Title: Leveraging K-Means Clustering for Movie Recommendation Systems: A Feature-Reduction Approach Using IMDb Tags

Christian Schmitz, Lu Knoblich

I. Introduction

THE abundance of digital media today has turned the task of selecting a film to watch into a daunting exercise. Amidst this sea of choices, personalized movie recommendation systems provide a much-needed compass, guiding users towards films that align with their tastes and preferences. As a response to this growing demand, our project leverages machine learning, with a particular emphasis on clustering, to construct a robust movie recommendation model.

The Movie Lens Database (ML) serves as a comprehensive archive of movie metadata, covering films spanning different periods and genres along with ratings and tags given by individual users. This project focuses on utilizing this dataset, specifically the user-generated tags, to develop a predictive model. The genome scores provide a value that indicated the degree of relevance that exists between each tag and movie. These user tags encapsulate various aspects of the film, such as its genre, themes, narrative devices, and unique identifiers. While these tags are insightful, due to the human nature by which these tags were created, their sheer volume and diversity pose a challenge to the development of a manageable and efficient prediction model.

vvv I'll be back vvv

In order to solve the prediction problem the k-means clustering algorithm will be utilized. By interpreting the profuse tag space geometrically, k-means clustering presents an intuitive and efficient solution to the movie recommendation problem. This approach will be utilized to form 'genre spaces' or clusters, predicated on tag similarity, thereby laying the groundwork for movie predictions.

The goal of this project is to create a model for movie recommendations based on a user's previous watching history. The process involves preprocessing the ML database tags to reduce feature size, and subsequently applying k-means clustering. With this model, we aspire to provide effective guidance to users across the expansive cinematic landscape

II. Data Acquisition and Preprocessing

The selection and subsequent preprocessing of the dataset are crucial steps in the development of our recommendation system. This chapter provides a detailed examination of the chosen data.

A. Dataset

The MovieLens dataset , which is publicly accessible and provided by GroupLens, was utilized in this project. The dataset contains user ratings and a wide collection of user-generated tags from the IMDb dataset. Its relationship with IMDb's comprehensive movie metadata makes it a particularly suitable choice for our project.

The tags and their relevance to each movie represent different characteristics of them, such as genre, themes and narrative devices, among other categorical and descriptive identifiers.

The main dataset to be used for the characterisation of each movie consists of three columns: movieId, tagId and relevance.

For both the movieId and tagId values, there is a respective reference file connecting the Ids to the movie titles and tag strings.

```
==> ./ml-25m/genome-scores.csv <==
movieId,tagId,relevance
1,1,0.028749999999999998
1,2,0.023749999999999993
1,3,0.0625
1,4,0.07574999999999998
...

==> ./ml-25m/genome-tags.csv <==
tagId,tag
1,007
2,007 (series)
3,18th century
4,1920s
...

==> ./ml-25m/movies.csv <==
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Chi...
2,Jumanji (1995),Adventure|Children|Fanta...
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Ro...
...
```

The relevance values of the movies stem from the tags given by the users, which can be found as part of the dataset in the following form:

```
==> ./ml-25m/tags.csv <==
userId,movieId,tag,timestamp
3,260,classic,1439472355
3,260,sci-fi,1439472256
4,1732,dark comedy,1573943598
4,1732,great dialogue,1573943604
```

For the characterization of the individual users, the utilized data consisted of four columns: userId, movieId, rating, and timestamp.

- `userId`: Represents the unique identifier for each user.
- `movieId`: Denotes the unique identifier for each movie in the IMDb database.
- `rating`: Personal rating given by the user for the Movie.
- `timestamp`: Indicates the time (in UNIX timestamp format) at which the user tagged the movie.

```
==> ./ml-25m/ratings.csv <==
userId,movieId,rating,timestamp
2,60756,funny,1445714994
2,60756,Highly quotable,1445714996
2,60756,will ferrell,1445714992
2,89774,Boxing story,1445715207
2,89774,MMA,1445715200
```

This data extract shows user 2 tagging two movies (with IDs 60756 and 89774) with various tags, with each tagging action being logged with a corresponding timestamp.

An essential aspect to consider within the context of our project is the inherent variability of user-generated tags in the dataset. These tags are not uniformly distributed and, more critically, they exhibit considerable inconsistency and redundancy. For instance, a user might tag a film with the word ”zombi,” while another might use ”zombie,” and yet another might use ”zomby” - all intending to signify the same concept. This variation could extend beyond spelling inconsistencies to include differences in phrasing, abbreviations, semantics, and language use.

B. Feature Reduction through Preprocessing

The high dimensionality of the dataset, particularly the user tags, presents a significant challenge. The sheer number of tags (1128) is a considerable obstacle to the development of a recommendation system. The tags are also highly variable, with a large number of them being redundant or irrelevant. This variability is a significant source of noise, which can adversely impact the performance of our recommendation model. Hence, the necessity for preprocessing becomes apparent, with the aim to standardize and consolidate the tags, reducing the feature size and complexity.

The ability to distill this data into a format that is reduced but still retains its meaningful information is crucial for the complexity and performance of a recommendation model.
The Python script we utilized for data preprocessing follows this approach:

1) **Loading data:** We start by loading multiple CSV files from the ML dataset using the pandas library. We create dictionaries to map identifiers to their respective tags and movie titles, thereby preparing our data for the subsequent preprocessing steps.
2) **Identifying relevant tags:** To reduce dimensionality and retain the most valuable information, we first isolate ’relevant’ tags based on a mean relevance threshold. Tags, whose mean relevance across all movies exceeds this threshold, are considered ’relevant’ for our analysis. This operation significantly reduces the tag count, allowing us to focus on the most indicative tags.
3) **Finding synonyms:** After extracting relevant tags, we further explore correlations among these tags to identify potential synonyms — tags that essentially signify the same concept. We compute the correlation matrix and identify highly correlated tag pairs, which we consider as synonyms. This step again contributes to the reduction in feature size.
4) **Replacing synonyms:** In the final step, we replace the synonyms identified in the previous step with a single representative tag. We choose the most relevant tag from each group of synonyms as the representative tag. The representative tag will then assume the maximum value among the synonymous tags for each movie, and the synonymous tags are dropped from the dataset. This operation further reduces the dimensionality of our tag space.

After preprocessing, the data is significantly reduced in dimensionality, while retaining meaningful tags for analysis. Some representative examples of the processed tag groups are as follows:

```
["greed", "corruption", "morality"]
["satirical", "satire", "sarcasm", "irreverent"]
["crude humor", "hilarious", "stupid", "funny"]
["divorce", "marriage", "infidelity"]
```

Each array denotes a synonym group of tags that have been discerned and summarized through the preprocessing steps. The tags grouped within a single array were established as synonyms via correlation analysis, indicative of analogous thematic undertones. Taking the first array as an example, the tags ”greed,” ”corruption,” and ”morality” all encapsulate the primary themes of the associated film. This reduction in tag representation, through synonym identification, substantially lowers the dimensionality of the input space, contributing to a more efficient implementation of the k-means clustering algorithm.

In the finalized format of our dataset, the primary components consist of film titles and their corresponding tag group relevances. The film titles function as unique identifiers in the y-axis, effectively representing individual data points. In parallel, the x-axis is composed of the tag relevances resulting from our preprocessing steps. This matrix of film titles and their tag relevances are the input for the k-means clustering algorithm. This data arrangement allows the algorithm to observe patterns in tag relevances across different movies, and accordingly, cluster films that exhibit similar thematic elements.

III. K-Means Clustering

In the domain of unsupervised learning techniques, K-Means clustering stands out for its simplicity and efficiency. This algorithm, introduced by Stuart Lloyd and Edward W. Forgy in the mid-twentieth century, has been widely adopted for its ability to quickly segment unlabeled datasets into distinct groups, or clusters, of related instances [2].

A. Concept behind k-means

The idea behind K-Means clustering lies in the notion of 'centroid-based clustering'. The algorithm begins by defining 'k' centroids randomly in the feature space, where 'k' is the number of clusters to be identified. Each instance in the dataset is then assigned to the cluster whose centroid it is closest to, thereby creating 'k' groups of related instances.

The centroids are then updated based on the mean feature values of the instances in their respective clusters, and the instances are reassigned to the closest centroid once more. This process of updating centroids and reassigning instances continues iteratively until the centroids no longer change, indicating that the algorithm has identified the optimal grouping of instances into 'k' clusters. [2]

In the next sections, we will discuss how we implemented the K-Means algorithm in our study and highlight some of the results we obtained.

B. Solution Implementation

In this section, the implementation of the k-means clustering method will be discussed. The solution employs a k-means clustering algorithm for its implementation, utilizing the scikit-learn library. The Python script detailing this implementation can be found in the Appendix.

Upon initialization, the preprocessed data is loaded from a CSV file, identifying unique identifiers for each movie and considering the rest of the columns as tag relevances. The k-means clustering process is then applied to this dataset, which organizes movies into groups based on the similarities in their tag relevances.

The implementation further includes capabilities to predict the cluster of new instances, and retrieve the assigned cluster of a movie, contributing to the flexibility and applicability of the solution.

C. Clustering Results

TODO:: Show and discuss clustring results

IV. Model implementation for Recommendation Service

In order to provide a recommendation, an instance of a user need to be created in order to analyze the user's preferences. This instance is created by selecting a user from the database and extracting the list of movies that the user has seen along with the rating that the user gave to each movie. The rating is normalized between 0 and 1.

A. Characterization of the users' preference vector

The available data from a given User requires further processing in order to be used for the recommendation algorithm. The first step is to define the clusters to which each of the films belongs to, which is performed using the model described in the previous section.

The movies where grouped into the different clusters and the sum of ratings for each group was calculated. This provided a measure of the rating a user would most likely give to a movie belonging to a certain cluster relative to the other clusters. These values were then normalized so their sum was equal to 1. This resulted in a vector of length equal to the number of clusters representing the individual preferences of each user for each cluster based on their past behavior.

This method allows the characterization of each user in terms of the relevance of each cluster for each user. With the resulting data, it can be assumed that the user's preferences and the movies' characteristics are represented by the same vector space.

With this information, it can be assumed that the movies a user would like to see the most are the ones that belong to the cluster that the user has the highest preference for, followed by the movies that belong to the cluster that the user has the second highest preference for, and so on.

B. Recommendation algorithm

The recommendation algorithm is based on the assumption that the user's current preferences have not changed too greatly since the last time the user rated a movie. This assumption is reasonable since the user's preferences are based on the user's behavior over time.

This process is implemented as follows:

The first step is to define the cluster from which to take the movie recommendation from using a random number generator with the user's preference vector as the probability distribution This is followed by selecting a movie from the cluster that the user has not seen yet.

If the user has seen all the movies from the cluster, then a new random cluster is selected and the process is repeated.

V. Assessment of the Proposed Solution

The method used for this project was only one of the many possible approaches to the problem of movie recommendations. Many other methods exist, and each has its own advantages and disadvantages.

This section aims to provide a general assessment of the advantages and disadvantages of the proposed solution, as well as to discuss alternative paths that could have been taken and are worth exploring.

A. Advantages

The proposed solution was able to provide a personalized recommendation service for users based on their past behavior. The system was able to recommend movies that the user would most likely enjoy, based on the user's past behavior. The system was also able to provide recommendations for movies that the user has not seen yet, based on the user's preferences.

1) Scalability: One key advantage of this approach was its simplicity. The model was relatively easy to implement and required minimal computational resources. The use of K-Means clustering allowed us to group movies into categories based on their tag relevances, highlighting similar thematic elements.

B. Simplicity

Likewise, the preprocessing of the MovieLens dataset was straightforward. The dataset was already well-structured, so that the main task was to simplify the data removing unnecessary information and some level of redundancy due to the human nature of the data collection process and vague nature of the tags compared to the potential complexity of the movies.

1) Versatility: The model was able to accommodate both explicit and implicit data without requiring specific treatments for each, offering application versatility. The way in which the data was processed allowed the model to take advantage of semantic relation between the tags and their use, which could not have been achieved with hardcoded definitions of the tags and the synonyms (in the literally sense).

C. Disadvantages

Despite its merits, the proposed solution also has several drawbacks:

- it does not relate the behavior of one user to the behavior of other users, which could be useful to identify users with similar preferences and to provide recommendations based on the preferences of other users with similar watch behavior.

- it does not take into account all the metadata available for each movie, such as the cast, the director, the production company, the year of release, etc. This information could be used in cases where a user shown a clear preference towards a specific producer, actor or director, whose movies could encompass a wide range of genres and themes, rendering the clustering based solely on the tags less effective.

D. Alternative Paths That Could Have Been Taken and Are Worth Exploring

An alternative approach to this problem would be to use a different machine learning algorithm, such as Multilayer Perceptron (MLP) or Support Vector Machine (SVM). These algorithms are able to model more complex non-linear relationships between the data points and could can provide better results in terms of accuracy and precision. However, they require more computational resources, data and time to train the model, as well as an extensive hyperparameter tuning process to achieve the best results.

Another alternative approach would be to use a different dataset, such as the IMDb dataset, which contains more information about the movies, such as the cast, the director, the production company, the year of release, etc. This information could lead to more accurate recommendations, as it would allow the model to take into account relationships between the movies that are not captured by the tags alone.

Nevertheless, this would increase the required processing of the metadata and the comlexity of the model itself, possibly requiring a different machine learning algorithm to be used, since the different possible attributes contained in a movie's metadata would have to be interpreted and encoded in a way that the model can understand.

A final proposal, which would make use of the approach taken in this project, would be to combine output of different K-means clustering models using different walues for K. This would allow the model to take into account different levels of granularity when grouping the movies, which could lead to more accurate recommendations, possibly allowing the user to have the option to choose the level of granularity of the recommendations in order to avoid a positive feedback loop in which the user is only recommended movies that are very similar to the ones they have already seen and isn't able to discover new movies that they might enjoy.

VI. Conclusion

In conclusion, this project has successfully demonstrated a novel approach for movie recommendations using machine learning. The implementation of K-Means clustering on the MovieLens dataset offered a way to group films into categories based on their tag relevances, highlighting similar thematic elements. This formed the basis for our personalized recommendation system. While the system isn't revolutionary, it offers an alternative to more complex and resource intensive solutions, and provides chance to apply machine learning techniques to a real-world problem resulting interesting insights.

REFERENCES

[1] https://grouplens.org/datasets/movielens/ (Accessed: 14.05.23)
[2] Aurélien Géron, Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems, 2nd ed. O'Reilly Media, Inc., 2019.

APPENDIX

Code example 1: implementation of the k-means clustering algorythm

```
class KMCModel:
    def __init__(self, file):
        # Load data from CSV file
        self.data = pd.read_csv(file)
        self.movies = self.data['movieId']
        self.tag_relevances = self.data.drop('movieId', axis=1)

        # Create results directory if not exists
        if not os.path.exists("results"):
            os.makedirs("results")

    def plot_clusters(self, column, method_name):
        print(f'running plot_clusters')
        # Reduce dimensionality to 2D using PCA
        pca = PCA(n_components=2)
        pca_result = pca.fit_transform(self.tag_relevances)

        # Create a scatter plot of the two principal components with cluster labels as color
        plt.scatter(pca_result[:, 0], pca_result[:, 1], c=self.data[column], cmap='viridis')
        plt.xlabel('Principal Component 1')
        plt.ylabel('Principal Component 2')
        plt.title(f'{method_name} Clustering')
        plt.savefig(f'results/{method_name}_clustering_2.png')
        plt.clf()

    def predict(self,*args,**kwargs):
        return self.kmeansModel.predict(*args,**kwargs)

    def getFilmCategory(self, filmId):
        if filmId not in self.movies.values:
            return None
        # return self.data.loc[self.data['movieId']==filmId,'Category'].values[0]
        return self.data.loc[self.data['movieId']==filmId,'Category'].values[0]

    def kmeans_clustering(self, n=10, max_iter=100):
        print(f'running kmeans_clustering')
        # Apply KMeans Clustering
        kmeans = KMeans(n_clusters=n, init='k-means++', max_iter=max_iter, n_init=1)
        kmeans.fit(self.tag_relevances)
        self.data["Category"] = kmeans.predict(self.tag_relevances)
        self.plot_clusters("Category", "KMeans")
        self.kmeansModel = kmeans

    def save_results(self, file):
        print(f'running save_results')
        # Save results to a new CSV file
        self.data.to_csv(file, index=False)
```