analysis

In [48]:
```
print("collaborative similarity", compute_mean_sim_score(400,result[2],dpdf,movies)[1])
print("collaborative Diversity",compute_mean_sim_score(400,result[2],dpdf,movies)[2])
print("collaborative coverage", compute_mean_sim_score(400,result[2],dpdf,movies)[3])
```

```
collaborative similarity 0.06625481199399376
collaborative Diversity 0.9337451880060063
collaborative coverage 0.6813333333333333
```

Hybrid based recommendations analysis

In [49]:
```
print("Hybrid based similarity", compute_mean_sim_score(400,result[0],dpdf, movies)[1])
print("Hybrid based Diversity",compute_mean_sim_score(400,result[0],dpdf, movies)[2])
print("Hybrid based coverage", compute_mean_sim_score(400,result[0],dpdf, movies)[3])
```

```
Hybrid based similarity 0.07296290370859333
Hybrid based Diversity 0.9270370962914066
Hybrid based coverage 0.6946666666666667
```

Similarity & diversity comparison Between Hybrid and content-based model

In [50]:
```
# compute_mean_sim_score(400,result[0],dpdf,movies)
print("Mean sim score for Hybrid recommendations", (compute_mean_sim_score(400,result[0
print("Mean sim score for Content_based recommendations", (compute_mean_sim_score(400,r
```

```
Mean sim score for Hybrid recommendations 0.07296290370859333
Mean sim score for Content_based recommendations 0.46492154055899637
```

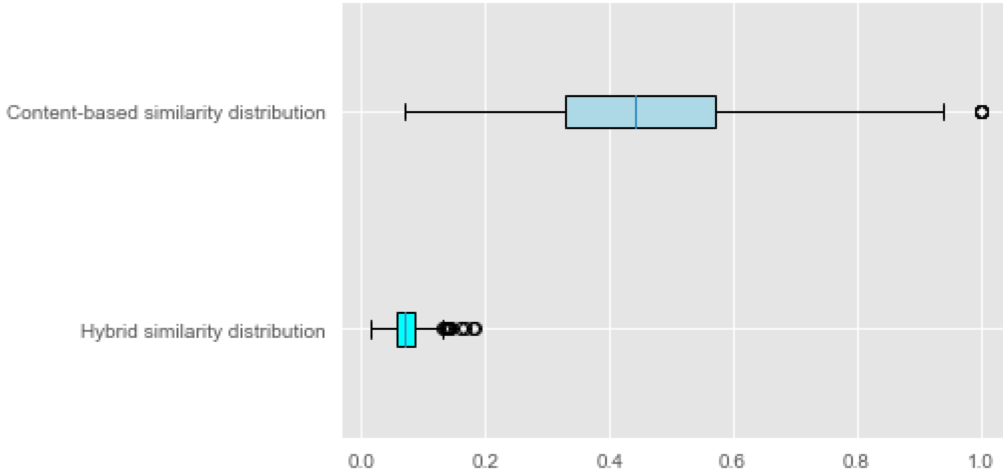
In [51]:
```
# compute_mean_sim_score(400,result[0],dpdf,movies)
print("Diversity for Hybrid recommendations", np.mean(compute_mean_sim_score(400,result
print("Diversity for Content_based recommendations", np.mean(compute_mean_sim_score(400
print("% increase in diversity by Hybrid model",np.mean(compute_mean_sim_score(400,resu
```

```
Diversity for Hybrid recommendations 0.9270370962914066
Diversity for Content_based recommendations 0.5350784594410036
% increase in diversity by Hybrid model 0.391958636850403
```

In [52]:
```
box_plot_data=[compute_mean_sim_score(400,result[0],dpdf,movies)[0],compute_mean_sim_sc
box= plt.boxplot(box_plot_data,vert=0,patch_artist=True,labels=['Hybrid similarity dist
            )
colors = ['cyan', 'lightblue']

for patch, color in zip(box['boxes'], colors):
    patch.set_facecolor(color)

plt.show()
```



Tests the significance in the difference of the two distributions using paired Student's t-Test

In [53]:
```
from scipy.stats import ttest_rel
data1 = compute_mean_sim_score(400,result[0],dpdf, movies)[0]
data2 = compute_mean_sim_score(400,result[1],dpdf, movies)[0]
# compare samples
stat, p = ttest_rel(data1, data2)
print('Statistics=%.3f, p=%.20f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Same distributions (fail to reject H0)')
else:
    print('Different distributions (reject H0)')
```

```
Statistics=-40.312, p=0.00000000000000000000
Different distributions (reject H0)
```

Coverage comparison between Hybrid and Collaborative models

% Difference in coverage, Hybrid vs Collaborative

At 0% data sparsity we there is 1.333 % difference in coverage b/n the hybrid and collaborative model

In [54]:
```
(compute_mean_sim_score(400,result[0],dpdf,movies)[3] - compute_mean_sim_score(400,resu
```

Out[54]: 1.3333333333333308

Varying data sparsity

Build new Hybrid and collaborative models to intake different Knn functions for eah step

In [55]:
```
def collaborative_recommender2(n_users,df, col_fun):
```

```
    sample = df.copy()
    list_of_recommendations=[]
    for user in range(n_users+1):
        if user == 0:
            pass
        else:
            sample["estimated_rating"]= movies.apply(lambda x: col_fun.predict(str(user
            list_of_recommendations.append(list(sample.sort_values('estimated_rating',a
    return list_of_recommendations
```

In [56]:
```
def hybrid_recommender2(df,n_users,n_items, history, top_n_conent, top_n_coll, sim_m, c
    content_based = modified_content_recommender(history,n_items, df)
    collaborative_filter = collaborative_recommender2(n_users,df,col_fun)
    total_recommendations = []
    for i in range(len(content_based )):
        total_recommendations.append(list(set(random.sample(content_based [i],top_n_con
    return total_recommendations, content_based, collaborative_filter
```

Varying data sparsity allows analysis of how the hybrid model performs compared to a purely collaborative model

For each step...

1. Generate new sparse data for the selected level of sparsity
2. Train a new collaborative model
3. Input the new model into the hybrid model
4. Make purely content and collaborative recommendations
5. Make hybrid recommendations
6. Compare coverage between Hybrid and Collaborative model
7. Assess how diversity is impacted by the hybrid models in each steps

Coverage comparison at 16.7% sparsity

Train the model with sparse data

In [57]:
```
n_users = 401
n_movies = 2500
#generate a rating for each user/movie combination
data1 = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id f
np.random.seed(12)
randratings = np.random.randint(1,6, data1.shape[0])
data1['rating'] = randratings
data1["user_id"] = ratings["user_id"].astype(str)
data1["movie_id"] = ratings["movie_id"].astype(str)
data1.dtypes, data1.shape
```

Out[57]:
```
(user_id     object
 movie_id    object
 rating       int32
```

```
dtype: object,
(1000000, 3))
```

Sparsity level

In [58]:
```
(len(ratings) - len(data1))/len(ratings) # % sparcity
```

Out[58]:
```
0.16666666666666666
```

Build new collaborative model on sparse data

In [59]:
```
our_seed = 14
reader1 = Reader(rating_scale=(1,5)) # defaults to (0,5)
d1 = Dataset.load_from_df(data1, reader1)
knn1 = KNNBasic(k= 4, n_jobs=-1, verbose=False)
random.seed(our_seed)
np.random.seed(our_seed)
knn_cv1 = cross_validate(knn1, d1, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv1)
knn_RMSE1 = np.mean(knn_cv1['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE1}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5791  1.5809  1.5782  1.5802  1.5794  1.5796  0.0009
Fit time          12.39   10.72   11.71   10.03   12.30   11.43   0.92
Test time         52.35   41.82   41.44   40.28   47.06   44.59   4.53
{'test_rmse': array([1.57911029, 1.58091823, 1.57821139, 1.58022001, 1.57938965]), 'fit_
time': (12.391668558120728, 10.724379777908325, 11.713493347167969, 10.031909227371216,
12.297916173934937), 'test_time': (52.34842324256897, 41.821391105651855, 41.44197487831
116, 40.28261470794678, 47.05648970603943)}

The RMSE across five folds was 1.5795699139520485
```

Extract highly rated movies for each user based on the new sparse data

In [60]:
```
highest_rated1 = data1.groupby(['user_id'])['rating'].transform(max) == data1['rating']
highest_rated_df1 = data1[highest_rated1]
total_highest_rated_movies1 = highest_rated_df1.groupby(['user_id','movie_id'])['rating
highest_rated_movies1 = total_highest_rated_movies1.groupby('user_id').apply(lambda x:
highest_rated_movies1['user_id'] = highest_rated_movies1['user_id'].astype(int)
highest_rated_movies1 = highest_rated_movies1.sort_values("user_id")
```

In [61]:
```
users_history1 = extract_users_movie(movies, highest_rated_movies1)
```

Make a hybrid recommendations based on new highly rated movies for content-based recommendations and newly trained collaborative model

In [62]:
```
result1 = hybrid_recommender2(df,400,10,users_history1,2,8,dpdf,knn1)
```

In [63]:
```
print("Collaborative coverage", compute_mean_sim_score(400,result1[2],dpdf, movies)[3])
print("Hybrid based coverage", compute_mean_sim_score(400,result1[0],dpdf, movies)[3])
print("% difference in coverage", (compute_mean_sim_score(400,result1[0],dpdf, movies)[
```

```
Collaborative coverage 0.62
Hybrid based coverage 0.6236666666666667
% difference in coverage 0.3666666666666707
```

Coverage comparison at 33.3% sparsity

In [64]:
```
n_users = 401
n_movies = 2000
#generate a rating for each user/movie combination
data2 = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id f
np.random.seed(12)
randratings = np.random.randint(1,6, data2.shape[0])
data2['rating'] = randratings
data2["user_id"] = data2["user_id"].astype(str)
data2["movie_id"] = data2["movie_id"].astype(str)
data2.dtypes, data2.shape
```

Out[64]:
```
(user_id     object
 movie_id    object
 rating       int32
 dtype: object,
 (800000, 3))
```

In [65]:
```
(len(ratings)-len(data2))/len(ratings)
```

Out[65]:
```
0.3333333333333333
```

In [66]:
```
our_seed = 14
reader2 = Reader(rating_scale=(1,5)) # defaults to (0,5)
d2 = Dataset.load_from_df(data2, reader2)
knn2 = KNNBasic(k= 4, n_jobs=-1, verbose=False)
random.seed(our_seed)
np.random.seed(our_seed)
knn_cv2 = cross_validate(knn2, d2, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv2)
knn_RMSE2 = np.mean(knn_cv2['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE2}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5847  1.5816  1.5831  1.5819  1.5841  1.5831  0.0012
Fit time          11.47   12.07   12.52   13.00   11.88   12.19   0.53
Test time         42.78   40.74   39.97   41.89   40.48   41.17   1.02
{'test_rmse': array([1.58467458, 1.58160778, 1.58306427, 1.58192259, 1.58414689]), 'fit_
time': (11.469969749450684, 12.067939281463623, 12.51714825630188, 12.996087789535522, 1
1.882978200912476), 'test_time': (42.78344368934631, 40.741952657699585, 39.972779273986
816, 41.891682147979736, 40.47976732254028)}

The RMSE across five folds was 1.5830832199358247
```

In [67]:
```
highest_rated2 = data2.groupby(['user_id'])['rating'].transform(max) == data2['rating']
highest_rated_df2 = data2[highest_rated2]
total_highest_rated_movies2 = highest_rated_df2.groupby(['user_id','movie_id'])['rating
highest_rated_movies2 = total_highest_rated_movies2.groupby('user_id').apply(lambda x:
highest_rated_movies2['user_id'] = highest_rated_movies2['user_id'].astype(int)
highest_rated_movies2 = highest_rated_movies2.sort_values("user_id")
```

In [68]:
```
users_history2 = extract_users_movie(movies, highest_rated_movies2)
```

In [69]:
```
result2 = hybrid_recommender2(df,400,10,users_history2,2,8,dpdf,knn2)
```

In [70]:
```
print("Collaborative coverage", compute_mean_sim_score(400,result2[2],dpdf, movies)[3])
print("Hybrid based coverage", compute_mean_sim_score(400,result2[0],dpdf, movies)[3])
print("% difference in coverage", (compute_mean_sim_score(400,result2[0],dpdf, movies)[
```

```
Collaborative coverage 0.541
Hybrid based coverage 0.5986666666666667
% difference in coverage 5.766666666666664
```

Coverage comparison at 50 % sparsity

In [71]:
```
n_users = 401
n_movies = 1500
#generate a rating for each user/movie combination
data3 = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id f
np.random.seed(12)
randratings = np.random.randint(1,6, data3.shape[0])
data3['rating'] = randratings
data3["user_id"] = data3["user_id"].astype(str)
data3["movie_id"] = data3["movie_id"].astype(str)
data3.dtypes, data3.shape
```

Out[71]:
```
(user_id     object
 movie_id    object
 rating       int32
 dtype: object,
 (600000, 3))
```

In [72]:
```
(len(ratings) - len(data3))/len(ratings)
```

Out[72]:
```
0.5
```

In [73]:
```
our_seed = 14
reader3 = Reader(rating_scale=(1,5)) # defaults to (0,5)
d3 = Dataset.load_from_df(data3, reader3)
knn3 = KNNBasic(k= 4, n_jobs=-1, verbose=False) #the default for k is 40, we're also se
random.seed(our_seed)
np.random.seed(our_seed)
knn_cv3 = cross_validate(knn3, d3, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv3)
knn_RMSE3 = np.mean(knn_cv3['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE3}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5761  1.5798  1.5808  1.5812  1.5827  1.5801  0.0022
Fit time          8.61    9.53    9.30    8.99    8.69    9.02    0.35
Test time         35.51   33.88   32.02   35.44   38.53   35.07   2.15
{'test_rmse': array([1.57611062, 1.57982727, 1.58080716, 1.58120509, 1.58266862]), 'fit_
time': (8.612277030944824, 9.53093409538269, 9.297731876373291, 8.98525881767273, 8.6883
81433486938), 'test_time': (35.507365226745605, 33.877373456954956, 32.018059968948364,
35.43669366836548, 38.53493666648865)}

The RMSE across five folds was 1.580123751584545
```

In [74]:
```
highest_rated3 = data3.groupby(['user_id'])['rating'].transform(max) == data3['rating']
highest_rated_df3 = data3[highest_rated3]
```

```
total_highest_rated_movies3 = highest_rated_df3.groupby(['user_id','movie_id'])['rating
highest_rated_movies3 = total_highest_rated_movies3.groupby('user_id').apply(lambda x:
highest_rated_movies3['user_id'] = highest_rated_movies3['user_id'].astype(int)
highest_rated_movies3 = highest_rated_movies3.sort_values("user_id")
```

In [75]:
```
users_history3 = extract_users_movie(movies, highest_rated_movies3)
```

In [76]:
```
result3 = hybrid_recommender2(df,400,10,users_history3,2,8,dpdf,knn3)
```

In [77]:
```
print("Collaborative coverage",compute_mean_sim_score(400,result3[2],dpdf, movies)[3])
print("Hybrid based coverage", compute_mean_sim_score(400,result3[0],dpdf, movies)[3])
print("% difference in coverage", (compute_mean_sim_score(400,result3[0],dpdf, movies)[
```

```
Collaborative coverage 0.442
Hybrid based coverage 0.5393333333333333
% difference in coverage 9.733333333333333
```

Coverage comparison at 66.66 % sparsity

In [78]:
```
n_users = 401
n_movies = 1000
#generate a rating for each user/movie combination
data4 = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id f
np.random.seed(12)
randratings = np.random.randint(1,6, data4.shape[0])
data4['rating'] = randratings
data4["user_id"] = data4["user_id"].astype(str)
data4["movie_id"] = data4["movie_id"].astype(str)
data4.dtypes, data4.shape
print("Sparsity level", (len(ratings) - len(data4))/len(ratings) * 100)
```

```
Sparsity level 66.66666666666666
```

In [79]:
```
our_seed = 14
reader4 = Reader(rating_scale=(1,5))
d4 = Dataset.load_from_df(data4, reader4)
knn4 = KNNBasic(k= 4, n_jobs=-1, verbose=False)
random.seed(our_seed)
np.random.seed(our_seed)
knn_cv4 = cross_validate(knn4, d4, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv3)
knn_RMSE4 = np.mean(knn_cv4['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE4}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5766  1.5767  1.5789  1.5841  1.5798  1.5792  0.0027
Fit time          6.31    6.29    6.19    6.07    6.20    6.21    0.09
Test time         22.06   23.89   21.13   22.10   22.89   22.42   0.92
{'test_rmse': array([1.57611062, 1.57982727, 1.58080716, 1.58120509, 1.58266862]), 'fit_
time': (8.612277030944824, 9.53093409538269, 9.297731876373291, 8.98525881767273, 8.6883
81433486938), 'test_time': (35.507365226745605, 33.877373456954956, 32.018059968948364,
35.43669366836548, 38.53493666648865)}

The RMSE across five folds was 1.579211162818686
```

In [80]:
```
highest_rated4 = data4.groupby(['user_id'])['rating'].transform(max) == data4['rating']
highest_rated_df4 = data4[highest_rated3]
```

```
total_highest_rated_movies4 = highest_rated_df4.groupby(['user_id','movie_id'])['rating
highest_rated_movies4 = total_highest_rated_movies4.groupby('user_id').apply(lambda x:
highest_rated_movies4['user_id'] = highest_rated_movies4['user_id'].astype(int)
highest_rated_movies4 = highest_rated_movies4.sort_values("user_id")
```

In [81]:
```
users_history4 = extract_users_movie(movies, highest_rated_movies4)
```

In [82]:
```
result4 = hybrid_recommender2(df,400,10,users_history4,2,8,dpdf,knn4)
```

In [83]:
```
print("Collaborative coverage",compute_mean_sim_score(400,result4[2],dpdf, movies)[3])
print("Hybrid based coverage", compute_mean_sim_score(400,result4[0],dpdf, movies)[3])
print("% difference in coverage", (compute_mean_sim_score(400,result4[0],dpdf, movies)[
```

```
Collaborative coverage 0.31833333333333336
Hybrid based coverage 0.44633333333333336
% difference in coverage 12.8
```

Coverage comparison at 83.33 % sparsity

In [84]:
```
n_users = 401
n_movies = 500
#generate a rating for each user/movie combination
data5 = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id f
np.random.seed(12)
randratings = np.random.randint(1,6, data5.shape[0])
data5['rating'] = randratings
data5["user_id"] = data5["user_id"].astype(str)
data5["movie_id"] = data5["movie_id"].astype(str)
data5.dtypes, data4.shape
print("Sparsity level", (len(ratings) - len(data5))/len(ratings) * 100)
```

```
Sparsity level 83.33333333333334
```

In [85]:
```
our_seed = 14
reader5 = Reader(rating_scale=(1,5)) # defaults to (0,5)
d5 = Dataset.load_from_df(data5, reader5)
knn5 = KNNBasic(k= 4, n_jobs=-1, verbose=False)
random.seed(our_seed)
np.random.seed(our_seed)
knn_cv5 = cross_validate(knn5, d5, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv5)
knn_RMSE5 = np.mean(knn_cv5['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE5}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5784  1.5812  1.5830  1.5794  1.5880  1.5820  0.0034
Fit time          2.56    2.92    2.70    2.88    2.81    2.77    0.13
Test time         10.14   10.56   11.18   10.07   10.07   10.41   0.43
{'test_rmse': array([1.5783887 , 1.58122628, 1.58296136, 1.57937938, 1.58798607]), 'fit_
time': (2.5628745555877686, 2.9153695106506348, 2.696699619293213, 2.8804097175598145,
2.8139145374298096), 'test_time': (10.141149759292603, 10.559873580932617, 11.1838972568
51196, 10.071245908737183, 10.074365854263306)}

The RMSE across five folds was 1.5819883564771986
```

In [86]:
```
highest_rated5 = data5.groupby(['user_id'])['rating'].transform(max) == data5['rating']
highest_rated_df5 = data5[highest_rated5]
```

```
total_highest_rated_movies5 = highest_rated_df5.groupby(['user_id','movie_id'])['rating
highest_rated_movies5 = total_highest_rated_movies5.groupby('user_id').apply(lambda x:
highest_rated_movies5['user_id'] = highest_rated_movies5['user_id'].astype(int)
highest_rated_movies5 = highest_rated_movies5.sort_values("user_id")
```

In [87]:
```
users_history5 = extract_users_movie(movies, highest_rated_movies5)
result5 = hybrid_recommender2(df,400,10,users_history5,2,8,dpdf,knn5)
```

In [88]:
```
print("Collaborative coverage",compute_mean_sim_score(400,result5[2],dpdf, movies)[3])
print("Hybrid based coverage", compute_mean_sim_score(400,result5[0],dpdf, movies)[3])
print("% difference in coverage", (compute_mean_sim_score(400,result5[0],dpdf, movies)[
```

```
Collaborative coverage 0.16666666666666666
Hybrid based coverage 0.337
% difference in coverage 17.033333333333335
```

Coverage comparison at 91.66 % sparsity

In [89]:
```
n_users = 401
n_movies = 250
#generate a rating for each user/movie combination
data6 = pd.DataFrame(np.array(np.meshgrid([userid for userid in range(1,n_users)],[id f
np.random.seed(12)
randratings = np.random.randint(1,6, data6.shape[0])
data6['rating'] = randratings
data6["user_id"] = data6["user_id"].astype(str)
data6["movie_id"] = data6["movie_id"].astype(str)
data6.dtypes, data6.shape
print("Sparsity level", (len(ratings) - len(data6))/len(ratings) * 100)
```

```
Sparsity level 91.66666666666666
```

In [90]:
```
our_seed = 14
reader6 = Reader(rating_scale=(1,5))
d6 = Dataset.load_from_df(data6, reader6)
knn6 = KNNBasic(k= 4, n_jobs=-1, verbose=False)
random.seed(our_seed)
np.random.seed(our_seed)
knn_cv6 = cross_validate(knn6, d6, measures=['RMSE'], cv=5, verbose=True)
print(knn_cv6)
knn_RMSE6 = np.mean(knn_cv6['test_rmse'])
print(f'\nThe RMSE across five folds was {knn_RMSE6}')
```

```
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5765  1.5722  1.5748  1.5883  1.5803  1.5784  0.0056
Fit time          1.32    1.24    1.23    1.39    1.20    1.28    0.07
Test time         4.49    4.66    5.31    4.59    4.43    4.70    0.32
{'test_rmse': array([1.57653474, 1.57224313, 1.57483776, 1.58830412, 1.58031657]), 'fit_
time': (1.3204686641693115, 1.2416791915893555, 1.2316641807556152, 1.392242193222046,
1.204784631729126), 'test_time': (4.485004186630249, 4.661531686782837, 5.30979800224304
2, 4.586774587631226, 4.4321067333221436)}

The RMSE across five folds was 1.578447265269169
```

In [91]:
```
highest_rated6 = data6.groupby(['user_id'])['rating'].transform(max) == data6['rating']
highest_rated_df6 = data6[highest_rated6]
total_highest_rated_movies6 = highest_rated_df6.groupby(['user_id','movie_id'])['rating
highest_rated_movies6 = total_highest_rated_movies6.groupby('user_id').apply(lambda x:
```

```
highest_rated_movies6['user_id'] = highest_rated_movies6['user_id'].astype(int)
highest_rated_movies6 = highest_rated_movies6.sort_values("user_id")
```

In [92]:
```
users_history6 = extract_users_movie(movies, highest_rated_movies6)
result6 = hybrid_recommender2(df,400,10,users_history6,2,8,dpdf,knn6)
```

In [93]:
```
print("Collaborative coverage",compute_mean_sim_score(400,result6[2],dpdf, movies)[3])
print("Hybrid based coverage", compute_mean_sim_score(400,result6[0],dpdf, movies)[3])
print("% difference in coverage", (compute_mean_sim_score(400,result6[0],dpdf, movies)[
```

```
Collaborative coverage 0.08333333333333333
Hybrid based coverage 0.25766666666666665
% difference in coverage 17.433333333333334
```

Collect coverage values

In [94]:
```
collaborative_coverage_values= [compute_mean_sim_score(400,result[2],dpdf, movies)[3],
                                compute_mean_sim_score(400,result1[2],dpdf, movies)[3],
                                compute_mean_sim_score(400,result2[2],dpdf, movies)[3],
                                compute_mean_sim_score(400,result3[2],dpdf, movies)[3],
                                compute_mean_sim_score(400,result4[2],dpdf, movies)[3],
                                compute_mean_sim_score(400,result5[2],dpdf, movies)[3],
                                compute_mean_sim_score(400,result6[2],dpdf, movies)[3]]

Hybrid_coverage_values = [compute_mean_sim_score(400,result[0],dpdf, movies)[3],
                                compute_mean_sim_score(400,result1[0],dpdf, movies)[3],
                                compute_mean_sim_score(400,result2[0],dpdf, movies)[3],
                                compute_mean_sim_score(400,result3[0],dpdf, movies)[3],
                                compute_mean_sim_score(400,result4[0],dpdf, movies)[3],
                                compute_mean_sim_score(400,result5[0],dpdf, movies)[3],
                                compute_mean_sim_score(400,result6[0],dpdf, movies)[3]]
Coverage_difference = list()
for item1, item2 in zip(Hybrid_coverage_values ,collaborative_coverage_values):
    item = item1 - item2
    Coverage_difference.append(item* 100)

Sparsity_level = [0,16.7,33.3,50, 66.67, 83.33, 91.67]
# Coverage_difference = Hybrid_coverage_values - collaborative_coverage_values
```

In [95]:
```
Coverage_data = pd.DataFrame(
    {'% Data sparsity level':Sparsity_level ,'Collaborative coverage': collaborative_co
     'Hybrid coverage': Hybrid_coverage_values ,
     '% Difference in coverage by hybrid model vs Collab-model': Coverage_difference
    })
Coverage_data
```

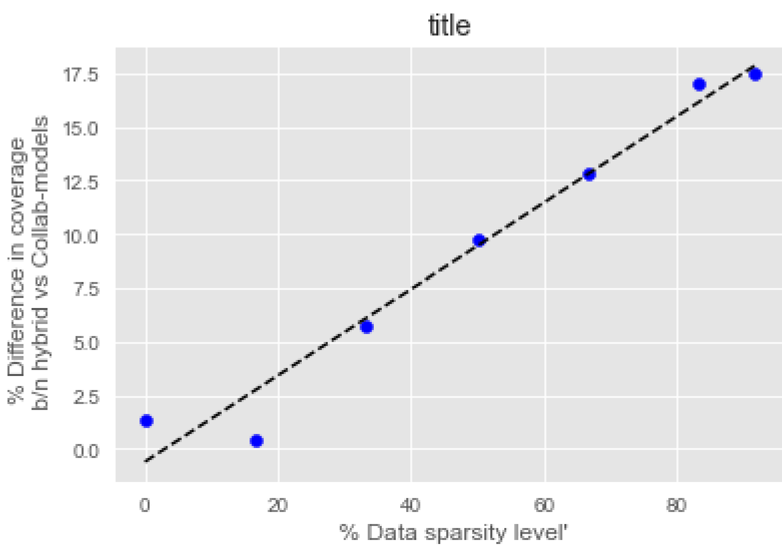
Out[95]:

	% Data sparsity level	Collaborative coverage	Hybrid coverage	% Difference in coverage by hybrid model vs Collab-model
0	0.00	0.681333	0.694667	1.333333
1	16.70	0.620000	0.623667	0.366667
2	33.30	0.541000	0.598667	5.766667
3	50.00	0.442000	0.539333	9.733333
4	66.67	0.318333	0.446333	12.800000
5	83.33	0.166667	0.337000	17.033333

	% Data sparsity level	Collaborative coverage	Hybrid coverage	% Difference in coverage by hybrid model vs Collab-model
6	91.67	0.083333	0.257667	17.433333

In [96]:
```
plt.scatter(Coverage_data['% Data sparsity level'], Coverage_data['% Difference in cove
z = np.polyfit(Coverage_data['% Data sparsity level'], Coverage_data['% Difference in c
p = np.poly1d(z)
plt.plot(Coverage_data['% Data sparsity level'],p(Coverage_data['% Data sparsity level'
plt.title("title")
plt.xlabel(" % Data sparsity level'")
plt.ylabel("% Difference in coverage \n b/n hybrid vs Collab-models")
plt.show()

# plt.show()
```



Diversity comparison

In [97]:
```
content_diversity_values= [compute_mean_sim_score(400,result[1],dpdf, movies)[2],
                                compute_mean_sim_score(400,result1[1],dpdf, movies)[2],
                                compute_mean_sim_score(400,result2[1],dpdf, movies)[2],
                                compute_mean_sim_score(400,result3[1],dpdf, movies)[2],
                                compute_mean_sim_score(400,result4[1],dpdf, movies)[2],
                                compute_mean_sim_score(400,result5[1],dpdf, movies)[2],
                                compute_mean_sim_score(400,result6[1],dpdf, movies)[2]]

Hybrid_diversity_values = [compute_mean_sim_score(400,result[0],dpdf, movies)[2],
                                compute_mean_sim_score(400,result1[0],dpdf, movies)[2],
                                compute_mean_sim_score(400,result2[0],dpdf, movies)[2],
                                compute_mean_sim_score(400,result3[0],dpdf, movies)[2],
                                compute_mean_sim_score(400,result4[0],dpdf, movies)[2],
                                compute_mean_sim_score(400,result5[0],dpdf, movies)[2],
                                compute_mean_sim_score(400,result6[0],dpdf, movies)[2]]
diversity_difference= list()
for item1, item2 in zip(Hybrid_diversity_values ,content_diversity_values):
    item = item1 - item2
    diversity_difference.append(item* 100)
```

```
Sparsity_level = [0,16.7,33.3,50, 66.67, 83.33, 91.67]
# Coverage_difference = Hybrid_coverage_values - collaborative_coverage_values
```

In [98]:

```
diversity_data = pd.DataFrame(
    {'% Data sparsity level':Sparsity_level ,'Content model diversity': content_diversi
     'Hybrid model diversity': Hybrid_diversity_values,
     '% Difference in diversity': diversity_difference
    })
diversity_data
```

Out[98]:

	% Data sparsity level	Content model diversity	Hybrid model diversity	% Difference in diversity
0	0.00	0.535078	0.927037	39.195864
1	16.70	0.428490	0.912906	48.441589
2	33.30	0.526261	0.925958	39.969780
3	50.00	0.518247	0.924863	40.661618
4	66.67	0.538587	0.922692	38.410487
5	83.33	0.526765	0.923430	39.666460
6	91.67	0.523503	0.917408	39.390465

In [99]:

```
diversity_data[['Content model diversity', 'Hybrid model diversity']].to_numpy()
```

Out[99]:

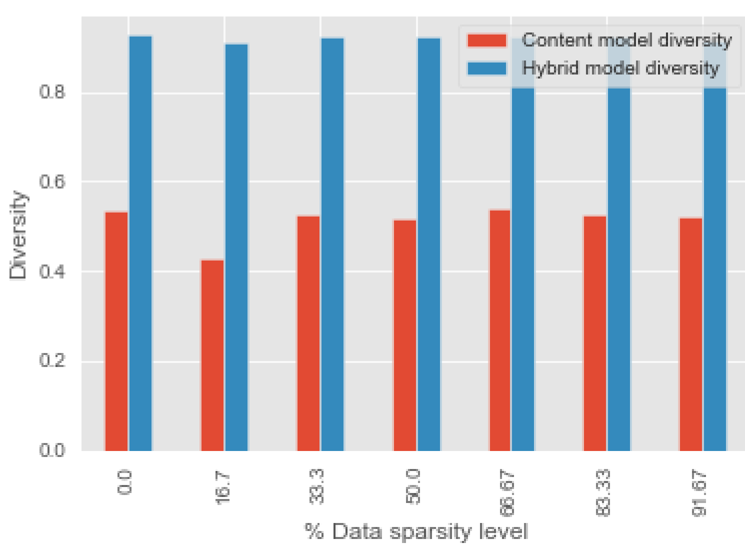
```
array([[0.53507846, 0.9270371 ],
       [0.4284904 , 0.91290629],
       [0.52626051, 0.92595832],
       [0.5182469 , 0.92486308],
       [0.53858691, 0.92269178],
       [0.52676508, 0.92342969],
       [0.5235031 , 0.91740776]])
```

In [100…

```
diversity_data.plot(x= "% Data sparsity level", y =["Content model diversity","Hybrid m
```

Out[100…

```
Text(0, 0.5, 'Diversity')
```



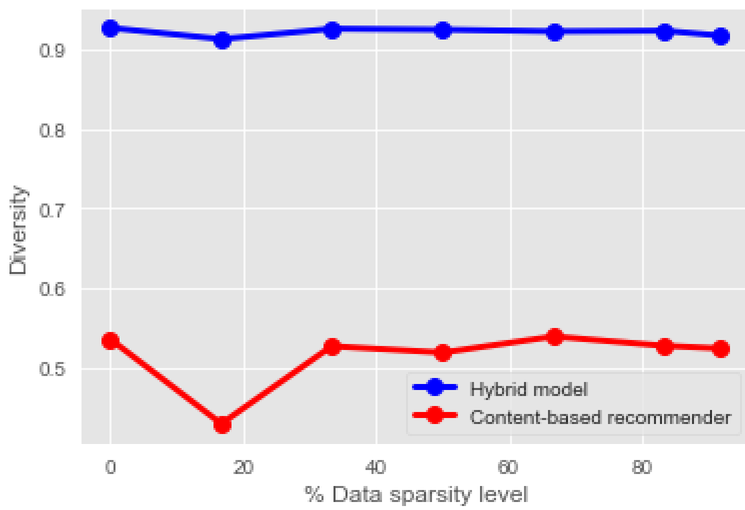
In [112…

```
x = diversity_data['% Data sparsity level']
y = diversity_data['Content model diversity']
z = diversity_data['Hybrid model diversity']
```

```
# Plot a simple line chart
plt.plot(x, z, 'b', label='Hybrid model', marker='o', markersize=8,linewidth=3)

plt.plot(x, y, 'r', label='Content-based recommender',marker='o',markersize=8,linewidth

# Plot another line on the same chart/graph
plt.xlabel('% Data sparsity level')
plt.ylabel('Diversity')
# plt.legend(loc='upper right')
plt.legend()
plt.show()
```

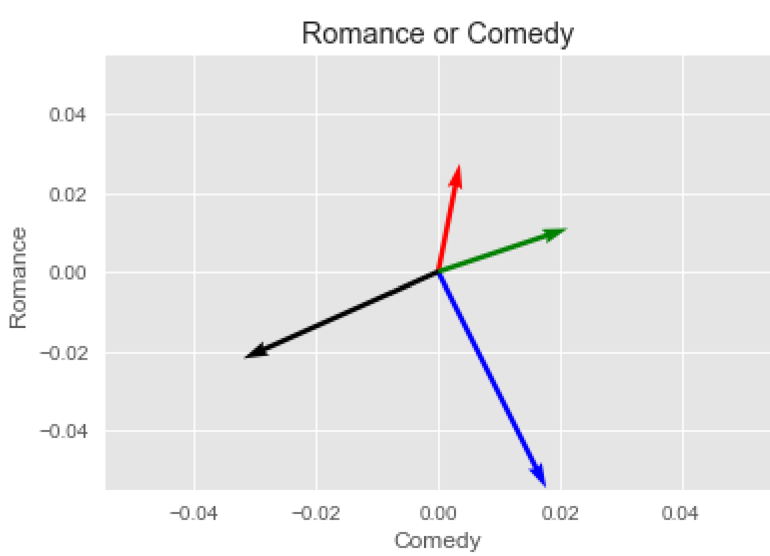


Sample cosine similarity illustration

In [101…

```
V = np.array([[1,5], [5,-10], [6,2], [-9,-4]])
origin = np.array([[0, 0, 0, 0],[0, 0, 0, 0]]) # origin point

plt.quiver(*origin, V[:,0], V[:,1], color=['r','b','g','black'], scale=31)
plt.ylabel('Romance')
plt.xlabel('Comedy')
plt.title('Romance or Comedy')
plt.show()
```



hybrid recommender gives a higher diversity than the content based recommender. resolves the filter bubble.

Hybrid recommender gives a higher coverage than collaborative filtering due to its collaboration with a conent based recommendet and thus considering items with no rating values. resolving the data sparsity