Problem formulation

In this project, we will be building our recommender system, similar to the one used by Netflix. Some of the business questions which we will seek to answer include the following:

1. Given a user's history or movie preferences, which movie is the user likely to be interested in?
2. Should we adopt a binary class approach (recommend or not recommend) or a multi-class approach (ratings - 1, 2, 3, 4, and 5)
3. If there is more than one movie to recommend, what ranking system should be used to determine the order of the movie list?

Data sets:

Using TMDB Data Set

The first dataset contains the following features:

- crew
- cast
- movie id
- title

The second dataset has the following features:

- budget
- genre
- homepage
- id
- keywords
- original_language
- original_title
- overview
- popularity
- production_companies
- production_countries
- release_date
- revenue
- runtime
- status
- tagline
- title
- vote_average
- vote_count

Saved Models: "SVDppModel.pickle" and "modelSVDppLoo_1M.pickle".

The [MovieLens] data set

	id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...
...	...	...	...	...
4798	9367	El Mariachi	[{"cast_id": 1, "character": "El Mariachi", "c...	[{"credit_id": "52fe44eec3a36847f80b280b", "de...
4799	72766	Newlyweds	[{"cast_id": 1, "character": "Buzzy", "credit_...	[{"credit_id": "52fe487dc3a368484e0fb013", "de...
4800	231617	Signed, Sealed, Delivered	[{"cast_id": 8, "character": "Oliver O\u2019To...	[{"credit_id": "52fe4df3c3a36847f8275ecf", "de...
4801	126186	Shanghai Calling	[{"cast_id": 3, "character": "Sam", "credit_id...	[{"credit_id": "52fe4ad9c3a368484e16a36b", "de...
4802	25975	My Date with Drew	[{"cast_id": 3, "character": "Herself", "credi...	[{"credit_id": "58ce021b9251415a390165d9", "de...

4803 rows × 4 columns

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
5	1	70	3.0	964982400
6	1	101	5.0	964980868
7	1	110	4.0	964982176
8	1	151	5.0	964984041
9	1	157	5.0	964984100

Data Preparation

1. Merging TMDB Data Sets

```
df_all = pd.merge (df1 , df2, how = 'inner' , on='id')
df_all.drop("original_title" , axis = 1)
df_all.drop("title_y" , axis = 1)
```

2. Cleaning TMDB Data Set

```
# Function to convert all strings to lower case and strip names of spaces
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

```
[ ] # Apply clean_data function to your features.
    features = ['cast', 'keywords', 'director', 'genres']

    for feature in features:
        df_all[feature] = df_all[feature].apply(clean_data)
```

```
# removing timestamp from dataframe 'ratings'
newRatings = ratings.iloc[:,:-1]
newRatings.head(10)
```

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0
5	1	70	3.0
6	1	101	5.0
7	1	110	4.0
8	1	151	5.0
9	1	157	5.0

3. Check null values in rating data and scaling it

```
print('The count of null values, after dropping null values:')
newRatings.isna().sum()
```

```
The count of null values, after dropping null values:
userId      0
movieId     0
rating      0
dtype: int64
```

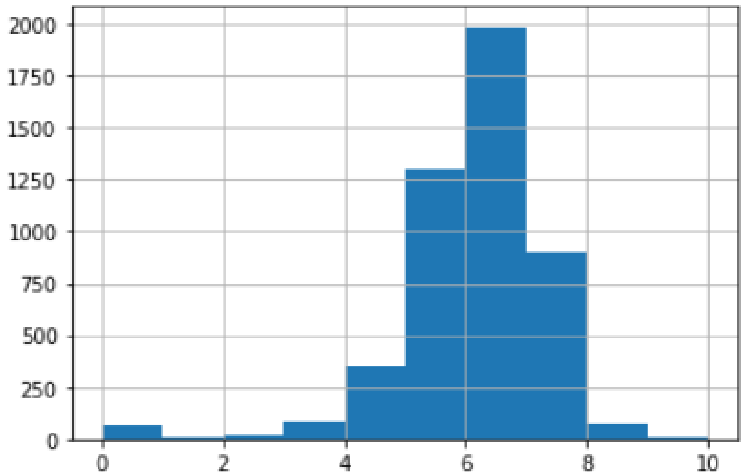
```
# Scaling rating from 0.5 into 5.0
reader = Reader(rating_scale=(0.5, 5.0))
sRatings = Dataset.load_from_df(newRatings[['userId', 'movieId', 'rating']], reader)
```

Methods:

Demographic Filtering :

calculate the mean which is our c variable

```
hist = df_all['vote_average'].hist(bins=10)
```

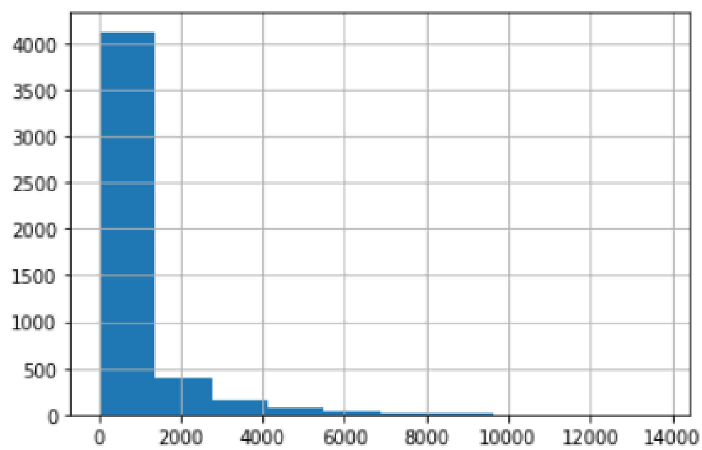


Calculate the min votes which is m

```
m= df_all['vote_count'].quantile(0.9)
m
```

```
1838.4000000000015
```

```
hist = df_all['vote_count'].hist(bins=10)
```



Filtering out the movies that qualify for the chart

```
[ ]  q_movies = df_all.copy().loc[df_all['vote_count'] >= m]
     q_movies.shape
```

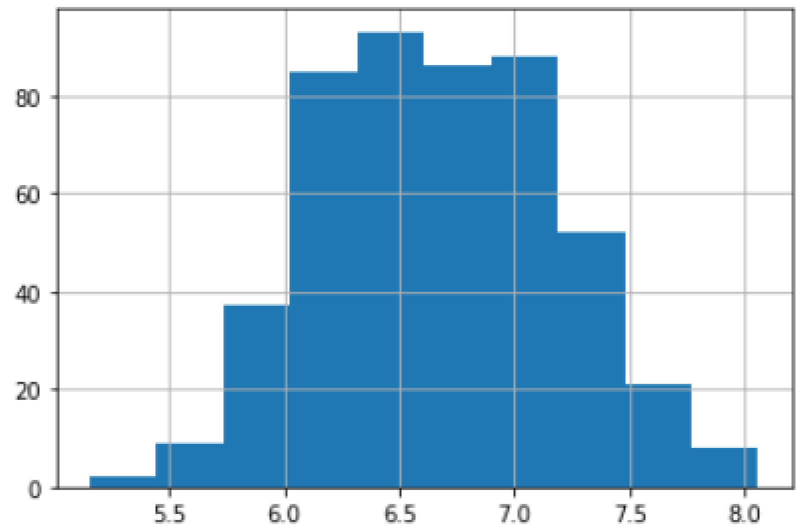
```
(481, 23)
```

sortting movies

	title_x	vote_count	vote_average	score
1881	The Shawshank Redemption	8205	8.5	8.059258
662	Fight Club	9413	8.3	7.939256
65	The Dark Knight	12002	8.2	7.920020
3232	Pulp Fiction	8428	8.3	7.904645
96	Inception	13752	8.1	7.863239
3337	The Godfather	5893	8.4	7.851236
95	Interstellar	10867	8.1	7.809479
809	Forrest Gump	7927	8.2	7.803188
329	The Lord of the Rings: The Return of the King	8064	8.1	7.727243
1990	The Empire Strikes Back	5879	8.2	7.697884
262	The Lord of the Rings: The Fellowship of the Ring	8705	8.0	7.667341
2912	Star Wars	6624	8.1	7.663813
1818	Schindler's List	4329	8.3	7.641883

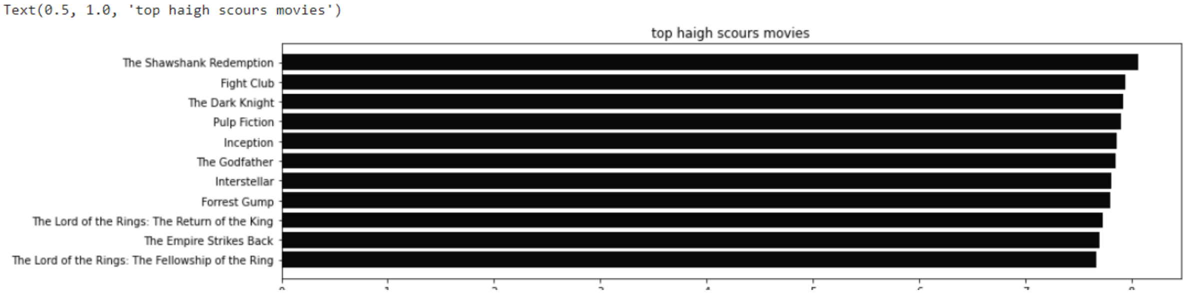
Define score for the movies

```
hist = q_movies['score'].hist(bins=10)
```



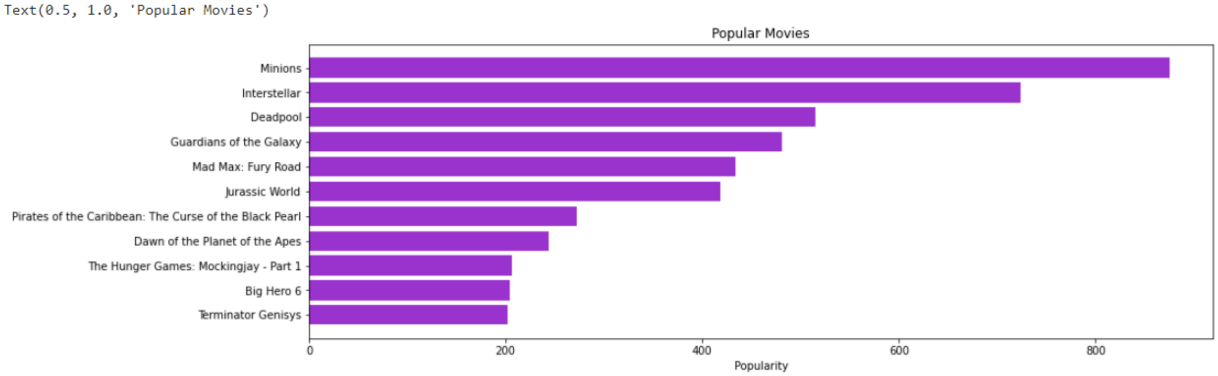
Showing the top 10 high scours movies

```
Text(0.5, 1.0, 'top haigh scours movies')
```



Showing the top 10 popular movies

```
Text(0.5, 1.0, 'Popular Movies')
```



Content Based Filtering

```
df_all['overview'].head(8)
```

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
5    The seemingly invincible Spider-Man goes up ag...
6    When the kingdom's most wanted-and most charmi...
7    When Tony Stark tries to jumpstart a dormant p...
Name: overview, dtype: object
```

Finding similarity from the movie title and overview only

```
get_recommendations('The Dark Knight Rises')
```

```
65                               The Dark Knight
299                               Batman Forever
428                               Batman Returns
1359                                      Batman
3854    Batman: The Dark Knight Returns, Part 2
119                                Batman Begins
2507                                   Slow Burn
9             Batman v Superman: Dawn of Justice
1181                                         JFK
210                               Batman & Robin
Name: title_x, dtype: object
```

Finding similarity from the Credits, Genres and Keywords

```
# Print the new features of the first 5 films
df_all[['title_x', 'cast', 'director', 'keywords', 'genres']].head(5)
```

	title_x	cast	director	keywords	genres
0	Avatar	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	James Cameron	[culture clash, future, space war]	[Action, Adventure, Fantasy]
1	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	Gore Verbinski	[ocean, drug abuse, exotic island]	[Adventure, Fantasy, Action]
2	Spectre	[Daniel Craig, Christoph Waltz, Léa Seydoux]	Sam Mendes	[spy, based on novel, secret agent]	[Action, Adventure, Crime]
3	The Dark Knight Rises	[Christian Bale, Michael Caine, Gary Oldman]	Christopher Nolan	[dc comics, crime fighter, terrorist]	[Action, Crime, Drama]
4	John Carter	[Taylor Kitsch, Lynn Collins, Samantha Morton]	Andrew Stanton	[based on novel, mars, medallion]	[Action, Adventure, Science Fiction]

Text Feature Engineering

TF-IDF Vectorizer

TF-IDF Vectorizer

```
[ ] #Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
    tfidf = TfidfVectorizer(stop_words='english')

    #Replace NaN with an empty string
    df_all['overview'] = df_all['overview'].fillna('')

    #Construct the required TF-IDF matrix by fitting and transforming the data
    tfidf_matrix = tfidf.fit_transform(df_all['overview'])

    #Output the shape of tfidf_matrix
    tfidf_matrix.shape
```

```
(4803, 20978)
```

CountVectorizer instead of TF-iDF:

This is because we do not want to down-weight the presence of an actor/director if he or she has acted or directed in relatively more movies. It doesn't make much intuitive sense.

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df_all['soup'])
```

```
# Compute the Cosine Similarity matrix based on the count_matrix

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```
# Reset index of our main DataFrame and construct reverse mapping as before
df_all = df_all.reset_index()
indices = pd.Series(df_all.index, index=df_all['title_x'])
```

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

```
65                 The Dark Knight
119                  Batman Begins
4638    Amidst the Devil's Wings
1196                  The Prestige
3073             Romeo Is Bleeding
3326                Black November
1503                        Takers
1986                        Faster
303                       Catwoman
747                 Gangster Squad
Name: title_x, dtype: object
```

Collaborative Filtering

We used some algorithms in surprise library:

- Matrix Factorization-base models: SVD, SVDpp
- Classification: KNNBaseline , KNNBasic, KNNWithMeans, KNNWithZScore
- Cluster: CoClustering

```
Classification (KNNBaseline(), KNNBasic(), KNNWithMeans(), KNNWithZScore())
Cluster(CoClustering())
algors = [KNNBaseline(), KNNBasic(), KNNWithMeans(), KNNWithZScore(),
              CoClustering(), SVD(), SVDpp()]
benchmark = []
for alg in algors:
    print("Starting: " ,str(alg))
    # Perform cross validation
    results = cross_validate(alg, sRatings, measures=['RMSE'], cv=3, verbose=False)
    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(alg).split(' ')[0].split('.')[-1]], index=['Algorithm']))
    benchmark.append(tmp)
    print("Done: " ,str(alg), "\n\n")
```

```
getting RMSE and time of training and testing for all classification, cluster, and Matrix Factorization-based models
surprise_results = pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
surprise_results
```

Algorithm	test_rmse	fit_time	test_time
SVDpp	0.890262	69.757271	2.499507
SVD	0.899533	1.458985	0.110629
KNNBaseline	0.916795	0.056012	0.580003
KNNWithZScore	0.935753	0.046119	0.484627
KNNWithMeans	0.942184	0.027898	0.446881
CoClustering	0.996825	0.670121	0.055480
KNNBasic	1.035113	0.020917	0.407386

Split dataset After Scaling to train champoin model:

```
# split dataSets 'newRatings' into trainSet (70) and testSet(30)
trainSet, testSet = train_test_split(sRatings, test_size=0.3)
trainingParams = {}
```

We chose the champion model (SVDpp):

The reason is the test_rmse of SVDpp model is the smallest value, but it is taking a lot of time so we will apply GridSearch CV to get the best values of parameters in SVDpp Model:

```
print(trainingParams)

print("------------STARTING-----------\n\n")
start = datetime.now()

startTraining = datetime.now()
print("> Training...")

algor = SVDpp(n_epochs = trainingParams['n_epochs'], lr_all = trainingParams['lr_all'], reg_all = trainingParams['reg_all'])

algor.fit(trainSet)

endTraining = datetime.now()
print("> OK \t\t It Took: ", (endTraining-startTraining).seconds, "seconds")

end = datetime.now()
print ("> > DONE \t\t It Took", (end-start).seconds, "seconds" )
```

```
{'n_epochs': 15, 'lr_all': 0.005, 'reg_all': 0.01}
------------STARTING-----------


> Training...
> OK            It Took:  418 seconds
>> DONE                 It Took 418 seconds
```

After training the champion model (SVDpp), we saved it in ‘SVDppModel.pickle’.
So we shouldn’t train SVDpp model again.

```
## SAVING TRAINED MODEL
model_filename = "./SVDppModel.pickle"
print (">> Starting dump")
# Dump algorithm and reload it.
file_name = os.path.expanduser(model_filename)
dump.dump(file_name, algo=algor)
print (">> Dump done")
print(model_filename)
```

```
>> Starting dump
>> Dump done
./SVDppModel.pickle
```

Evalution on and all All models and champion model (SVD):

Algorithm	test_rmse	fit_time	test_time
SVDpp	0.890262	69.757271	2.499507
SVD	0.899533	1.458985	0.110629
KNNBaseline	0.916795	0.056012	0.580003
KNNWithZScore	0.935753	0.046119	0.484627
KNNWithMeans	0.942184	0.027898	0.446881
CoClustering	0.996825	0.670121	0.055480
KNNBasic	1.035113	0.020917	0.407386

Load champion Model (SVDapp)

```
# loading model 'SVDppModel.pickle'
loadModel = load_model('SVDppModel.pickle')
```

```
>> Loading dump (SVDppModel.pickle)
>> Done
```

```
# getting predictions, after loading SVDpp model and using 30% (testSet) from dataSet 'sRatings'
predictions = loadModel.test(testSet)
```

By predicting by SVDpp, we can determine the best and worst predictions.

```
# The best Predictions for champion Model (SVDapp)
best_predictions
```

	uid	iid	rui	est	details	Iu	Ui	err
8149	275	909	5.0	5.0	{'was_impossible': False}	282	15	0.0
24288	513	750	5.0	5.0	{'was_impossible': False}	22	66	0.0
12792	275	1300	5.0	5.0	{'was_impossible': False}	282	7	0.0
18084	561	1196	5.0	5.0	{'was_impossible': False}	354	152	0.0
19294	474	912	5.0	5.0	{'was_impossible': False}	1472	66	0.0
29124	275	2797	5.0	5.0	{'was_impossible': False}	282	63	0.0
1794	282	527	5.0	5.0	{'was_impossible': False}	172	159	0.0
4543	367	858	5.0	5.0	{'was_impossible': False}	119	130	0.0
23550	275	3083	5.0	5.0	{'was_impossible': False}	282	13	0.0
23181	275	1247	5.0	5.0	{'was_impossible': False}	282	51	0.0

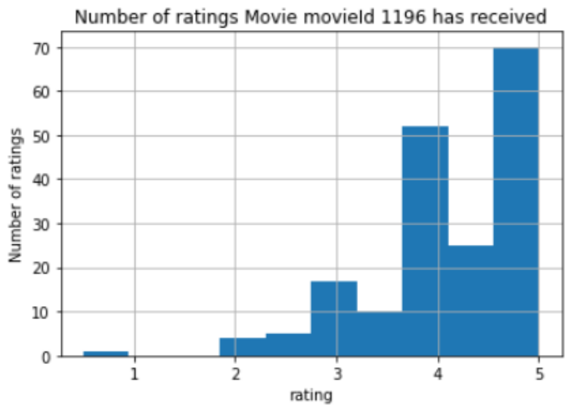
```
# # The worst Predictions for champion Model (SVDapp)
worst_predictions
```

	uid	iid	rui	est	details	Iu	Ui	err
27291	308	1203	0.5	4.647482	{'was_impossible': False}	79	43	4.147482
4517	510	34405	0.5	4.726259	{'was_impossible': False}	78	38	4.226259
26328	153	106916	0.5	4.744578	{'was_impossible': False}	125	12	4.244578
22899	153	102125	0.5	4.824737	{'was_impossible': False}	125	22	4.324737
10396	153	1580	0.5	4.856587	{'was_impossible': False}	125	122	4.356587
10242	308	2959	0.5	4.931781	{'was_impossible': False}	79	161	4.431781
179	153	2571	0.5	5.000000	{'was_impossible': False}	125	206	4.500000
18926	153	1198	0.5	5.000000	{'was_impossible': False}	125	131	4.500000
19449	153	2959	0.5	5.000000	{'was_impossible': False}	125	161	4.500000
2730	153	1196	0.5	5.000000	{'was_impossible': False}	125	152	4.500000

In the lift table, the best predictions are not lucky guesses. Because Ui is anywhere between 13 to 104, they are not really small, meaning that significant number of users have rated the target movie.

Plot Number of ratings Movie movieId 1196 (The worst Predictions for champion Model (SVDapp)) has received

```
filter_RatingsDS.loc[filter_RatingsDS['movieId'] == 1196]['rating'].hist()
plt.xlabel('rating')
plt.ylabel('Number of ratings')
plt.title('Number of ratings Movie movieId 1196 has received')
plt.show()
```



Example to test champion model SVDpp:

Those are the liked movies for the user with id 26

```
] # getting 10 top movies userID 26.
  getTopMovieRS(26)
```

```
43             Seven (a.k.a. Se7en) (1995)
123                       Apollo 13 (1995)
126                  Batman Forever (1995)
138     Die Hard: With a Vengeance (1995)
176                      Waterworld (1995)
192                      Disclosure (1994)
260                       Quiz Show (1994)
302     Ace Ventura: Pet Detective (1994)
398                   Fugitive, The (1993)
510       Silence of the Lambs, The (1991)
Name: title, dtype: object
```

Error Analysis:

```
# using One million (1,000,000) records of 25M in Error Analysis
testdataError = FullrecomSystem.iloc[:1000000,:]
```

```
# # rescaling 1M
reader = Reader(rating_scale=(0.5, 5))
data = Dataset.load_from_df(testdataError[['userId', 'movieId', 'rating']], reader)
```

We removed one of these movies (Leave-One-Out cross-validation).

To evaluate top-10, we used hit rate, that is, if a user rated one of the top-10 we recommended, we consider it is a "hit".

```
LOOCV = LeaveOneOut(n_splits=1, random_state=1)
modelSVDppLoo = SVDpp(n_epochs = trainingParams['n_epochs'],
                      lr_all = trainingParams['lr_all'],
                      reg_all = trainingParams['reg_all'],random_state=1)
for trainSet, testSet in LOOCV.split(data):
    # Train model without left-out ratings
    modelSVDppLoo.fit(trainSet)
    # Predicts ratings for left-out ratings only
    leftOutPredictions = modelSVDppLoo.test(testSet)
```

We can save the training model of Leave-One-Out cross-validation in 'modelSVDppLoo_1M.pickle', so we shouldn't train it.

```
## Saving Training MODEL of
model_filename = "./modelSVDppLoo_1M.pickle"
print (">> Starting dump")
# Dump algorithm and reload it.
file_name = os.path.expanduser(model_filename)
dump.dump(file_name, algo=modelSVDppLoo)
print (">> Dump done")
print(model_filename)
```

```
# loading model 'modelSVDppLoo_1M.pickle'
loadModelLoo = load_model('modelSVDppLoo_1M.pickle')
for _,testSet in LOOCV.split(data):
  # Predicts ratings for left-out ratings only
 leftOutPredictions = loadModelLoo.test(testSet)
```

```
>> Loading dump (modelSVDppLoo_1M.pickle)
>> Done
```

We used **Hit Rate by Rating Value on ‘**modelSVDppLoo_1M.pickle**’**

By using the predicted rating values, we can deconstruct hit rate. In an ideal world, we would be able to predict how well-liked a movie will be by its audience.

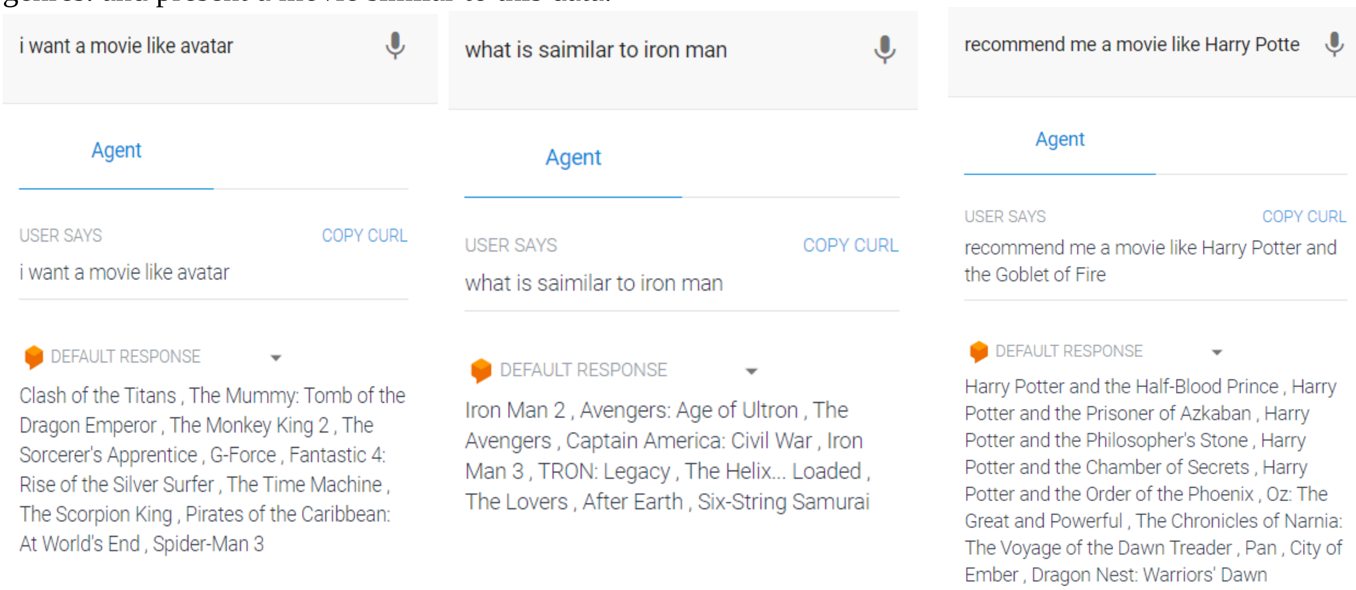
```
def RatingHitRate(topNPredicted, leftOutPredictions):
    hits = defaultdict(float)
    total = defaultdict(float)
    # For each left-out rating
    for userID, leftOutMovieID, actualRating, estimatedRating, _ in leftOutPredictions:
        # Is it in the predicted top N for this user?
        hit = False
        for movieID, predictedRating in topNPredicted[int(userID)]:
            if (int(leftOutMovieID) == movieID):
                hit = True
                break
        if (hit) :
            hits[actualRating] += 1
        total[actualRating] += 1

    # Compute overall precision
    for rating in sorted(hits.keys()):
        print(rating, hits[rating] / total[rating])
print("Hit Rate by Rating value: ")
RatingHitRate(all_pred, leftOutPredictions)
```

```
Hit Rate by Rating value:
1.0 0.05555555555555555
1.5 0.14285714285714285
2.0 0.08571428571428572
3.0 0.03759398496240601
3.5 0.03571428571428571
4.0 0.06875
4.5 0.05555555555555555
5.0 0.08870967741935484
```

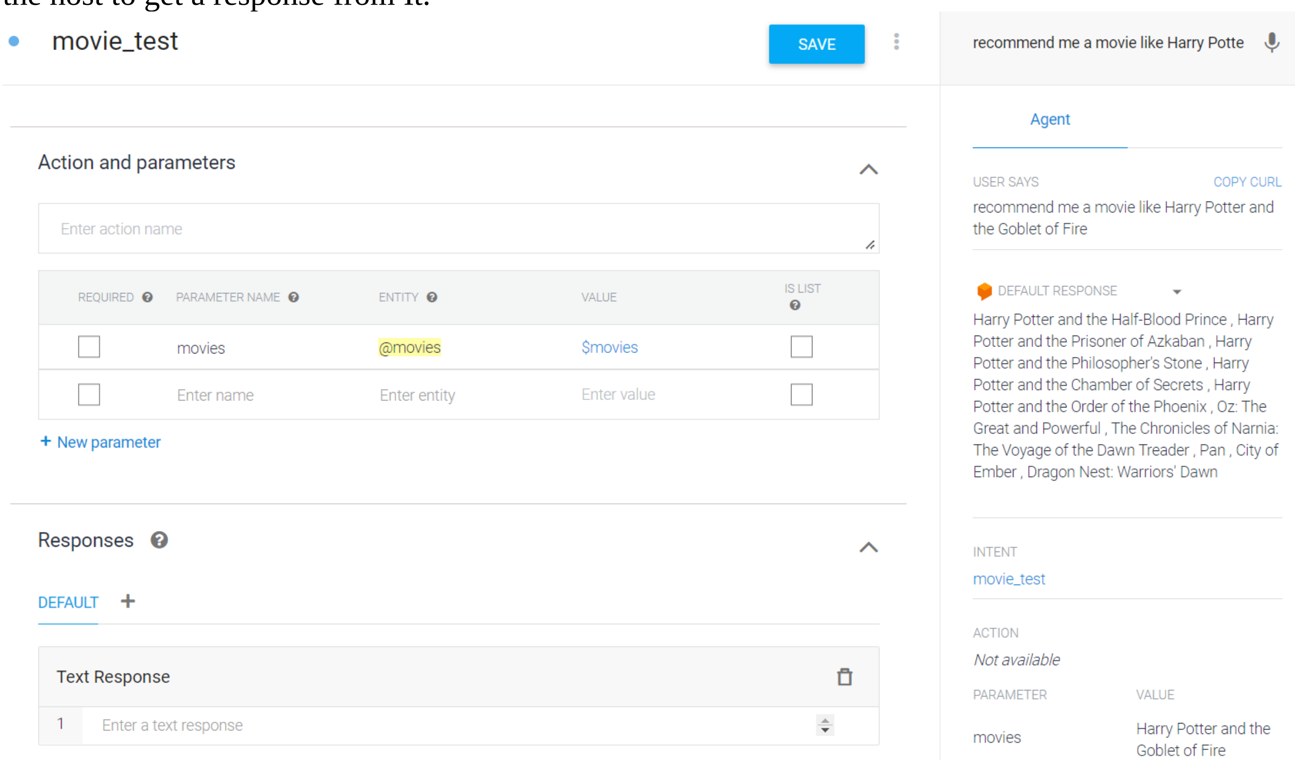
Our hit rate distribution matches my expectations completely. The rating score 5 has a substantially better hit rate than the others. Consequently, our champion model (SVDpp) can anticipate it correctly.

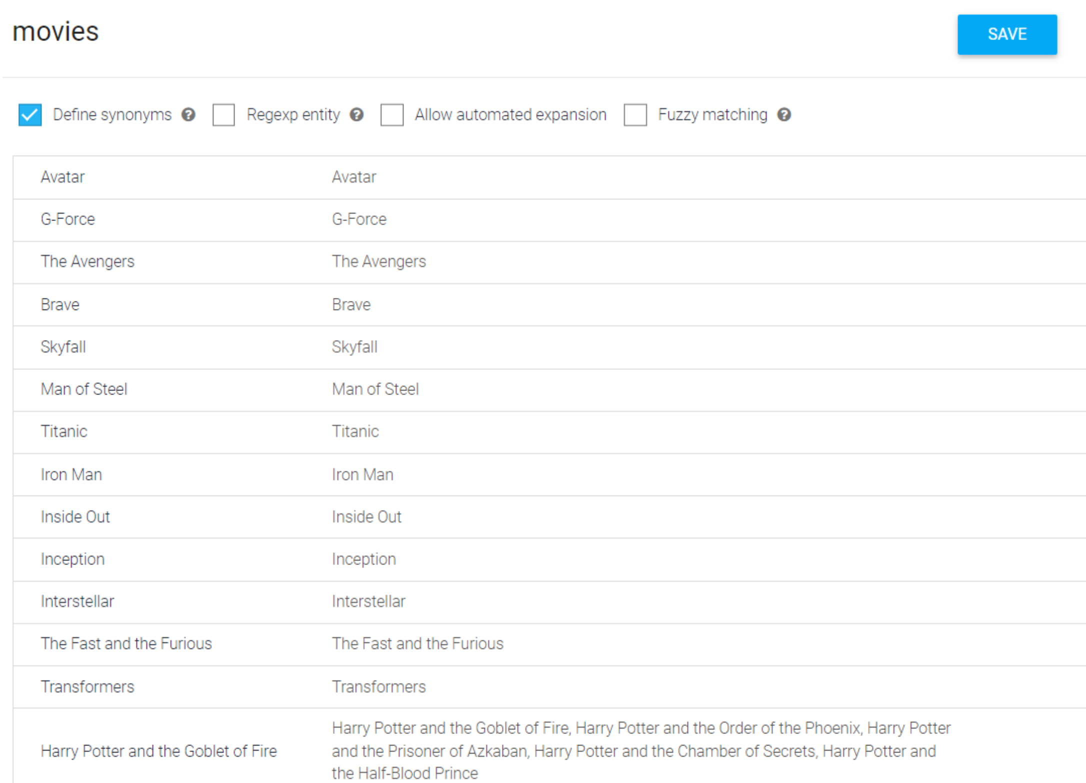
Visualization of results, and graphical intuition and analysis

Connecting our code with Dialog flow chatbot via **NGROK**
First, let's test content-based recommendation which is recommending a movie similar to a movie you ask for, and work to analyze the movie metadata: cast, keywords, director, and genres. and present a movie similar to this data.



As we see in this figure there is no response to any question. It extracts any movie name that we have provided in the entity names and then communicates directly to the host to get a response from It.





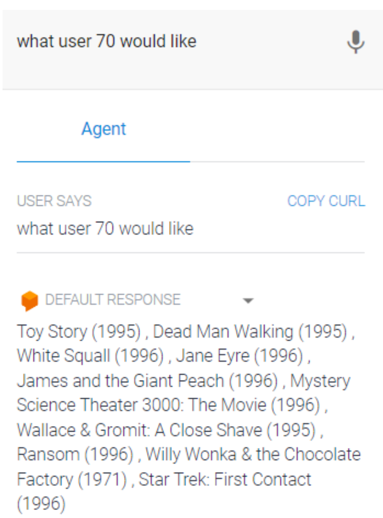
movies

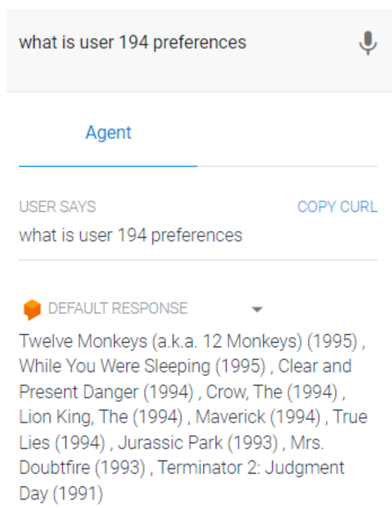
SAVE

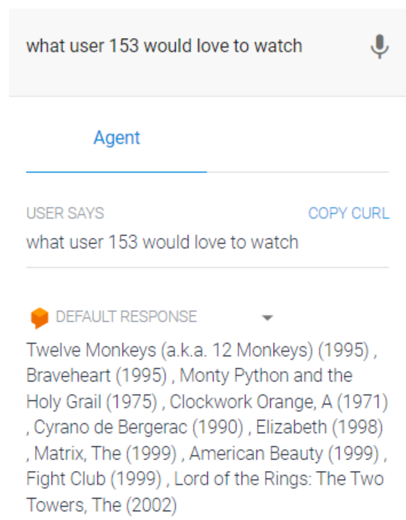
Define synonyms | Regexp entity | Allow automated expansion | Fuzzy matching

Avatar	Avatar
G-Force	G-Force
The Avengers	The Avengers
Brave	Brave
Skyfall	Skyfall
Man of Steel	Man of Steel
Titanic	Titanic
Iron Man	Iron Man
Inside Out	Inside Out
Inception	Inception
Interstellar	Interstellar
The Fast and the Furious	The Fast and the Furious
Transformers	Transformers
Harry Potter and the Goblet of Fire	Harry Potter and the Goblet of Fire, Harry Potter and the Order of the Phoenix, Harry Potter and the Prisoner of Azkaban, Harry Potter and the Chamber of Secrets, Harry Potter and the Half-Blood Prince

This entity contains some examples of our data set to catch any movie in our dataset immediately
Second, let's test collaborative filtering by knowing users what would like to watch according to their history







In this feature, the chatbot extracts the number of the user and gets preferences for him

[Dialogflow (google.com)](https://dialogflow.cloud.google.com)

Innovativeness

We applied every recommendation like non-personalized and personalized, starting with the non-personalized, which is a probability estimation similarity using the IMDB formula. Then we jumped into the personalization and we applied to content based on the movie's output. As a feature, we will add the score from the weighted rate formula that we calculated to improve the recommendation. Then we apply the content with our Metadata ("cast, crew, directory, and keywords") by using cosine similarity with TFiDF to make the recommendation more accurate. Also, we used a technique called Collaborative Based Filtering, which is an approach where user ratings come into account, and hence there can be different outputs possible on the basis of the reviews given to the items or movies in focus. We compared even more algorithms (SVD, SVDpp, KNNBaseline, KNNBasic, KNNWithMeans, KNNWithZScore, and CoClustering) to select the best champion model (SVDpp). We applied GridSearch to reduce the training time for the SVDpp problem. This champion can extract movies that are more suitable for a particular user. We integrated these methods that applied more models such as cosine similarity and SVDpp with a chatbot that can recommend top movies to users by some previous standards.