POLITECNICO
MILANO 1863

Movie Recommender System

Numerical Analysis for Machine Learning

Nicolò Tocalli (10716843)

Professor: Edie Miglio
Politecnico di Milano

September 2, 2024

Contents

1 Introduction **2**

2 Technology **2**
2.1 Unsupervised learning and Clustering . . . 2
2.2 Movie recommender systems . . . 3
2.3 Clustering algorithms . . . 3
2.3.1 K-Means algorithm . . . 3
2.3.2 Birch algorithm . . . 4
2.3.3 Minibatches K-means . . . 4
2.3.4 Mean-Shift Clustering . . . 4
2.3.5 Affinity propagation . . . 5
2.3.6 Agglomerative Clustering . . . 5
2.3.7 Spectral Clustering . . . 6

3 Dataset overview **7**

4 Data processing **8**
4.1 Creation of data points . . . 8
4.2 Training and Test set separation . . . 9

5 Cluster analysis **9**
5.1 Choosing optimal number of clusters K . . . 10
5.2 Model fitting . . . 10
5.3 Model evaluation and cluster quality . . . 10
5.3.1 Dunn's index . . . 11
5.3.2 Average similarity . . . 12
5.3.3 Davies-Bouldin Index . . . 13
5.3.4 Calinski-Harabasz Index . . . 13
5.3.5 Computational time . . . 14
5.4 Results . . . 15

6 A simple Movie Recommender System **16**
6.1 The algorithm . . . 16
6.2 Recommender Evaluation . . . 17
6.2.1 Mean Absolute Error (MAE) . . . 17
6.2.2 Root Mean Squared Error (RMSE) . . . 18
6.2.3 Precision . . . 18
6.3 Results . . . 19

7 Adding more dimensions **19**
7.1 Data preparation . . . 19
7.2 *PCA* Analysis . . . 19
7.3 Results . . . 21

8 Conclusions **22**

1 Introduction

In today's digital age, Recommender systems play a pivotal role in personalizing user experiences across various platforms, from e-commerce to social media and, notably, online streaming services. These systems analyze vast amounts of data to suggest products, content, or services that align with individual preferences, thereby enhancing user satisfaction and engagement. In the realm of movie recommendations, these systems have evolved to manage and curate immense libraries of content, ensuring users can easily discover films that match their tastes. Techniques such as collaborative filtering, content-based filtering, and hybrid models are commonly used to generate these recommendations. Among these, clustering-based methods have gained attention for their ability to group users or movies into clusters based on shared characteristics, offering personalized suggestions by considering the collective behavior of these groups.
In this scenario the paper [1] I had to analyze and reproduce focuses on finding the best algorithm that optimizes the clustering process by evaluating and comparing different clustering techniques using several metrics.
By identifying the most effective clustering approach, the authors aim to enhance the quality of movie recommendations, thereby providing a more personalized and efficient user experience.

2 Technology

2.1 Unsupervised learning and Clustering

Clustering is a fundamental unsupervised learning technique in data analysis that involves grouping a set of objects so that those within the same group, or cluster, are more similar to each other than to those in other groups. Unlike supervised learning, clustering does not require labeled data, making it especially useful for discovering patterns and structure in large, unlabeled datasets. This method is widely applied in various fields, including machine learning, pattern recognition, and data mining, to facilitate data interpretation and inform decision-making.

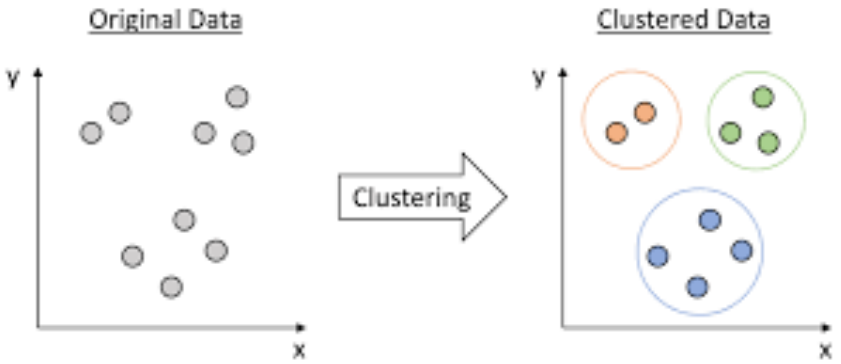

Figure 1: *Example of clustering*

[1] https://www.mdpi.com/2073-8994/12/2/185

2.2 Movie recommender systems

A movie recommender system is a tool designed to help users discover films that match their preferences by analyzing their past behavior, such as movie ratings and viewing history. These systems aim to reduce the time and effort users spend searching for movies by suggesting options that are likely to align with their tastes.
Clustering is a key technique used in movie recommender systems to group users with similar preferences. By applying clustering algorithms, the system identifies groups of users who have rated or liked similar movies. Once these clusters are formed, the system can recommend movies to a user based on the preferences of others in the same cluster as shown in Figure 2. This approach improves the accuracy of recommendations by leveraging the collective preferences of like-minded individuals.

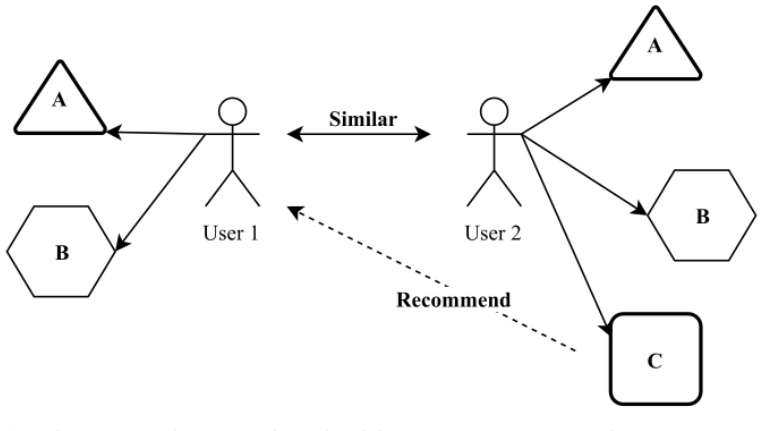

Figure 2: *Similarities within people to build a movie recommendation system for users.*

2.3 Clustering algorithms

In this study, I compared the performances of 7 clustering techniques:

2.3.1 K-Means algorithm

The K-Means algorithm partitions a dataset into **K** clusters by minimizing the within-cluster sum of squares **(WCSS)**. It operates as follows:

1. Initialization: Randomly select K initial centroids from the dataset.
2. Assignment: Assign each data point to the nearest centroid based on the Euclidean distance.
3. Update: Recalculate the centroids as the mean of all data points assigned to each cluster.
4. Iteration: Repeat the assignment and update steps until the centroids converge, meaning there is no significant change in their positions.

The algorithm converges to a solution where the total variance within clusters is minimized, providing **K** well-defined clusters.

2.3.2 Birch algorithm

The BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm is designed for clustering large datasets efficiently by building a hierarchical tree structure called the CF **(Clustering Feature)** tree. Here's how it works:

1. Initialization: Create a CF tree with a specified branching factor and maximum number of nodes.
2. Insertion: Insert each data point into the CF tree. Data points are summarized into Clustering Features (CFs), which are tuples of the number of points, the linear sum of points, and the squared sum of points. These CFs are stored in nodes of the tree.
3. Tree Maintenance: Periodically or as needed, the CF tree is adjusted by merging or splitting nodes to ensure it remains balanced and that CFs represent clusters effectively.
4. Clustering: Once the CF tree is built, a clustering algorithm like K-Means is applied to the CFs in the tree to refine clusters and produce the final clustering result.

BIRCH effectively manages large datasets by summarizing data into a tree structure, allowing for scalable clustering with reduced computational complexity.

2.3.3 Minibatches K-means

Mini-Batch K-Means is a variant of the K-Means algorithm designed to handle large datasets more efficiently. Here's how it works:

1. Initialization: Randomly select **K** initial centroids from the dataset.
2. Mini-Batch Iteration: Instead of using the entire dataset, process small, random subsets (mini-batches) of the data in each iteration.
3. Assignment: For each mini-batch, assign each data point to the nearest centroid based on Euclidean distance.
4. Update: Update the centroids by computing the mean of the data points assigned to each centroid within the mini-batch.
5. Iteration: Repeat the mini-batch processing and centroid updating steps for a fixed number of iterations or until convergence.

Mini-Batch K-Means reduces computational cost and memory usage by approximating the K-Means clustering with smaller, manageable data subsets, making it suitable for large-scale datasets.

2.3.4 Mean-Shift Clustering

Mean-Shift Clustering is a non-parametric clustering algorithm that does not require specifying the number of clusters in advance. The algorithm works as follows:

1. **Initialization**: Place a kernel window (e.g., a circular or spherical window) centered at each data point in the dataset.

2. **Mean Calculation**: For each data point, compute the mean of all points within the kernel window. This mean is used to update the position of the kernel window's center.

3. **Shift**: Move the kernel window's center to the computed mean position.

4. **Iteration**: Repeat the mean calculation and shifting steps until convergence, where the kernel centers no longer change significantly.

5. **Cluster Formation**: Once converged, points that converge to the same kernel center are assigned to the same cluster.

Mean-Shift Clustering identifies clusters by finding dense regions in the feature space, adapting to the data's distribution without needing predefined cluster numbers.

2.3.5 Affinity propagation

Affinity Propagation is a clustering algorithm that identifies exemplars, which are representative data points, without requiring the number of clusters to be specified in advance. The algorithm operates as follows:

1. **Initialization**: Initialize messages between data points. These messages include *responsibility* and *availability* messages that reflect the likelihood of one point being an exemplar and the suitability of one point being assigned to another point.

2. **Message Update**:

   - *Responsibility* ($r(i, k)$): Update the responsibility message from point i to candidate exemplar k, reflecting how well-suited k is to be the exemplar for i.
   - *Availability* ($a(i, k)$): Update the availability message indicating how appropriate it is for point k to be the exemplar for point i.

3. **Iteration**: Iterate the message updates until convergence, where the responsibility and availability messages stabilize.

4. **Cluster Formation**: After convergence, assign each data point to the cluster of its selected exemplar. The exemplars are the points that receive the highest availability and responsibility values.

Affinity Propagation finds clusters by exchanging messages between data points and identifying representative exemplars based on the input similarities, without requiring a predefined number of clusters.

2.3.6 Agglomerative Clustering

Agglomerative Clustering is a hierarchical clustering algorithm that builds a hierarchy of clusters by iteratively merging the closest pairs of clusters. The algorithm operates as follows:

1. **Initialization**: Start with each data point as its own individual cluster.

2. **Distance Calculation**: Compute the distance (or dissimilarity) between all pairs of clusters. The distance metric can be Euclidean distance or other measures depending on the linkage criterion.

3. **Merge Clusters**: Identify the pair of clusters that are closest (based on the distance metric) and merge them into a single cluster.

4. **Update Distances**: Update the distance matrix to reflect the distances between the newly formed cluster and the remaining clusters. The update method depends on the linkage criterion:

   - **Single Linkage**: Minimum distance between any pair of points in the two clusters.
   - **Complete Linkage**: Maximum distance between any pair of points in the two clusters.
   - **Average Linkage**: Average distance between all pairs of points in the two clusters.
   - **Ward's Linkage**: Minimizes the variance within clusters by merging clusters that result in the smallest increase in total within-cluster variance.

5. **Iteration**: Repeat the merge and update steps until all data points are merged into a single cluster or until a desired number of clusters is achieved.

6. **Cluster Formation**: The resulting hierarchy can be represented as a dendrogram, from which clusters can be extracted based on a chosen cut-off level.

Agglomerative Clustering constructs a hierarchy of clusters by progressively merging the closest pairs of clusters, providing a flexible and intuitive way to group data based on similarity.

2.3.7 Spectral Clustering

Spectral Clustering is a technique that uses the eigenvalues of a similarity matrix to perform dimensionality reduction before applying a clustering algorithm. It operates as follows:

1. **Similarity Matrix Construction**: Construct a similarity matrix S for the data points, where S_{ij} represents the similarity between data points i and j. Common similarity measures include Gaussian (RBF) kernel or cosine similarity.

2. **Graph Construction**: Construct a graph G from the similarity matrix, where each node represents a data point and edges represent the similarities between them.

3. **Laplacian Matrix Calculation**: Compute the Laplacian matrix L of the graph. There are several forms of the Laplacian matrix, including:

   - **Unnormalized Laplacian**: $L = D - S$, where D is the degree matrix.
   - **Normalized Laplacian**: $L_{sym} = I - D^{-1/2}SD^{-1/2}$ or $L_{rw} = I - D^{-1}S$, where D is the degree matrix.

4. **Eigen-decomposition**: Compute the eigenvalues and eigenvectors of the Laplacian matrix L. Select the k smallest eigenvalues and their corresponding eigenvectors to form the matrix U.

5. **Dimensionality Reduction**: Form a new representation of the data points using the k eigenvectors. This matrix U is used to reduce the dimensionality of the data.

6. **Clustering**: Apply a standard clustering algorithm, such as K-Means, to the rows of the matrix U to partition the data into k clusters.

Spectral Clustering leverages the properties of the graph Laplacian to capture the structure of the data, making it effective for identifying clusters that are not necessarily spherical or linearly separable.

3 Dataset overview

The dataset I chose (ml-latest-small)[2] describes 5-star rating and free-text tagging activity from MovieLens[3], a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.
Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.
The data are contained in the files `links.csv, movies.csv, ratings.csv and tags.csv`. For my research I didn't consider the `links.csv` file since the other 3 tables can be linked with the keys *movieId* and *userId*.

The selected dataset is a smaller version of the original Movielens dataset [4], which in a first trial caused me lot of trouble and computational issues due to my limited resources, so I decided to go for a reduced version of it which also makes data visualization easier without affecting the purpose of this study.

Movie information is contained in the file `movies.csv`. Each line of this file after the header row represents one movie, and has the following format: movieId,title,genres

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Figure 3: *portion of* `movies.csv` *file*

[2] `https://www.kaggle.com/datasets/shubhammehta21/movie-lens-small-latest-dataset`
[3] `https://movielens.org/`
[4] `https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset`

All **ratings** are contained in the file `ratings.csv`. Each line of this file after the header row represents one rating of one movie by one user, and has the following format: userId,movieId,rating,timestamp. The lines within this file are ordered first by userId, then, within user, by movieId. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

	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

Figure 4: *portion of* ***`ratings.csv`*** *file*

All **tags** are contained in the file 'tags.csv'. Each line of this file after the header row represents one tag applied to one movie by one user, and has the following format: userId,movieId,tag,timestamp The lines within this file are ordered first by userId, then, within user, by movieId. Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user.

	userId	movieId	tag
0	2	60756	funny
1	2	60756	Highly quotable
2	2	60756	will ferrell
3	2	89774	Boxing story
4	2	89774	MMA

Figure 5: *portion of* ***`tags.csv`*** *file*

Note that Timestamps have been removed as they are irrelevant for this study.

4 Data processing

4.1 Creation of data points

Following the paper, the clustering analysis focuses on **movie genres** and **movie tags** as clustering dimensions, developing in parallel this two aspects through all the phases of the study.
For every user of the dataset I calculated the average ratings for each movie genre and movie tag.

My first strategy followed strictly the paper approach and considered only **Top 3** favourite movie genres and tags for the clustering analysis in order to obtain more robust result and a better visualization, that would have been harder using more than 3 dimensions. So the created data points consist in the original dataset's users, characterized by their average rating for top 3 genres/tags.
Another intermediate step was to bias the dataset in order to obtain a more delimited subset of users by considering only the ones that have a positive rating for at least one of top 3 genres/tags. This made the dataset even easier to visualize.

Figure 6 shows the final datasets that was used to conduct the clustering analysis.

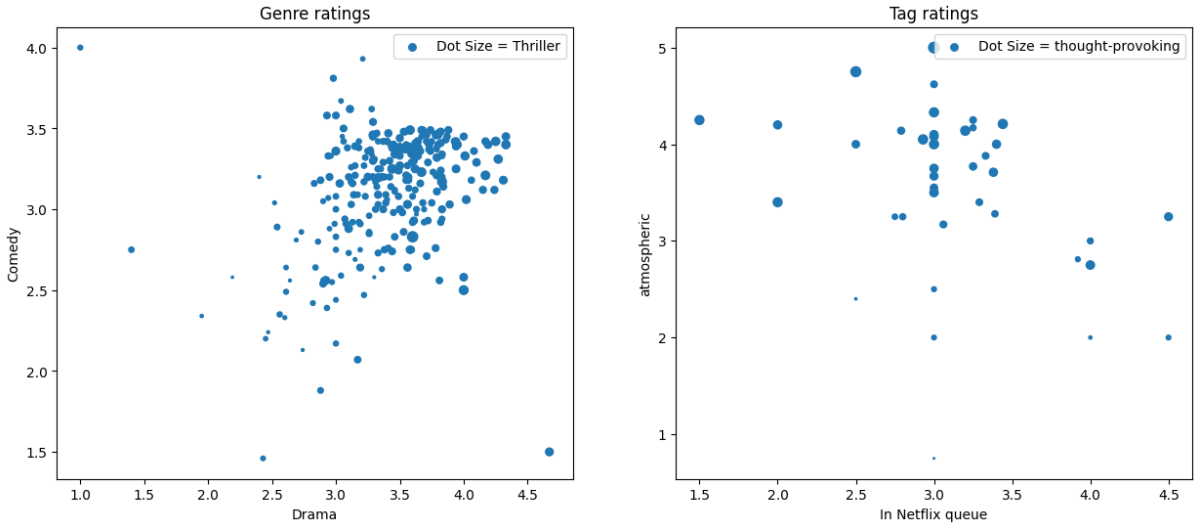

Figure 6: *Average ratings for Top 3 genres/tags*

The dataset depicted in the aforementioned image does not exhibit distinct separability into well-defined clusters and, as you will see, the 7 clustering techniques in some cases struggled to obtain clear and consistent results in terms of cluster's quality and visualization.

4.2 Training and Test set separation

Clustering analysis is an unsupervised learning technique, so there isn't a well defined idea of Ground Truth Labels, nevertheless the dataset was split into Training and Test set, following a classic 80-20 approach. The training set has been used to create and train cluster models, while test set was crucial to asses the recommendation quality.

5 Cluster analysis

Finally the two datasets were clustered using seven distinct algorithms. The resulting models and their corresponding recommendations were then evaluated based on various criteria.

5.1 Choosing optimal number of clusters K

Among the 7 selected clustering algorithms, some of them (K-Means, Birch, Minibatches K-means, Agglomerative) need to manually choose how many clusters should form the model while the others automatically choose the best number. Following the paper I adopted Silhouette score plot to determine the best K number of clusters.
Silhouette Score is a metric used to evaluate the quality of clustering in data analysis. It measures how similar an object is to its own cluster compared to other clusters, providing an indication of cluster cohesion and separation. Ranging from -1 to 1, higher Silhouette Scores suggest well-defined clusters, while lower scores indicate overlapping or poorly defined clusters. This metric is particularly useful in determining the optimal number of clusters by comparing scores across different clustering solutions, guiding the selection of the most appropriate cluster count for a given dataset.
Figure 7 shows an example of the Silhouette score analysis for K-Means with best genres/tags.

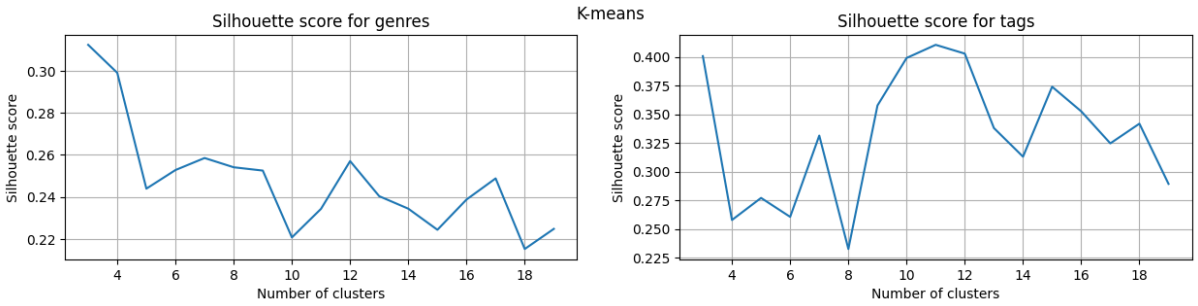

Figure 7: *Silhouette score for K-Means - 3-20 clusters*

It is easy to see that the best number K of clusters for genres is 3 clusters while for tags the number which provides optimal clustering is 11.

5.2 Model fitting

After having selected the optimal number of clusters for the algorithms which needed it, the models were fitted to the datasets.
As an example the plots of the clustered data resulting from two of the seven clustering algorithms are displayed below 8 9.

While the data separated in a fewer number of groups provides a discrete visualization of the clusters, as the number increases the distinct clusters not always are well defined and easy to distinguish.

5.3 Model evaluation and cluster quality

In this section, the evaluation of the clustering models is conducted using a comprehensive set of metrics to assess their performance and effectiveness. Specifically, the quality

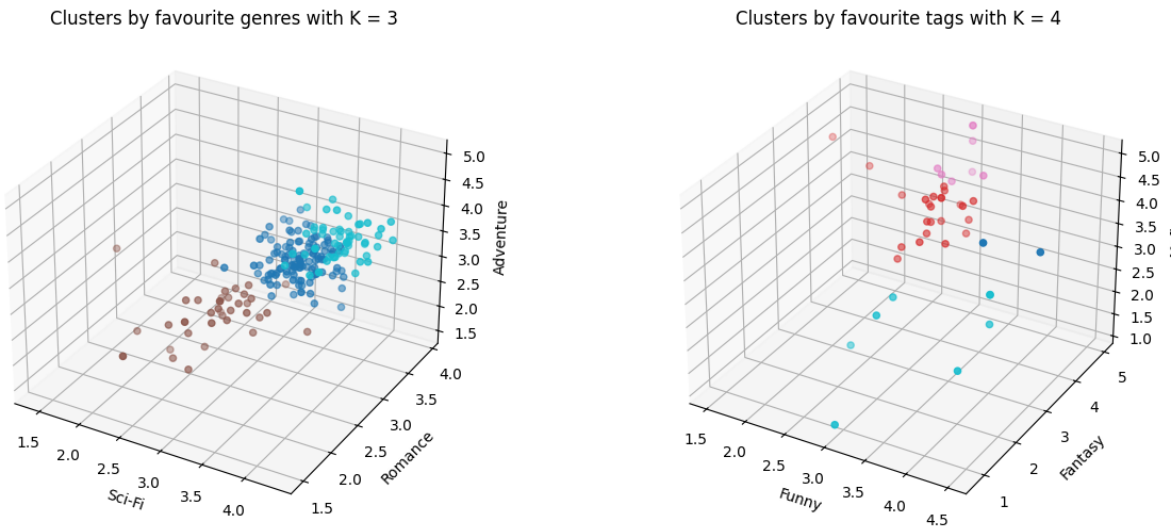

Figure 8: *Visualization of clustered data using Minibatches K-Means algorithm*

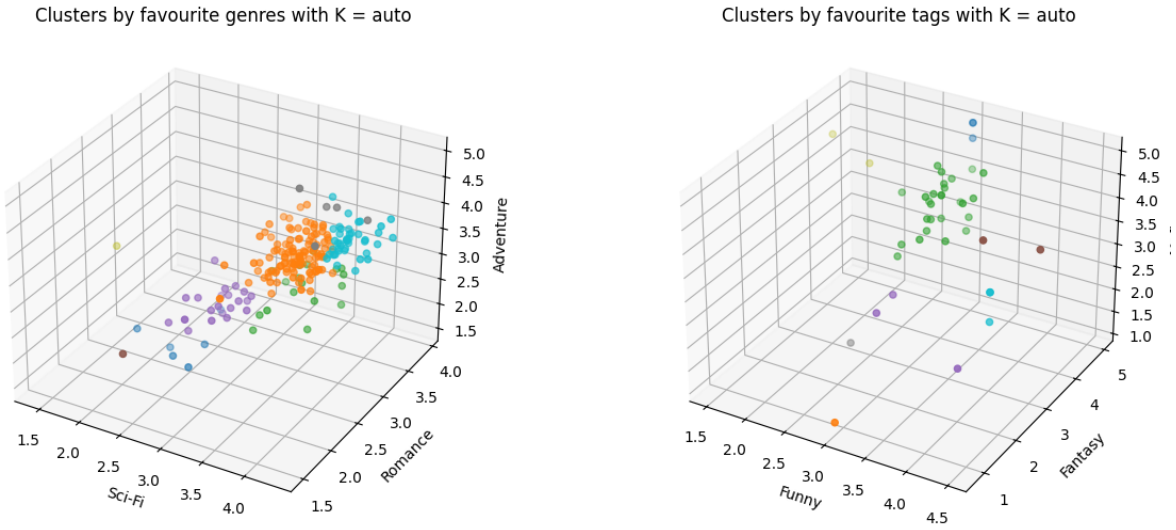

Figure 9: *Visualization of clustered data using Spectral clustering*

and distinctiveness of the clusters are measured using Dunn's Index, Average Similarity, Davies-Bouldin Index, and Calinski-Harabasz Index. Additionally, computational efficiency is considered by analyzing the computational time required by each algorithm. These metrics provide a robust framework for comparing and contrasting the clustering outcomes, ensuring a thorough evaluation of each model's performance.

5.3.1 Dunn's index

Dunn's Index is a metric used to evaluate the quality of clustering by measuring the ratio between the minimum inter-cluster distance and the maximum intra-cluster distance. It is designed to identify well-separated and compact clusters. A higher Dunn's Index indicates better clustering, with well-separated and dense clusters. Formally, Dunn's Index (D) is defined as:

$$D = \frac{\min_{1 \leq i < j \leq k} d(c_i, c_j)}{\max_{1 \leq l \leq k} \Delta(c_l)}$$

where:

- $d(c_i, c_j)$ is the distance between the centroids of clusters c_i and c_j,
- $\Delta(c_l)$ is the diameter of cluster c_l, defined as the maximum distance between any two points within the cluster,
- k is the number of clusters.

A higher value of Dunn's Index suggests that clusters are well-separated and internally compact, making it a useful metric for assessing clustering effectiveness.

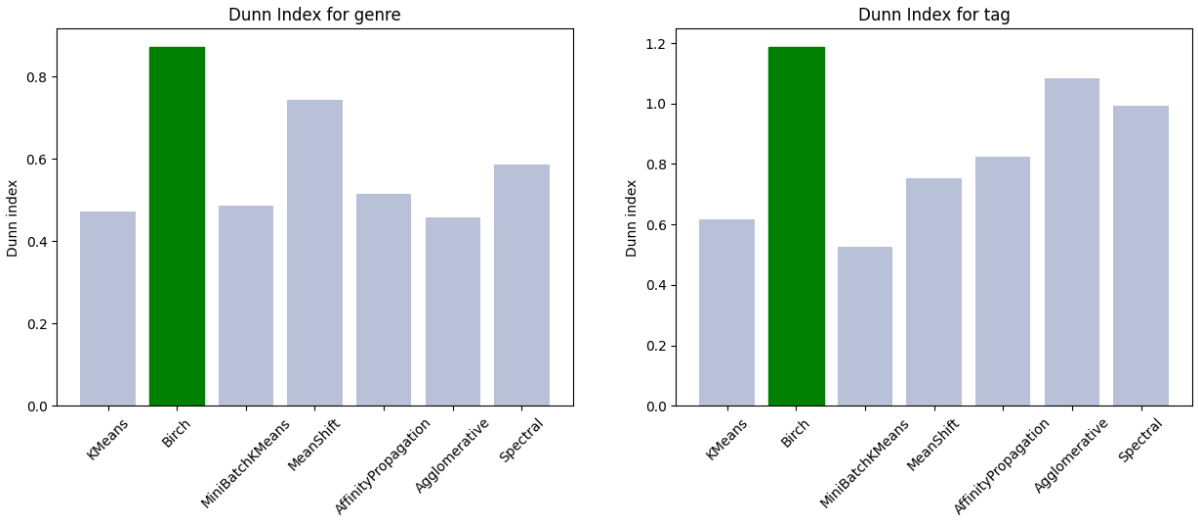

Figure 10: *Dunn's index results*

5.3.2 Average similarity

Average Similarity is a metric used to evaluate the internal consistency of clusters by measuring the average pairwise similarity between data points within the same cluster. This metric helps assess the cohesiveness of the clusters, with higher values indicating that points within a cluster are more similar to each other. Formally, Average Similarity (S) is defined as:

$$S = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|C_i|(|C_i| - 1)} \sum_{x,y \in C_i, x \neq y} \text{sim}(x, y)$$

where:

- k is the number of clusters,
- C_i is the set of points in cluster i,
- $|C_i|$ is the number of points in cluster i,
- $\text{sim}(x, y)$ is the similarity between points x and y.

Higher values of Average Similarity indicate more cohesive clusters, which are desirable in clustering results.

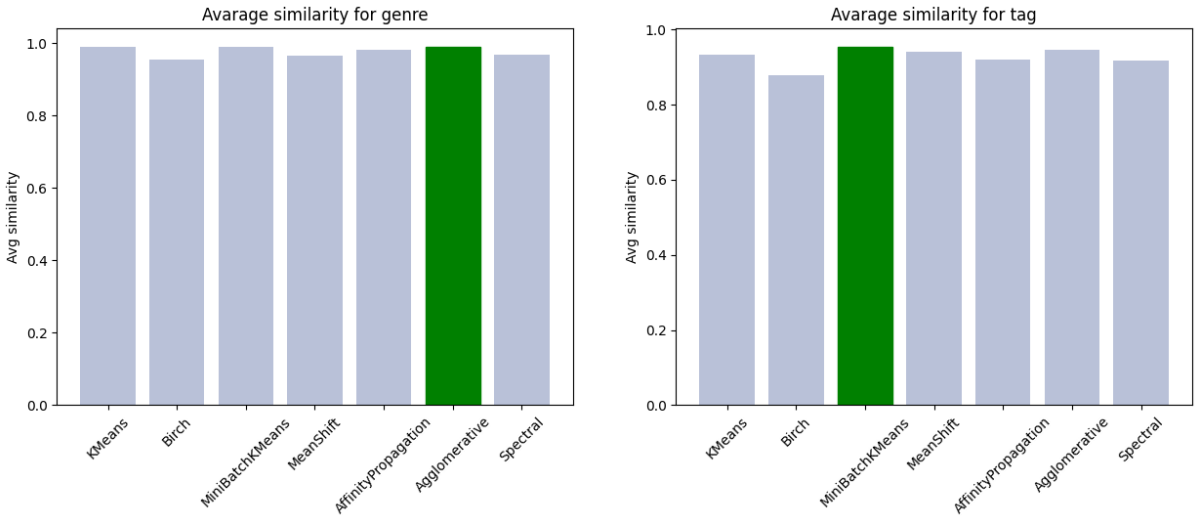

Figure 11: *Average similarity results*

5.3.3 Davies-Bouldin Index

The Davies-Bouldin Index (DBI) is a metric that evaluates clustering quality by assessing the average similarity ratio of each cluster with its most similar cluster. A lower Davies-Bouldin Index indicates better clustering, as it suggests that clusters are well-separated and distinct from each other. Formally, the Davies-Bouldin Index (DBI) is defined as:

$$DBI = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\Delta(C_i) + \Delta(C_j)}{d(C_i, C_j)} \right)$$

where:

- k is the number of clusters,
- $\Delta(C_i)$ is the average distance of all points in cluster i to the centroid of C_i,
- $d(C_i, C_j)$ is the distance between the centroids of clusters C_i and C_j.

A lower Davies-Bouldin Index indicates better clustering with well-separated clusters.

5.3.4 Calinski-Harabasz Index

The Calinski-Harabasz Index, also known as the Variance Ratio Criterion, evaluates the quality of clustering by comparing the ratio of the sum of between-cluster dispersion and within-cluster dispersion. A higher Calinski-Harabasz Index suggests better-defined clusters with distinct separation. Formally, the Calinski-Harabasz Index (CH) is defined as:

$$CH = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \times \frac{n - k}{k - 1}$$

where:

- $\mathrm{Tr}(B_k)$ is the trace of the between-cluster dispersion matrix,

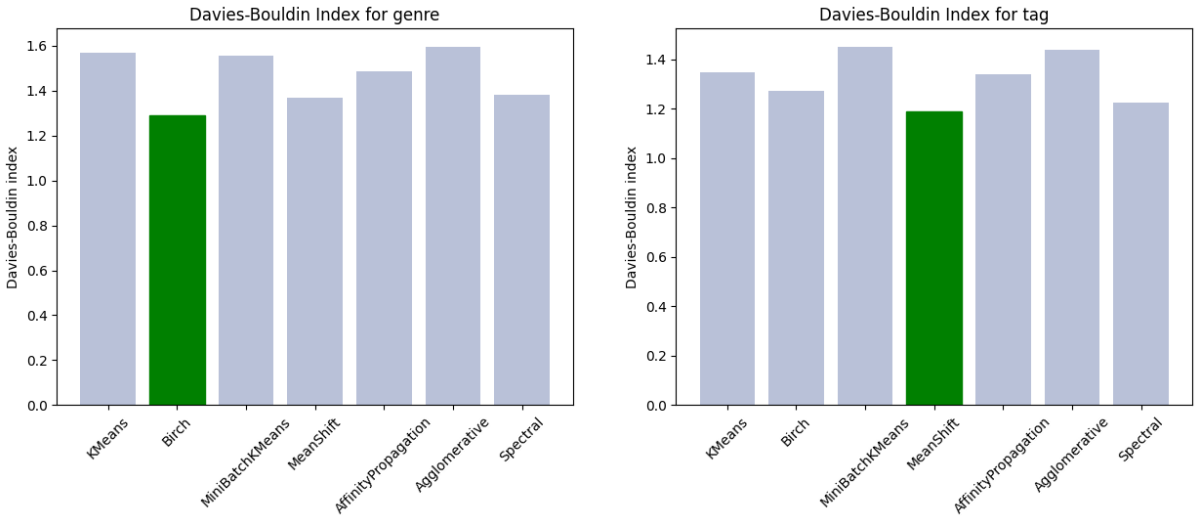

Figure 12: *Davies-Bouldin Index results*

- $\mathrm{Tr}(W_k)$ is the trace of the within-cluster dispersion matrix,
- n is the total number of data points,
- k is the number of clusters.

Higher values of the Calinski-Harabasz Index indicate better clustering performance with clear cluster distinctions.

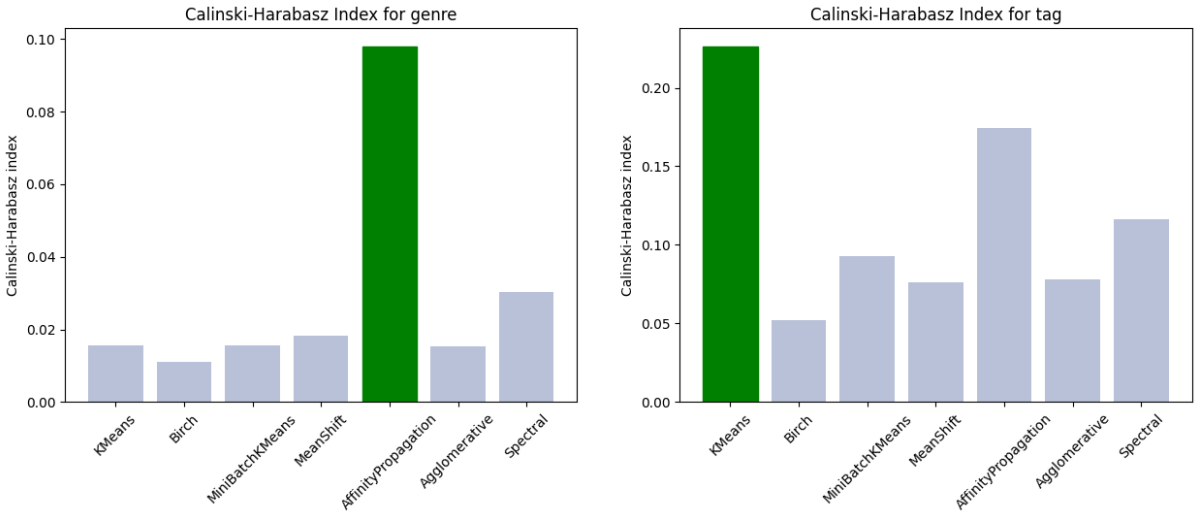

Figure 13: *Calinski-Harabasz Index results*

5.3.5 Computational time

Computational Time is a critical metric used to evaluate the efficiency of clustering algorithms. It measures the total time taken to fit a clustering model to a dataset. This metric is particularly important when dealing with large datasets or when the clustering algorithm has a high computational complexity.

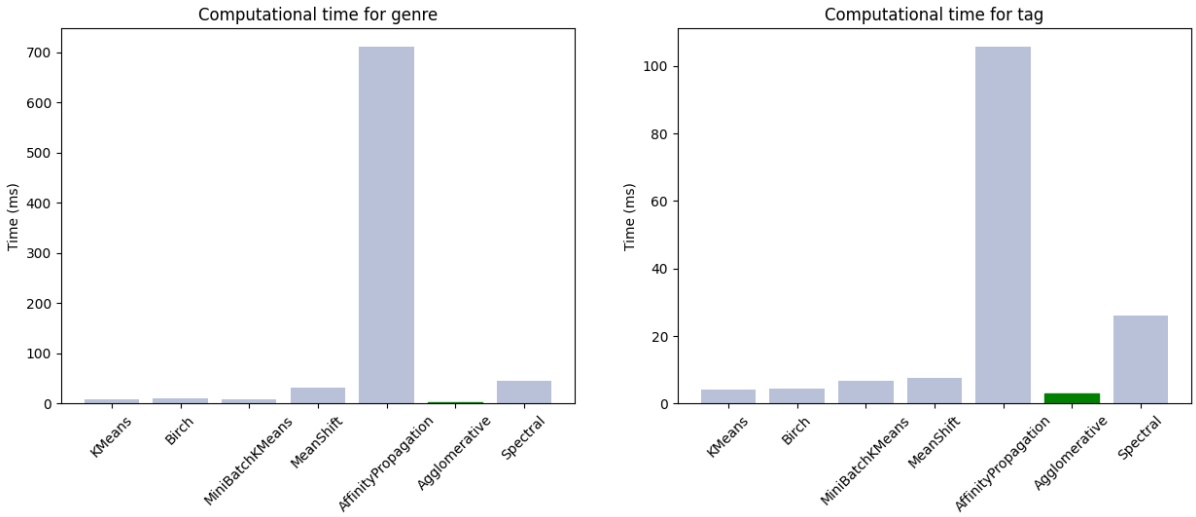

Figure 14: *Computational time results*

5.4 Results

- ***K-Means*** got intermediate performances for Dunn's index and Davies-Bouldin index, while it did better with average similarity, Calinski-Harabasz index and computational time.
- ***Birch*** showed great results for Dunn's index both for genres and tags, and had also the best scoring of davies bouldin index with genres. Computational time was also quite good while average similarity and Calinski-Harabasz index were a little below the average results.
- ***Minibatches K-Means*** performed great with Average similarity and Computational time while a little less with the other criteria, however showing acceptable results.
- ***Mean-shift*** did well with average similarity, got the best result with Davies-bouldin index with tags and also a decent result in terms of time consumption, while it did not so great with the other metrics.
- ***Affinity propagation*** outperformed all the other algorithms in terms of calinski-Harabasz index with genre and did also pretty good with the other metrics but certainly was the most time consuming algorithm among the 7 tested, resulting not suitable for large datasets, typical of movie recommender systems.
- ***Agglomerative clustering*** showed great performances in terms of average similarity with tags and also with computational time resulting in being the fastest algorithm. On the other end the other metric results were a bit below the average.
- ***Spectral clustering*** got appreciable results in almost all the scoring, but time consumption was not as good as its competitors, making it a sub-optimal option for this purpose.

6 A simple Movie Recommender System

6.1 The algorithm

The concept behind this recommender system, which leverages user clustering, is to utilize the similarity between users to generate personalized recommendations. Initially, a new user is assigned to the most similar existing cluster, with similarity determined by calculating the Euclidean distance between data points. Subsequently a function that predicts users ratings, produces top N recommended movies for a users by selecting first N highest rated movies.

```
def TOP_N_MOVIES(train_set, test_set, model,user_id, N, model_name):
    if(user_id in test_set['userId'].values):
        X = test_set.copy()
        X = X.reset_index()
        new_user = X[X['userId'] == user_id].index[0]
        cluster = predict_new_data(test_set, model, train_set, model_name)[
    new_user]
        cluster_ids = train_set[model.labels_ == cluster]['userId']
        user_rated_movies = ratings[ratings['userId'] == test_set.iloc[
    new_user]['userId']]
    elif(user_id in train_set['userId'].values):
        X = train_set.copy()
        X = X.reset_index()
        new_user = X[X['userId'] == user_id].index[0]
        cluster = model.labels_[new_user]
        cluster_ids = train_set[model.labels_ == cluster]['userId']
        user_rated_movies = ratings[ratings['userId'] == train_set.iloc[
    new_user]['userId']]
    else:
        print('User not found in Test or Train set, please try another user...
    ')
        return
    cluster_movies_not_rated = ratings[ratings['userId'].isin(cluster_ids) & ~
    ratings['movieId'].isin(user_rated_movies['movieId'])].drop('userId', axis
    =1).groupby('movieId').mean().sort_values('rating', ascending=False).head(N
    )
    for(index, _) in cluster_movies_not_rated.iterrows():
        cluster_movies_not_rated.at[index, 'title'] = movies[movies['movieId']
     == index]['title'].values[0]
    return cluster_movies_not_rated.drop('rating', axis=1).reset_index()
```

The core of this function is the user ratings prediction which takes place by using as prediction the user's cluster average ratings of each movie.
The function above provides the Top N Movies list15 given a model between the 14 obtained in the previous phase and a userId. Recommendations can be obtained both for training set users and test set ones while the recommendation evaluation was obviously made only with the test set to avoid over-fitting and to obtain unbiased results.

0	3246	Malcolm X (1992)
1	37733	History of Violence, A (2005)
2	2379	Police Academy 2: Their First Assignment (1985)
3	2380	Police Academy 3: Back in Training (1986)
4	2381	Police Academy 4: Citizens on Patrol (1987)
5	2382	Police Academy 5: Assignment: Miami Beach (1988)
6	142488	Spotlight (2015)
7	2463	Ruthless People (1986)
8	101765	Perfect Plan, A (Plan parfait, Un) (2012)
9	2490	Payback (1999)
10	96811	End of Watch (2012)
11	2792	Airplane II: The Sequel (1982)
12	2936	Sullivan's Travels (1941)
13	2968	Time Bandits (1981)
14	95167	Brave (2012)
15	3198	Papillon (1973)
16	305	Ready to Wear (Pret-A-Porter) (1994)
17	2303	Nashville (1975)
18	273	Mary Shelley's Frankenstein (Frankenstein) (1994)
19	267	Major Payne (1995)

Figure 15: *Example of Top 20 movies produced using Minibatches K-Means model with tags*

6.2 Recommender Evaluation

To assess the effectiveness of the recommendation system, a set of well-established metrics has been employed. These metrics provide a comprehensive evaluation of the system's performance by measuring its accuracy and reliability. The criteria used in this analysis are Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Precision.

6.2.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a fundamental metric that quantifies the average magnitude of the errors in predictions, without considering their direction. It is calculated as the average of the absolute differences between the predicted and actual values:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{1}$$

where $\hat{y}_i$ represents the predicted value and y_i represents the actual value. MAE provides a clear indication of the average prediction error, with lower values signifying better predictive accuracy.

6.2.2 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) measures the square root of the average squared differences between predicted and actual values:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{2}$$

RMSE is sensitive to large errors due to the squaring of the differences and is often preferred when large errors are particularly undesirable. It provides a measure of the standard deviation of the prediction errors, with lower values indicating a more accurate model.

6.2.3 Precision

Precision evaluates the proportion of relevant recommendations among the total number of recommendations made. It is calculated as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3}$$

where True Positives refers to the number of relevant items that were recommended, and False Positives refers to the number of irrelevant items that were recommended. High precision indicates that the majority of recommendations are relevant, thus enhancing the overall effectiveness of the recommendation system.

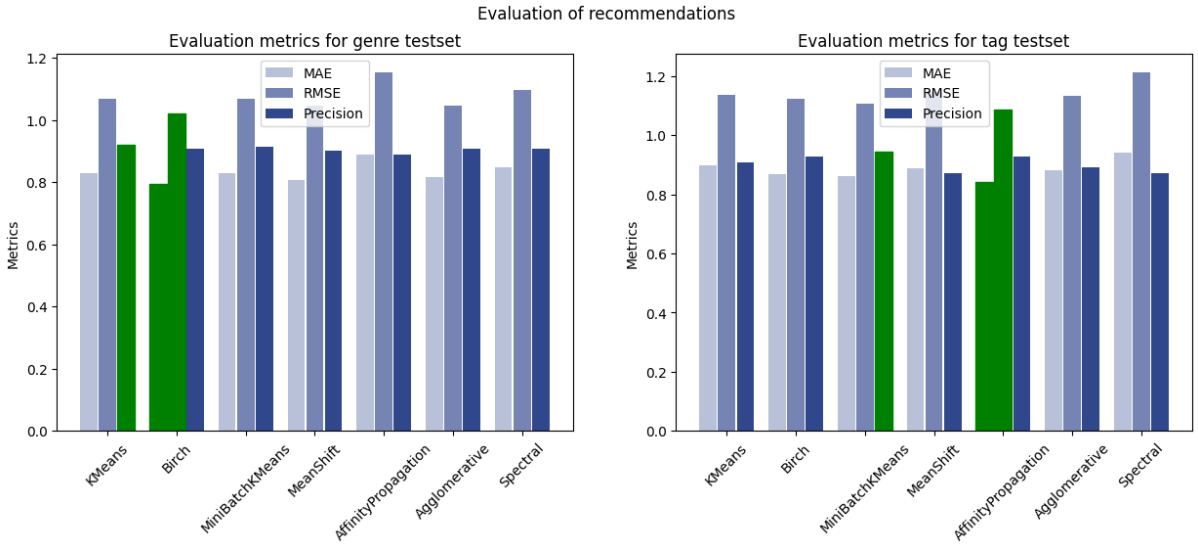

Figure 16: *Recommender evaluation results*

6.3 Results

Movie genres clustering showed similar results in terms of ***MAE*** for all the algorithms oscillating around a value of 0.8, and the same stood for tags clustering but with a slightly increment in the error which almost approached 0.9.
Also for ***RMSE*** all the algorithms performed similarly oscillating above the value of 1, seeing also in this case a slight increment of the error for tag clustering.
PRECISION over recommendation settled at a value around 90% for all the models which is a pretty decent result considering the simplicity of the recommendation logic and the use of a small dataset.
Overall, Birch had the lowest error values and K-Means the highest precision for genre clustering while Affinity propagation gained the lowest error values and Minibatches K-means got the highest precision for tag clustering.

Considering the aggregated performances of clustering and recommendation, the best clustering algorithm using movie's genres as clustering dimensions is Birch which has the best clustering results and also high recommendations quality.
For tags the result is not so clear since the best clustering algorithm is Agglomerative Clustering which has good clustering results but the recommendations quality is not the highest. On the other hand Mini-Batch K-Means has the highest recommendations quality. In this case I personally chose Minibatches K-means as the best tag based clustering method since the study focuses on Building a movie recommender system and I gave more importance to this aspect.

7 Adding more dimensions

After having found the best models using top 3 genres/tags, I decided to extend the original paper analysis by considering all 20 genres and top 20 tags, and then performing *dimensionality reduction* through the use of *PCA* to extract fundamental relations within the dataset features and to find out if Adding more dimensions would bring some benefit in the clustering and recommendation process. Then I evaluated the performance of the clustering algorithms using the same metrics as before and looked if any substantial improvement in the clustering and recommendation results came in.

7.1 Data preparation

The dataset of the first cluster analysis was extended to include all 20 movie genres and first 20 preferred tags. The Null values of unrated genres/tags were filled with the user's mean rating values.

7.2 *PCA* Analysis

Clustering data with 20 dimensions can introduce redundancy and noise, complicating the clustering process. Principal Component Analysis (PCA) is employed to reduce the feature space by transforming the original variables into a set of uncorrelated components that capture the most significant variance in the data. By selecting the principal components that retain at least 90% of the variance and that contribute most to the data's structure, PCA effectively reduces the dimensionality, enabling more efficient and

accurate clustering.
The following picture 17 shows the percentage of retained variance for the different number of principal components kept.

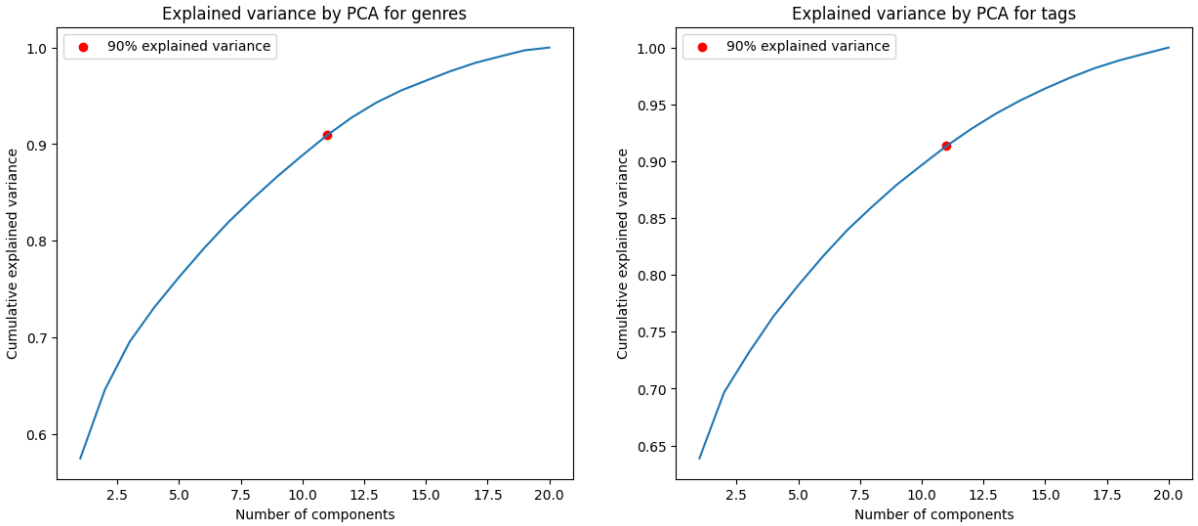

Figure 17: *Explained variance for different number of principal components*

From the regular shape of the above functions it can be noticed that the original features contribute almost equally to the data structure, with the *PCA analysis* resulting in a poor dimension reduction, keeping still 11 features out of 20.

From now on the very same analysis done before was conducted to find the optimal models for genre/tag clustering and movie recommendations using PCA.
For the sake of visualisation below 18 you can find an example of the first 3 principal components, grouped using Agglomerative clustering.

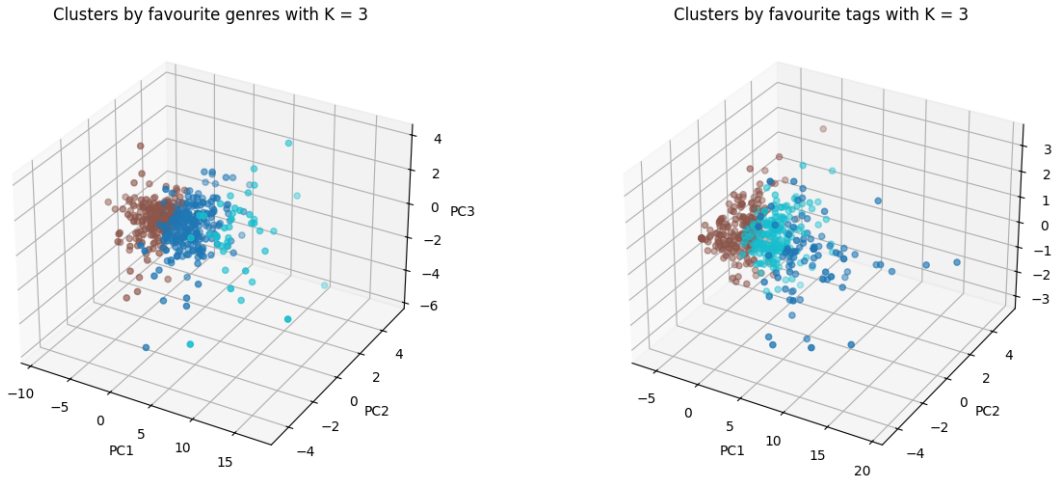

Figure 18: *First 3 principal components clustered with Agglomerative algorithm*

7.3 Results

Below 19 all the analysis results are shown.

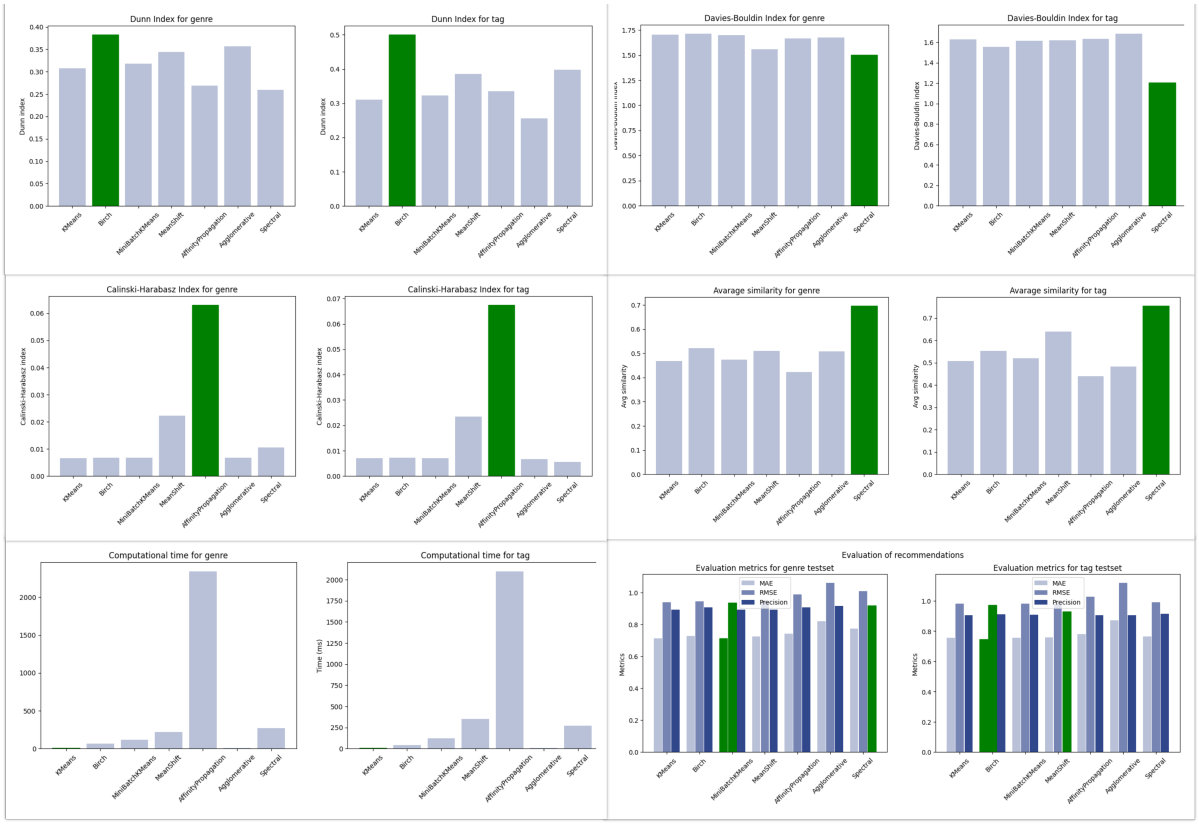

Figure 19: *PCA Analysis results*

Considering the overall performances for different metrics, the best clustering algorithm using movies's genres as clustering dimensions and PCA to reduce dimensionality, is Spectral Clustering, which had the best clustering results and also high recommendations quality.

For tags the result was not so clear since the best clustering algorithm was Spectral Clustering which had good clustering results and decent recommendations quality but Birch Clustering got the highest recommendations score. Also this time I weighted heavier recommendation quality over cluster robustness for the sake of the study resulting in Birch being elected best model for tag clustering.

Eventually I compared the best clustering algorithms found for the two strategies using the same metrics as before to see if the PCA approach brought substantial improvements or if there wasn't any benefit in adopting a more complex method.

	Model	Silhouette Score	Dunn Index	Davies-Bouldin Index	Calinski-Harabasz Index	Time (ms)	MAE	RMSE	Precision
0	Birch with 3 genres	0.494244	0.872285	1.291014	0.011011	9.737253	0.794414	1.021977	0.907692
1	MiniBatchKMeans with 3 tags	0.318345	0.486601	1.555922	0.015673	8.680105	0.860426	1.107188	0.945455
2	Spectral Clustering with PCA using genres	0.073591	0.259077	1.502277	0.010536	268.220901	0.847564	1.097574	0.907692
3	Birch with PCA using tags	0.306971	0.500901	1.553256	0.007192	39.945602	0.866573	1.123112	0.927273

Figure 20: *Numerical comparison of the two strategies*

As we can see above 20, the two strategies have comparable results for clustering quality, with the first approach having a slight advantage in terms of recommendation quality. This results, in addition to the high computational time, the low interpretability and the greater complexity of the Pca approach, makes the first strategy the optimal for building a movie recommender system in this specific settings. The analysis shows also that choosing movies tags as clustering dimensions provides a slightly worse clustering quality but a better recommendation quality compared to genres.

8 Conclusions

In the initial phases of this project I had a lot of trouble in understanding the paper I was assigned, because of the language that sometimes was cryptic and coloured with syntactical errors that made the path to the end tortuous. The superficiality of some explanations and the poor clarity of some results made me struggle to undertake the right way for my analysis. Eventually I hopefully managed to achieve some appreciable results by adopting some changes in the evaluation part and by extending the study with the use of *PCA*, even if it didn't reveal to be useful, at least in this specific settings, due to the structure and nature of the data.
It can be observed that *Birch* algorithm consistently achieved the best performance across both analyses, thereby providing a basis for meaningful comparison between the two studies.
My project attempts to reproduce a scientific paper not always in a faithful way, primarily due to computational limitations, a problematic understanding of the original results, and a topic certainly not trivial at all. The results achieved may not be astonishing and could also result somehow approximate as they only scrapes the surface of the recommender systems universe but I am still pretty happy with them considered what said above.