Recommender Systems

Iordanis Koutsopoulos - professor (jordan@aueb.gr)

Project: Recommender systems optimization for coverage, diversity and serendipity

George Karoulas
ID:P3351805

George Koroniotis
ID:P3351808

Panagiotis Rallis
ID:P3351816

Maria Stamouli
ID:P3351820

Menelaos Fragkiadakis
ID:P3351822

Table of Contents

1.Introduction

For the purpose of our group project we will implement a recommendation system in the light of several recommender algorithms and evaluation metrics such as coverage, diversity and serendipity. We are using items of several categories which will be recommended to users. Specifically, the items we examine are movies which have been rated from each individual user. The first problem we address is, given a set of ratings for each user and each movie, generated through a conventional recommender system algorithm, our goal is to recommend movies to users such that the ratings of items in the recommendation lists are maximized subject to maintaining sufficient coverage and diversity for each set of movies.

The second problem is to maximize the total serendipity of recommended items subject to guaranteeing a given minimum serendipity for each set of items. Serendipity can be defined in various ways. Here we define it through a combination of item popularity and its difference from items a user has experienced so far. We show that these problems may be viewed as assignment problems. We will perform extensive numerical experiments with real data that verify the findings of our approach.

2.Problem A: Recommendation for Diversity and Coverage

2.1 Data Sources Description

In order to solve Problem A, we are using the dataset MovieLens and more precisely the *Movielens ml-latest-small*. The Ratings file includes the below information.

- userID : refers to each user
- movieID : the corresponding Id of the rated movie
- rating : user rate in a 5 star scale (0,5 to 5 with a step of 0,5)
- timestamp : represent seconds since midnight

This dataset contains 100,000 ratings from 671 users and 9,125 movies.

Furthermore, the Movie file is used to retrieve the name of each movie.

2.2 Data Preprocessing

Firstly it was necessary to filter the dataset by removing the movies that have less than fifty ratings and the corresponding users who have rated less than thirty movies.

In order to apply the constraints of Problem A, the movies were splitted in 10 random pseudo categories.

A sample of the dataset

	userId	movieId	title	rating	timestamp	Category
0	1	1	Toy Story (1995)	4.0	1970-01-01 00:00:00.964982703	7
1	1	3	Grumpier Old Men (1995)	4.0	1970-01-01 00:00:00.964981247	2
2	1	6	Heat (1995)	4.0	1970-01-01 00:00:00.964982224	2
3	1	47	Seven (a.k.a. Se7en) (1995)	5.0	1970-01-01 00:00:00.964983815	10
4	1	50	Usual Suspects, The (1995)	5.0	1970-01-01 00:00:00.964982931	1
5	1	70	From Dusk Till Dawn (1996)	3.0	1970-01-01 00:00:00.964982400	5
6	1	110	Braveheart (1995)	4.0	1970-01-01 00:00:00.964982176	10
7	1	163	Desperado (1995)	5.0	1970-01-01 00:00:00.964983650	7
8	1	223	Clerks (1994)	3.0	1970-01-01 00:00:00.964980985	9
9	1	231	Dumb & Dumber (Dumb and Dumber) (1994)	5.0	1970-01-01 00:00:00.964981179	2

Item-Item Collaborative Filtering

Collaborative Filtering (CF) represents today's widely adopted strategy to build recommendation engines.

Idea of Item-Item CF: Find similar items to those that I have previously liked, in order to predict the rate of an unseen movie.

There is a variety of similarity methods that could be used. Most popular of them are **Pearson Correlation** and **Cosine Similarity**. The below results refer to the similarity of movies using the current ratings.

Pearson Correlation sample:

	1	2	3	6	7	10	11	16	17	19
1	1.000000	0.179770	0.112437	0.098310	0.072752	0.167058	0.147498	0.066013	0.129882	0.196285
2	0.179770	1.000000	0.179294	0.085122	0.102948	0.235787	0.156321	0.029587	0.059945	0.357226
3	0.112437	0.179294	1.000000	0.114399	0.364542	0.127868	0.162518	0.201292	0.107414	0.217006
6	0.098310	0.085122	0.114399	1.000000	0.089559	0.222340	0.135839	0.397297	0.071550	0.080929
7	0.072752	0.102948	0.364542	0.089559	1.000000	0.139871	0.360419	-0.006098	0.265248	0.104413
10	0.167058	0.235787	0.127868	0.222340	0.139871	1.000000	0.199985	0.083894	0.040940	0.268414
11	0.147498	0.156321	0.162518	0.135839	0.360419	0.199985	1.000000	0.096104	0.314683	0.061964
16	0.066013	0.029587	0.201292	0.397297	-0.006098	0.083894	0.096104	1.000000	0.071823	0.112561
17	0.129882	0.059945	0.107414	0.071550	0.265248	0.040940	0.314683	0.071823	1.000000	0.033050
19	0.196285	0.357226	0.217006	0.080929	0.104413	0.268414	0.061964	0.112561	0.033050	1.000000

k=20 Nearest Neighbors of movieID 2

```
array([ 231,  316,  367,  480,  500,  552,  592,  596,  648,  673,  736,
        780, 1073, 1580, 2012, 2054, 2115, 2617, 2628, 3489])
```

Cosine Similarity sample:

	1	2	3	6	7	10	11	16	17	19
1	1.000000	0.452472	0.284195	0.383202	0.249859	0.460831	0.367282	0.332650	0.326486	0.421742
2	0.452472	1.000000	0.303040	0.307868	0.235917	0.443118	0.326830	0.243159	0.228368	0.498329
3	0.284195	0.303040	1.000000	0.243198	0.422817	0.267442	0.265045	0.306103	0.204777	0.320032
6	0.383202	0.307868	0.243198	1.000000	0.218177	0.421781	0.301758	0.522424	0.229176	0.273427
7	0.249859	0.235917	0.422817	0.218177	1.000000	0.272398	0.434746	0.124233	0.342567	0.219808
10	0.460831	0.443118	0.267442	0.421781	0.272398	1.000000	0.369454	0.296258	0.222404	0.437600
11	0.367282	0.326830	0.265045	0.301758	0.434746	0.369454	1.000000	0.253462	0.411902	0.226118
16	0.332650	0.243159	0.306103	0.522424	0.124233	0.296258	0.253462	1.000000	0.214655	0.280713
17	0.326486	0.228368	0.204777	0.229176	0.342567	0.222404	0.411902	0.214655	1.000000	0.182756
19	0.421742	0.498329	0.320032	0.273427	0.219808	0.437600	0.226118	0.280713	0.182756	1.000000

k=20 Nearest Neighbors of movieID 2

```
array([   1,  110,  231,  316,  356,  367,  457,  480,  500,  590,  592,
        648,  780, 1073, 1580, 2012, 2054, 2115, 2617, 2628])
```

Comparing the above results we can realise that 15 of the 20 most similar movies of movieID 2, are the same in both similarity methods.

The similarity method that will be used, does not affect the scope of Problem A and it is just needed in order to predict the unrated movies. Thus it was selected the Pearson Correlation method to be used by the prediction expression.

Rating Matrix

The next step is to compute the predicted rating of unseen movies using the similarity metrics and a K Nearest Neighbors approach, using the expression below.

$$r_{xi} = \frac{\sum_{j \epsilon N(i;x)} s_{ij} r_{xj}}{\sum_{j \epsilon N(i;x)} s_{ij}}$$

$\mathbf{s_{ij}}$… similarity of items **i** and **j**
$\mathbf{r_{xj}}$…rating of user **x** on item **j**
N(i;x)… set items rated by **x** similar to **i**

Finally, after all the preprocess assumptions, the Item-Item Collaborative Filter method results in a Rating Matrix with shape 387(users) x 450(movies).

A sample of the rating matrix. NaN values indicate movies which were already watched and rated.

movieId	1	2	3	6	7	10	11	16	17	19
userId										
1	NaN	3.72	NaN	NaN	4.08	3.75	4.01	4.51	4.22	4.09
4	3.73	3.17	3.32	3.39	2.91	3.05	3.11	3.28	3.27	3.10
5	NaN	3.51	3.29	3.46	3.31	3.19	3.28	3.61	3.45	3.45
6	4.02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	3.52	3.42	3.96	3.55	3.51	3.69	3.25	3.22	3.59
8	3.57	NaN	3.16	3.65	3.51	NaN	NaN	3.72	3.57	3.33
10	3.24	3.28	3.03	2.87	3.21	3.58	3.16	2.93	3.14	3.24
11	4.01	3.53	3.55	NaN	3.70	NaN	3.94	4.01	4.02	3.53
14	2.83	2.96	3.04	3.70	NaN	3.11	3.62	3.59	3.86	NaN
15	NaN	3.24	3.24	4.02	3.37	3.53	3.48	3.74	3.24	3.15

Baseline Recommendation

The baseline recommendation list consists of the top L rated items of each user.

The 10 size Baseline Recommendation list of userID = 1

	movieId	title	Category	Rating
351	5952	Lord of the Rings: The Two Towers, The (2002)	2	4.88
441	91529	Dark Knight Rises, The (2012)	2	4.88
336	4993	Lord of the Rings: The Fellowship of the Ring,...	6	4.87
433	76093	How to Train Your Dragon (2010)	5	4.87
381	8874	Shaun of the Dead (2004)	2	4.86
408	51255	Hot Fuzz (2007)	3	4.85
402	48516	Departed, The (2006)	9	4.83
424	68358	Star Trek (2009)	8	4.83
436	80463	Social Network, The (2010)	4	4.83
334	4973	Amelie (Fabuleux destin d'Amélie Poulain, Le) ...	4	4.82

The 5 size Baseline Recommendation list of userID = 4

	movieId	title	Category	Rating
127	953	It's a Wonderful Life (1946)	7	4.34
272	2804	Christmas Story, A (1983)	5	4.27
150	1207	To Kill a Mockingbird (1962)	4	4.23
125	923	Citizen Kane (1941)	5	4.15
219	1997	Exorcist, The (1973)	9	4.11

Dissimilarity among Users

The same approach was used in order to compute the dissimilarity of users, according to their ratings.

Pearson Correlation sample:

	1	4	5	6	7	8	10	11	14	15
1	1.000000	0.238971	0.096821	0.027975	0.065004	0.059860	-0.121533	0.076933	0.061545	0.097805
4	0.238971	1.000000	0.018433	-0.037615	0.027164	-0.015816	-0.044287	-0.018221	-0.030066	-0.046168
5	0.096821	0.018433	1.000000	0.449113	0.022191	0.483117	-0.011442	0.197870	0.250355	0.064170
6	0.027975	-0.037615	0.449113	1.000000	-0.047477	0.537333	-0.075488	0.338383	0.479118	-0.058249
7	0.065004	0.027164	0.022191	-0.047477	1.000000	0.016436	0.173451	0.177209	-0.011631	0.210606
8	0.059860	-0.015816	0.483117	0.537333	0.016436	1.000000	-0.044258	0.224597	0.443382	0.015190
10	-0.121533	-0.044287	-0.011442	-0.075488	0.173451	-0.044258	1.000000	-0.027362	-0.020371	0.143160
11	0.076933	-0.018221	0.197870	0.338383	0.177209	0.224597	-0.027362	1.000000	0.213674	0.017032
14	0.061545	-0.030066	0.250355	0.479118	-0.011631	0.443382	-0.020371	0.213674	1.000000	0.005380
15	0.097805	-0.046168	0.064170	-0.058249	0.210606	0.015190	0.143160	0.017032	0.005380	1.000000

2.3 Model Building and Assessment

The main subject of this project is to change the baseline recommended list, applying the constraints of Coverage and Diversity, with the minimum changing cost.

We need to maximize the below expression,

$$\max \frac{1}{L|U|} \sum_{i \in I} \sum_{u \epsilon U} r_{iu} x_{iu}$$

fulfil the constraints:

1. $\sum_c \sum_{i \epsilon Ic} x_{iu}$ = L, for each user
2. $\mathrm{Div}_c \geq \mathrm{D}$,
3. $\mathrm{Cov}_c = \mathrm{K}_c$

Coverage

The Coverage is defined as the percentage of times an item of category c is assigned to users' list. More specifically, an item is covered if it is recommended at least to a certain number of users.

$$\mathrm{Cov}_c\ (\mathrm{x}) = \sum_{i \in Ic} \sum_{u \in U} x_{iu} = \mathrm{K}_c \qquad (1)$$

where

- Ic : the number of movies in each category
- Kc : the coverage percentage of each category

According to the expression (1), there is a need to filter the unknown variable X for each category.

Below is an explanation of this implementation.

Considering the X array with 2 users and 3 movies. For each category, it was created an array Category_Index with shape 2x3, with 1 if the movie belongs to the category and 0 if not.

Category: Pandemic

	The Flu	Titanic	Contagion
user 1	1	0	1
user 2	1	0	1

Category: Drama

	The Flu	Titanic	Contagion
user 1	0	1	0
user 2	0	1	0

X Matrix:

	The Flu	Titanic	Contagion
user 1	$X_{1,\text{THE FLU}}$	$X_{1,\text{Titanic}}$	$X_{1,\text{Contagion}}$
user 2	$X_{2,\text{THE FLU}}$	$X_{2,\text{Titanic}}$	$X_{2,\text{Contagion}}$

In order to filter the X matrix, it is needed for each category, to multiply each element of X matrix with each element of the corresponding category matrix.

```
Category: 0 , Count: 39
Category: 1 , Count: 48
Category: 2 , Count: 58
Category: 3 , Count: 52
Category: 4 , Count: 43
Category: 5 , Count: 48
Category: 6 , Count: 41
Category: 7 , Count: 36
Category: 8 , Count: 46
Category: 9 , Count: 39
```

Summarizing the items of each category which have been recommended for the Baseline and using the constraints, we can observe that the movies of most popular categories, such as category 6, were reduced using the Coverage Constraint and the movies of less popular categories were increased. This means that the algorithm tries to balance the recommended lists according to the categories.

Baseline Recommendation

1	443
2	309
3	269
4	417
5	327
6	561
7	458
8	410
9	337
10	339

Optimization Using Coverage and L constraint Kc = 0.2 and L=10

1	445
2	318
3	293
4	431
5	311
6	525
7	439
8	403
9	349
10	356

Below examples depict the effect of coverage constraint to the recommendation lists.

USER ID: 20

Baseline Recommendation top 10 list

	movieId	title	Category	Rating
157	1220	Blues Brothers, The (1980)	4	4.59
160	1225	Amadeus (1984)	2	4.56
127	953	It's a Wonderful Life (1946)	3	4.54
0	1	Toy Story (1995)	10	4.54
240	2324	Life Is Beautiful (La Vita è bella) (1997)	10	4.52
143	1197	Princess Bride, The (1987)	5	4.52
150	1207	To Kill a Mockingbird (1962)	4	4.49
172	1278	Young Frankenstein (1974)	9	4.48
15	39	Clueless (1995)	7	4.47
254	2599	Election (1999)	7	4.47

Optimization using Coverage=0.2 and L=10

	movieId	title	Category	Rating
4	1220	Blues Brothers, The (1980)	4	4.59
5	1225	Amadeus (1984)	2	4.56
0	1	Toy Story (1995)	10	4.54
1	953	It's a Wonderful Life (1946)	3	4.54
2	1197	Princess Bride, The (1987)	5	4.52
8	2324	Life Is Beautiful (La Vita è bella) (1997)	10	4.52
3	1207	To Kill a Mockingbird (1962)	4	4.49
7	1278	Young Frankenstein (1974)	9	4.48
6	1259	Stand by Me (1986)	5	4.47
9	2599	Election (1999)	7	4.47

As it can be observed, the movie Clueless of category 7 was replaced by the movie Stand by me of category 5. However, both movies have the same rating.

For the same user > *Baseline Recommendation top 5 list*

	movieId	title	Category	Rating
157	1220	Blues Brothers, The (1980)	4	4.59
160	1225	Amadeus (1984)	2	4.56
127	953	It's a Wonderful Life (1946)	3	4.54
0	1	Toy Story (1995)	10	4.54
240	2324	Life Is Beautiful (La Vita è bella) (1997)	10	4.52

Optimization using Coverage=0.2 and L=5

	movieId	title	Category	Rating
3	1220	Blues Brothers, The (1980)	4	4.59
4	1225	Amadeus (1984)	2	4.56
0	1	Toy Story (1995)	10	4.54
1	953	It's a Wonderful Life (1946)	3	4.54
2	1197	Princess Bride, The (1987)	5	4.52

The movie Life is Beautiful of category 10 was replaced by the movie The Princess Bride of category 5.

Trying to explain the effect of different Coverage values to the recommended lists, the same example was performed for the same user with Kc = 0.02.

Optimization using Coverage=0.02 and L=5

	movieId	title	Category	Rating
3	1220	Blues Brothers, The (1980)	4	4.59
4	1225	Amadeus (1984)	2	4.56
0	1	Toy Story (1995)	10	4.54
1	953	It's a Wonderful Life (1946)	3	4.54
2	1197	Princess Bride, The (1987)	5	4.52

As we can observe, the same movie was replaced by the Princess Bride.

Looking deeper on the Coverage constraint and on distinct movies of each category,

distinct movies of each category:

```
Category: 0 , Count: 39
Category: 1 , Count: 48
Category: 2 , Count: 58
Category: 3 , Count: 52
Category: 4 , Count: 43
Category: 5 , Count: 48
Category: 6 , Count: 41
Category: 7 , Count: 36
Category: 8 , Count: 46
Category: 9 , Count: 39
```

So, we observe that for the K constraint, indeed the per category recommendations to the user change, consistently for different values of K. But the total Average Rating for all users and all items remains the same. We can explain this, because next available movies to be recommended to the user have a very close Rating value.

Diversity

In the diversity metric we are interested in assigning items in Ic in as diverse a set of users as possible in order to expand their reach to new audiences.

$$\mathrm{Div_c} = \frac{2}{|I_c|} \frac{\sum_{i \in I_c} \sum_{u \in U} \sum_{u \epsilon U: v \neq u} d_{uv} x_{iu} x_{iv}}{K_c |I_c| \times (K_c |I_c| - 1)}$$

For the calculation of diversity we could not programmatically compute the solution of the formed quadratic equation, therefore we implemented an alternative solution for the calculation of diversity.

According to the definition of diversity, one category should be recommended to dissimilar users. While the variable of category was created manually and contains a small range of numbers, the constraint of diversity is already achieved by the baseline recommended list. Therefore we implement a constraint about the diversity of movies. If two users are dissimilar, it should be recommended to them for a relatively similar set of movies. As a movie belongs to one category, this assures that this implementation achieves the diversity of categories too.

Assuming that the new recommended list for user u is X_u and for user v is X_v. In order to compute the similarity of X_u and X_v, we used the norm 2 distance of the two vectors.

cp.norm(Xu-Xv)

If the two users are dissimilar, we need the distance of two vectors to be near to zero. On the other hand, for similar users we need a norm near to 1.

Therefore the inequality of constraint was set to the division:

D / dissimilarity(u,v)

where D is a given constant.

For example, if we set D = 0.4, for two dissimilar users the value of dissimilarity will be near to 1.

Therefore the constraint

$$\text{cp.norm(Xu-Xv)} <= 0.4/1$$

assures that the two vectors should be almost similar. In other words, the recommended movies in two dissimilar users will be the same.

On the other hand for two similar users, the value of dissimilarity will be near to zero. In our case the dissimilarity values range from 0.4 to 0.99.

Therefore the constraint will be

$$\text{cp.norm(Xu-Xv)} <= 0.4/0.4$$

and the recommendations will be not affected.

Even with the aforementioned solution, the application of the above constraint for 370 users took hours, therefore we applied it only to the userId=6 with all other users, in order to confirm our actions. More specifically the constraint assures that the recommended list of a dissimilar user to user 6 is close enough to the recommended list of user 6. The choice of user 6 was performed because it has similar and dissimilar users as well.

UserID: 6

Baseline top 10 list

	movieId	title	Category	Rating
210	1721	Titanic (1997)	9	4.33
129	1035	Sound of Music, The (1965)	9	4.26
116	786	Eraser (1996)	5	4.25
110	733	Rock, The (1996)	5	4.20
94	586	Home Alone (1990)	8	4.19
137	1101	Top Gun (1986)	10	4.16
197	1580	Men in Black (a.k.a. MIB) (1997)	9	4.16
314	4022	Cast Away (2000)	7	4.14
106	648	Mission: Impossible (1996)	10	4.12
17	48	Pocahontas (1995)	3	4.08

optimization Kc=0.2, L=10, D=0.3

	movieId	title	Category	Rating
3	1721	Titanic (1997)	9	4.33
1	1035	Sound of Music, The (1965)	9	4.26
0	786	Eraser (1996)	5	4.25
2	1101	Top Gun (1986)	10	4.16
4	4022	Cast Away (2000)	7	4.14
6	4995	Beautiful Mind, A (2001)	4	4.02
5	4896	Harry Potter and the Sorcerer's Stone (a.k.a. ...	1	3.97
8	6942	Love Actually (2003)	3	3.93
7	6377	Finding Nemo (2003)	9	3.92
9	76093	How to Train Your Dragon (2010)	3	3.91

As we can observe, there is a significant change of movies on user's 6 recommended list, applying the diversity constraint. We need to compare the recommended list of a dissimilar user of user 6, in order to approve that in dissimilar users, it was recommended the same movies.

Dissimilarity between user 6 and user 1 is 0.97202.

UserID = 1

optimization Kc=0.2, L=10, D=0.3

	movieId	title	Category	Rating
9	76093	How to Train Your Dragon (2010)	3	4.87
5	4995	Beautiful Mind, A (2001)	4	4.82
8	59315	Iron Man (2008)	2	4.82
7	6942	Love Actually (2003)	3	4.77
4	4896	Harry Potter and the Sorcerer's Stone (a.k.a. ...	1	4.75
6	6377	Finding Nemo (2003)	9	4.72
3	4022	Cast Away (2000)	7	4.67
0	1035	Sound of Music, The (1965)	9	4.52
1	1101	Top Gun (1986)	10	4.48
2	1721	Titanic (1997)	9	4.35

As we can observe there are 9/10 recommended movies to both dissimilar users.

Now we need to compare two similar users, to validate that there is not an impact to the recommended lists.

Dissimilarity between user 6 and user 8 is 0.46266744

UserId = 8

optimization Kc=0.2, L=10, D=0.3

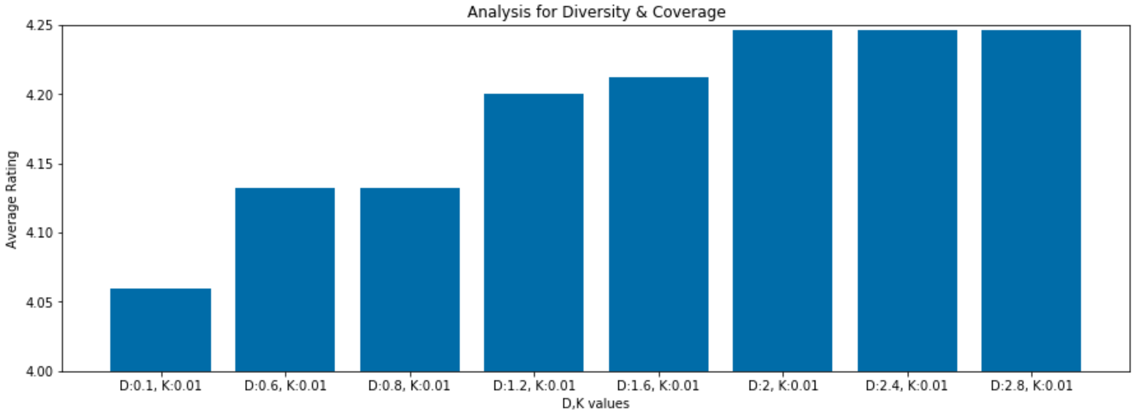
	movieId	title	Category	Rating
9	116797	The Imitation Game (2014)	6	4.47
8	99114	Django Unchained (2012)	8	4.27
0	858	Godfather, The (1972)	7	4.21
4	6016	City of God (Cidade de Deus) (2002)	10	4.21
7	79132	Inception (2010)	7	4.21
2	2959	Fight Club (1999)	5	4.20
6	74458	Shutter Island (2010)	5	4.20
1	2858	American Beauty (1999)	10	4.19
3	4973	Amelie (Fabuleux destin d'Amélie Poulain, Le) ...	6	4.18
5	58559	Dark Knight, The (2008)	10	4.17

Comparing the recommended list of user 8 with the relatively similar user 6, there is no movie which is recommended to both of them.

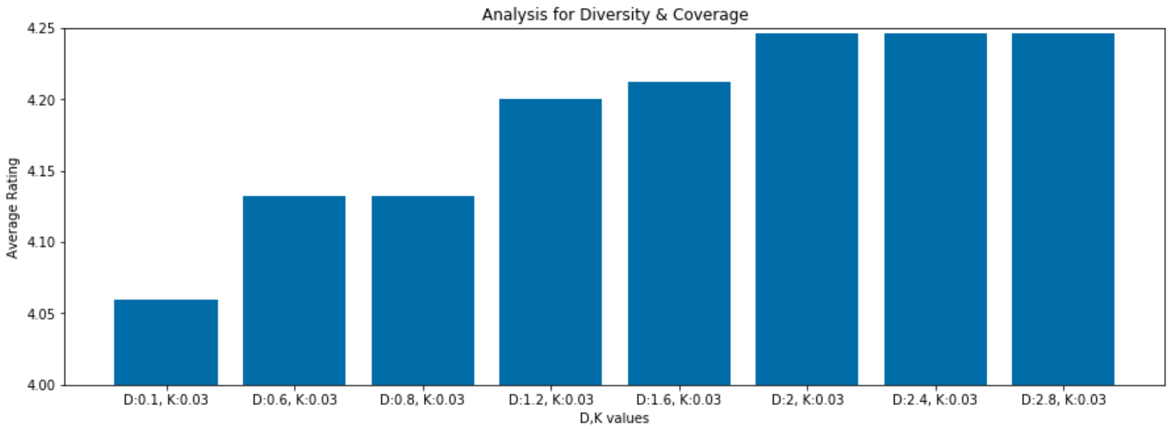
2.4 Plots & Results

For D= [0.1,0.6,0.8,1.2,1.6,2,2.4,2.8] , L=[5,10] and K= [0.01,0.03,0.05,0.07,0.1,0.15]

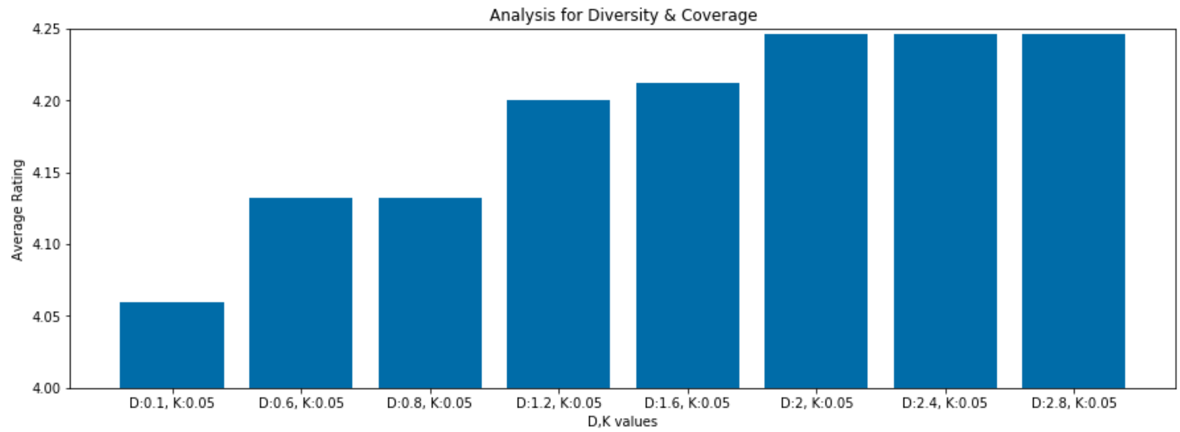
L=5, different D,K values , continuous values, default cvxpy solver



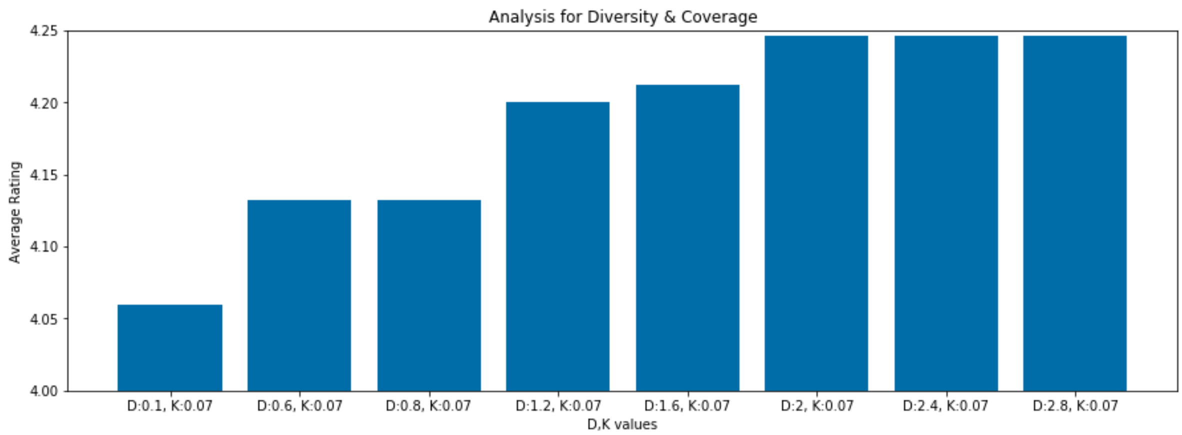
	D_K	D	K	Average_Rating
0	D:0.1, K:0.01	0.1	0.01	4.060
1	D:0.6, K:0.01	0.6	0.01	4.132
2	D:0.8, K:0.01	0.8	0.01	4.132
3	D:1.2, K:0.01	1.2	0.01	4.200
4	D:1.6, K:0.01	1.6	0.01	4.212
5	D:2, K:0.01	2.0	0.01	4.246
6	D:2.4, K:0.01	2.4	0.01	4.246
7	D:2.8, K:0.01	2.8	0.01	4.246



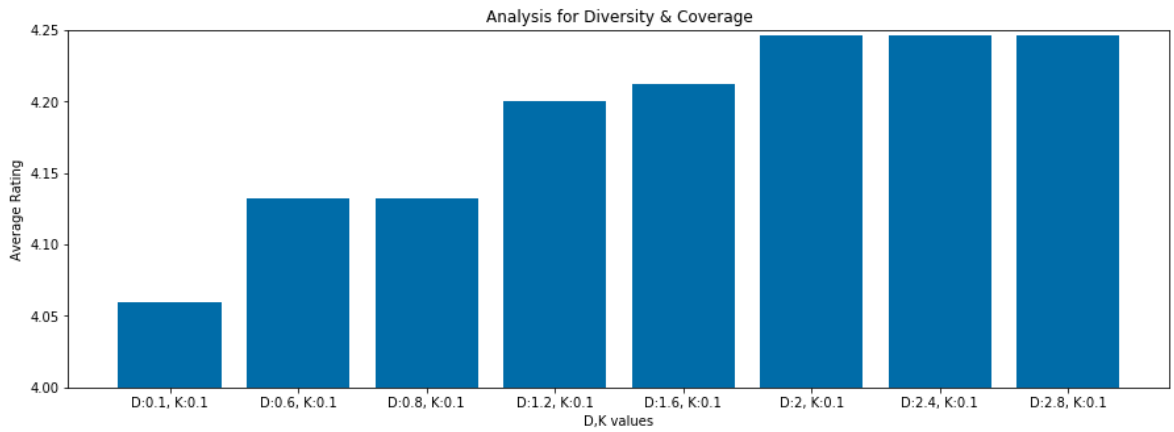
	D_K	D	K	Average_Rating
0	D:0.1, K:0.03	0.1	0.03	4.060
1	D:0.6, K:0.03	0.6	0.03	4.132
2	D:0.8, K:0.03	0.8	0.03	4.132
3	D:1.2, K:0.03	1.2	0.03	4.200
4	D:1.6, K:0.03	1.6	0.03	4.212
5	D:2, K:0.03	2.0	0.03	4.246
6	D:2.4, K:0.03	2.4	0.03	4.246
7	D:2.8, K:0.03	2.8	0.03	4.246



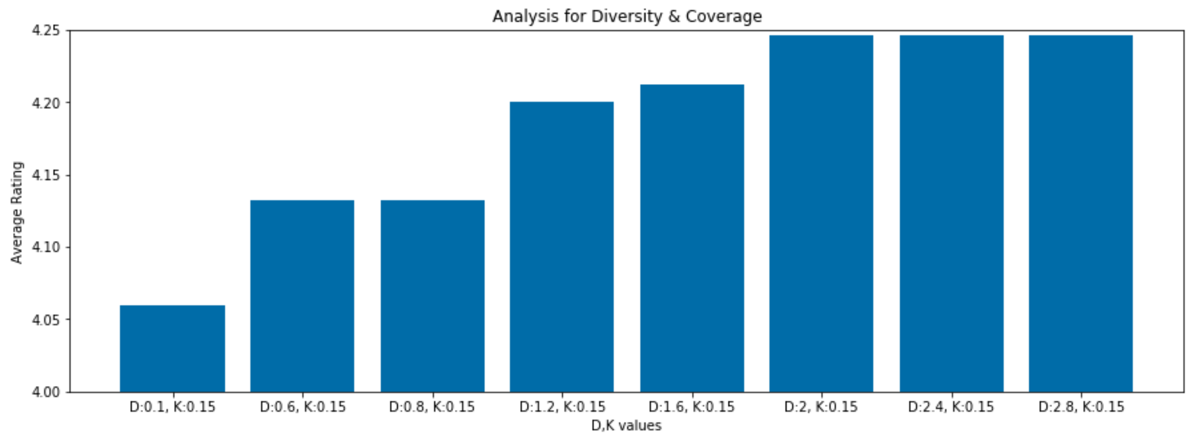
	D_K	D	K	Average_Rating
0	D:0.1, K:0.05	0.1	0.05	4.060
1	D:0.6, K:0.05	0.6	0.05	4.132
2	D:0.8, K:0.05	0.8	0.05	4.132
3	D:1.2, K:0.05	1.2	0.05	4.200
4	D:1.6, K:0.05	1.6	0.05	4.212
5	D:2, K:0.05	2.0	0.05	4.246
6	D:2.4, K:0.05	2.4	0.05	4.246
7	D:2.8, K:0.05	2.8	0.05	4.246



	D_K	D	K	Average_Rating
0	D:0.1, K:0.07	0.1	0.07	4.060
1	D:0.6, K:0.07	0.6	0.07	4.132
2	D:0.8, K:0.07	0.8	0.07	4.132
3	D:1.2, K:0.07	1.2	0.07	4.200
4	D:1.6, K:0.07	1.6	0.07	4.212
5	D:2, K:0.07	2.0	0.07	4.246
6	D:2.4, K:0.07	2.4	0.07	4.246
7	D:2.8, K:0.07	2.8	0.07	4.246

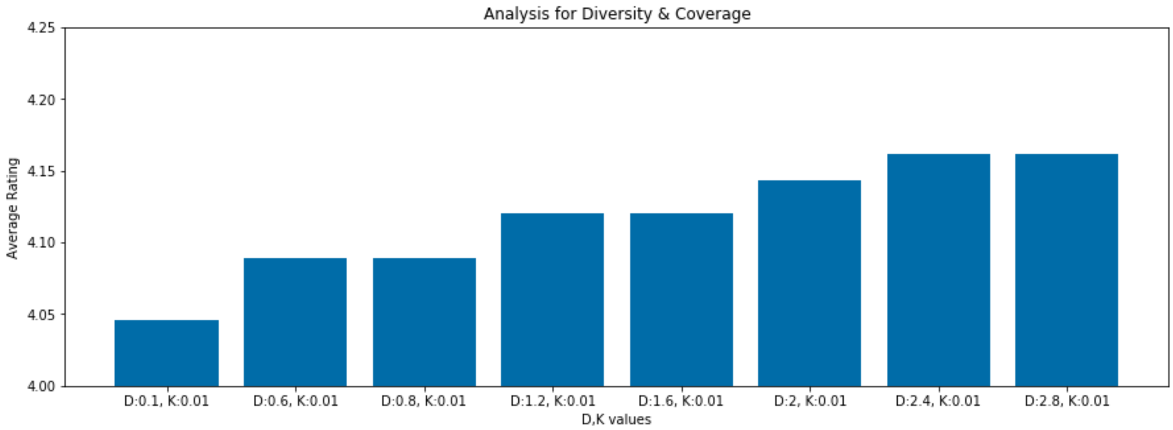


	D_K	D	K	Average_Rating
0	D:0.1, K:0.1	0.1	0.1	4.060
1	D:0.6, K:0.1	0.6	0.1	4.132
2	D:0.8, K:0.1	0.8	0.1	4.132
3	D:1.2, K:0.1	1.2	0.1	4.200
4	D:1.6, K:0.1	1.6	0.1	4.212
5	D:2, K:0.1	2.0	0.1	4.246
6	D:2.4, K:0.1	2.4	0.1	4.246
7	D:2.8, K:0.1	2.8	0.1	4.246

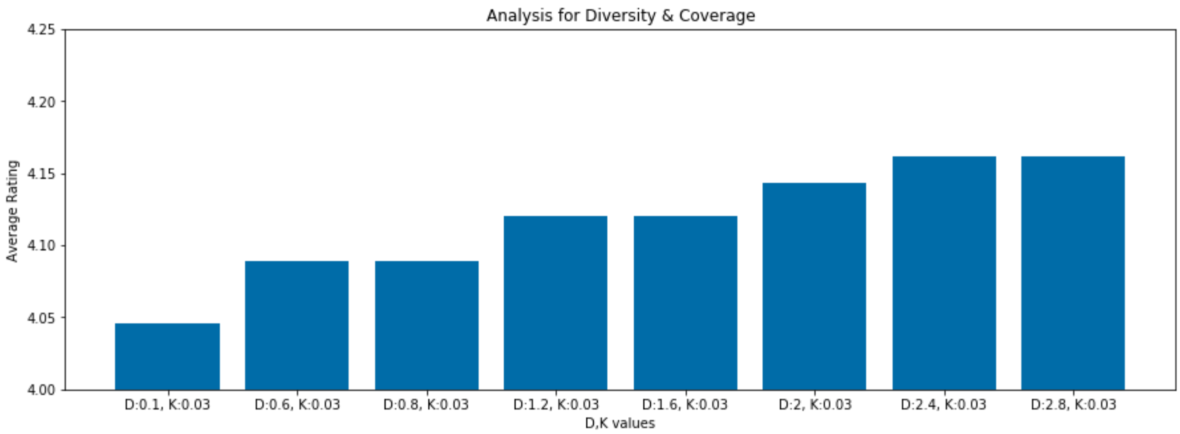


	D_K	D	K	Average_Rating
0	D:0.1, K:0.15	0.1	0.15	4.060
1	D:0.6, K:0.15	0.6	0.15	4.132
2	D:0.8, K:0.15	0.8	0.15	4.132
3	D:1.2, K:0.15	1.2	0.15	4.200
4	D:1.6, K:0.15	1.6	0.15	4.212
5	D:2, K:0.15	2.0	0.15	4.246
6	D:2.4, K:0.15	2.4	0.15	4.246
7	D:2.8, K:0.15	2.8	0.15	4.246

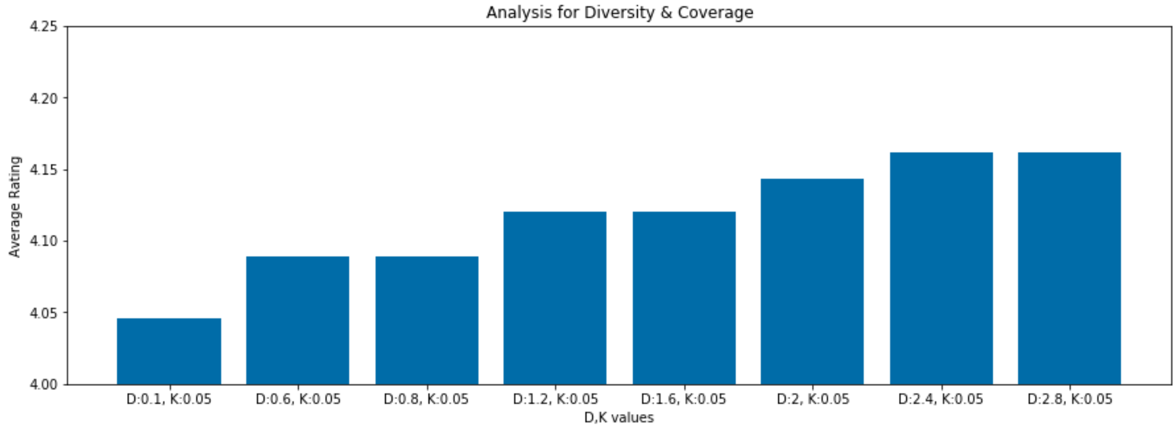
L=10, different D,K values , continuous values, default cvxpy solver



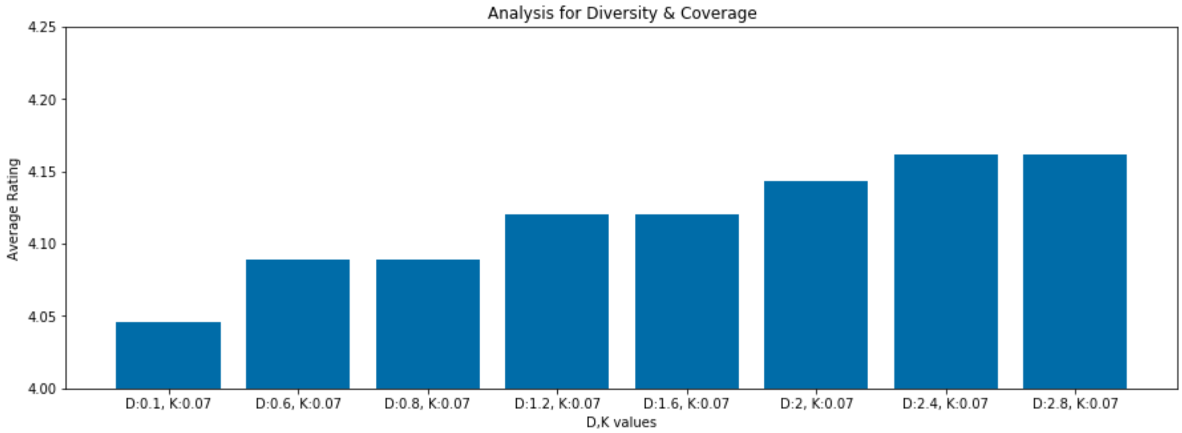
	D_K	D	K	Average_Rating
0	D:0.1, K:0.01	0.1	0.01	4.046
1	D:0.6, K:0.01	0.6	0.01	4.089
2	D:0.8, K:0.01	0.8	0.01	4.089
3	D:1.2, K:0.01	1.2	0.01	4.120
4	D:1.6, K:0.01	1.6	0.01	4.120
5	D:2, K:0.01	2.0	0.01	4.143
6	D:2.4, K:0.01	2.4	0.01	4.162
7	D:2.8, K:0.01	2.8	0.01	4.162



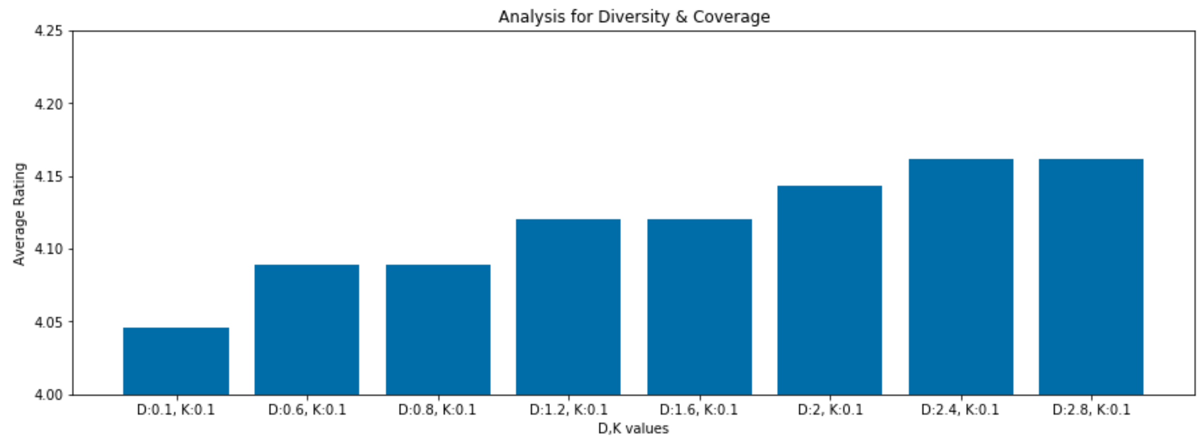
	D_K	D	K	Average_Rating
0	D:0.1, K:0.03	0.1	0.03	4.046
1	D:0.6, K:0.03	0.6	0.03	4.089
2	D:0.8, K:0.03	0.8	0.03	4.089
3	D:1.2, K:0.03	1.2	0.03	4.120
4	D:1.6, K:0.03	1.6	0.03	4.120
5	D:2, K:0.03	2.0	0.03	4.143
6	D:2.4, K:0.03	2.4	0.03	4.162
7	D:2.8, K:0.03	2.8	0.03	4.162



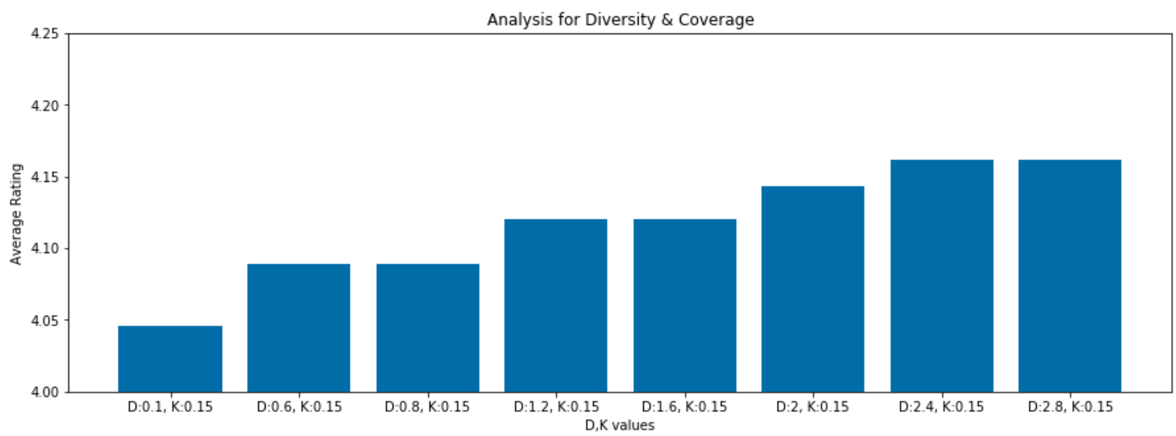
	D_K	D	K	Average_Rating
0	D:0.1, K:0.05	0.1	0.05	4.046
1	D:0.6, K:0.05	0.6	0.05	4.089
2	D:0.8, K:0.05	0.8	0.05	4.089
3	D:1.2, K:0.05	1.2	0.05	4.120
4	D:1.6, K:0.05	1.6	0.05	4.120
5	D:2, K:0.05	2.0	0.05	4.143
6	D:2.4, K:0.05	2.4	0.05	4.162
7	D:2.8, K:0.05	2.8	0.05	4.162



	D_K	D	K	Average_Rating
0	D:0.1, K:0.07	0.1	0.07	4.046
1	D:0.6, K:0.07	0.6	0.07	4.089
2	D:0.8, K:0.07	0.8	0.07	4.089
3	D:1.2, K:0.07	1.2	0.07	4.120
4	D:1.6, K:0.07	1.6	0.07	4.120
5	D:2, K:0.07	2.0	0.07	4.143
6	D:2.4, K:0.07	2.4	0.07	4.162
7	D:2.8, K:0.07	2.8	0.07	4.162

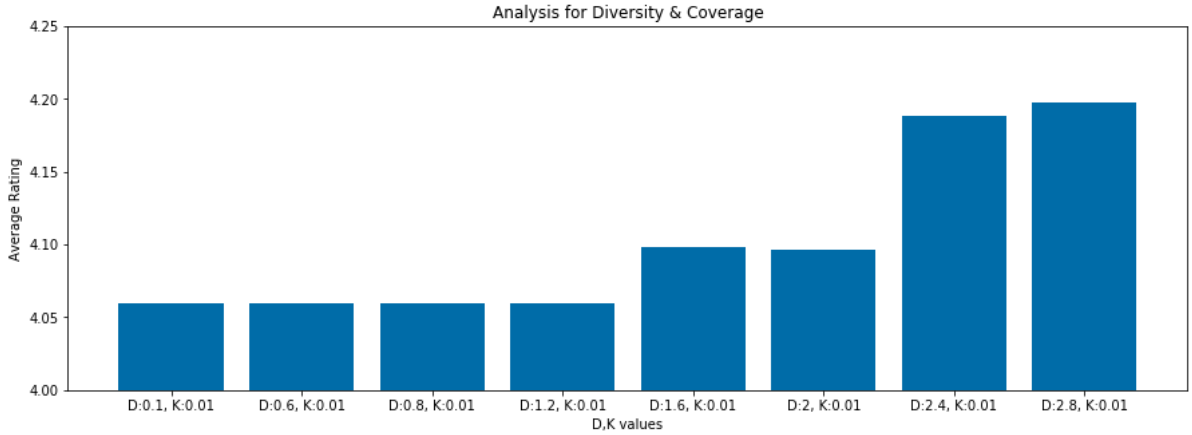


	D_K	D	K	Average_Rating
0	D:0.1, K:0.1	0.1	0.1	4.046
1	D:0.6, K:0.1	0.6	0.1	4.089
2	D:0.8, K:0.1	0.8	0.1	4.089
3	D:1.2, K:0.1	1.2	0.1	4.120
4	D:1.6, K:0.1	1.6	0.1	4.120
5	D:2, K:0.1	2.0	0.1	4.143
6	D:2.4, K:0.1	2.4	0.1	4.162
7	D:2.8, K:0.1	2.8	0.1	4.162

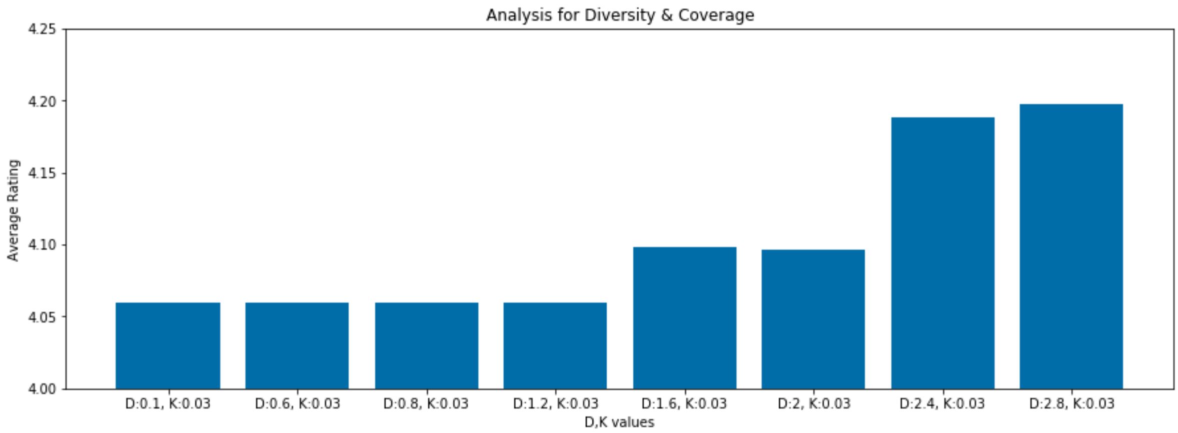


	D_K	D	K	Average_Rating
0	D:0.1, K:0.15	0.1	0.15	4.046
1	D:0.6, K:0.15	0.6	0.15	4.089
2	D:0.8, K:0.15	0.8	0.15	4.089
3	D:1.2, K:0.15	1.2	0.15	4.120
4	D:1.6, K:0.15	1.6	0.15	4.120
5	D:2, K:0.15	2.0	0.15	4.143
6	D:2.4, K:0.15	2.4	0.15	4.162
7	D:2.8, K:0.15	2.8	0.15	4.162

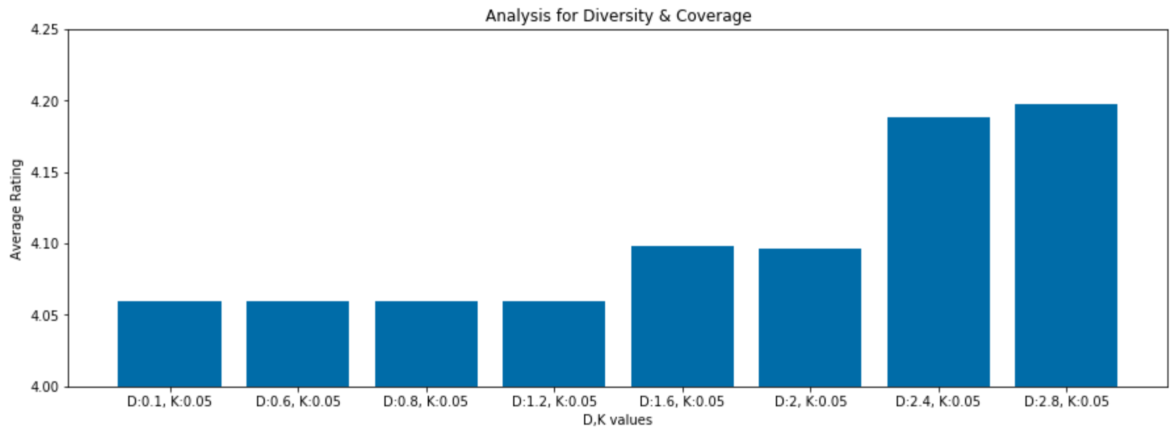
L=5, different D,K values , discrete values, Gurobi solver



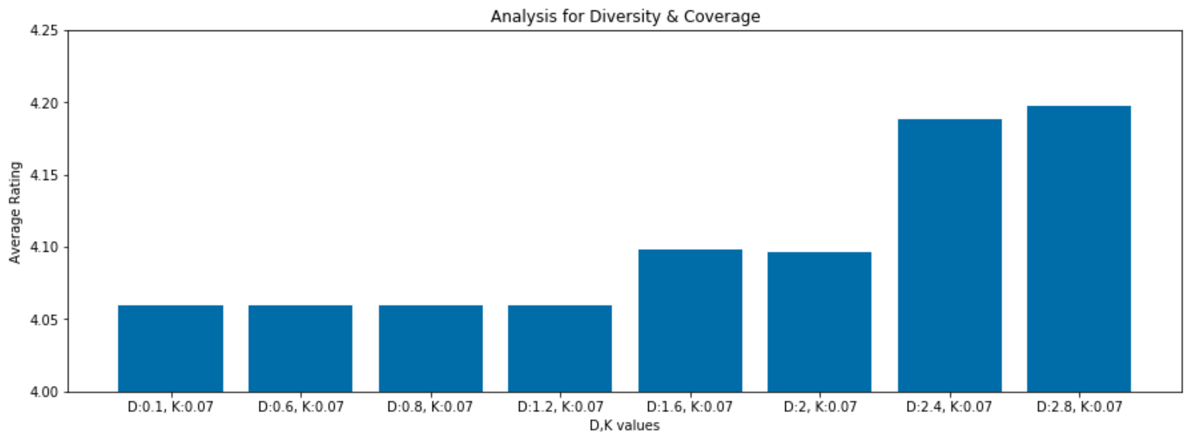
	D_K	D	K	Average_Rating
0	D:0.1, K:0.01	0.1	0.01	4.060
1	D:0.6, K:0.01	0.6	0.01	4.060
2	D:0.8, K:0.01	0.8	0.01	4.060
3	D:1.2, K:0.01	1.2	0.01	4.060
4	D:1.6, K:0.01	1.6	0.01	4.098
5	D:2, K:0.01	2.0	0.01	4.096
6	D:2.4, K:0.01	2.4	0.01	4.188
7	D:2.8, K:0.01	2.8	0.01	4.198



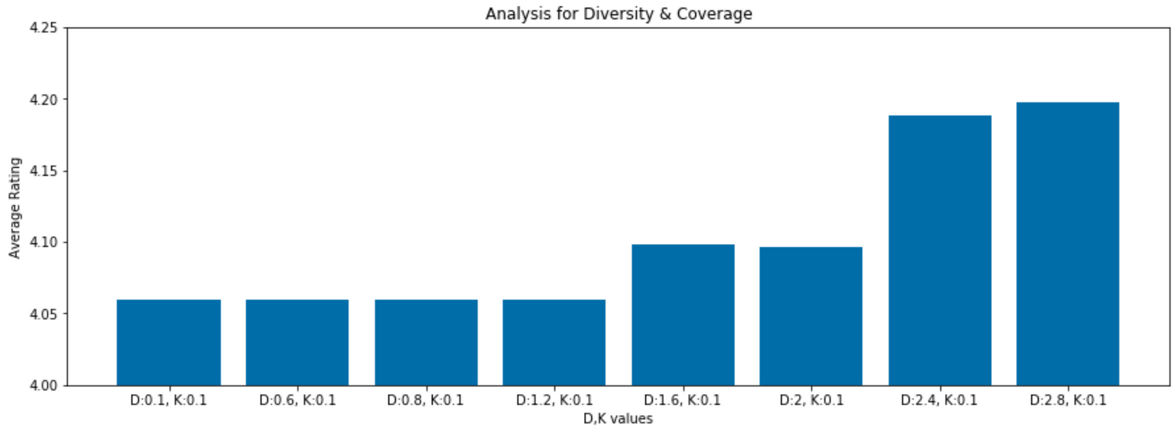
	D_K	D	K	Average_Rating
0	D:0.1, K:0.03	0.1	0.03	4.060
1	D:0.6, K:0.03	0.6	0.03	4.060
2	D:0.8, K:0.03	0.8	0.03	4.060
3	D:1.2, K:0.03	1.2	0.03	4.060
4	D:1.6, K:0.03	1.6	0.03	4.098
5	D:2, K:0.03	2.0	0.03	4.096
6	D:2.4, K:0.03	2.4	0.03	4.188
7	D:2.8, K:0.03	2.8	0.03	4.198



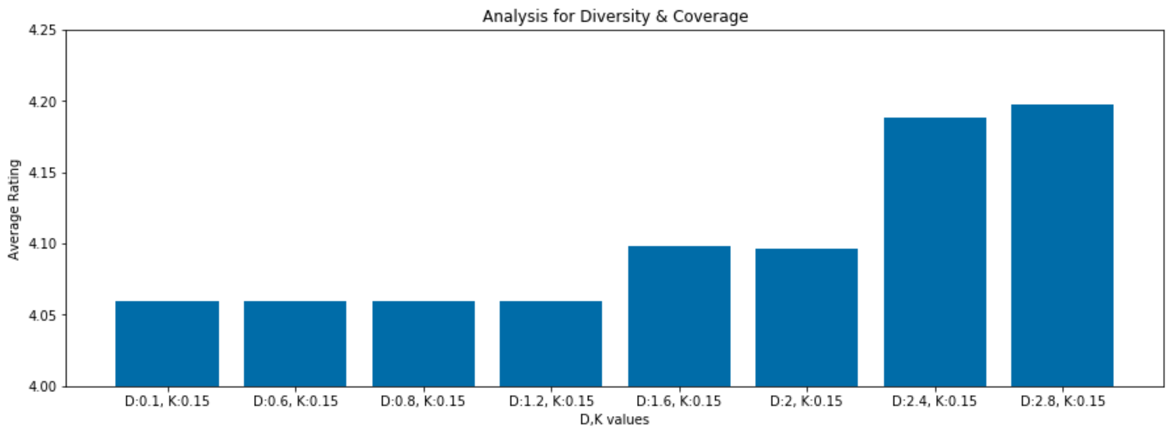
	D_K	D	K	Average_Rating
0	D:0.1, K:0.05	0.1	0.05	4.060
1	D:0.6, K:0.05	0.6	0.05	4.060
2	D:0.8, K:0.05	0.8	0.05	4.060
3	D:1.2, K:0.05	1.2	0.05	4.060
4	D:1.6, K:0.05	1.6	0.05	4.098
5	D:2, K:0.05	2.0	0.05	4.096
6	D:2.4, K:0.05	2.4	0.05	4.188
7	D:2.8, K:0.05	2.8	0.05	4.198



	D_K	D	K	Average_Rating
0	D:0.1, K:0.07	0.1	0.07	4.060
1	D:0.6, K:0.07	0.6	0.07	4.060
2	D:0.8, K:0.07	0.8	0.07	4.060
3	D:1.2, K:0.07	1.2	0.07	4.060
4	D:1.6, K:0.07	1.6	0.07	4.098
5	D:2, K:0.07	2.0	0.07	4.096
6	D:2.4, K:0.07	2.4	0.07	4.188
7	D:2.8, K:0.07	2.8	0.07	4.198

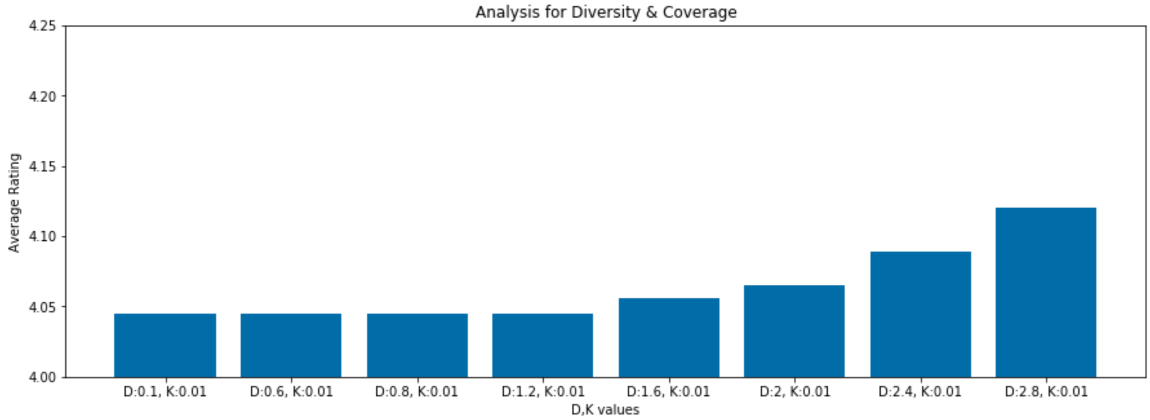


	D_K	D	K	Average_Rating
0	D:0.1, K:0.1	0.1	0.1	4.060
1	D:0.6, K:0.1	0.6	0.1	4.060
2	D:0.8, K:0.1	0.8	0.1	4.060
3	D:1.2, K:0.1	1.2	0.1	4.060
4	D:1.6, K:0.1	1.6	0.1	4.098
5	D:2, K:0.1	2.0	0.1	4.096
6	D:2.4, K:0.1	2.4	0.1	4.188
7	D:2.8, K:0.1	2.8	0.1	4.198

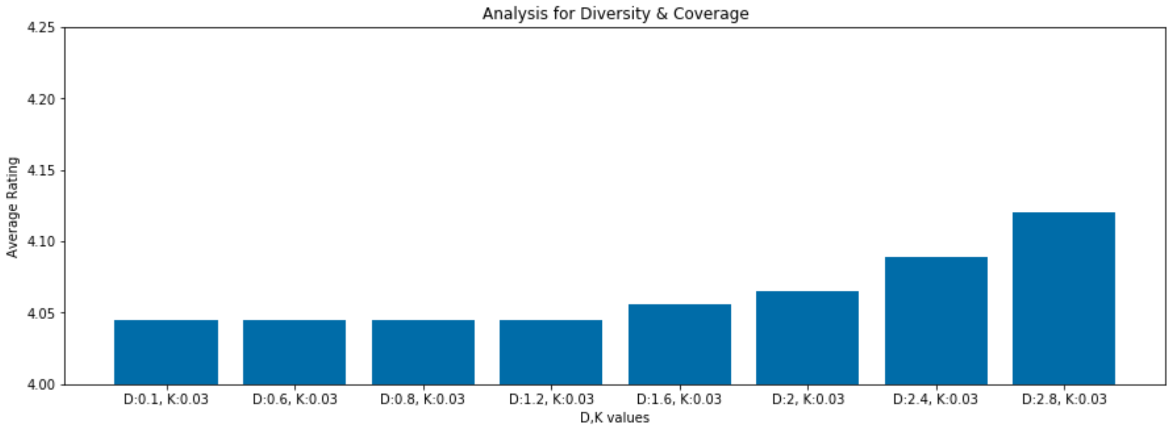


	D_K	D	K	Average_Rating
0	D:0.1, K:0.15	0.1	0.15	4.060
1	D:0.6, K:0.15	0.6	0.15	4.060
2	D:0.8, K:0.15	0.8	0.15	4.060
3	D:1.2, K:0.15	1.2	0.15	4.060
4	D:1.6, K:0.15	1.6	0.15	4.098
5	D:2, K:0.15	2.0	0.15	4.096
6	D:2.4, K:0.15	2.4	0.15	4.188
7	D:2.8, K:0.15	2.8	0.15	4.198

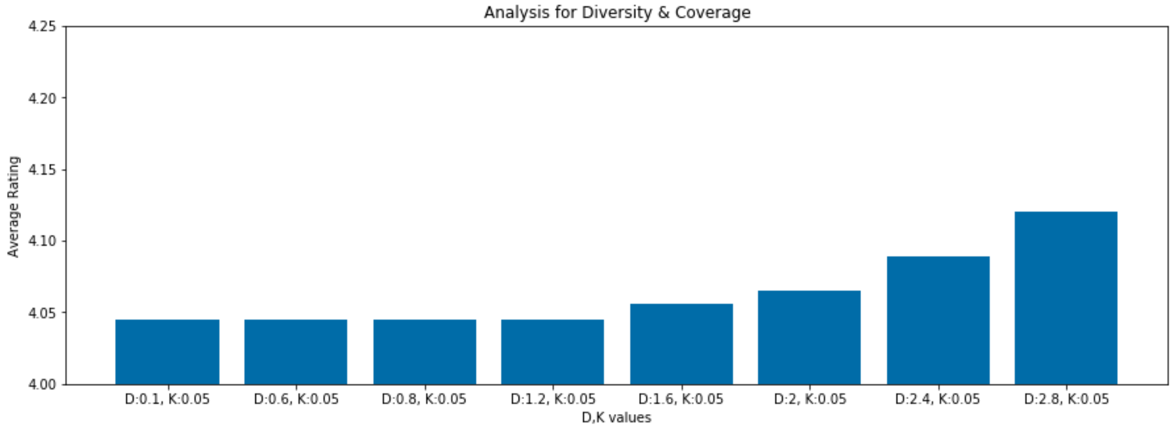
L=10, different D,K values , discrete values, Gurobi solver



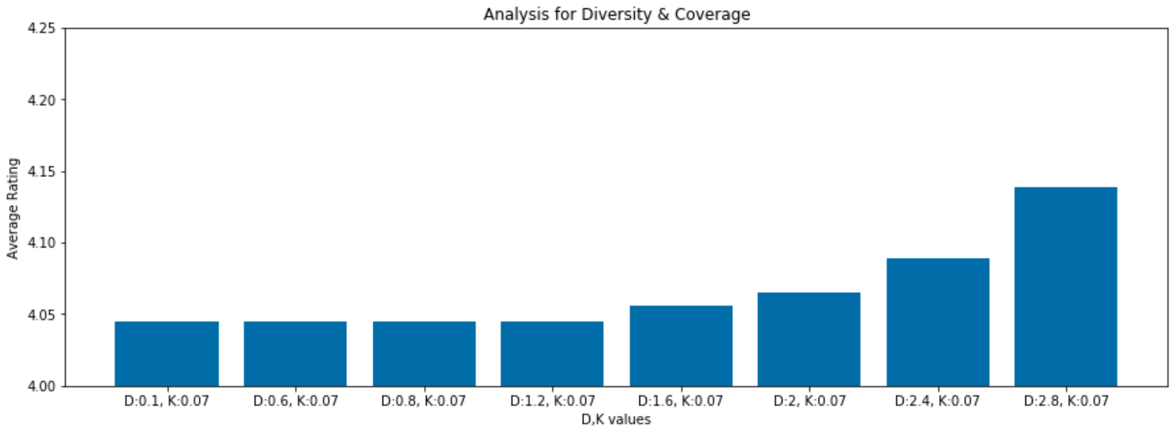
	D_K	D	K	Average_Rating
0	D:0.1, K:0.01	0.1	0.01	4.045
1	D:0.6, K:0.01	0.6	0.01	4.045
2	D:0.8, K:0.01	0.8	0.01	4.045
3	D:1.2, K:0.01	1.2	0.01	4.045
4	D:1.6, K:0.01	1.6	0.01	4.056
5	D:2, K:0.01	2.0	0.01	4.065
6	D:2.4, K:0.01	2.4	0.01	4.089
7	D:2.8, K:0.01	2.8	0.01	4.120



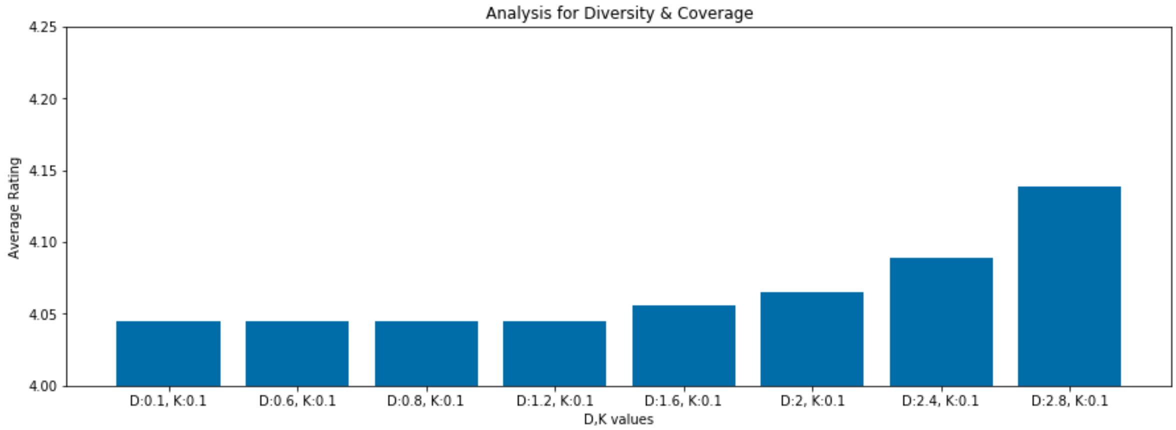
	D_K	D	K	Average_Rating
0	D:0.1, K:0.03	0.1	0.03	4.045
1	D:0.6, K:0.03	0.6	0.03	4.045
2	D:0.8, K:0.03	0.8	0.03	4.045
3	D:1.2, K:0.03	1.2	0.03	4.045
4	D:1.6, K:0.03	1.6	0.03	4.056
5	D:2, K:0.03	2.0	0.03	4.065
6	D:2.4, K:0.03	2.4	0.03	4.089
7	D:2.8, K:0.03	2.8	0.03	4.120



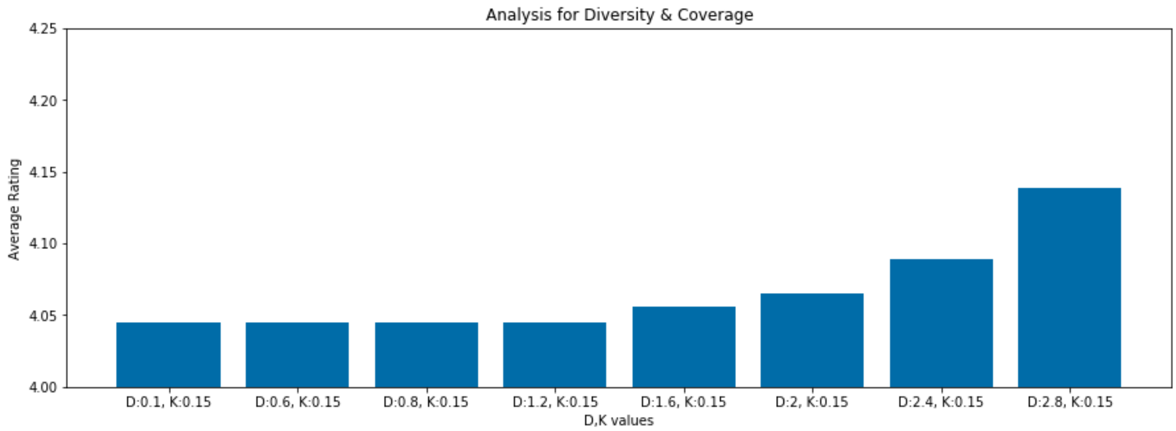
	D_K	D	K	Average_Rating
0	D:0.1, K:0.05	0.1	0.05	4.045
1	D:0.6, K:0.05	0.6	0.05	4.045
2	D:0.8, K:0.05	0.8	0.05	4.045
3	D:1.2, K:0.05	1.2	0.05	4.045
4	D:1.6, K:0.05	1.6	0.05	4.056
5	D:2, K:0.05	2.0	0.05	4.065
6	D:2.4, K:0.05	2.4	0.05	4.089
7	D:2.8, K:0.05	2.8	0.05	4.120



	D_K	D	K	Average_Rating
0	D:0.1, K:0.07	0.1	0.07	4.045
1	D:0.6, K:0.07	0.6	0.07	4.045
2	D:0.8, K:0.07	0.8	0.07	4.045
3	D:1.2, K:0.07	1.2	0.07	4.045
4	D:1.6, K:0.07	1.6	0.07	4.056
5	D:2, K:0.07	2.0	0.07	4.065
6	D:2.4, K:0.07	2.4	0.07	4.089
7	D:2.8, K:0.07	2.8	0.07	4.139



	D_K	D	K	Average_Rating
0	D:0.1, K:0.1	0.1	0.1	4.045
1	D:0.6, K:0.1	0.6	0.1	4.045
2	D:0.8, K:0.1	0.8	0.1	4.045
3	D:1.2, K:0.1	1.2	0.1	4.045
4	D:1.6, K:0.1	1.6	0.1	4.056
5	D:2, K:0.1	2.0	0.1	4.065
6	D:2.4, K:0.1	2.4	0.1	4.089
7	D:2.8, K:0.1	2.8	0.1	4.139



	D_K	D	K	Average_Rating
0	D:0.1, K:0.15	0.1	0.15	4.045
1	D:0.6, K:0.15	0.6	0.15	4.045
2	D:0.8, K:0.15	0.8	0.15	4.045
3	D:1.2, K:0.15	1.2	0.15	4.045
4	D:1.6, K:0.15	1.6	0.15	4.056
5	D:2, K:0.15	2.0	0.15	4.065
6	D:2.4, K:0.15	2.4	0.15	4.089
7	D:2.8, K:0.15	2.8	0.15	4.139

From all the above plots, we observe that Average Rating is increasing as the values of D increase. This is normal, given the Diversity constraint we have introduced, the increase of D, decreases recommendations diversity in our system, resulting in higher Average Ratings.

In general we experimented with numerous python libraries(cvxpy, qcp, qcqp, dccp, cplex, gurobi) trying to apply the quadratic constraint of diversity.

For the continuous values problem we used the default solver of cvxpy which, for the discrete values case we utilized the Gurobi solver in cvxpy, since it was both faster and produced solid results. Comparing the two problems in terms of average Rating, both had equivalent results. Moving forward though, the continuous variable problem has equivalent precision and produces results in a notably smaller time than the discrete.

USE CASE: Increasing categories count

We need to test the aforementioned process for more categories (range 1-50), in order to apply the diversity constraint to categories and not to specific movies as above.

UserID: 6 baseline

Baseline top 10 list

	movieId	title	Category	Rating
210	1721	Titanic (1997)	6	4.33
129	1035	Sound of Music, The (1965)	31	4.26
116	786	Eraser (1996)	17	4.25
110	733	Rock, The (1996)	22	4.20
94	586	Home Alone (1990)	33	4.19
137	1101	Top Gun (1986)	5	4.16
197	1580	Men in Black (a.k.a. MIB) (1997)	43	4.16
314	4022	Cast Away (2000)	33	4.14
106	648	Mission: Impossible (1996)	19	4.12
17	48	Pocahontas (1995)	27	4.08

UserID:6

optimization Kc:0.01, D=0.01, L=10

	movieId	title	Category	Rating
2	1721	Titanic (1997)	6	4.33
0	1035	Sound of Music, The (1965)	31	4.26
1	1101	Top Gun (1986)	5	4.16
3	4022	Cast Away (2000)	33	4.14
5	4995	Beautiful Mind, A (2001)	16	4.02
4	4896	Harry Potter and the Sorcerer's Stone (a.k.a. ...	11	3.97
7	6942	Love Actually (2003)	34	3.93
6	6377	Finding Nemo (2003)	15	3.92
9	76093	How to Train Your Dragon (2010)	45	3.91
8	59315	Iron Man (2008)	15	3.81

UserID:1 dissimilar

optimization Kc:0.01, D=0.01, L=10

	movieId	title	Category	Rating
9	76093	How to Train Your Dragon (2010)	45	4.87
5	4995	Beautiful Mind, A (2001)	16	4.82
6	5989	Catch Me If You Can (2002)	18	4.81
8	6942	Love Actually (2003)	34	4.77
4	4896	Harry Potter and the Sorcerer's Stone (a.k.a. ...	11	4.75
7	6377	Finding Nemo (2003)	15	4.72
3	4022	Cast Away (2000)	33	4.67
0	1035	Sound of Music, The (1965)	31	4.52
1	1101	Top Gun (1986)	5	4.48
2	1721	Titanic (1997)	6	4.35

As it can be observed, 9/10 categories (**5,6,11,15,16,31,33,34,45** **not 18**)are similar in both recommendation lists for two dissimilar users.

UserID:8 similar

optimization Kc:0.01, D=0.01, L=10

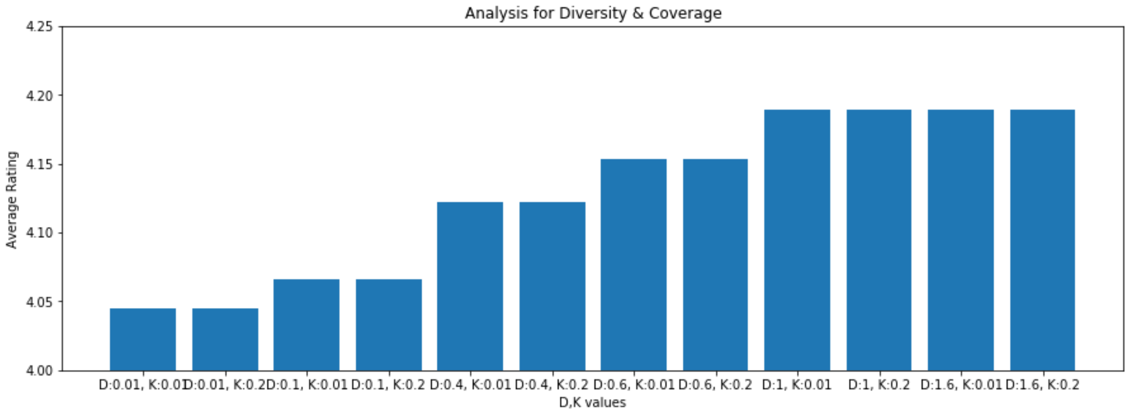
	movieId	title	Category	Rating
9	116797	The Imitation Game (2014)	3	4.47
8	99114	Django Unchained (2012)	18	4.27
0	858	Godfather, The (1972)	4	4.21
4	6016	City of God (Cidade de Deus) (2002)	33	4.21
7	79132	Inception (2010)	45	4.21
2	2959	Fight Club (1999)	30	4.20
6	74458	Shutter Island (2010)	28	4.20
1	2858	American Beauty (1999)	24	4.19
3	4973	Amelie (Fabuleux destin d'Amélie Poulain, Le) ...	15	4.18
5	58559	Dark Knight, The (2008)	43	4.17

One the other hand, 7/10 categories of recommendation lists for two approximate similar users are different.

different categories 3,4,18,24,28,30,43

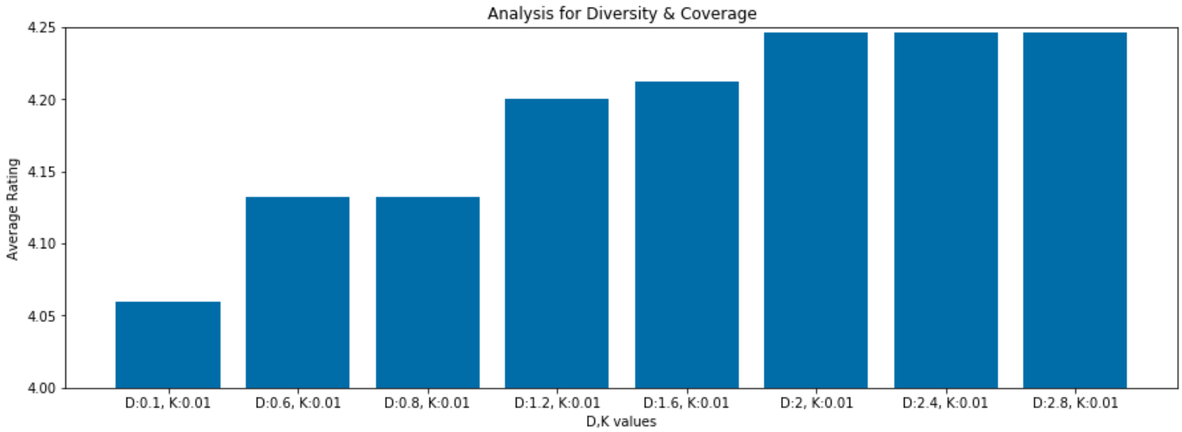
same 15,33,45

While the number of categories is higher than the previous result, there is a need to use lower values of D in order to assure the similarity of recommended lists for two dissimilar users.



	D_K	D	K	Average_Rating
0	D:0.01, K:0.01	0.01	0.01	4.045
1	D:0.01, K:0.2	0.01	0.20	4.045
2	D:0.1, K:0.01	0.10	0.01	4.066
3	D:0.1, K:0.2	0.10	0.20	4.066
4	D:0.4, K:0.01	0.40	0.01	4.122
5	D:0.4, K:0.2	0.40	0.20	4.122
6	D:0.6, K:0.01	0.60	0.01	4.153
7	D:0.6, K:0.2	0.60	0.20	4.153
8	D:1, K:0.01	1.00	0.01	4.189
9	D:1, K:0.2	1.00	0.20	4.189
10	D:1.6, K:0.01	1.60	0.01	4.189
11	D:1.6, K:0.2	1.60	0.20	4.189

Previous Results



	D_K	D	K	Average_Rating
0	D:0.1, K:0.01	0.1	0.01	4.060
1	D:0.6, K:0.01	0.6	0.01	4.132
2	D:0.8, K:0.01	0.8	0.01	4.132
3	D:1.2, K:0.01	1.2	0.01	4.200
4	D:1.6, K:0.01	1.6	0.01	4.212
5	D:2, K:0.01	2.0	0.01	4.246
6	D:2.4, K:0.01	2.4	0.01	4.246
7	D:2.8, K:0.01	2.8	0.01	4.246

To conclude, the diversity constraint seems to perform better for a use case in which we have a wide range of different categories. In both cases, the total average rating does not change for different values of K (Coverage Constraint). In the case of a wide range of categories, there is a need to re-evaluate the values of constant D, while we need lower D to assure the diversity of users. Also the upper bound of constant D was decreased to 1 as the average rating does not change for values over 1.

3.Problem B: Serendipity in Recommender systems

Moving on to problem b we examine serendipity, the third metric that is considered to have a significant impact on user quality of experience. Serendipity is defined as the accident of finding something good or useful while not specifically searching for it. In other words, serendipity is concerned with the novelty of recommendations and in how far recommendations may positively surprise users.

3.1 Data Sources Description

For this problem, we are using **serendipity-sac2018.zip** dataset from grouplens.org site. Specifically, we use the datasets **movies.csv** and **answers.csv**.

The **movies.csv** dataset contains the below attributes:

- *movieId* - movie id
- *title* - movie title
- *directedBy* - directors separated by commas
- *starring* - cast separated by commas
- *genres* - genres separated by commas

A preview of the dataset is depicted below

movieId	title	directedBy	starring	genres
1	Toy Story (1995)	John Lasseter	Tim Allen, Tom Hanks, Don Rickles, Jim Varney, John Ratzenberger, Wallace Shawn, Laurie Metc	Adventure,Animation,Children,Comedy,Fantasy
2	Jumanji (1995)	Joe Johnston	Jonathan Hyde, Bradley Pierce, Robin Williams, Kirsten Dunst	Adventure,Children,Fantasy
3	Grumpier Old Men (1995)	Howard Deutch	Jack Lemmon, Walter Matthau, Ann-Margret , Sophia Loren	Comedy,Romance
4	Waiting to Exhale (1995)	Forest Whitaker	Angela Bassett, Loretta Devine, Whitney Houston, Lela Rochon	Comedy,Drama,Romance
5	Father of the Bride Part II (1995)	Charles Shyer	Steve Martin, Martin Short, Diane Keaton, Kimberly Williams, George Newbern, Kieran Culkin	Comedy
6	Heat (1995)	Michael Mann	Robert De Niro, Al Pacino, Val Kilmer, Jon Voight, Tom Sizemore, Ashley Judd, Diane Venora, Nat	Action,Crime,Thriller
7	Sabrina (1995)	Sydney Pollack	Harrison Ford, Greg Kinnear, Nancy Marchand, Julia Ormond	Comedy,Romance
8	Tom and Huck (1995)	Peter Hewitt	Jonathan Taylor Thomas, Brad Renfro, Eric Schweig, Charles Rocket, Amy Wright, Michael McSh	Adventure,Children
9	Sudden Death (1995)	Peter Hyams	Raymond J. Barry, Powers Boothe, Jean-Claude Van Damme, Whittni Wright	Action
10	GoldenEye (1995)	Martin Campbell	Pierce Brosnan, Sean Bean, Famke Janssen, Izabella Scorupco, Joe Don Baker, Judi Dench, Robbi	Action,Adventure,Thriller
11	American President, The (1995)	Rob Reiner	Michael Douglas, Michael J. Fox, Martin Sheen, Annette Bening, Richard Dreyfuss	Comedy,Drama,Romance
12	Dracula: Dead and Loving It (1995)	Mel Brooks	Peter MacNicol, Leslie Nielsen, Steven Weber, Amy Yasbeck	Comedy,Horror
13	Balto (1995)	Simon Wells	Kevin Bacon, Jim Cummings, Bob Hoskins, Bridget Fonda	Adventure,Animation,Children
14	Nixon (1995)	Oliver Stone	Anthony Hopkins, Joan Allen, Powers Boothe, Ed Harris, Bob Hoskins, E.G. Marshall, David Paym	Drama
15	Cutthroat Island (1995)	Renny Harlin	Maury Chaykin, Frank Langella, Matthew Modine, Geena Davis	Action,Adventure,Romance
16	Casino (1995)	Martin Scorsese	Robert De Niro, Joe Pesci, James Woods, Sharon Stone	Crime,Drama
17	Sense and Sensibility (1995)	Ang Lee	Hugh Grant, Alan Rickman, Emma Thompson, Kate Winslet	Drama,Romance
18	Four Rooms (1995)	Allison Anders, Alexandre Rc	Sammi Davis, Amanda De Cadenet, Valeria Golino, Madonna, Ione Skye, Lili Taylor, Alicia Witt, J	Comedy

The **answers.csv** dataset contains the below attributes:

- *userId* - user id
- *movieId* - movie id
- *timestamp* - timestamp, which indicates when the user gave the rating.
- *s1* -The first time I heard of this movie was when MovieLens suggested it to me.
- *s2* - MovieLens influenced my decision to watch this movie.
- *s3* - I expected to enjoy this movie before watching it for the first time.

- *s4* - This is the type of movie I would not normally discover on my own; I need a recommender system like MovieLens to find movies like this one.
- *s5* - This movie is different (e.g., in style, genre, topic) from the movies I usually watch.
- *s6* - I was (or, would have been) surprised that MovieLens picked this movie to recommend to me.
- *s7* - I am glad I watched this movie.
- *s8* - Watching this movie broadened my preferences. Now I am interested in a wider selection of movies.

As we can see, the above formulation of the questions declares the surprise of the users. The higher the rating in the scale is given by the user, the bigger the surprise for him.

As mentioned above the dataset contains *481 users and 1,678 movies* and a preview of the dataset is posted below.

userId	movieId	timestamp	s1	s2	s3	s4	s5	s6	s7	s8
205229	108979	14861278:	1	1	3	4	2	2	5	5
205229	6947	14861212:	1	1	3	4	4	2	5	4
205229	117444	14861278:	1	4	4	2	2	2	4	2
205229	150548	14861278:	2	2	4	2	4	2	4	1
205229	136542	14861280:	1	1	5	1	1	2	5	2
117112	77455	14919106(	1	2	2	2	4	4	4	4
144726	1303	14891186!	1	1	NA	4	3	2	5	3
144726	103306	14856401	NA	NA	3	2	1	2	4	2
144726	2060	14881682:	1	1	4	2	1	1	5	2
144726	135534	14864381(	2	1	5	1	1	1	5	2
144726	128542	14864468(	1	1	4	3	1	2	5	2
200400	26939	14906798!	1	2	4	2	2	2	4	1
200400	40491	14889589!	1	2	4	2	2	2	4	1
125112	104337	14903695:	NA	NA	NA	NA	NA	NA	NA	NA
125112	162082	14871643!	2	NA	3	5	5	NA	4	4
125112	96966	14903698(	NA	NA	NA	NA	NA	NA	NA	NA
125112	165551	14903689:	1	3	4	3	3	2	5	4

Furthermore, for the popularity metric, we are using the **rating.csv** dataset from **ml-latest-small.zip** which is extracted from MovieLens datasets as described on Problem A.

3.2 Methodology

To begin with, the packages/libraries we used to implement the recommendation in serendipity were:

- Pandas
- Numpy

- Cvxpy
- matplotlib

In order to compute the serendipity index, we first need to create a training dataset from the **movies.csv** and **answers.csv** data, which we train a linear model.

For each movie *i* viewed by the user u in the answers.csv dataset, we calculated the

1. Similarity sij between this movie i and other movies j that user u had rated
2. Popularity metric of the movie
3. Index Serendipity of the movie

Similarity metric

To calculate the similarity of movie *i* for user *u* in regard to the movies the user had previously seen we joined the datasets Answers and Movies and we implemented Jaccard similarity with the features from the movies.csv. More specifically the features 'genres', 'starring' and 'directed by'. Jaccard similarity was used to measure the similarity between two sets of elements. The Jaccard similarity between two sets was computed as:

$$J(X,Y) = |X \cap Y| \; / \; |X \cup Y|$$

An important attribute that we took under consideration in the calculation of the similarity was the "timestamp". We applied the aforementioned calculation of Jaccard similarity to a movie *i* in regard to movies that the user had seen in the past in relevance of the specific movie *i* .

The result of the similarity per user was a NxN item matrix, where N is the number of movies that the corresponding user had seen. For example the user 111751 had watched the movies 2, 55442, 135569, 166461 and 166528.

	2	55442	135569	166461	166528
2	NaN	NaN	NaN	NaN	0.212698
55442	0.040650	NaN	0.042735	NaN	0.047619
135569	0.164706	NaN	NaN	NaN	0.297619
166461	0.311111	0.096206	0.121693	NaN	0.155844
166528	NaN	NaN	NaN	NaN	NaN

Finally, we set the similarity of movie *i* as the average of the similarities in respect to the older movies the user had seen.

userId	movieId	similarity
111751	2	0.212698
111751	55442	0.043668
111751	135569	0.231162
111751	166461	0.171214
111751	166528	0.000000

Given a movie *i* 55442, the similarity of the movie for the specific user is the average of the similarities from the movies 2, 135569 and 166525 which the user has watched before 55442.

All those calculations were implemented in the **'similarityCalculation'** function which was developed in the corresponding python part.

Popularity metric

The next attribute, we computed for our training dataset, was the movie popularity. For the movie popularity, we utilized the "ratings.csv" so as to implement the calculation.

$$p_i = \frac{number\ of\ users\ that\ have\ rated\ the\ itemi\ i}{number\ of\ users\ that\ have\ rated\ movies}$$

In addition, we applied the $-logp_i$ to take the final popularity metric. A preview of the popularity per movie is shown below.

movieId	popularity
1	1.042821
2	1.712979
3	2.462215
4	4.467549
5	2.521639
...	...
193581	6.413459
193583	6.413459
193585	6.413459
193587	6.413459
193609	6.413459

All those calculations were implemented in the '**calculate_popularity**' function which was developed in the corresponding python part.

Serendipity Index

The third attribute of our training dataset is Serendipity Index Si of a movie *i*. We computed the serendipity by taking the average of the answers from the users to questions s1, s2, s3, s4, s5, s6, s7 and s8. In addition, the null or na values in S1, S2,..., S8 attributes were replaced with the average of each question. The answers of these questions are on the "answers.csv" dataset. It is noteworthy that the NA values are replaced with the average value per question.

Below is a preview of the serendipity metric per movie.

movieId	serendipity
2	2.250
7	3.000
10	2.000
11	2.375
12	2.375
...	...
170705	2.500
170725	2.750
170743	3.125
170839	2.625
171129	3.125

All those calculations were implemented in the '**calculate_serendipity**' function which was developed in the corresponding python part.

Linear Regression

Finally, having computed the similarity, popularity and serendipity our training dataset can be seen at the following table:

movieId	userId	serendipity	popularity	randNumCol	similarity
123534	200683	2.3750	5.128896	9	0.321958
3114	126536	2.5000	1.838748	7	0.088687
45928	148258	2.6250	4.804021	4	0.087719
77201	149789	2.5625	6.413459	3	0.000000
26170	114454	3.8750	5.128896	9	0.096667
...	...	...	...	...	...
55063	103561	1.8750	5.128896	2	0.105263
25771	148065	3.0000	5.027165	4	0.061568
2360	113591	2.6875	3.928552	8	0.144360
167746	144824	2.5000	4.467549	5	0.111979
116855	113031	2.3750	5.128896	1	0.118860

2149 rows × 4 columns

The training dataset was used to train our linear model with two attributes, the **similarity** to past watched movies, and the movie **popularity**.

The model we expected to use should have the following form:

$$Y = ax1 + bx2 + c$$

where:

x1: similarity metric

x2: popularity metric

a, b coefficients and c the intercept.

In fact, the X1 X2 are two vectors and their size is (2149,1) each. Thus, the y should be a vector as well with the same size and predicts the serendipity.

Moreover, we train a linear model with three attributes, the **similarity** to past watched movies, the movie **popularity** and the predicted rating.

After having applied the model to the regressor and fitted the data, we got the following regression model:

$$Y = ax1 + bx2 + cx3 + d$$

The models and Akaike Information Criteria are shown below:

Model	Coefficients	AIC
Similarity, popularity	Y = -0.14885282 x1 +0.11686286 x2	3201.77064
Similarity, popularity, predicted rating	Y = -0.15618393 x1+0.13013616 x2 + 0.12617188 x3	3163.95283

Given the specific model we predicted the serendipity per user for movies that he had not watched. We created a similar dataset like the training one, so as to feed our model and predict the serendipity. The predicted dataset contains the movies that the user had not seen, with the necessary attributes the model had been trained, popularity and similarity.

The popularity metric per movie was integrated to the dataset. For similarity though, we made the assumption that similarity for a movie *i* , that the user had not seen, was the metric of the average similarity for the specific movie from all other users. All the other users had seen that movie though. Next, we computed the average of similarity scores per user and finally we calculated the mean of those averages.

Our dataset for the prediction of the serendipity can be seen below as a preview:

	movieId	userId	serendipity	popularity	predictedRating	similarity
0	163645	143883	2.703125	1.882240	3.601264	0.084404
1	164909	148369	2.854167	1.831087	3.948199	0.187440
2	166705	205356	2.750000	2.785330	3.533969	0.232906
3	159817	118322	3.125000	1.940232	4.488367	0.064103
4	165549	144922	2.633929	2.086360	4.038167	0.277083

It is worth mentioning that we proceeded to further preprocessing at this point in order to eliminate the number of non rating movies and get better results. We opted for movies which had been watched by five users at least.

As a result, the final dataset we had to work with was the following:

movieId	userId	serendipity	popularity	similarity	randNumCol
122922	125000	2.471154	1.442907	0.000000	3
122922	189816	2.471154	1.442907	0.092106	3
122922	168844	2.471154	1.442907	0.170467	3
122922	109335	2.471154	1.442907	0.136641	3
122922	116072	2.471154	1.442907	0.082707	3
...	...	...	...	...	...
167570	142972	2.700000	2.785330	0.000000	2
167570	193483	2.700000	2.785330	0.000000	2
167570	149956	2.700000	2.785330	0.000000	2
167570	143272	2.700000	2.785330	0.000000	2
167570	149815	2.700000	2.785330	0.000000	2

206 rows × 5 columns

However, we wanted to have the data in a pivot view (users-movies), thus, we created two pivot tables. The first one was as for the serendipity (see picture below):

movieId / userId	122922	152077	158783	159817	160874	161922	161966	162082	162414	162602	...	164981	165549	165551	166461	166528	166635	166643
100200	0.0	0.00	0.0000	0.000	0.000	0.0	0.0	0.00	0.000000	2.833333	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
101049	0.0	0.00	3.4375	0.000	0.000	0.0	0.0	3.75	0.000000	0.000000	...	0.0	2.633929	0.0	0.0	0.0	0.0	2.68125
101263	0.0	0.00	0.0000	0.000	0.000	0.0	0.0	0.00	0.000000	2.833333	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
101318	0.0	0.00	0.0000	0.000	0.000	0.0	0.0	0.00	0.000000	0.000000	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
101579	0.0	0.00	0.0000	3.125	0.000	0.0	0.0	0.00	0.000000	0.000000	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
206030	0.0	0.00	0.0000	0.000	3.075	0.0	0.0	0.00	0.000000	0.000000	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
206188	0.0	0.00	0.0000	0.000	0.000	0.0	0.0	0.00	0.000000	2.833333	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
206190	0.0	2.75	0.0000	0.000	0.000	0.0	0.0	0.00	0.000000	0.000000	...	0.0	0.000000	0.0	0.0	0.0	0.0	0.00000
206554	0.0	0.00	0.0000	0.000	0.000	0.0	0.0	0.00	2.973684	0.000000	...	0.0	2.633929	0.0	0.0	0.0	0.0	0.00000
206808	0.0	0.00	0.0000	0.000	0.000	0.0	0.0	0.00	0.000000	0.000000	...	0.0	2.633929	0.0	0.0	0.0	0.0	0.00000

139 rows × 26 columns

and the second one was as for the similarity (see picture below):

movieId / userId	122922	152077	158783	159817	160874	161922	161966	162082	162414	162602	...	164981	165549	165551	166461	166528	166635
100200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	...	NaN	NaN	NaN	NaN	NaN	NaN
101049	NaN	NaN	0.159091	NaN	NaN	NaN	NaN	0.0	NaN	NaN	...	NaN	0.162176	NaN	NaN	NaN	NaN
101263	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.167262	...	NaN	NaN	NaN	NaN	NaN	NaN
101318	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
101579	NaN	NaN	NaN	0.066667	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
206030	NaN	NaN	NaN	NaN	0.069444	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
206188	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.174603	...	NaN	NaN	NaN	NaN	NaN	NaN
206190	NaN	0.066667	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
206554	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.277282	NaN	...	NaN	0.207858	NaN	NaN	NaN	NaN
206808	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	0.109848	NaN	NaN	NaN	NaN

139 rows × 26 columns

As anyone can easily observe, the first pivoted dataset has no NaN values since we filled the NaN values with zeros. However, the second pivoted dataset about the similarity, we chose to leave NaNs within so as to keep the information what movies had been watched by user *u* and what movies had not been watched by user *u* .

Next, we constructed a new dataset named ‘**user_movie_serendipity_small**’ and replaced the values greater than zero with NaNs leaving as are the rest values zeros. By doing that, we achieved a matrix with unknown serendipity values. That matrix, though, was going to be used for the function called ‘**replace_unseen_movies_with_predicted_serendipity**’ in which we applied the regression model to predict the serendipities for movies each user had not watched yet. The result was:

movieId	122922	152077	158783	159817	160874	161922	161966	162082	162414	162602	...	164981	165549	165551	166461	1665
userId																
101049	2.665658	2.657468	0.000000	2.648191	2.644144	2.632578	2.629837	0.000000	2.624218	2.622312	...	2.641441	0.000000	2.621320	2.651057	2.6776
101263	2.662986	2.654796	2.630739	2.645518	2.641472	2.629905	2.627165	2.630924	2.621546	0.000000	...	2.638769	2.632730	2.618648	2.648384	2.6749
101579	2.670473	2.662283	2.638226	0.000000	2.648959	2.637392	2.634652	2.638411	2.629033	2.627127	...	2.646256	2.640217	2.626135	2.655871	2.6824
101818	2.662036	2.653845	2.629789	2.644568	2.640521	2.628955	2.626214	2.629974	0.000000	2.618690	...	2.637819	2.631780	2.617697	2.647434	2.6739
102355	2.661640	2.653449	2.629393	2.644172	2.640126	2.628559	2.625819	2.629578	2.620200	2.618294	...	2.637423	2.631384	2.617301	2.647038	2.6735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
206030	2.670266	2.662076	2.638019	2.652799	0.000000	2.637186	2.634445	2.638205	2.628826	2.626920	...	2.646049	2.640010	2.625928	2.655665	2.6822
206188	2.668008	2.659818	2.635761	2.650541	2.646494	2.634928	2.632187	2.635947	2.626568	0.000000	...	2.643791	2.637752	2.623670	2.653407	2.6799
206190	2.670473	0.000000	2.638226	2.653005	2.648959	2.637392	2.634652	2.638411	2.629033	2.627127	...	2.646256	2.640217	2.626135	2.655871	2.6824
206554	2.657381	2.649191	2.625134	2.639914	2.635867	2.624300	2.621560	2.625319	0.000000	2.614035	...	2.633164	0.000000	2.613043	2.642780	2.6693
206808	2.667259	2.659069	2.635012	2.649792	2.645745	2.634178	2.631438	2.635197	2.625819	2.623913	...	2.643042	0.000000	2.622921	2.652658	2.6792

106 rows × 26 columns

Note, here the serendipity for watched movies was replaced with the value 0. We decided to replace them with zero since we were going to take the higher serendipity scores and we did not want to collect the movies that each user had seen before.

3.4 Model Building and Assessment

Our objective, different from that of Problem A, is to do the recommendation so as to maximize the amount of serendipity in the system. As we have already noted, serendipity expresses the element of surprise a recommended movie provokes to the user.

The goal is to maximize total average serendipity to users:

$$\max \frac{1}{|U|} \sum_{u \in U} \left(\sum_{i \epsilon I} s_{iu} x_{iu} + \sum_{i \in I} \sum_{j \epsilon I} d_{ij} x_{iu} x_{ju} \right)$$

subject to the following constraints:

$$\sum_c \sum_{i \in Ic} x_{iu} = L_s$$

$$S_c = \frac{1}{|I_c|} \sum_{u \in U} \left(\sum_{i \in I} s_{iu} x_{iu} + \sum_{i \in I} \sum_{j \in I} d_{ij} x_{iu} x_{ju} \right) \geq \theta$$

where θ the minimum average serendipity for items of category c.

In order to implement the above objective functions and the corresponding constraints, we utilized **cvxpy** optimization Python library.

Regarding the above objective and constraint, we assumed the dissimilarity dij between the items, equal to zero. That means that the objective function is simplified as following:

$$\max \frac{1}{|U|} \sum_{u \in U} \sum_{i \in I} s_{iu} x_{iu}$$

and the constraint will be:

$$\bar{S}c = \frac{1}{|I_c|} \sum_{u \in U} \sum_{i \in I} s_{iu} x_{iu} \geq \theta$$

We consider that x is between 0 and 1.

Before proceeding to the optimization, we constructed the categories for movies. In other words, we assigned random categories to the movies. We consider that 4 categories are enough for 26 unique movies contained in our dataset. The category attribute here is called '**randNumCol**'.

	movieId	userId	serendipity	popularity	similarity	randNumCol
0	122922	125000	2.471154	1.442907	0.000000	3
1	122922	189816	2.471154	1.442907	0.092106	3
2	122922	168844	2.471154	1.442907	0.170467	3
3	122922	109335	2.471154	1.442907	0.136641	3
4	122922	116072	2.471154	1.442907	0.082707	3
..	...	...	...	...	...	...
59	166643	119863	2.681250	1.785330	0.180556	3
60	166643	143636	2.681250	1.785330	0.000000	3
61	166643	201027	2.681250	1.785330	0.125940	3
62	166643	144628	2.681250	1.785330	0.391304	3
63	166643	171987	2.681250	1.785330	0.391304	3

For example, in category 1 we have 46 observations of movieIds. Some movies appear with a higher frequency in our dataset, so we consider to construct equal categories.

Furthermore, we constructed a function called '**runOptimizatonPartB**' to run the optimization process.

```
runOptimizatonPartB (L,theta):
```

That function gets two parameters as input. The first parameter is the *L* which is the number of serendipity recommendations per user whereas the parameter *theta* is a coefficient used by the second constraint of the objective function.

The process that function follows is to maximize the objective function as can easily be seen at the picture below:

```
objective= cp.Maximize( (1/U)*cp.sum(cp.multiply(siu,X)))
```

where, X (namely xij) be the matrix that determines which movie will be recommended to which user and its shape is `(106, 26)` and siu is the pivot matrix with movies on the horizontal axis (columns) and users on the vertical axis (rows) and its shape is `(106, 26)` as well.

This objective function takes into account the following constraints:

- 0 <= X <= 1 and Xij are continuous variables.
- cp.sum(X, axis=1) == L
- for each category cp.sum(cp.multiply(category_indexes,X))>=theta*ic where *theta* is a minimum average amount of serendipity.

After running the '**runOptimizatonPartB**', we can see for the user **101049** the results of Xij will be:

```
[9.99999161e-01 1.73886440e-07 4.47826626e-10 7.37881721e-08
 5.90087525e-08 3.32938492e-08 3.06143391e-08 4.46676420e-10
 2.64120278e-08 2.83686838e-08 7.04018981e-08 7.19514533e-08
 5.33199854e-08 4.28350301e-08 9.99999837e-01 7.14254920e-08
 5.20406783e-08 4.46676420e-10 2.77146625e-08 8.97228704e-08
 9.99999910e-01 5.99046493e-08 4.47827439e-10 2.64872773e-08
 5.84394059e-08 4.11377443e-08]
```

From the above result we figure out that the variable xij is continuous and it returns 3 movies very close to the value 1 among 26 movies there exist.

At this point, we know what movies are recommended by the optimizer but we needed to find out the exact position (meaning their indexes). In order to gain that result, we constructed the function called '**recSerMovies**'

```
recSerMovies(result, Ls):
```

As we can see, the '**recSerMovies**' function gets two parameters as input. The result and the Ls. The result is the output of the aforementioned function '**runOptimizatonPartB**' and more specifically the values of that output. On the other hand, the Ls is the number of recommended movies to each user. The process of the function is to map the higher Xij to the user-movie index. What the function returns is a dataframe with columns the recommendations (i.e. 3 columns whether we recommend to users 3 serendipity movies or 5 columns whether we recommend to users 5 serendipity movies accordingly).

Thus, the output in this case would be:

	movie 1	movie 2	movie 3
101049	162414	161966	167570
101263	162082	161966	167570
101579	161966	162082	167570
101818	161966	162082	167570
102355	161966	162082	167570
...	...	...	...
206030	161966	162082	167570
206188	161966	162082	167570
206190	161966	162082	167570
206554	161966	162082	167570
206808	162082	161966	167570

106 rows × 3 columns

Test scenario for user [**101049**]:

We recommended **L = 3** movie Ids with **θ = 0.1**

- Recommendations for Model with similarity and popularity:

movieId	title	releaseDate	directedBy	starring	imdbId	tmdbId	genres
162414	Moonlight	1969-12-31	Barry Jenkins	Naomie Harris,Mahershala Ali,Andre Holland	4975722	376867.0	Drama
161966	Elle (2016)	2016-05-25	Paul Verhoeven	Isabelle Huppert,Laurent Lafitte,Anne Consigny...	3716530	337674.0	Thriller
167570	The OA	1969-12-31	NaN	NaN	4635282	432192.0	NaN

- Recommendations for Model with similarity, popularity and predicted rating:

movieId	title	releaseDate	directedBy	starring	imdbId	tmdbId	genres
165551	Lion (2016)	2016-11-24	Garth Davis	Dev Patel,Rooney Mara,Nicole Kidman,Nawazuddin...	3741834	334543.0	Drama
162414	Moonlight	1969-12-31	Barry Jenkins	Naomie Harris,Mahershala Ali,Andre Holland	4975722	376867.0	Drama
161966	Elle (2016)	2016-05-25	Paul Verhoeven	Isabelle Huppert,Laurent Lafitte,Anne Consigny...	3716530	337674.0	Thriller

Last, we created one more function to calculate the Total Serendipity for the plot, The function is called '**getTotalSer**' and takes three parameters as inputs (as can be seen from the picture below):

```
getTotalSer(Ls, siu,result):
```

The Ls parameter is the number of the recommendations per user, the siu is the serendipity matrix for user movies and the result is the output from the optimizer. This function thus calculates the total serendipity score by aggregating all serendipities.

<u>For example:</u>

Model with similarity and popularity gives total serendipity 927.2141

Model with similarity, popularity and predicted rating gives total serendipity 925.1779

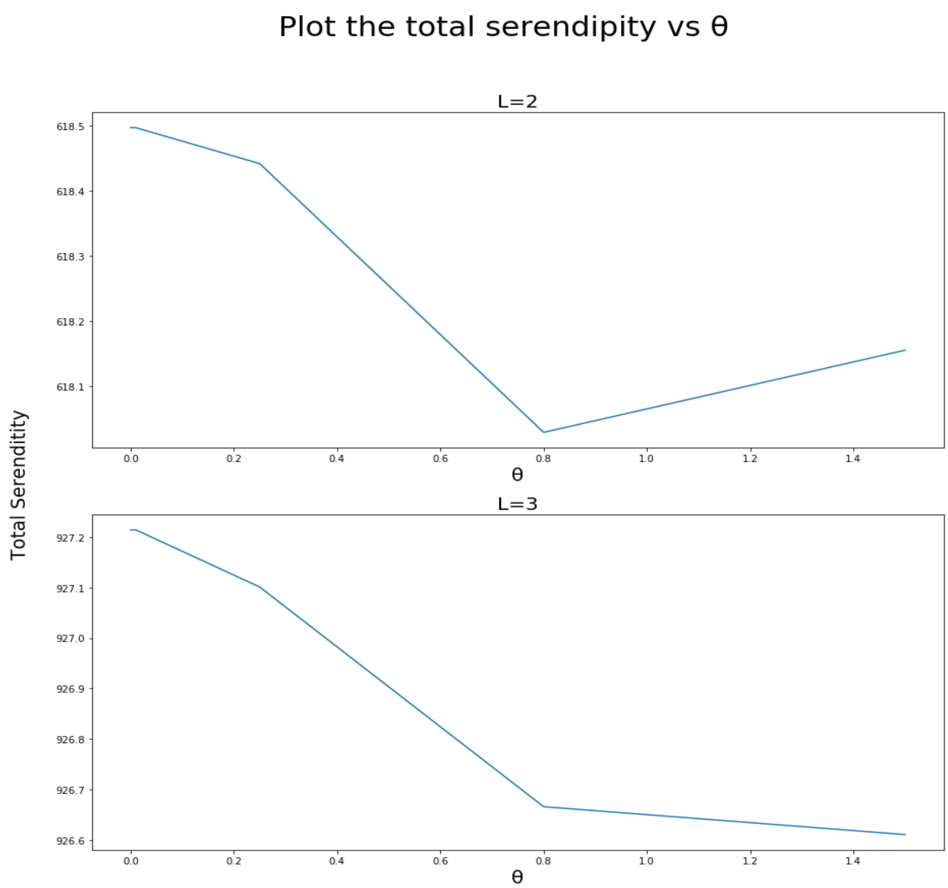
3.5 Plots and results

By plotting the total serendipity for vs ϑ where ϑ = [0, 0.01, 0.25, 0.80, 1.5], we get the following results:

Model with similarity and popularity

	L	theta	Total Serendipity
0	2	0.00	618.497351
1	2	0.01	618.497351
2	2	0.25	618.433652
3	2	0.80	617.948422
4	2	1.50	617.954882
5	3	0.00	927.214143
6	3	0.01	927.214143
7	3	0.25	927.090555
8	3	0.80	926.519412
9	3	1.50	926.519412

And the plots where L = 2 or L = 3 are:

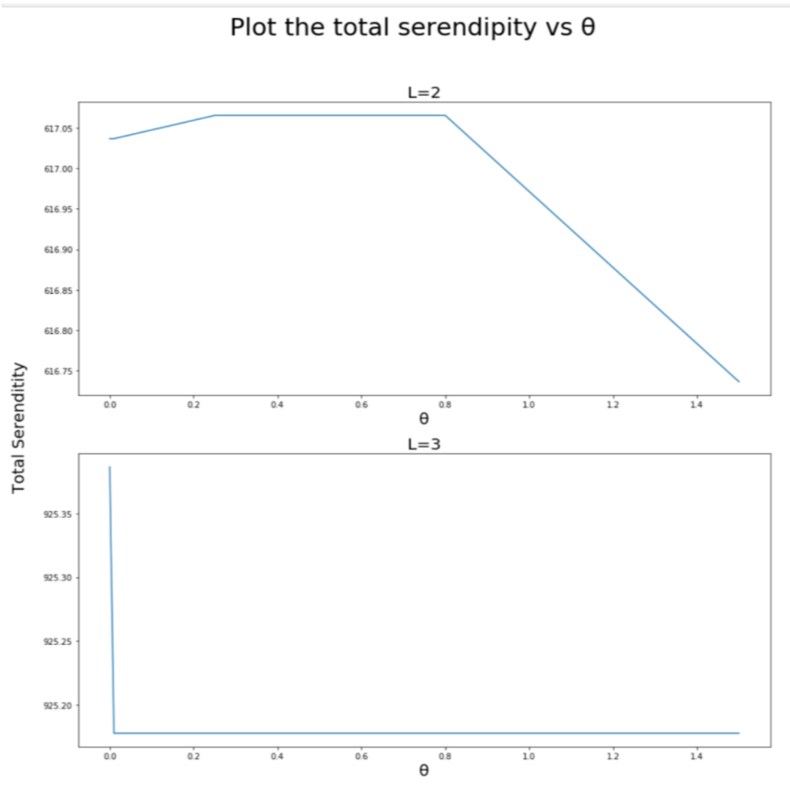


From the above graphs, we can infer at a glance that the total serendipity differs in relation to the L (L is the number of recommendations to each user). For L = 2 the total serendipity ranges from 618.15 to 618.50 while for L = 3 the total serendipity ranges from 926.61 to 927.21. What provokes interest though is the number of total serendipity behaves differently in the two cases. Specifically, in the first case (where the L=2), the total serendipity decreases as long as the ϑ increases up to the point 0.8. From the value θ = 0.8 and next, the total serendipity seems to increase again.

However, the previous case is not the same for L = 3. For all values of theta the total serendipity keeps decreasing. The difference here needs to be mentioned though is the total serendipity decreases at a smaller pace after the value of θ = 0.8.

Model with similarity, popularity and predicted rating

By plotting the total serendipity for vs ϑ where ϑ = [0, 0.01, 0.25, 0.80, 1.5], we get the following results:



	L	theta	Total Serendipity
0	2	0.00	617.036809
1	2	0.01	617.036809
2	2	0.25	617.065501
3	2	0.80	617.065501
4	2	1.50	616.736358
5	3	0.00	925.386386
6	3	0.01	925.177961
7	3	0.25	925.177961
8	3	0.80	925.177961
9	3	1.50	925.177961

In this figure in the case of L=2 the total serendipity is reduced for values of θ >= 0.8. Moreover, in the case of L = 3 recommendations the maximum total serendipity is achieved for θ = 0.1