Exploring Recommendation System Techniques on Movie Datasets - A Review

Marios Andreas Koutras
ECE UTH Volos
kmarios-@uth.gr

CONTENTS

LIST OF FIGURES

Exploring Recommendation System Techniques on Movie Datasets - A Review

***Abstract*—The constantly growing online content and data makes it difficult to select specific information from data pools. Personalization / Recommendation systems are important since they aim to reduce this complexity by helping users find content that suits their preferences and needs. Recommendation systems also help providers by increasing user engagement, satisfaction and loyalty.**

In this paper, we explore techniques of recommendation systems, such as collaborative filtering, content-based filtering and hybrid methods and their challenges. We demonstrate the use of libraries such as pandas, `numpy`, and `scikit-learn`, `nltk`, `PyTorch` to preprocess data, compute similarity measures, and generate recommendations.

I. Introduction

This paper delves into the intricacies of recommendation systems exploring various techniques such as collaborative filtering, content-based filtering and hybrid methods. Each of these techniques has its advantages and drawbacks which we will discuss in detail.

In the initial phase of our research, we focus on the utilization of the TMBD 5000 dataset within the context of content-based filtering. The dataset contains 4807 unique movies along with their relevant information. The primary objective of this segment of our study is to harness textual data and explore methodologies for its transformation into actionable information

Specifically: In this phase of the study we exploit the most relevant movie information included in the `movies.csv` and `credits.csv` datasets and create a Recommendation System which will allow users to filter out movies based on their specific preferences and queries (such as cast, directors, release dates) finding personalized and relevant content. In this dataset we will showcase how to preprocess text data, use appropriate libraries to extract text data, represent it by a numeric vector, calculate similarity metrics from these vectors and finally extract predictions from it. Finally the algorithm will either recommend movies similar to the user's chosen movie or movies with titles closer to the user's query.

The second part of the research focuses on the MovieLens dataset containing information on user ratings of movies with 610 unique users and 9724 unique movies. This kind of numeric data is perfect for training neural networks and extracting latent factors which influence the choices of the users and capture relationships between movies, but are invisible to the data analyst. The second part of the research focuses on exploiting these latent factors to recommend similar content.

We will use the TMDB dataset to propose a content-based filtering method to solve the cold start problem. The relevant information of a movie will be represented as a string which will be converted to a numeric vector, which will be passed as input to our cosine similarity function. The MovieLens dataset will be preprocessed differently depending on the methods applied. Initially, a sparse user-item matrix will be constructed, forming the basis for training diverse models such as Nearest Neighbors, Singular Value Decomposition (SVD), Non-Negative Matrix Factorization (NMF), and Truncated SVD. These models will be employed for rating predictions, utilizing frameworks like Keras and PyTorch. The resultant rating predictions will then be employed to generate recommendations and predict user preferences. Furthermore, neural networks implemented in PyTorch will be trained to extract movie embeddings. These embeddings will subsequently serve as inputs to clustering algorithms, such as KMeans, facilitating the grouping of movies based on the similarity of their latent factors. Finally, a hybrid matrix factorization technique, integrating SVD, PyTorch and KMeans, will be proposed as a comprehensive approach to enhance performance. This multi-faceted methodology aims to offer a holistic solution, addressing the cold start problem while concurrently optimizing the accuracy and diversity of movie recommendations.

II. Recommendation Systems and Terminology

Recommendation algorithms [1] are usually classified in the following categories: 1) Content based filtering recommendation algorithms [2], 2) Collaborative filtering recommendation algorithms [3] and 3) Hybrid recommendation algorithms [4].

A. Content Based Filtering

Content based recommendation algorithms are the earliest proposed in large scale applications. Such algorithms propose items with a high similarity between them based on their features or on the likes of the user. This method tries to extract characteristics and information such as browsing history records and user logs (ex. the history of viewed items until a user reached a specific item). Afterwards it creates *preference documents* and compares the features of the items to be recommended. The items with the greatest relevance are selected for recommendation [5]. The TF-IDF algorithm [6] is generally used to compute feature weights. The more often keyword k appears in document D the greater the importance of k. Additionally, the more often keyword k appears in other documents, the less the potential of keyword k to distinguish between documents. Assuming a number of documents N, a number of documents containing keyword K_i in the document set being n_i and f_{ij} being the number of times keyword K_i appears in document d_j, the term frequency TF_{ij} of K_i in

document d_j is defined as:

$$TF_{ij} = \frac{f_{ij}}{max_z f_{zj}} \tag{1}$$

z being the keyword appearing in document d_j.

The inverse frequency IDF_i of k_i in the document set is defined by:

$$IDF_i = \log \frac{N}{n_i} \tag{2}$$

The K-dimensional vectors $d_j = (w_{j1}, w_{j2}, \ldots, w_{jk})$ and $d_c = (w_{1c}, w_{2c}, \ldots, w_{kc})$ represent the project document and configuration document of user c, and each component in each vector is calculated by:

$$w_j = TF_j \cdot IDF_i = \frac{f_{ij}}{\max f_j} \cdot \log \frac{N}{n_i} \tag{3}$$

The similarity between the project document and the user is obtained by [7]:

$$sim(c, d_j) = cos(d_c, d_j) = \frac{\sum_{i=1}^{k} w_i c w_i j}{\sqrt{\sum_{i=1}^{k} w_i^2 c}\sqrt{\sum_{i=1}^{k} w_i^2 j}} \tag{4}$$

B. Collaborative Filtering

This method was proposed by Goldberc, Nicols, Oki, Tery in 1992 [8]. Collaborative filtering is categorized into 1) Memory-based algorithms [9] and 2) model-based algorithms [10]. Memory-based algorithms for recommendation systems mainly collect user ratings for items, create a (sparse) user-item matrix which contains the ratings of every user towards the items they have rated and finally use cosine similarity to calculate the similarity between users. Finally it recommends items to users based on the items of similar users. Assuming Iij the item set rated by users i and j and I_i, I_j denote the itemset scored by users i and j respectively. Then the similarity score between users i and j is:

$$sim = \frac{\sum_{c \in I_{ij}} (R_{i,c} - \bar{R}_i)(R_{j,c} - \bar{R}_j)}{\sqrt{\sum_{c \in I_i} (R_{i,c} - \bar{R}_i)^2}\sqrt{\sum_{c \in I_j} (R_{j,c} - \bar{R}_j)^2}} \tag{5}$$

where $R_{i,c}$ represents the rating of item c by user i and R_i, R_j represent the average rating of item c by users i and j. With the upscaling of services and the Internet, the increase of registered users and available products, memory based recommendation algorithms are bound by memory limitations or cold start problems. New users for example have not rated many (or any) items. To solve these limitations content-based nearest neighbor techniques are employed [11].

Model based recommendation algorithms are employed when the amount of data is too large for memory based approaches. Their strengths are in 1) Scalability: They can handle large datasets and make predictions quickly, 2) Improved Accuracy: Model-based recommendation systems have been shown to to provide more accurate recommendations compared to other techniques, 3) Sparsity: Model-based systems can handle sparse data by exploiting latent factors to predict the preference scores of unseen items. [12].

C. Hybrid Approach

The hybrid approach is the combination of more than one recommendation methods. For example, collaborative-filtering algorithms aim at finding the relationship between items, establish a user-item rating matrix based on the user's preferences and predict user scores for an unseen item. This approach however has the problem of cold start. Content-based recommendations on the other hand can solve this problem and reduce the impact of sparsity on the system's functionality [13], [14]. A hybrid algorithm attempts to keep the advantages of the ensemble of used algorithms and minimize their weaknesses.

D. Deep Learning for Recommender Systems

A not very traditional but upcoming and promising approach that deserves to be mentioned is the neural network and deep learning approach for recommendation generation. Deep learning has led to more precise and personalized recommendations and has significantly enhanced user experience [15].

In the collaborative filtering domain [16] applied deep neural collaborative filtering for personalized educational service recommendations. [17] used gated and attentive neural collaborative filtering for user-generated list recommendations. The data sparsity problem has been tackled with [18]. The researchers proposed a contextual-boosted deep neural collaborative-filtering (CDNC) model which uses user ratings along with item introductions to alleviate the cold start problem and provide transparent item recommendations.

Recommendation Systems heavily rely on embeddings. Embeddings capture the underlying patterns between users and items which are often passed as inputs to deep learning architectures / machine learning models. This is a byproduct of the training process on user behavior data [19]. More advanced recommendation systems may include contextual embeddings [20] allowing for more personalized recommendations. Deep learning algorithms have been researched to detect low dimensional embeddings for user-item features such as DeepFM [19], [21], [22], [23], [24]. DeepFM combines factorization algorithms with deep learning to model feature interactions.

Embeddings and deep learning have been revolutionary in the field of recommendation systems significantly improving results. Therefore deep learning has become a crucial component of state of the art recommendation systems.

III. Literature and Recent Works

One important similarity metric which finds use in Recommendation Systems is the cosine similarity metric. Research [25] proposes a such a system which involves sentiment classification and cosine similarity for personalized recommendations. Their proposed methodology involves several key components, including data collection from Kaggle and Wikipedia, data analysis to reduce the dataset to relevant features, combining features to create a similarity matrix, and applying Cosine Similarity to calculate the similarity of movie vectors. The sentiment analysis is conducted on collected reviews to determine the sentiments associated with

each movie. The system also incorporates API calls to obtain movie information and reviews, contributing to the generation of accurate and personalized recommendations. The implementation of the Movie Recommendation System using Cosine Similarity with Sentiment Analysis yielded promising results. The system successfully provided accurate and efficient movie recommendations, taking into account user preferences and sentiments. The integration of Cosine Similarity and sentiment analysis contributed to the system's ability to generate personalized recommendations, addressing the limitations of existing models and providing a comprehensive solution for movie enthusiasts.

[26] has implemented neural networks using the `PyTorch` framework to develop a Movie Recommendation System on the MovieLens dataset. They employed deep learning techniques for user based and item based filtering. Their proposed model exploited the latent factors, the *weights* of the embeddings, which capture relationships between users and movies as a byproduct of the training process. Additionally they have showcased the fit of the network using the RMSE and MAE metrics and the network performance for different activation functions. The proposed model had an accuracy of 80-85%.

The researchers in [27] discuss a proposed Ensemble of Deep Neural Networks (EDNN) method to predict Quality of Service (QoS) values for cloud services in order to recommend the best-suited service for customers. The paper argues that conventional machine learning algorithms and statistical analysis methods are not efficient in learning the complex correlation between data elements, while deep learning models have proven to be practical and precise. The proposed EDNN method is of the hybrid type, the fusion of neighborhood-based and neural network model-based methods. The output obtained from both the models is combined using another different neural network model. The results of this research show that the EDNN approach is better at learning latent features and outperforms conventional machine learning models marginally.

Research [28] revolves around Session-Based Recommendation Systems. It explains the baseline approaches in Recommender Systems such as the Simple Heuristics Method, Nearest Neighbors, Recurrent NNs and Factorization based methods. The study does not consider these algorithms as a baseline since they do not produce competitive results in session based recommendation scenarios. The researchers propose the Gated Recurrent Unit Neural Networks Model (GRU4REC). The GRU4REC is built using a variant of Recurrent Neural Networks (RNN), called Gated Recurrent Unit (GRU). The model considers each item in a session as a word and the catalog of items as the total vocabulary. The user's history of interactions with items is treated as a sequence of "words" in the session. The GRU4REC model uses a series of gates to selectively regulate and store relevant information over time, such as the user's interests or preferences. The model is optimized with a novel ranking-based loss function that aims to minimize the difference between the predicted and the actual next item that the user has interacted with. The proposed model has shown to be effective in generating session-based recommendations and achieving the state-of-the-art performance on several benchmark datasets.

The researchers in [20] propose a neural network framework called Textual and Contextual Embedding-based Neural Recommender (TCENR) for Point of Interest (POI) recommendations. They emphasize the limitation of traditional collaborative filtering techniques in regards to highly sparse environments like POI recommendation and their inability to generalize. The proposed model's performance was compared against seven baselines, HPF, NMF, Geo-SAGE, LCARS, Pace, DEEPCoNN and NeuMF. TCENR outperforms all state of the art baselines in terms of accuracy and MSE.

IV. Datasets: Features and Preprocessing

In this section we will present the datasets used and the applied preprocessing techniques.

A. TMDB 5000 Dataset

The TMBD 5000 dataset contains two integral files: `credits.csv` and `movies.csv`, both serving as repositories of comprehensive information for each movie within the dataset. The `credits.csv` file meticulously documents the cast and crew associated with each movie, while the `movies.csv` file consolidates all pertinent non-cast details, including genres, keywords, overviews, popularity metrics, and other relevant information. This dataset has been identified as particularly conducive to the implementation of content-based filtering, which will serve as our initial methodological approach. Significantly, this approach is proposed as an effective solution to mitigate the cold start problem, ensuring robust and personalized recommendations for users.

Upon merging the data based on the movie titles, we consolidate the information into a singular dataframe. The ensuing step involves the initiation of our preprocessing pipeline, presented in Fig. 1. This pipeline is designed to extract useful information from an initially disordered dataset. Specifically, we concatenate the content from the `keywords`, `overview`, `tagline`, and `title` columns into a unified, tokenized column. This consolidated column is subsequently represented as a vector, a key component in the computation of cosine similarity between movies. This methodology enhances the efficiency of content-based filtering, allowing for a more refined and cohesive measure of similarity between movies.

B. MovieLens Dataset

The MovieLens dataset comprises four files, of which we will focus on utilizing two: `ratings.csv` and `movies.csv`. In the context of our neural network rating prediction, we partition 80% of this dataset for training purposes, reserving the remaining 20% for testing and validation. Conversely, for our KNN and factorization techniques, we utilize the dataset in its entirety. The consolidated dataset, post-merging movies and ratings, is configured to include the columns `userId`, `movieId`, `rating`, and `title`. A critical aspect of preprocessing involves the encoding of user and movie values to rectify label inconsistencies. This step is particularly vital for the subsequent Keras review prediction

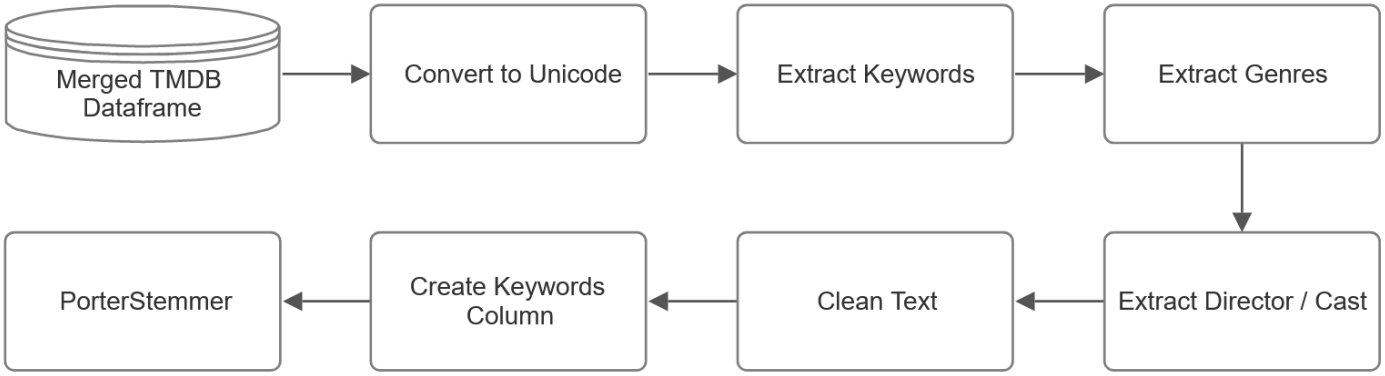


Fig. 1. TMDB Processing Pipeline

model. For our PyTorch models, we leverage the data formatting classes provided by the library, specifically `Dataset` and `DataLoader`. As depicted in Fig. 2, the preprocessing pipeline for the MovieLens dataset encapsulates these essential steps, ensuring the integrity and uniformity of the dataset for subsequent model training and evaluation.

For datasets and code management, detailed instructions will be given in the end of the report.

V. Methodology

In this section we will give the user a brief overview of the algorithms used, highlighting the key points of each.

A. Matrix Factorization Techniques

Matrix factorization is a technique for decomposing a matrix into two lower dimensional matrices. A user matrix and an item matrix. The objective of such algorithms is to predict a user's rating on items they have not interacted with. Matrix factorization algorithms are able to identify latent structures in the sparse user-item interaction matrices allowing more accurate recommendations.

1) Singular Value Decomposition: SVD was first used by [29] in 1998 in recommendation service. SVD is applicable to any rectangular matrix R. Assuming m users and n items then the score then $R \in \mathbb{R}^{m \times n}$. This matrix is decomposed as $R = U \cdot \Sigma \cdot V^T$. Then:

- R is a $m \times n$ user item matrix
- U us a $m \times r$ matrix where R the rank of R. The columns are called the *left singular vectors of* R
- Σ is an $r \times r$ diagonal matrix of the singular values of R, arranged in descending order. They measure the importance of each factor.
- V^T is the transpose of V. Its columns are the right singular values of R

2) Truncated SVD: TSVD is a variation of SVD where only the first k singular values and their corresponding vectors are kept, the rest are discarded. SVD decomposes a matrix into three matrices while TSVD retains only the most important components, reducing the dimensionality of the decomposition.

3) Non-Negative Matrix Factorization: NMF [30], [31], has been gradually developed over the years. The objective is similar to SVD but this technique is applied to matrices with non negative elements. Assuming m users and n items the score matrix $V \in \mathbb{R}^{m \times n}$ is decomposed as $V = W \cdot H$. Then:

- W is a $m \times k$ user (or feature) matrix. Each row corresponds to a user and each row corresponds to a latent feature.
- H is a $k \times n$ item (or coefficient) matrix for latent features. Each row corresponds to a latent feature and each column to an item.

B. Clustering Algorithms

1) Nearest Neighbors: The unsupervised KNN [32] algorithm was introduced by Evelyn Fix and Joseph Hodges in 1951 [33], [34], [35] and is mostly used as a classification algorithm. For a new input, the K-nearest neighbors are calculated and the majority of the neighboring data decides the class of the new input. This algorithm is instrumental in identifying proximate data points within a dataset without the reliance on labeled information. Leveraging distance metrics, such as Euclidean distance, KNN systematically locates the k-nearest neighbors of a given data point or all points within a specified radius. Specifically: Assuming a set **D** of training samples $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, the algorithm calculates the distance (or similarity) between every sample $z = (x_0, y_0)$ and every training sample $(x, y) \in D$ in order to determine the list of closest neighbors D_z. When the list of closest neighbors is computed the sample is categorized depending on the majority of closest neighbors

$$y' = \underset{u}{\operatorname{argmax}} = \sum_{(x_i, y_i) \in D} I(u = y_i) \tag{6}$$

A high level summary of the method is given by Algorithm 1 [36].

2) KMeans Clustering: In the context of unsupervised machine learning, the KMeans algorithm is a pivotal tool with notable applications in scientific research and recommendation systems. It was introduced by MacQueen in 1967 and was used to solve the *problem of the well-known cluster* [37]. The

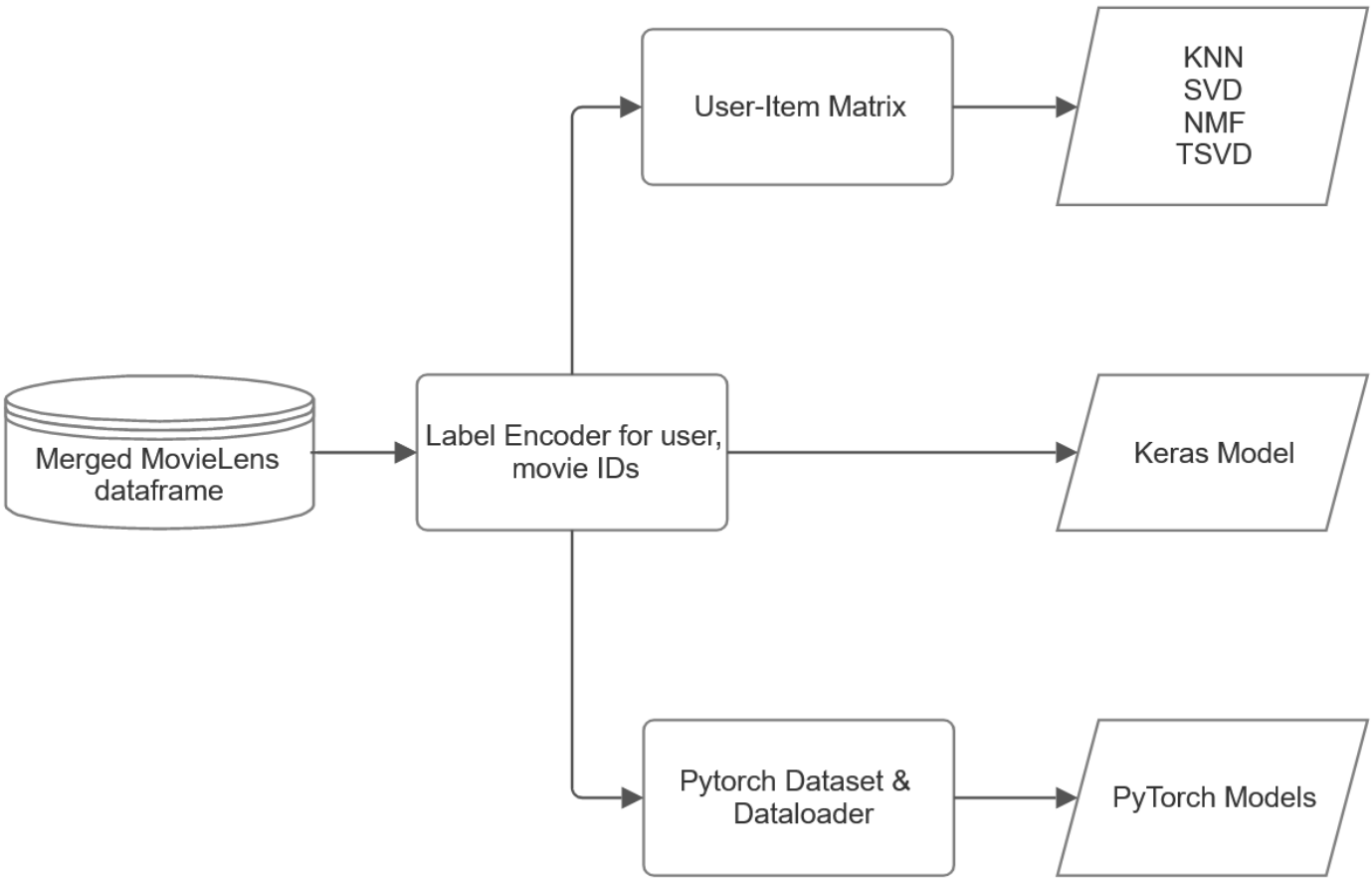


Fig. 2. MovieLens Processing Pipeline

Algorithm 1: Nearest Neighbors

Data: Input data X, Number of neighbors k
Result: Distances, Indices
Initialize NearestNeighbors with k;
Fit NearestNeighbors model with input data X;
for *each query point* q **do**
    Find k-nearest neighbors of q using the trained model;
return Distances, Indices;

K-means clustering algorithm is an iterative algorithm that divides a group of n datasets into k non-overlapping subgroups ($k \leq n$) based on the mean distance from the centroid. The algorithm starts by randomly selecting k points as the initial centroids. Then, each data point x_i is assigned to the nearest centroid c_j using the Euclidean distance metric:

$$d(x_i, c_j) = \sqrt{\sum_{i=1}^{n} (x_i - c_j)^2} \tag{7}$$

After all points are assigned, the centroids are updated by calculating the mean of all points in the cluster:

$$c_j = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{8}$$

The process of assignment and update is repeated until the centroids do not change significantly or a maximum number of iterations is reached.

The resulting cluster assignments furnish insights into inherent structures within the data, aiding researchers in uncovering patterns and relationships. This clustering mechanism, encapsulated within KMeans, thereby contributes significantly to the unsupervised exploration and understanding of complex datasets in scientific investigations. A high level summary of the method is given by Algorithm 2.

C. Deep Learning for embedding extraction

In this research, we propose novel deep learning models that concentrate on the latent attributes inherent in the dataset. The primary objective is to discover the implicit associations between users and items. A significant portion of this paper is dedicated to an in-depth examination of item embeddings and their utility in facilitating precise predictions. Our focus

Algorithm 2: KMeans Clustering

Data: Input data X, Number of clusters K
Result: Clusters, Centroids
Initialize K centroids randomly;
repeat
  for *each data point x in X* **do**
    Assign x to the cluster with the nearest centroid;
  for *each cluster C* **do**
    $new\_centroid \leftarrow$ mean of all data points in C;
    Update centroid of C to $new\_centroid$;
until *convergence (centroids do not change significantly)*;
return Clusters, Centroids;

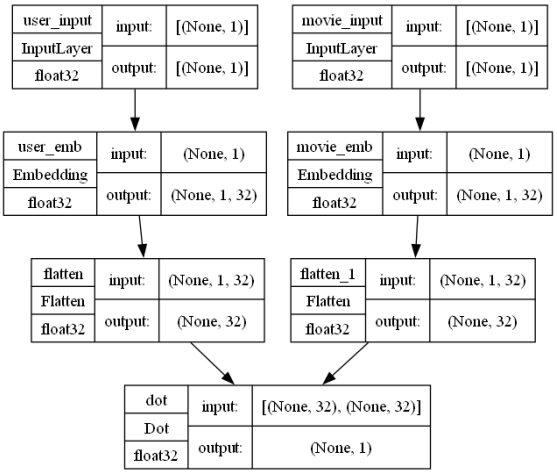


Fig. 3. Proposed Keras Model

is intended to contribute to the advancement of the field and provide valuable insights for future research.

1) Proposed Keras Model: Our first deep learning attempt in collaborative filtering will be the architectural representation of the Keras model is shown in Fig. 3. Let u denote a user, m denote a movie, and r_{um} signify the rating assigned by user u to movie m. The chosen embedding size is denoted as emb_size, serving as a parameter for the definition of the Embedding Layer. Subsequently, the model incorporates a Dot Product Layer, expressed as:

$$dot\_prod(u, m) = dot(user\_emb(u), movie\_emb(m)) \quad (9)$$

In a summary, the Keras embedding model computes the dot product between user and movie embeddings, providing the predicted user rating for a given movie. The model optimization process employs the widely utilized Adam optimizer.

2) Proposed PyTorch Model: Similar to the Keras model, our PyTorch model implements a collaborative filtering recommendation system using a more complex neural network architecture. The network model is designed to capture latent representations between users and items to predict ratings for

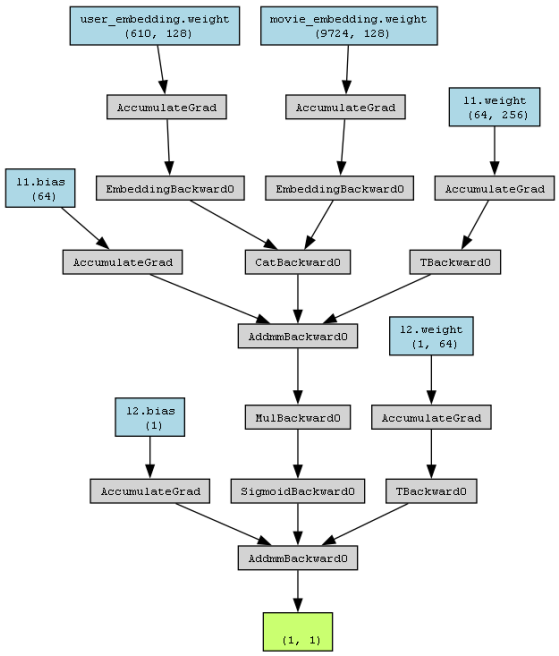


Fig. 4. Proposed PyTorch Model

unseen items. The model representation can be found in Fig. 4.

The dataset is represented as a class `ratings_dataset`, which inherits from PyTorch's `Dataset` class. This class is responsible for encoding user and movie IDs and preparing the data for training and testing. Assuming $\mathcal{U}$: Set of users, $\mathcal{M}$: Set of movies, $\mathcal{R}$: Set of ratings, u_i = user_encoder(u_i), m_j = movie_encoder(m_j), each slice of the dataset is an object of $(u_i, m_j, r_{ij}) \in \mathcal{U} \times \mathcal{M} \times \mathcal{R}$. The dataset consists of triples (u_i, m_j, r_{ij}) where r_{ij} is the rating given by user u_i to movie m_j.

The neural network model is defined as `rating_prediction_nn`. It learns embeddings for users and movies, concatenates them, and passes them through two linear layers with a non-linear activation in between. If $E_u : \mathcal{U} \to \mathbb{R}^d$ and $E_m : \mathcal{M} \to \mathbb{R}^d$, then

$$P(u_i, m_j) = \sigma(\text{Linear}(E_u(u_i), E_m(m_j))) \quad (10)$$

DataLoader objects (`ratings_train_loader` and `ratings_test_loader`) are created to efficiently iterate through batches of data during training and testing.

3) PyTorch - SVD: Ensemble Factorization Model: Our matrix factorization PyTorch model again computes the dot product in a similar way to (9). Given a user-item rating dataset, the class `ratings_dataset` is designed to preprocess the data for matrix factorization. It is represented by matrices $\mathbf{U} \in \mathbb{R}^{n_{\text{users}} \times k}$ and $\mathbf{M} \in \mathbb{R}^{n_{\text{movies}} \times k}$, where k is the number of latent factors. In our training process we use the MSELoss function and, again, the Adam optimization.

Finally, after training we perform K-Means Clustering on the movie Embeddings. The logic behind this is that, since the model has captured the underlying latent structure of movie

similarity as a byproduct of the training process, we can now cluster our movies together by their embeddings.

This algorithm combines matrix factorization and K-Means clustering for movie recommendation, providing both personalized predictions and grouping movies into clusters based on their learned embeddings. The combination aims to enhance the interpretability and diversity of recommendations.

VI. Results

A. Cold Start Content Based Models

As mentioned before, it would be difficult for a user item matrix of extreme sparsity to provide accurate results with collaborative filtering techniques. It would therefore be reasonable to employ content based filtering to alleviate this problem.

After cleaning and preparing the TMDB dataset. We begin by creating smaller *helper functions* that will eventually be combined to create one final prediction function. Such functions are:

1) `recommend_by_column()`: Recommends movies sorted by rating or popularity. Popularity is a more accurate metric and we will be using that from now on.
2) `recommend_by_director()`: Recommends movies based on their director.
3) `recommend_by_cast()`: Recommends movies based on their cast.
4) `recommend_by_genre()`: Recommends movies depending on the selected genres.
5) `recommend_by_date()`: Recommends movies based on date.
6) `recommend_similar_movies()`: Recommends movies that are similar depending on the cosine similarity metric in eq. 4.
7) `recommend()`: Final function incorporating the functionality of all the above.

During our preprocessing of the data we convert all text to unicode and merge the `credits.csv` and `movies.csv` dataframes. Then, using the proper functionality from the `ast` (Abstract Syntax Trees) library we can convert the textual data of the dataframe columns into dictionaries and extract all relevant information from them. The *keywords*, *overview*, *tagline*, *title* columns of the dataframe are merged together into one whitespace separated string column called *keywords*. After this preprocessing is complete, we *stem* the keywords with the Porter Stemming algorithm [38], introduced by M.F. Porter in 1980 [39]. This technique reduces words to their *base* or *root* form by removing suffixes (ex. The words "runner", "running", "run" all get stemmed to "run"). Finally the *keywords* are tokenized using `CountVectorizer` from `sklearn`, resulting a matrix of vectors, each representing a movie. The cosine similarity will be calculated using this matrix as input.

We can evaluate the functionality of the algorithm by testing different movies and seeing if the results make sense. For example the most similar movies of *Iron Man* will be high in similarity with *Iron Man 2*, *The Avengers* etc. which are all movies by Marvel. Movies like *Pirates of the Caribbean: At World's End* will correctly recommend more movies of the same franchise along with other titles related to pirates. The system's predictions make sense. Our final recommend function is robust and can work even with sparsity of user input. In our example we can find the "Iron Man" movie without the name, just by providing the correct input using the `cast` and `genres`. The recommendation system also provides accurate results when we are looking for substrings of a movie title (ex. the query "*bourne*" will correctly return the Jason Bourne movies). The same technique is applied to the MovieLens dataset using the `brute` algorithm and `cosine` metric on the `NearestNeighbors` class from `sklearn` with the user rating data. Both applications provide reasonable and satisfactory recommendations when a user doesn't have enough data to apply collaborative filtering.

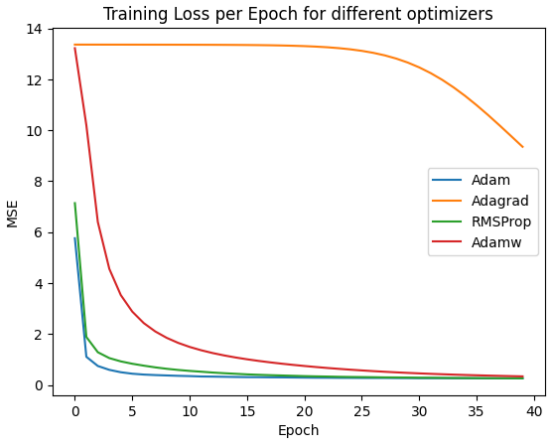


Fig. 5. Different Keras Optimizers for our training

B. Deep Learning Models

In our MovieLens dataset we begin by numerically encoding the values of movies and users using `LabelEncoder()`. 80% of the dataset will be used in the training process. We implement two architectures in different frameworks, one in Keras and one in PyTorch. Then we build our Keras model as described in Fig. 3 and V-C1 and our PyTorch model as described in Fig. 4 and V-C2. The objective of those models is to purely predict the rating of a user towards an unseen movie. The architecture of the two models differs in the sense that the Keras model purely computes the dot product while the PyTorch model is a deep learning model with activation functions to introduce non-linearity. We have selected the Adam optimizer because it converges faster and to smaller MSE values (Fig. 5). Our Keras model shows a good fit on the results with an average $MSE = 1.35$, average $RMSE = 1.16$ and average $MAE = 0.81$ on our 20% testing data. The PyTorch model again shows a good fit with the average losses being $MSE = 0.017$, $RMSE = 0.13$ and $MAE = 0.11$ for our testing set.

By extracting the movie embeddings from our model and further reducing their dimensionality with PCA (Fig. 7) and

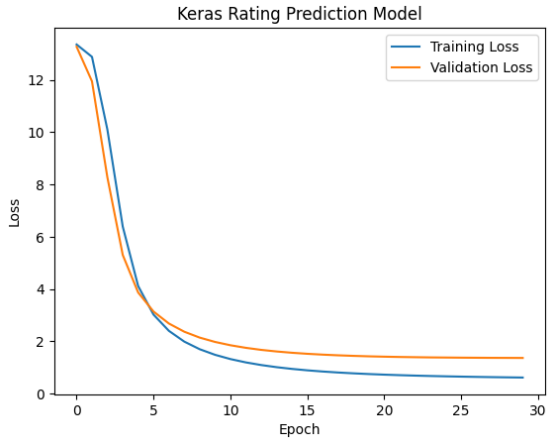


Fig. 6. Train and Validation MSE per epoch with Adam

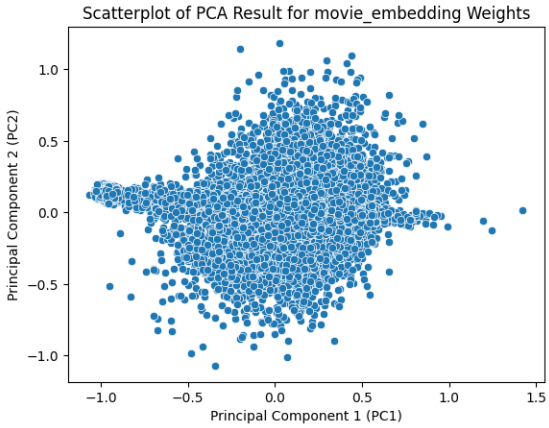


Fig. 7. PCA embedding dimensionality reduction

TSNE (Fig. 8) we can visualize how the embeddings help separate movie types.

In Fig. 6 we can see the losses per epoch for the training and testing set. Finally the proposed models can be used to make predictions.

C. Factorization Models

Factorization models serve as fundamental components in constructing our ensemble model. We present the application of Singular Value Decomposition (SVD), Non-negative Matrix Factorization (NMF), and Truncated Singular Value Decomposition (TSVD). Notably, these models exhibit comparable performance across our evaluation metrics. In our final ensemble model, we opt for the utilization of the SVD method due to its demonstrated efficacy.

The numerical encoding of user IDs and movie IDs, combined with their corresponding ratings, facilitates the construction of a user-rating sparse matrix. This matrix serves as the input for our decomposition models. In the subsequent analysis, we present the root mean square error (RMSE) associated with the reconstruction process across varying numbers of latent

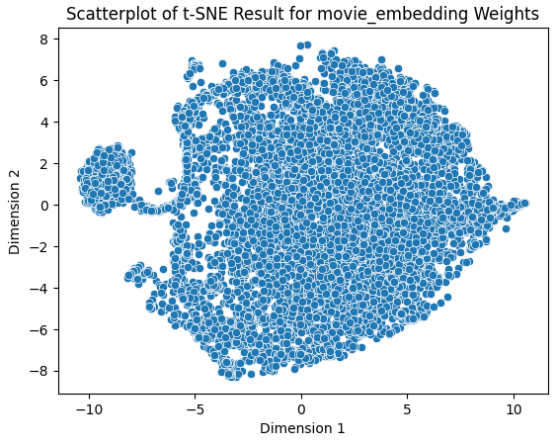


Fig. 8. TSNE embedding dimensionality reduction

factors. While the inclination may be to choose a substantial number of latent factors to minimize RMSE, we select 32 latent factors. Excessive latent factors can lead to overfitting, yielding irrelevant predictions.

Latent Factors	Reconstruction RMSE
16	2.43
32	2.19
64	1.86
128	1.4
256	0.8
512	0.14

D. Deep Learning for Matrix Factorization

Building our Ensemble model, a crucial step involves the implementation of the PyTorch Matrix Factorization model. In this phase, users and movies are numerically encoded. The user-item dataset, denoted as `ratings_dataset`, plays an important role in preprocessing the data, ensuring its conversion into a format suitable to our model's requirements. Subsequently, we introduce the `matrix_factorization` model, which, in accordance with equation (9), computes the dot product of the embeddings corresponding to movies and users. As an indirect outcome of the training process, we obtain the embeddings of movies. Recall that these embeddings encapsulate latent factors, discovering previously unseen relationships between movies.

The resulting matrix of latent factors, possessing a size of emb_size, undergoes clustering via the KMeans algorithm. This clustering mechanism aims to consolidate movies with similar embeddings, revealing hidden relationships. Our approach involves selecting 25 clusters, a decision made based on our assessment that this choice optimally balances a significant reduction in the inertia metric without resulting in excessively small clusters (Fig. 9).

E. Proposed Hybrid Model

In our ultimate model, we integrate the clustered movies obtained from the KMeans algorithm with the results of the

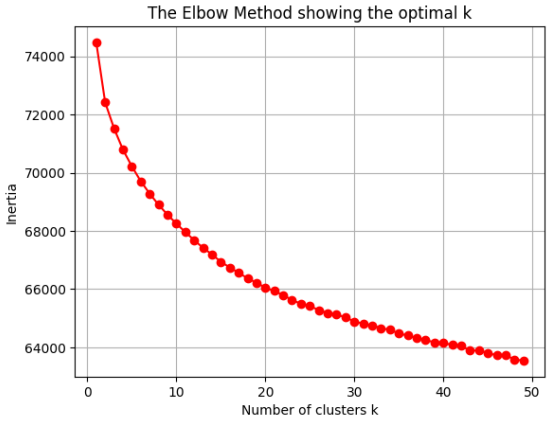


Fig. 9. KMeans Inertia per K

Singular Value Decomposition (SVD) factorization. Initially, we load the requisite models, followed by their incorporation into our `hybrid_recommendation()` function.

In the execution of this function, we identify the top 3 movies predicted by the SVD factorization for a given user. Subsequently, we determine the corresponding clusters to which these movies belong. Our rationale behind this approach is twofold: 1) to obtain a representative sample of the user's highest predicted movies, and 2) to further refine the selection of potential recommendations. By restricting the choices to movies within specific clusters, we aim to enhance the coherence and similarity among the recommended movies.

This multifaceted strategy is designed to achieve two primary objectives: 1) provide the user with a diverse yet personalized selection of top-rated movies and 2) narrow down the pool of potential recommendations, ensuring a high degree of thematic consistency and coherence among the suggested movies.

VII. Conclusion

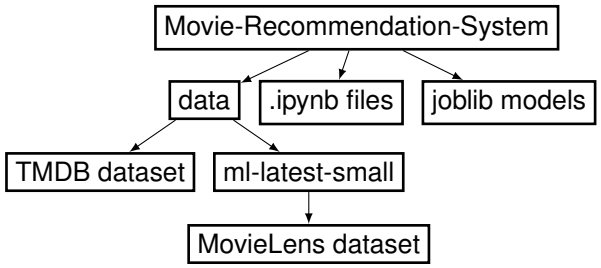
In this research we have proposed diverse algorithms and methodologies aimed at enchancing user experiences in content discovery. Through a comprehensive exploration of collaborative/content-based filtering, matrix factorization and hybrid models we have delved into the nuances of leveraging user-item interactions and latent factors to tailor recommendations.

We have developed content-based algorithms (KNN, KMeans, Cosine Similarity) that can tackle the cold start problem and can begin recommending appropriate content as the user browses an application. These algorithms are suitable for tackling the cold start problem for new users. Moreover we have proposed hybrid models that discover the relationships between users and items and of items with each other in order to provide more accurate and personalized recommendations.

Our findings underscore the significance of combining various recommendation strategies, harnessing the strengths of each to overcome individual limitations. The integration of clustering algorithms and deep learning methodologies along with traditional models such as SVD, stands out as a promising avenue fore achieving both diversity and coherence in personalized recommendations.

VIII. Code

All the research files can be found in the `Movie-Recommendation-System` directory. This directory should contain another directory called `data`. The `data` directory should contain the `ml-latest-small` (MovieLens) directory along with the TMDB files. The directory should look like this (The joblib models are not included to reduce the filesize. They will appear once the code is run):



The code is given in `.ipynb` files. Each file is named in the format `XX_info_dataset` where:

- XX: Order with which the files should run
- info: What the file implements
- dataset: On which dataset the method is implemented.

REFERENCES

[1] Y. Bo and Z. Pengfei. Review of recommendation algorithms. *Journal of Shanxi University (Natural Science Edition)*, 34(03):337–350, 2011.

[2] W. Yanran, C. Mei, W. Hanhu, and Z. Xin. A content filtering-based recommendation algorithm for scientific and technical literature. *Computer Technology and Development*, 21(02):66–69, 2011.

[3] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009, 2009.

[4] S. Renckes, H. Polat, and Y. Oysal. A new hybrid recommendation algorithm with privacy. *Expert Systems*, 29(1):39–55, 2012.

[5] S. Jiangbo, S. Xiaoxuan, L. Hai, Y. Baolin, and Z. Zhaoli. A content-based recommendation algorithm for learning resources. *Multimedia Systems*, 24(2), 2018.

[6] J. Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of the First Instructional Conference on Machine Learning*, volume 242, pages 29–48, 2003.

[7] F. Rahutomo, T. Kitasuka, and M. Aritsugi. Semantic cosine similarity. In *The 7th International Student Conference on Advanced Science and Technology (ICAST)*, volume 4, page 1, 2012.

[8] Hongwei Ma, Guangwei Zhang, and Peng Li. Review of collaborative filtering recommendation algorithm. *Journal of Small Microcomputer Systems*, 30(07):1282–1288, 2009.

[9] Fuping Yang, Hongguo Wang, Shuxia Dong, and Xuechen Zhao. Collaborative filtering recommendation algorithm based on memory effect. *Computer Engineering*, 38(23):63–66, 2012.

[10] Qingwen Liu. *Research on recommendation algorithm based on collaborative filtering*. PhD thesis, University of Science and Technology of China, 2013.

[11] S. Dhanabal and S. Chandramathi. A review of various k-nearest neighbor query processing techniques. *International Journal of Computer Applications*, 31(7):14–22, 2011.

[12] Abinash Pujahari and Dilip Singh Sisodia. Model-based collaborative filtering for recommender systems: An empirical survey. *First International Conference on Power, Control and Computing Technologies (ICPC2T)*, 2020.

[13] Y. H Guo and G. S Deng. Hybrid recommendation algorithm of item cold-start in collaborative filtering system. *Computer Engineering*, 34(23):11–13, 2008.

[14] F. Yang, Y. Zheng, and C. Zhang. Hybrid recommendation algorithm based on probability matrix factorization. *Journal of Computer Applications*, 2018.

[15] Kaiyang Guo and Chenyang Yang. Temporal-spatial recommendation for caching at base stations via deep reinforcement learning. *IEEE Access*, 7:58519–58532, 2019.

[16] Chao Yang, Lianhai Miao, Bin Jiang, Dongsheng Li, and Da Cao. Gated and attentive neural collaborative filtering for user generated list recommendation. *Knowledge-Based Systems*, 187:104839, 2020.

[17] Farhan Ullah, Bofeng Zhang, Rehan Ullah Khan, Tae-Sun Chung, Muhammad Attique, Khalil Khan, Salim El Khediri, and Sadeeq Jan. Deep edu: A deep neural collaborative filtering for educational services recommendation. *IEEE Access*, 8:110915–110928, 2020.

[18] Shuai Yu, Min Yang, Qiang Qu, and Ying Shen. Contextual-boosted deep neural collaborative filtering model for interpretable recommendation. *Expert Systems with Applications*, 136:365–375, 2019.

[19] V. Boppana and P. Sandhya. Web crawling based context-aware recommender system using optimized deep recurrent neural network. *J. Big Data*, 8(1):144, Dec. 2021.

[20] Omer Tal and Yang Liu. Tcenr: A deep learning framework for point-of-interest recommendation in location-based social networks. *Complexity*, 2019:1–16, 2019.

[21] Mengxin Ma, Guozhong Wang, and Tao Fan. Improved deepfm recommendation algorithm incorporating deep feature extraction. *Applied Sciences*, 12(23):11992, 2022.

[22] Y. Xiao, C. Li, and V. Liu. Dfm-gcn: A multi-task learning recommendation based on a deep graph neural network. *Mathematics*, 10(5):721, Feb. 2022.

[23] J. Xu, Z. Hu, and J. Zou. Personalized product recommendation method for analyzing user behavior using deepfm. *J. Inf. Process. Syst.*, 17(2):369–384, Apr. 2021.

[24] L. Chen, X. Bi, G. Fan, and H. Sun. A multitask recommendation algorithm based on deepfm and graph convolutional network. *Concurrency Comput., Pract. Exper.*, 35(2):e7498, Jan. 2023.

[25] H. Khatter, N. Goel, N. Gupta, and M. Gulati. Movie recommendation system using cosine similarity with sentiment analysis. 2021.

[26] Vijayshri Khedkar and Nirav Raval. A collaborative recommender system enhanced with a neural network. 2021.

[27] Parth Sahu, S. Raghavan, and K. Chandrasekaran. Ensemble deep neural network based quality of service prediction for cloud service recommendation. *Neurocomputing*, 465:476–489, 2021.

[28] T.K. Dang, Nguyen Nguyen, Q.P., and V.S. A study of deep learning-based approaches for session-based recommendation systems. sn comput. sci. 1, 216 (2020).

[29] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *ICML*, volume 98, pages 46–54, 1998.

[30] Wei Song and Xuesong Li. A non-negative matrix factorization for recommender systems based on dynamic bias. In Vicenç Torra, Yasuo Narukawa, Gabriella Pasi, and Marco Viviani, editors, *Modeling Decisions for Artificial Intelligence*, pages 151–163, Cham, 2019. Springer International Publishing.

[31] X. Zhang, X. Zhou, L. Chen, et al. Explainable recommendations with nonnegative matrix factorization. *Artificial Intelligence Review*, 56(Suppl 3):3927–3955, 2023.

[32] Kashvi Taunk, Sanjukta De, Srishti Verma, and Aleena Swetapadma. A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1255–1260, 2019.

[33] E. Fix and J. Hodges. An important contribution to nonparametric discriminant analysis and density estimation. *Int. Stat. Rev.*, 3(57):233–238, 1951.

[34] E. Fix and J. L. Hodges. Nonparametric discrimination: Consistency properties. Technical report, Randolph Field, Texas, Proj., 1951.

[35] E. Fix and J. L. Jr. Hodges. Discriminatory analysis-nonparametric discrimination: Small sample performance. 1952.

[36] Pan-Ning Tan, Michael Steinbach, Anij Karpatne, and Vipin Kumar. *Introduction to Data Mining*. 2nd edition, 2019.

[37] Jigui Sun, Jie Liu, and Lianyu Zhao. Clustering algorithms research. *Journal of Software*, 19(1):48–61, January 2008.

[38] Kadambini Swain and Ajit Kumar Nayak. A review on rule-based and hybrid stemming techniques. In *2018 2nd International Conference on Data Science and Business Analytics (ICDSBA)*, pages 25–29, 2018.

[39] M.F. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.