Capstone Project
Machine Learning Nano-Degree

Eslam Hathout
10/10/2019

Definition

Project Overview

“What movie should I watch this evening?” Have you ever had to answer this question at least once when you come home from work? As for me: yes, and more than once. From Netflix to Hulu, the need to build robust movie recommendation systems is extremely important given the huge demand for personalized content of modern consumers.

An example of a recommendation system is such as this:

- User A watches Game of Thrones and Breaking Bad.
- User B does search on Game of Thrones, then the system suggests Breaking Bad from data collected about user A.

Recommendation systems are used not only for movies but on multiple other products and services like Amazon (Books, Items), Pandora/Spotify (Music), Google (News, Search), YouTube (Videos), etc.

I decided to build a full Django website based on the idea of the movie recommendation systems. The website itself is not included in the scope of the project, but the movie recommendation system itself will be the intended scope in the capstone. You can read more about movie recommendation systems through this link https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223.

Problem Statement

The problem which I would like to find a solution for here in the capstone is to help users find the best suitable movie that matches their taste in movies.

The solution for the problem is to build clusters that could help to recommend movies for the users that exist in each cluster.

Metrics

I will be using the KMeans clustering algorithm for clustering the dataset and for comparing my capstone's model with the benchmark model, I will use Silhouette Coefficient because it works good in case of the If the ground truth labels are not known.

The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around zero indicate overlapping clusters and the score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

Analysis

Data Exploration

"The dataset(s) and/or input(s) to be used in the project are thoroughly described. Information such as how the dataset or input is (was) obtained, and the characteristics of the dataset or input, should be included. It should be clear how the dataset(s) or input(s) will be used in the project and whether their use is appropriate given the context of the problem.

The dataset which I will be using throughout the project is the Movielens dataset which is presented here: https://grouplens.org/datasets/movielens/. I will use the Full Movielens dataset which is a dataset of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. Includes tag genome data with 14 million relevance scores across 1,100 tags. Last updated 9/2018.

Exploratory Visualization

Fig#1: The plot below shows an initial trial to work with the data as it shows the average rating of the user to two genres of the movies Romance and Science Fiction. It also shows how a good cluster algorithm should cluster the data as visually we can indicate that the points that are more than 3 in SciFi and less than 2 in romance indicate that the people located in this area of

the graph tend to prefer sci-fi movies taste than romance movies.

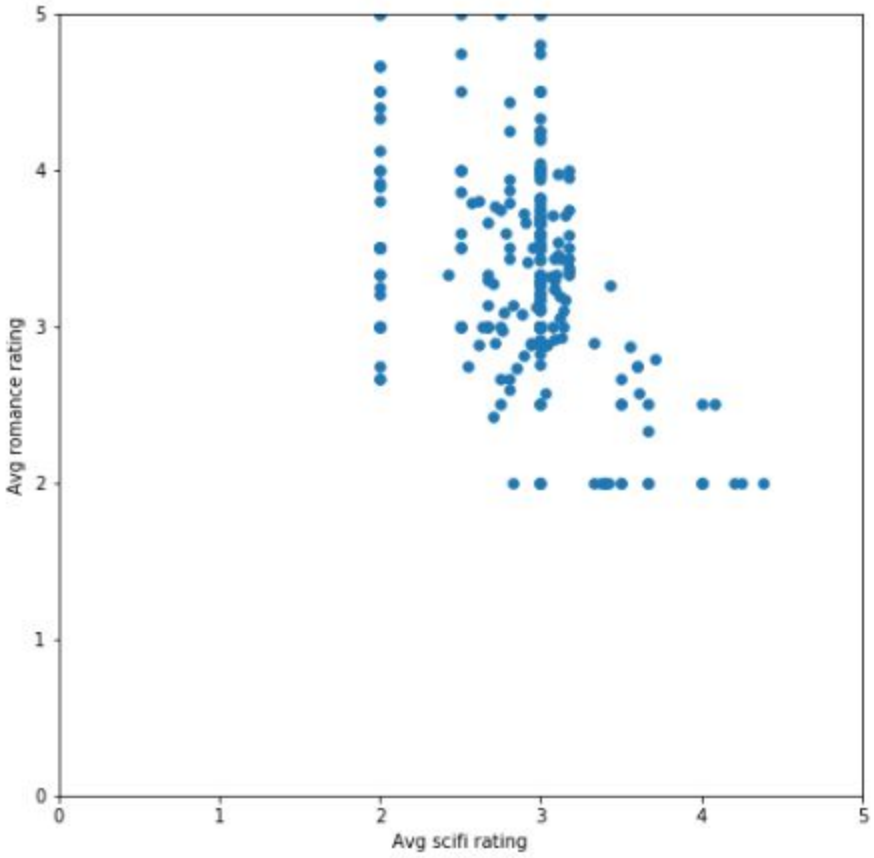
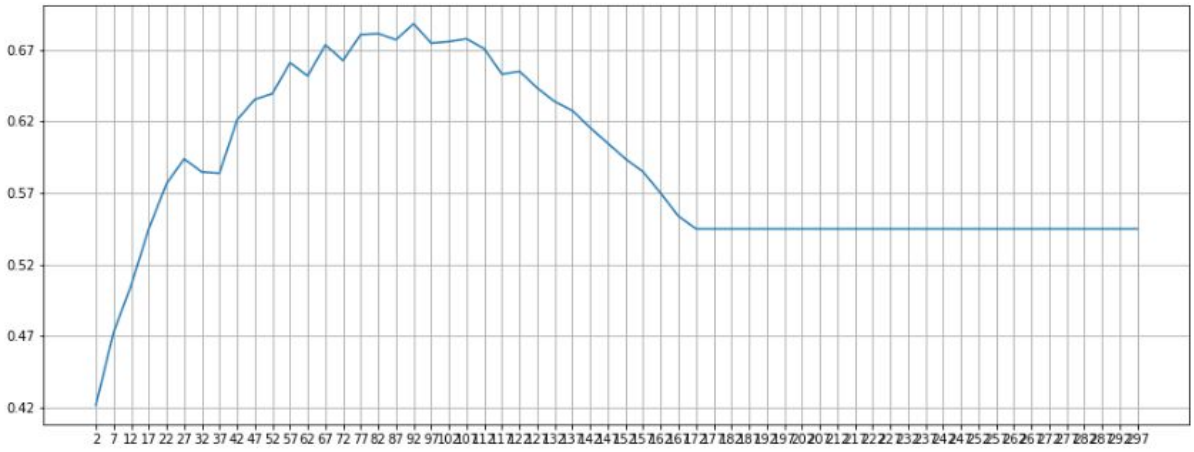


Fig #2: Shows the silhouette score across different values of K (X-axis represents the different values of K, Y-axis represents the score).



Algorithms and Techniques

The algorithm for clustering is the K-Means algorithm. It shows great results in the benchmark model and I think with some tuning of its parameters, I will achieve better results.

The average complexity is given by O(k n T), were n is the number of samples and T is the number of iteration.The worst-case complexity is given by O(n^(k+2/p)) with n = n_samples, p = n_features.

In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available).

The benchmark model, as well as the final model, uses the KMeans algorithm, but with different settings.

In the final model, I decided to check the best settings for the K value in the algorithm which determines the number of clusters that should be made by the algorithm. I build a simple solution to check the best suitable value for K by calculating the silhouette score across different values of K so that I can use the best value of K suitable for the problem.

Benchmark

The benchmark model which I will be looking for in the project is the K-means clustering project which was already implemented in the nano-degree on a smaller dataset.

The K-means clustering ‘the benchmark model’ with K =20 shows great results in successfully build clusters of users based on their ratings to different types of movies. Instead, I worked on a solution to not just set the value of K arbitraty, but I decided to use the silhouette score to adjust the value of K.

Methodology

Data Preprocessing

As the dataset contains a huge number of rows and it often returns MemoryError while working with data, I've decided to do many steps to reduce the number of rows in the dataset as much as I can as long as I am preserving good knowledge in the final dataset.

The first step of the preprocessing is to remove the rows the contain movie ratings less than 2.0, then, I decided to keep only the movies that produced in the 19th so I built a block of code to remove the movies other than 19th, then also removing the ratings for these movies.

Another level of preprocessing is converting the two datasets to one dataset containing the columns= 'title', values='rating' for each movie, then using csr_matrix on the pivot table to best adjust the data before clustering.

Implementation

I've implemented the final model based on the values of K produced by calculating the different values of the silhouette score. The best setting for K was with K=97, but I decided to go with a smaller value of K to best analysis each cluster 'K=37'.

After building the model, I did visualize some of the clusters using heatmaps, then checking one of the clusters and extracting the best 20 movies that rated by people within the same cluster based on the mean value of ratings.

The 3rd step was to make a new user and simulate some ratings for that user, then predicting the best possible cluster for that user and use that information to recommend movies for him/her.

Refinement

The refinement that I needed to add to the final model was to best adjust the value of K based on the silhouette score.

Another refinement was implemented by keeping the 19th movies and adjusting the rating system from 0-5 to 2:5.

Results

Model Evaluation and Validation

The final model is a KMeans model with a K value adjusted using the silhouette score. The other model parameter is the algorithm used in the cluster which I've decided to set it to 'Full' as the input for the model is sparse data and according to the algorithm's documentation, 'Full' is the best settings for sparse data.
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Justification

The key element for achieving the best results with the KMeans algorithm is to best adjust the value of K. The benchmark model arbitrary set it to 20.

I used the silhouette score to best address the value needed for K using the elbow method 'Comparing different values of accuracy against different levels of K'.

Conclusion

Free-Form Visualization

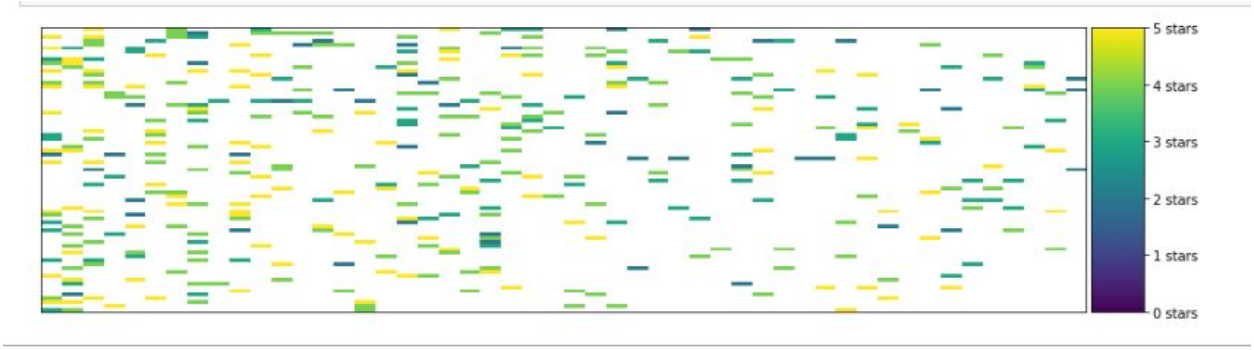
After building the final model, I decided to build a new user and feed it to the algorithm to predict a cluster for that user. The user will be any new user that will register to the final solution website and ask for some movie recommendations.

Each user should have the ability to rate any movie in our final dataset, then based on his/her rating to some movies, he/she will be assigned to a cluster. I decided that the movies that the new user would rate will be Starman (1984), Starman (1984), and Matrix, (1999) and the ratings for these movies would be 5 each.

After building the model and passing it's data to the KMeans algorithm to predict his/her cluster, I found that the user cluster would be 22.

I used that information to visualize the ratings of the people that exist in the same cluster using heatmaps and here are the results.

Fig #3: A heatmap shows the distribution of people ratings for a certain cluster for some movies.



Using the same piece of code, I visualized some of the clusters produced by the KMeans algorithm and here are some of them.

Fig #4: A heatmap shows the distribution of people ratings for cluster #16

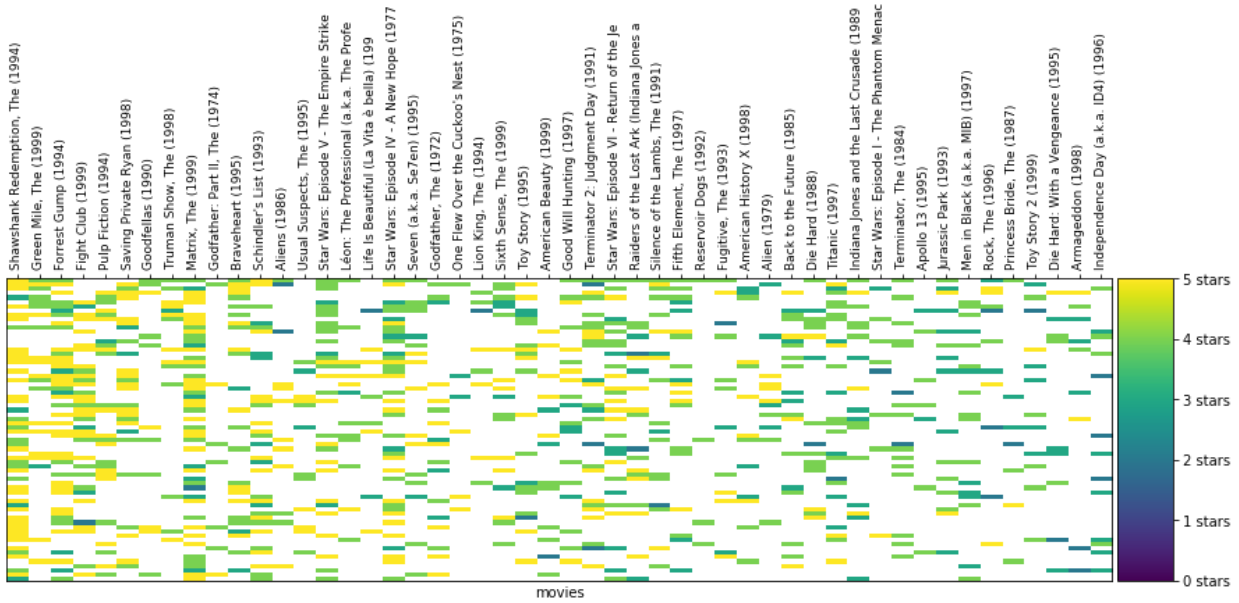


Fig #5: A heatmap shows the distribution of people ratings for cluster #22

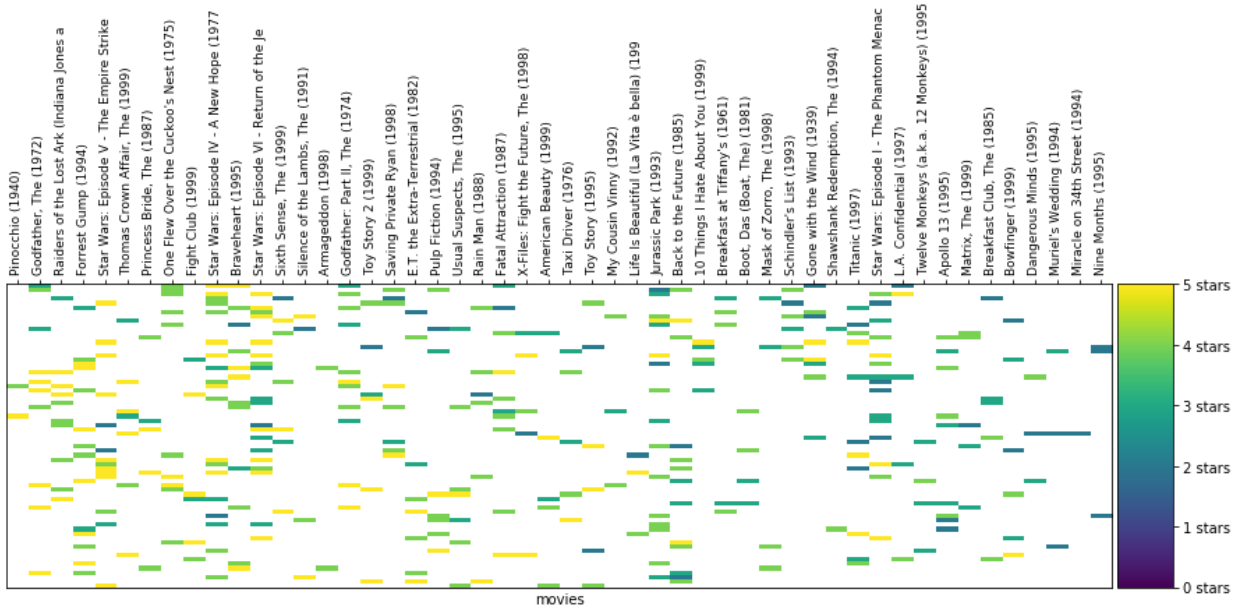
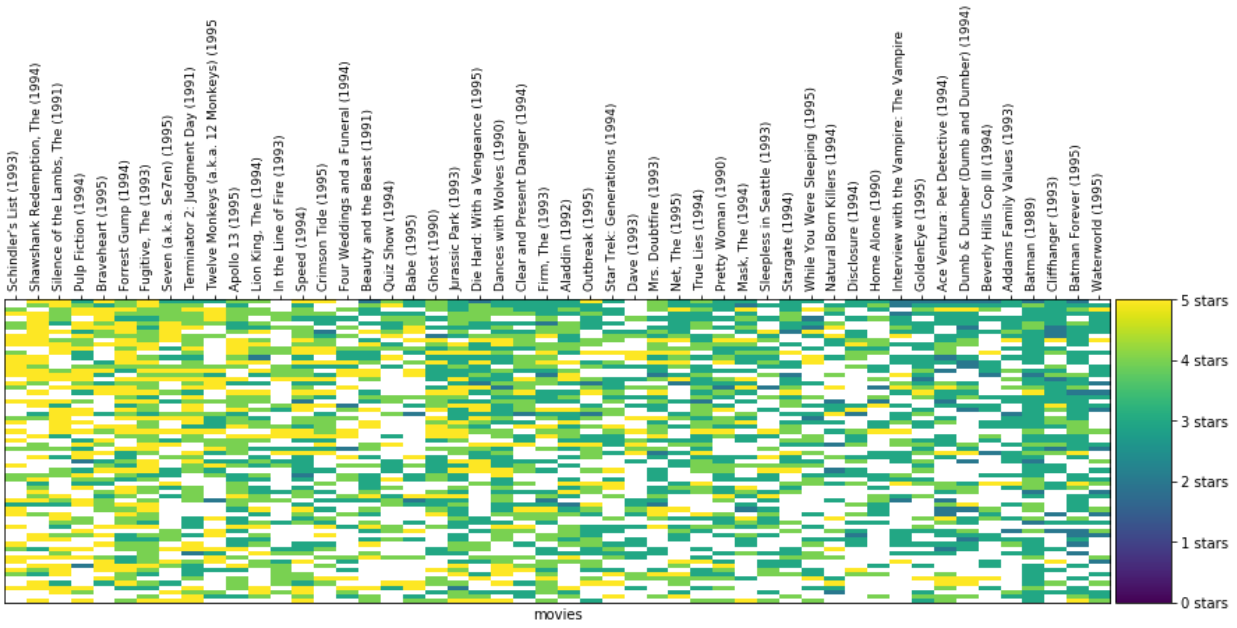
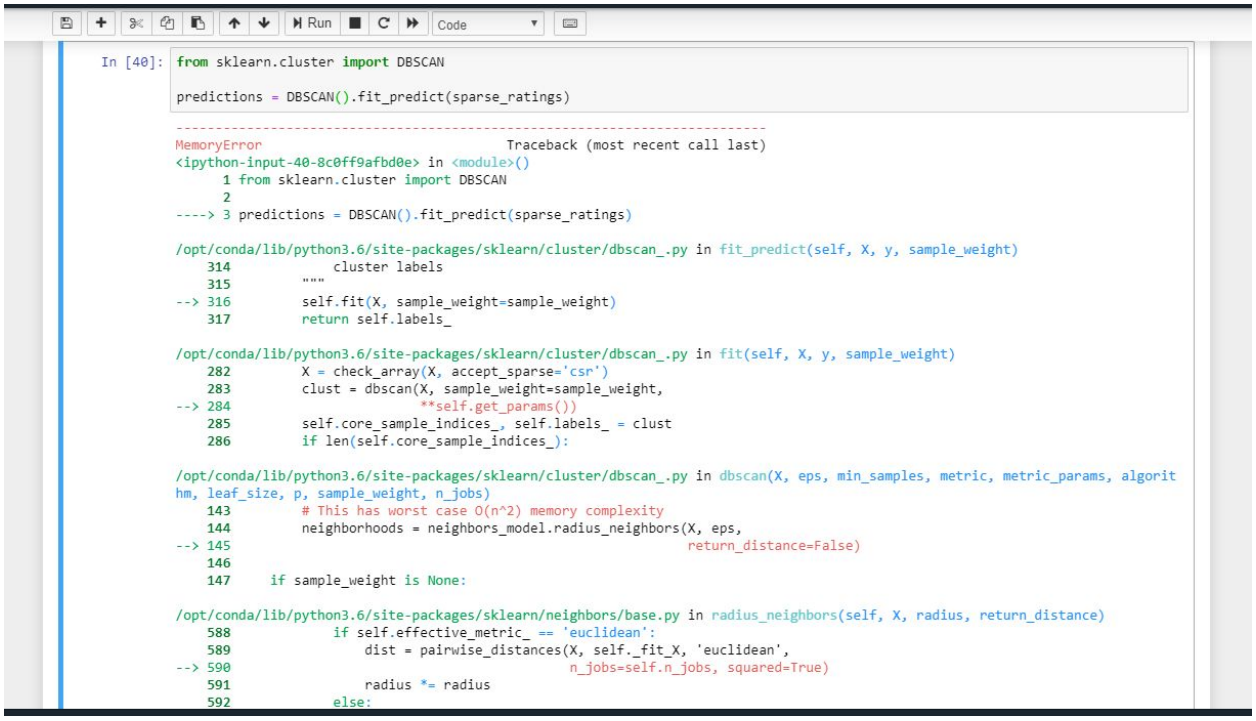


Fig #6: A heatmap shows the distribution of people ratings for cluster #12



Reflection

The main difficulty of the project was to deal with big data. In the beginning, I was getting MemoryError each step/action I perform on the entire dataset.



```
In [40]: from sklearn.cluster import DBSCAN

         predictions = DBSCAN().fit_predict(sparse_ratings)

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-40-8c0ff9afbd0e> in <module>()
      1 from sklearn.cluster import DBSCAN
      2
----> 3 predictions = DBSCAN().fit_predict(sparse_ratings)

/opt/conda/lib/python3.6/site-packages/sklearn/cluster/dbscan_.py in fit_predict(self, X, y, sample_weight)
    314             cluster labels
    315         """
--> 316         self.fit(X, sample_weight=sample_weight)
    317         return self.labels_

/opt/conda/lib/python3.6/site-packages/sklearn/cluster/dbscan_.py in fit(self, X, y, sample_weight)
    282         X = check_array(X, accept_sparse='csr')
    283         clust = dbscan(X, sample_weight=sample_weight,
--> 284                        **self.get_params())
    285         self.core_sample_indices_, self.labels_ = clust
    286         if len(self.core_sample_indices_):

/opt/conda/lib/python3.6/site-packages/sklearn/cluster/dbscan_.py in dbscan(X, eps, min_samples, metric, metric_params, algorit
hm, leaf_size, p, sample_weight, n_jobs)
    143         # This has worst case O(n^2) memory complexity
    144         neighborhoods = neighbors_model.radius_neighbors(X, eps,
--> 145                                                          return_distance=False)
    146
    147     if sample_weight is None:

/opt/conda/lib/python3.6/site-packages/sklearn/neighbors/base.py in radius_neighbors(self, X, radius, return_distance)
    588             if self.effective_metric_ == 'euclidean':
    589                 dist = pairwise_distances(X, self._fit_X, 'euclidean',
--> 590                                           n_jobs=self.n_jobs, squared=True)
    591                 radius *= radius
    592             else:
```

That's why I decided to build solutions to reduce some rows of the dataset as I started by changing the ratins scale to remove the ratings less than 2, then removing the movies that did not produce in the 19th and removing the ratings that the user made for the movies other than the 19th movies.

After such refinings that computer power that I am using didn't help in building the models so, I decided to slice the dataset and build the models using a slicing of the dataset, and the same settings would be applied to the entire dataset 'Future Work'.

So that was the trickest part of the project. Moving to the fun part, I was amazed by how easy it is to build a recommendation system for types of problems that heavily depends on recommendation such as movies. I am personally interested in completing the final model and feeding the final solution to a website and start to recommend movies for users/friends.

Improvement

The improvement which I am looking forward to working on later is to try different models that proved accuracy on recommendation systems or clustering, in general, to achieve the best suitable settings of clusters.

Another improvement that I am looking forward to making is to use the model on the full Movielense dataset on a more advanced machine to get insights about the entire knowledge exists in the dataset without the need to find ways to remove parts of the data for performance reasons.