# Project Report: Privacy Preserving with Preference Elicitation Process

Team Tatooine • 05.07.2020

# Progress

## Recent progress

- Data Gathering
- Data Cleaning
- Implemented Content Based Recommender System
- Implemented Hierarchical Clustering

## Work In-Progress

- Implementing Differential Privacy Mechanism

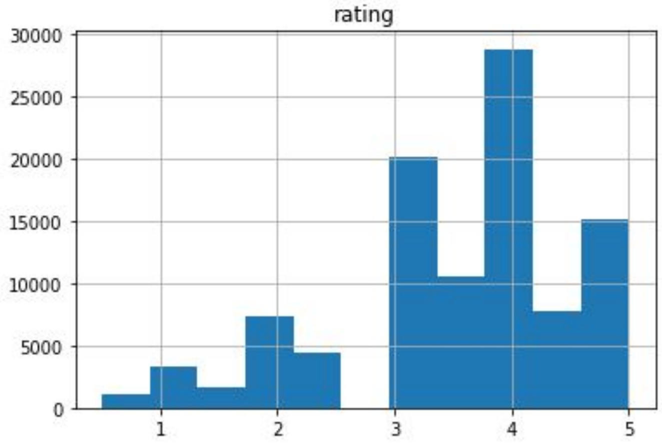
# Training Data Set

| | userId | movieId | rating | title | year | genre1 | genre2 | genre3 | genre4 | genre5 | genre6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 31 | 2.5 | Dangerous Minds | 1995.0 | Drama | NaN | NaN | NaN | NaN | NaN |
| 117 | 1 | 1129 | 2.0 | Escape from New York | 1981.0 | Action | Adventure | Sci-Fi | Thriller | NaN | NaN |
| 165 | 1 | 1172 | 4.0 | Cinema Paradiso | 1989.0 | Drama | NaN | NaN | NaN | NaN | NaN |
| 403 | 1 | 1343 | 2.0 | Cape Fear | 1991.0 | Thriller | NaN | NaN | NaN | NaN | NaN |
| 211 | 1 | 1263 | 2.0 | Deer Hunter, The | 1978.0 | Drama | War | NaN | NaN | NaN | NaN |
| 259 | 1 | 1287 | 2.0 | Ben-Hur | 1959.0 | Action | Adventure | Drama | NaN | NaN | NaN |
| 305 | 1 | 1293 | 2.0 | Gandhi | 1982.0 | Drama | NaN | NaN | NaN | NaN | NaN |
| 849 | 1 | 3671 | 3.0 | Blazing Saddles | 1974.0 | Comedy | Western | NaN | NaN | NaN | NaN |
| 84 | 1 | 1061 | 3.0 | Sleepers | 1996.0 | Thriller | NaN | NaN | NaN | NaN | NaN |
| 806 | 1 | 2968 | 1.0 | Time Bandits | 1981.0 | Adventure | Comedy | Fantasy | Sci-Fi | NaN | NaN |

# Training Data Set

```
no_of_ratings = movie_ratings.rating.unique()
print(no_of_ratings)
movie_ratings.hist(bins=11)
```

```
[2.5 2.  4.  3.  1.  3.5 5.  4.5 1.5 0.5]
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0f533b39e8>]],
      dtype=object)
```




```
diff_ratings = sorted_df[['genre1']]
rating_values = diff_ratings.genre1.unique()
print(rating_values)
```

```
['Drama' 'Action' 'Thriller' 'Comedy' 'Adventure' 'Animation' 'Fantasy'
 'Children' 'Crime' 'Mystery' 'Documentary' 'Horror' 'Musical' 'Film-Noir'
 'Sci-Fi' 'Romance' 'Western' 'War' '(no genres listed)']
```

- **10 unique rating values**
- **Maximum ratings between 3.5 and 4.5**

- **Genres: Adventure, Animation, Children, Action, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War and Western**

Movie Recommendation for a user based on his previous activity - movie ratings

```
In [1]: runfile('D:/CySA+Prep/RecommenderSystem_CIS700/RecSys-Materials/
RecSys-Materials/CollaborativeFiltering/SimpleUserCF.py', wdir='D:/CySA+Prep/
RecommenderSystem_CIS700/RecSys-Materials/RecSys-Materials/
CollaborativeFiltering')
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Inception (2010) 3.3
```

Measuring Accuracy of the Algorithm - Using Hit Rate as accuracy measure

```
In [2]: runfile('D:/CySA+Prep/RecommenderSystem_CIS700/RecSys-Materials/
RecSys-Materials/CollaborativeFiltering/EvaluateUserCF.py', wdir='D:/CySA
+Prep/RecommenderSystem_CIS700/RecSys-Materials/RecSys-Materials/
CollaborativeFiltering')
Reloaded modules: MovieLens
Loading movie ratings...

Computing movie popularity ranks so we can measure novelty later...
Estimating biases using als...
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
HR 0.055141579731 74367
```

# User based Collaborative Recommender System

**Hit Rate: We calculate the Top-N recommendations for a user, if that user has rated one movie from those recommendations, we consider that as a hit. So the hit rate will be the ratio of hits from the Top-N recommendations for all the users to the total number of users**

$$HR = totalhits/totalusers$$

| | userId | recommendedMovie | count | recommendedMovieRatng |
|---|---|---|---|---|
| 0 | 1 | 527 | 1 | 4.5 |
| 1 | 2 | 908 | 1 | 2.8 |
| 2 | 3 | 780 | 1 | 4.2 |
| 3 | 4 | 527 | 1 | 2.9 |
| 4 | 5 | 778 | 1 | 4.3 |
| ... | ... | ... | ... | ... |
| 666 | 667 | 50 | 1 | 3.3 |
| 667 | 668 | 356 | 1 | 2.6 |
| 668 | 669 | 2762 | 1 | 3.7 |
| 669 | 670 | 58559 | 1 | 2.9 |
| 670 | 671 | 135 | 1 | 2.0 |

671 rows × 4 columns

# Output of Recommender System

- Ran the recommender system over all the users from input dataset (ratings.csv)
- Mapped all the recommended movies to their movieId from the movies dataset (movies.csv)

# Recommender output analysis

| | userId | recommendedMovie | count | recommendedMovieRatng |
|---|---|---|---|---|
| **0** | 1 | 527 | 1 | 4.5 |
| **1** | 2 | 908 | 1 | 2.8 |
| **2** | 3 | 780 | 1 | 4.2 |
| **3** | 4 | 527 | 1 | 2.9 |
| **4** | 5 | 778 | 1 | 4.3 |
| **...** | ... | ... | ... | ... |
| **666** | 667 | 50 | 1 | 3.3 |
| **667** | 668 | 356 | 1 | 2.6 |
| **668** | 669 | 2762 | 1 | 3.7 |
| **669** | 670 | 58559 | 1 | 2.9 |
| **670** | 671 | 135 | 1 | 2.0 |

671 rows × 4 columns

```
[5] new_df = df1.groupby(['recommendedMovie'], as_index=False)['count'].sum()
    print(new_df)
```

```
     recommendedMovie  count
0                   1      5
1                   3      1
2                   5      8
3                   6      8
4                  10      2
..                ...    ...
110             68954      1
111             70286      2
112             79132     16
113             80463      1
114            109487      3

[115 rows x 2 columns]
```

# Hierarchical Clustering

Hierarchical clustering is used for clustering together movies that are similar to each other.

The algorithm uses an agglomerative(bottom up) approach to derive the clusters

# Using One hot encoding for creating vectors

| | | movieId | | | |
|---|---|---|---|---|---|
| | | m1 | m2 | ......... | mn |
| | u1 | 0 | 1 | ......... | 1 |
| | u2 | 1 | 0 | ......... | 1 |
| userId | | 1 | 1 | ......... | 1 |
| | un | 0 | 0 | ......... | 1 |

# Correlation Matrix

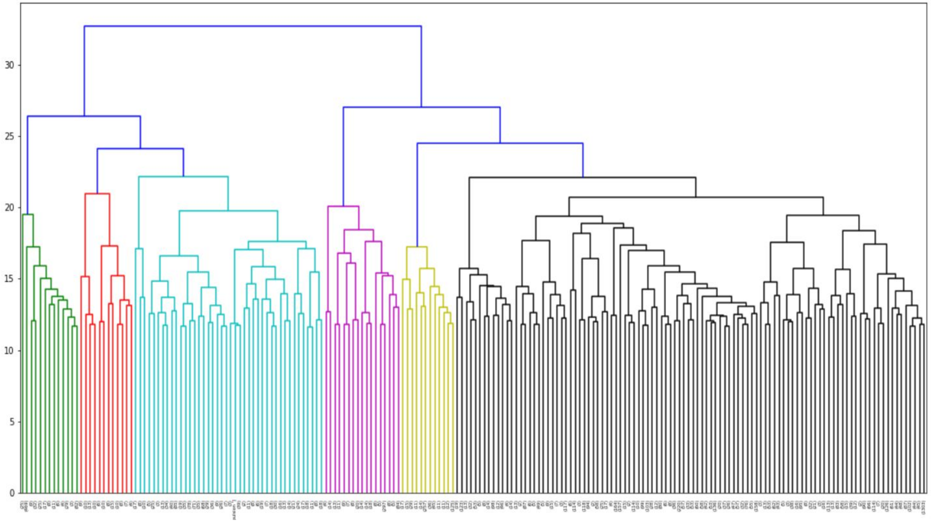
| movieId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| userId | | | | | | | | | | | | | | |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5 | 0.0 | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

A correlation matrix is created to define the similarities between each movie with every other movie.

A correlation of '1' defines movies that are the most similar

This is used to create hierarchical clusters

# Clustering Result




The scipy package is used to perform hierarchical clustering from the given matrix

A traditional dendogram is created with a limit of 200 clusters

Movies that are most similar to each other belong to the same cluster

# Differential Privacy Mechanism

$$\Pr[x \mid 4.5] = \frac{e^{\frac{(4.5,x)\varepsilon}{d}}}{\sum_{y[0,5]} e^{\frac{(4.5,y)\varepsilon}{d}}}$$

x = 4.5 (system generated rating)

y = k (user given rating for the same movie)

d = error distance of x|y over a scale of (0, 0.1, 0.2, 0.3, ………., 5.0)

eps = {0.1, 0.2, 0.3, ……, 1.0}

# Differential Privacy Mechanism

Approach 1:

- Alter Just one user record in the dataset, ratings.csv
- See if there is any change in the clusters.

Approach 2:

- Alter in batches of 5, 10, 15,..... User Records in the dataset
- See the changes in clusters (if any)

# Differential Privacy Mechanism

Approach 1:

- Generate fake data for 1 user

Concern:

- Should I be concerned about the distribution of data?

Approach 2:

- Generate fake data for 5 users

Concern:

- Same as Approach 1.

# Questions, comments and/or concerns?