MATHEMATICAL MODELING AND COMPUTING, Vol. 11, No. 3, pp. 785–797 (2024)

Large-scale recommender systems using Hadoop and collaborative filtering: a comparative study

Chafiki M. E.[1], Banouar O.[1], Benslimane M.[2]

[1]*Laboratory of Computer and Systems Engineering, Cadi Ayyad University, Marrakesh, Morocco*
[2]*Sciences, Engineering and Management Laboratory, Sidi Mohamed Ben Abdellah University, Fez, Morocco*

(Received 21 January 2024; Revised 16 August 2024; Accepted 18 August 2024)

With the rapid advancements in internet technologies over the past two decades, the amount of information available online has exponentially increased. This data explosion has led to the development of recommender systems, designed to understand individual preferences and provide personalized recommendations for desirable new content. These systems act as helpful guides, assisting users in discovering relevant and appealing information tailored to their specific tastes and interests. This study's primary objective is to assess and contrast the latest methods utilized in recommender systems within a distributed system architecture that relies on Hadoop. Our analysis will focus on collaborative filtering and will be conducted using a large dataset. We have implemented the algorithms using Python and PySpark, enabling the processing of large datasets using Apache Hadoop and Spark. The studied approaches have been implemented on the MovieLens dataset and compared using the following evaluation metrics: RMSE, precision, recall, and F1 score. Their training times have also been compared.

Keywords: *recommender system; collaborative filtering; Hadoop; Spark; similarity.*
2010 MSC: 68Txx **DOI:** 10.23939/mmc2024.03.785

1. Introduction

A recommender system is a mechanism that anticipates and proposes items or content that users will likely find interesting or valuable by predicting and suggesting them. These systems have gained popularity in recent years, revolutionizing the way we discover products, movies, music, articles, and even potential connections in social networks. By analyzing user preferences, their past behavior, and other relevant data, recommender systems can significantly enhance user experiences by providing personalized recommendations that match individual tastes and needs [1].

To meet the increasing demands for scalability and processing large datasets, modern recommender systems often rely on distributed computing frameworks such as Hadoop and Spark [2]. These frameworks provide powerful tools and infrastructure for processing and analyzing massive amounts of data in a distributed manner, enabling recommender systems to handle the immense scale and complexity of contemporary datasets. The combination of Hadoop and Spark offers numerous advantages. Hadoop provides a solid foundation with its Hadoop Distributed File System (HDFS) storage, batch processing using MapReduce and resource management using YARN, while Spark brings real-time analysis and interactive capabilities with its in-memory engine.

Indeed, these components work in synergy to provide a distributed framework capable of efficiently handling large amounts of data required for building recommender systems. HDFS provides scalable and fault-tolerant storage, Spark offers real-time and in-memory processing capabilities, while YARN allows for managing and sharing cluster resources among Spark applications. Together, they enable recommender systems to effectively handle the scale and complexity of contemporary datasets.

This work aims to perform an extensive comparative analysis (qualitative and quantitative) of collaborative filtering-based recommender systems under the framework of Hadoop and Spark. We aim to evaluate and compare various approaches in terms of effectiveness, efficiency, and applicability when used in distributed computing systems.

The remainder of the document is organized as follows: an overview of recommender system technology is given in Section 2, Section 3 presents each of the techniques used in the comparative study, Section 4 introduces the implementation and results, followed by a comparison of the results, Section 5 concludes the paper.

2. Recommender systems approaches: state of the art

A recommender system uses sophisticated techniques that filter through tons of data to determine connections between the user and other users or between different products and content. Based on the patterns it discovers, the system understands what the user may appreciate even without explicit requests. Three conventional recommender strategies that are frequently employed are content-based filtering, collaborative filtering, and hybrid filtering. Each recommender algorithm has its own advantages and disadvantages [3]. Another emerging recommender technique is contextual filtering [4].

2.1. Collaborative filtering (CF)

Collaborative filtering is a popular recommender system technique, which generates relevant recommendations based on previous rating information, reviews, and comments. This method operates without requesting external data, instead relying on the users' (or items') similarities [5]. With two dimensions (users and elements), the collection of preferences creates an evaluation matrix, where each row denotes a user and each column an element. The similarity between two users (elements) depends on the similarity of the evaluation history of these two users (elements). The collaborative filtering algorithms are mainly classified into two categories: memory-based and model-based.

Memory based approach uses historical user evaluation data to identify similarity between users or items. These techniques aim to establish a similarity metric between users or items, and find the most similar ones to recommend unseen items. While model-based creates a model using the dataset and gives the user recommendations based on the model. This method learns a model by using users' past evaluations and improves the performance of CF technique. CF offers a number of significant benefits, including the ability to function in domains with less article content, and where the content is difficult to analyze for a computer system [6].

Khalid Haruna et al. [7] presented a collaborative approach for a research paper recommender system. In order to tailor recommendations, it makes use of the benefits of collaborative filtering and publicly accessible contextual information to deduce the unnoticed relationships between research publications.

2.2. Content-based filtering (CBF)

In content-based filtering technique, user recommendations are based on the behavior and data of a single user. The description of items and the user's profile play an important role in content-based filtering technique. The following is the fundamental idea behind CBF recommender systems:

— Firstly, we examine the description of the preferred items by a specific user to figure out the preferences that can be used to characterize those items. These preferences are saved in a user profile.
— Secondly, each attribute is compared with the user's profile so that only relevant items that exhibit a high degree of similarity with the user's profile are recommended to that particular user.

A work by VIANA and SOARES [8] looks at how different metadata elements might be applied to improve the quality of recommendations for multimedia content, particularly for movies and TV shows. The studied metadata elements include title, genre, production date, and the list of directors and actors.

2.3. Context aware recommender systems (CARS)

The context is necessary in order to explain the setting in which a user and an application communicate. CARS are able to comprehend the unique needs of each user and offer customized services. Context in recommender systems is a key addition to tailor the recommendation provided to users.

By incorporating context, conventional two-dimensional recommender systems are expanded to three dimensions: context C, user U, and item I [9]. With this feature, CARS's objective is now to precisely prescribe actions to a specific user at the appropriate moment, in the appropriate setting, based on user-specific data, including their present activity and emotional state [10].

A notable example is CReS [11], proposed by Thaipisutikul and Shih in 2021 which captures consumers' dynamic preferences through the use of a deep sequential learning technique. CReS addresses the challenges of modeling the change in consumers' interests over time, incorporating long-term preferences, and providing interpretable recommendations. It employs a self-attention network to capture short-term and long-term interests, considering the hierarchical connections between items and contextual relations. This system aims to provide accurate and personalized recommendations while offering interpretability through its attention mechanism. Overall, CReS is an example of a context-aware recommender system that leverages deep learning techniques to enhance recommendation performance.

2.4. Hybrid recommender system approaches

The hybrid filtering method combines several recommender systems approaches for better system optimization in order to avoid certain limitations and issues of individual techniques. The hybrid filtering technique's concept is to combine methods that will offer recommendations that are superior and more effective than those made by a single algorithm since the drawbacks of one algorithm will be compensated for by another. There are different strategies to the hybrid filtering process that combine two or more techniques. The hybrid algorithm has a few benefits, including the capacity to solve issues with scalability, cold start, and new user concerns that other approaches cannot. The disadvantages are that it is complex and costly to implement [6].

Table 1. Classification of hybridization methods.

Technique name	Function
Weighted	Combine many recommender system components in a numerical manner
Switching	System alternates between recommender systems based on the circumstances at hand
Mixed	A group of recommenders is displayed together
Feature Combination	Combine the various features and assign them to a single technique
Feature Augmentation	One approach's output is utilized as an input feature by another approach
Cascade	Applies a recurrent improvement. One recommender improves upon the suggestion made by another
Meta-level	One method is used to create a small model, and then another method is applied to the input

3. Distributed approaches based on collaborative filtering for recommender systems

This section presents a thorough analysis of every method utilized. Implementation and results are presented in the following sections.

3.1. Regularized user and item effect (RUME)

This term is inspired by the SVD++ model [12], which emphasizes the importance of considering user and movie biases in collaborative filtering algorithms. By regularizing these biases, the model can prevent overfitting and improve generalization to new data.

First, a range of regularization parameters λ is to be initialized. Then we calculate the average value of the ratings in the training set μ. Then, for each λ value, we calculate the user and item biases. The item bias b_i is determined by taking the sum of the differences between the ratings given by users for that particular item r_ij and the overall average rating μ. This sum is then regularized by dividing it by how many ratings that particular item has $|R_i|$ plus the regularization value λ,

$$b_i = \frac{\sum_{j \in R_i}(r_{ij} - \mu)}{|R_i| + \lambda}, \tag{1}$$

R_i is the ratings for that item.

On the other hand, the user bias is determined by taking into account the ratings they have given to the items R_u and the biases of those items b_i. Regarding every item i that the user u has rated, we deduct the item bias b_i and the overall average rating μ from the user's rating r_ij of that particular item. We then sum up all these differences and divide by the total number of items rated by the user plus the regularization value. The final predictions p_ui are the sum of the average μ, the item bias b_i, and the user bias b_u,

$$b_u = \frac{\sum_{i \in R_u}(r_{ij} - \mu - b_i)}{|R_u| + \lambda}, \tag{2}$$

$$p_{ui} = \mu + b_i + b_u. \tag{3}$$

Finally, we store the results in a list to identify the best regularization value based on the root mean squared error (RMSE).

RMSE is one of the evaluation criteria of a regression model. It is a metric that shows the typical difference between the values in the dataset that are real and the values the model predicts. The better a model fits a dataset, the lower its RMSE [13],

$$\text{RMSE} = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (p_{ui} - r_{ui})^2}, \tag{4}$$

where p_{ui} is the prediction and r_{ui} is the real rating each pair of users and items (u, i) in the test dataset T. And the total number of user-item pairings in the test dataset is denoted by $|T|$.

3.2. Alternating least squares with weighted regularization (ALS-WR)

Introduced to solve scalability issues of massive datasets, ALS-WR [14] is one of the most well-known recommender system algorithms. Let R be a rating matrix with user items where each element r_ij represents the score that user i assigned to item j (or 0 if the user has not given the item a rating). ALS-WR takes as input the training data R and the test data $Test$ and has the goal to decompose the matrix R into two latent factor matrices, U and V. It initializes the matrices U and V randomly, the regularization parameter λ, the convergence threshold ε, the number of max iterations and the dimension of the matrices of latent factors are defined. The method keeps iterating until it hits either the maximum number of iterations or convergence.

In each iteration, it updates the corresponding rows of matrices U and V according to the next equations:

$$U_u = \left(V \times V^T + \lambda \times I\right)^{-1} \cdot V^T \times R_u, \tag{5}$$

$$V_i = \left(U \times U^T + \lambda \times I\right)^{-1} \cdot U^T \times R_i. \tag{6}$$

Then, it calculates the predictions P_{ui} for each pair of user u and item i in the test data,

$$P_{ui} = U_u \cdot V_i^T. \tag{7}$$

The RMSE is calculated based on the predictions. The variation in RMSE errors between two consecutive iterations, called $diff$, is also calculated. If $diff$ is less than the convergence threshold ε, it indicates that the algorithm has converged.

3.3. Memory based collaborative filtering (MCF)

In this algorithm, we compute a similarity matrix between the items (or users) and use it to estimate the missing ratings based on the rating history. This allows us to recommend the highest-rated items to each user. We put item-based and user-based collaborative filtering into practice, utilizing the Pearson correlation coefficient and cosine similarity, two distinct similarity metrics.

Cosine similarity. By definition, cosine similarity is a positive number between 0 and 1, 1 if X and Y are identical, and 0 if there is no similarity between the two. The cosine similarity is used to determine the cosine of the angle that exists between two vectors, or two documents in vector space. The smaller is the angle, the greater is the similarity. To find the cosine similarity between two vectors, x and y,

we divide their dot product by the product of their magnitudes,

$$\text{cosine}(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}. \tag{8}$$

Pearson correlation. It is used to measure how two sets of data are linearly related. The Pearson correlation is a statistic with values ranging from -1 to $+1$, signifying perfect negative and positive correlations, respectively. When there is no correlation between the variables under study, the value is zero [15]. It is calculated for two vectors, x and y, by dividing their covariance by the product of their standard variances,

$$r(x, y) = \frac{\text{cov}(x, y)}{\text{std}(x) \cdot \text{std}(y)}. \tag{9}$$

In user-based collaborative filtering, to start, we compute a similarity matrix S for every user included in the dataset. Then we keep only positive values in the matrix,

$$S = \max(0, x) \text{ for each } x \text{ in } S. \tag{10}$$

We then provide recommendations for the target user based on the ratings these similar users have provided. The predictions are determined by calculating the weighted average of the ratings given by the other users for the same item,

$$\text{Pred}_{u,i} = \sum_{j \in U} \frac{S_{u,j} \cdot R_{j,i}}{S_{u,j}}, \tag{11}$$

where $\text{Pred}_{u,i}$ is the prediction of the rating of the user u for the item i, U is the list of the users who have previously rated the item i, $S_{u,j}$ is the similarity between the users u and j, $R_{j,i}$ is the rating of the user j for the item i.

In contrast, item-based collaborative filtering involves selecting a target item and identifying other items that demonstrate similarity based on evaluations from users who have rated both items. The predictions are calculated by taking the weighted average of the ratings given by the current user for the other items.

3.4. K-means clustering + Memory based collaborative filtering (KMCF)

K-means clustering is a vector quantization technique that gained popularity for cluster analysis in data exploration. It was initially employed in signal processing. With K-means clustering, an observation is divided into k clusters, with each observation belonging to the cluster whose mean is closest to it, acting as the cluster prototype.

In K-means, we begin by initializing the k cluster centroids at random, denoted as $\mu_1, \mu_2, \ldots, \mu_k$. Then, for each data point x_i, we calculate the distance to each centroid μ_j using one of the distance measures below,

Euclidean distance:

$$d(x_i, \mu_j) = \sqrt{\sum_k (x_{i_k} - \mu_{j_k})^2}, \tag{12}$$

where $x_{i,k}$ and $\mu_{j,k}$ are the kth coordinates of x_i and μ_j, respectively.

Cosine distance:

$$d(x_i, \mu_j) = 1 - \text{cosine}(x_i, \mu_j), \tag{13}$$

where $\text{cosine}(x_i, \mu_j)$ is the cosine similarity between x_i and μ_j.

Next, we designate the cluster containing the nearest centroid for every data point x_i, based on the minimum distance:

$$C(x_i) = \arg\min(d(x_i, \mu_j)) \tag{14}$$

Next, we compute the mean of the data points allocated to each cluster in order to update the centroids of each cluster,

$$\mu_j = \frac{1}{|C(j)|} \sum_{x_i} x_i, \tag{15}$$

where x_i belongs to $C(j)$, and $|C(j)|$ denotes the number of data points in cluster $C(j)$.

Until the cluster assignments no longer significantly change or a maximum number of iterations is reached, we repeat those steps.

One of the constraints of this method is that it is necessary to determine the optimal number of clusters in advance. The distance metric, which is a crucial parameter of this method, currently offers two options: Euclidean distance and cosine distance. A visual depiction of the algorithm's process can be seen in Figure 1.

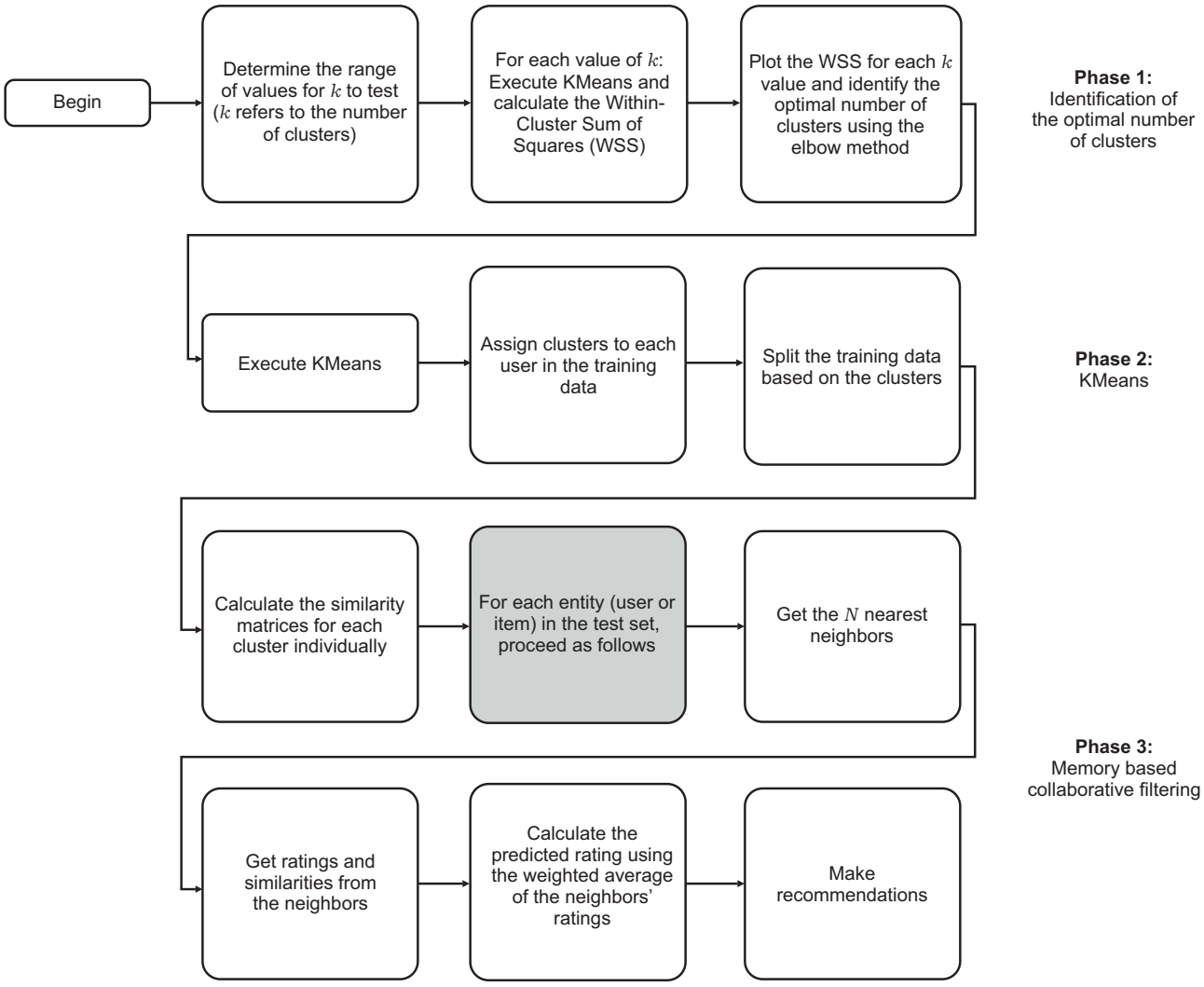


Fig. 1. Diagram of the process of the K-means clustering + Collaborative Filtering algorithm.

3.5. Decision Tree

Decision trees are a popular method for classification in data exploration. They help simplify the decision-making process by creating a tree-like structure that divides the data into smaller subsets [16]. The final result is a tree with decision nodes and leaves, where decision nodes have multiple branches and leaves represent a decision or classification. Decision trees can handle both categorical and numerical data.

This algorithm is mostly used to solve classification problems, while it can also be used in regression cases. It is composed of three types of nodes: the root node which acts as the starting point that further expands to various branches making it a tree-like structure [17] the leaf node, which is the output of decision nodes and has no further branches, and the decision node, which is used to make decisions and has several branches. Decision tree simply forks the tree into sub-trees on the basis of answer to question i.e. whether a yes or a no [17].

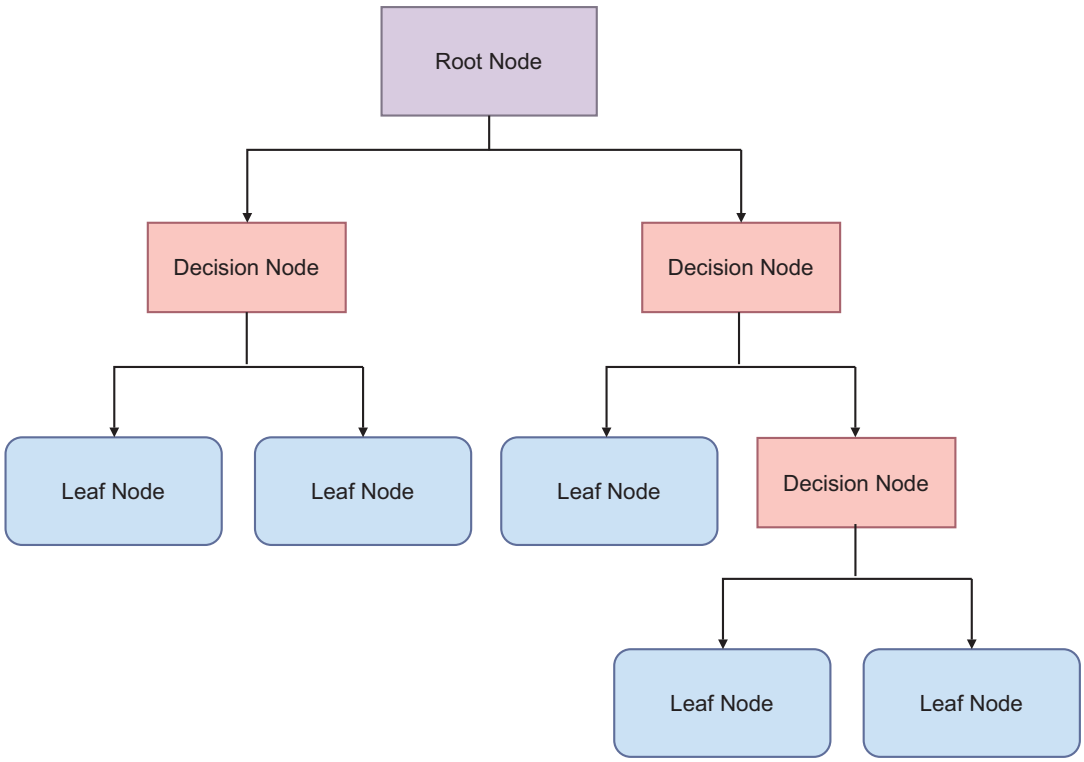


Fig. 2. Diagrammatic representation of a decision tree.

The first step of this approach treats the whole dataset as the decision tree's first beginning point. After that, it determines which feature is best for dividing the data while accounting for particular factors like the Gini index.

Gini index is a measure to assess a node's level of impurity. It measures the likelihood that a randomly chosen element will be incorrectly classified in light of the node's class distribution. The following is a definition of the Gini index equation:

$$\text{Gini} = 1 - \sum_j P_j^2, \tag{16}$$

where P_j is the likelihood that an item will fall into a specific class.

It then creates a decision node at that split point and divides the data into subsets according to the values of the feature. Until a stopping condition is satisfied, this procedure is continued recursively for every subset (e.g., reaching a maximum depth, minimum number of samples). At the leaf nodes, which represent the final predictions, the decision tree assigns a majority class or computes the average value of the target variable. During prediction, an input instance, depending on the feature values, proceeds down the decision path from the root node to a leaf node, and the anticipated result is determined by the corresponding leaf node's value.

In the context of a recommender system, decision trees are trained using product features. In the case of the MovieLens dataset, these features are represented by the genres of each film.

3.6. AutoRec: Autoencoders meet collaborative filtering

Deep learning can be viewed as a way to automate the process of performing prediction analyses due to its inherent simplicity. In contrast to the linear nature of typical machine learning algorithms, deep learning approaches consist of a hierarchy of increasing complexities and abstractions. Over time, neural networks have emerged in various forms, including back propagation neural networks (BPNN), convolutional neural networks (CNN), recurrent neural networks (RNN), and auto encoders (AE). Neural networks are considered the most accurate artificial representation of the functioning of the human brain [18].

AutoRec is an AE-based collaborative filtering model proposed by Sedhain et al. [19]. It is a model that takes as input the representative vector of each item $r(i)$ or user $r(u)$, projects it into a latent space of reduced dimension, and then reconstructs it in the output space of the initial dimension to predict missing ratings for recommendation purposes. The authors of [19] used a set of base parameters for the AutoRec model, which were then fine-tuned based on the RMSE metric. The best identified parameters are:

— Activation function for the encoder f: $f(z) = \frac{1}{1+e^{-z}}$;
— Activation function for the decoder g: $g(x) = x$;
— Regularization parameter λ: 0.001;
— Learning rate α: 0.0001;
— Number of epochs: 200.

However, the number of hidden units and batch size may vary depending on the size of the dataset used and the approach (user-based or item-based).

The algorithm operates as follows in a matrix of interactions with m users and n items, where $r(i) = (R_{1i}, \ldots, R_{mi})$ represents the ratings of user i for each item, $r(u) = (R_{u1}, \ldots, R_{un})$ represents the ratings of user u for each item, and h denotes the number of hidden layers: first, it initializes the weight matrix $W \in \mathbb{R}^{h \times n}$, bias vectors $b \in \mathbb{R}^h$, and $c \in \mathbb{R}^m$. Then, for each training epoch and each batch of input vectors $X(k)$:

1. Calculates the activation of the hidden layer for batch:
$$H^{(k)} = f\big(WX^{(k)} + b\big). \tag{17}$$
2. Calculates the reconstructed output vector for the batch:
$$X'^{(k)} = g\big(WH^{(k)} + c\big), \tag{18}$$
where $H^{(k)}$ is the hidden layer activation for a specific batch of input vectors $X^{(k)}$.
3. Calculates the masked MSE for the batch utilizing the subsequent equation:
$$L^{(k)} = \frac{1}{\sum_{i,j} M_{ij}^{(k)}} \sum_{i,j} M^{(k)} \big(X_{ij}^{(k)} - X_{ij}'^{(k)}\big), \tag{19}$$
where the mask matrix $M^{(k)}$ is a binary matrix of the same dimensions as the input vectors $X^{(k)}$; $M_{ij}^{(k)}$ is the element of the mask matrix corresponding to the i-th user and j-th item in the k-th batch, it is either 0 or 1, indicating whether the rating for that user-item pair is missing (0) or present (1).
4. Calculates the regularization term:
$$R = \frac{1}{2}\lambda \sum_{j=1}^{h} \sum_{k=1}^{n} W_{jk}^2. \tag{20}$$
5. Calculates the total loss for the batch:
$$L_{\text{total}}^{(k)} = L^{(k)} + R, \tag{21}$$
where $L^{(k)}$ is the masked MSE for the k-th batch and R is the regularization term.
6. Backpropagation: Calculate the gradients
$$\frac{\partial L_{\text{total}}^{(k)}}{\partial W}, \quad \frac{\partial L_{\text{total}}^{(k)}}{\partial b}, \quad \frac{\partial L_{\text{total}}^{(k)}}{\partial c}. \tag{22}$$
7. Update the weights and biases using gradient descent:
$$W \leftarrow W - \frac{\alpha}{B} \sum_{k=1}^{B} \left(\frac{\partial L_{\text{total}}^{(k)}}{\partial W} \right), \quad b \leftarrow b - \frac{\alpha}{B} \sum_{k=1}^{B} \left(\frac{\partial L_{\text{total}}^{(k)}}{\partial b} \right), \quad c \leftarrow c - \frac{\alpha}{B} \sum_{k=1}^{B} \left(\frac{\partial L_{\text{total}}^{(k)}}{\partial c} \right), \tag{23}$$
where B represents the batch size.

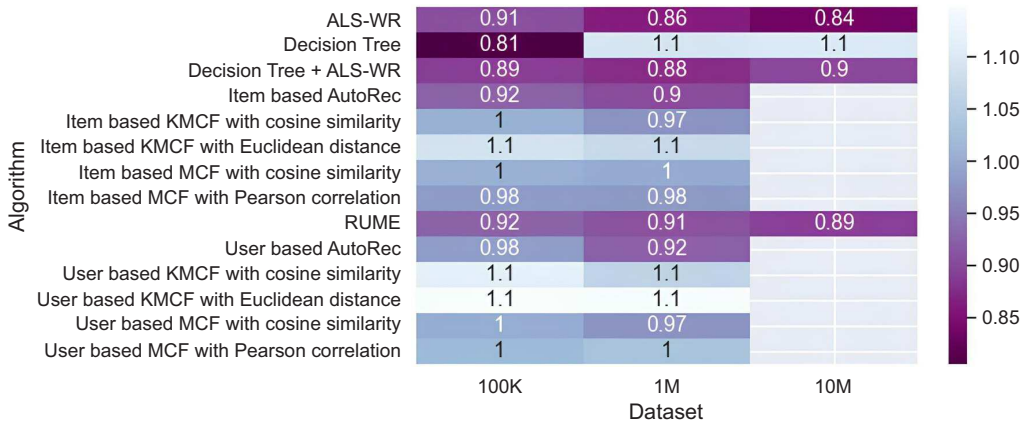
3.7. Decision tree + ALS-WR

This algorithm contains the implementation of the model proposed in [16]. It is a weighted hybrid model that combines decision tree-based classification, which considers movie properties, with collaborative filtering using ALS-WR matrix factorization. However, in cases where ALS-WR cannot provide a recommendation or suffers from the cold-start problem, we use decision tree predictions.

4. Implementation and results

Having introduced the different recommender techniques and their foundations, we now delve into a comprehensive examination of these techniques through a comparative study, encompassing both quantitative and qualitative evaluations. Our study involves the practical implementation and rigorous testing of each technique on the MovieLens dataset within a distributed environment. Specifically, we conduct our experiments on a Hadoop/Spark cluster comprising three machines, following a Master/slave architecture, with a total of 30 Gb ram and 70 Gb HDD Storage.

There were three MovieLens dataset variations used. MovieLens 100k, which has 100 000 user ratings for 1682 films submitted by 943 individuals. A total of 1 000 209 user ratings for 3 900 films are available on MovieLens 1M. Additionally, MovieLens 10M has 10 000 054 user ratings for 10 681 movies from 71 567 people. To quantify the effectiveness of each method and identify the ones that are the most efficient, we used different evaluation metrics, which are RMSE, precision, recall, f1 score and time. But first we had to construct a confusion matrix. In order to do that, we start from the assumption that any real rating above a certain threshold, such as 3.5, corresponds to a relevant element, and any real rating below the threshold is not relevant. In other words, we consider a recommendation as "relevant" if the user evaluates the article positively, and "non-relevant" if the user evaluates the article negatively. This allows us to construct a confusion matrix, which we can then use to calculate precision, recall, and F1 score. The following figures show the results of the study.

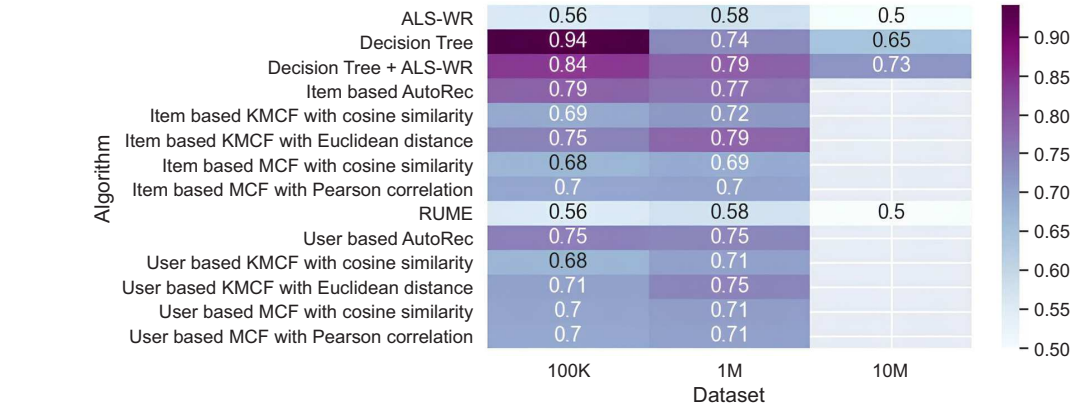


ALS-WR: Alternating least squares with weighted regularization
MCF: Memory based collaborative filtering
KMCF: K-Means + Memory based collaborative filtering
RUME: Regularized user and item effect

Fig. 3. RMSE results.

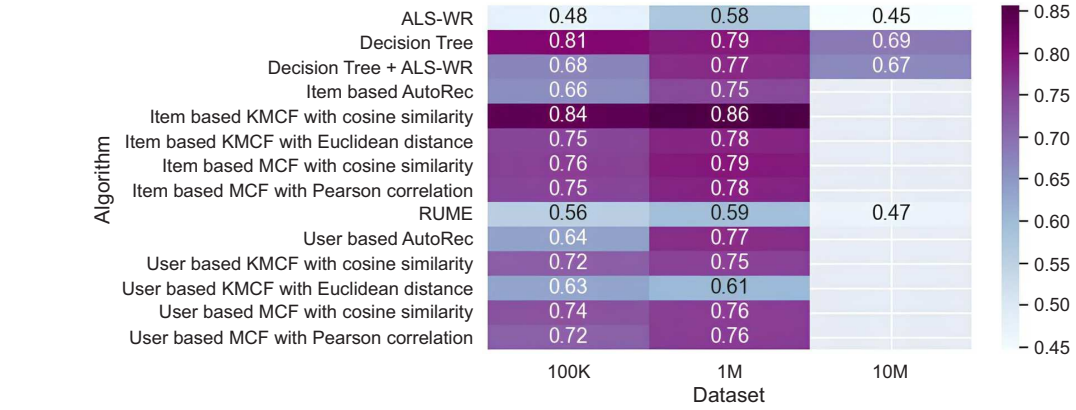
The evaluation of different recommender algorithms has yielded interesting results. Evaluation metrics such as RMSE, precision, recall, and F1 score provide an indication of the algorithms' performance on the MovieLens datasets.

Among the tested algorithms, ALS-WR and the RUME model showed good results in terms of RMSE. This indicates a high predictive capability of the ratings. However, in terms of precision, recall, and F1 score, these two algorithms obtained acceptable values but were lower than the other algorithms. In terms of training time and memory management, the results suggest that those two algorithms can be faster to train and more efficient in terms of resource management.



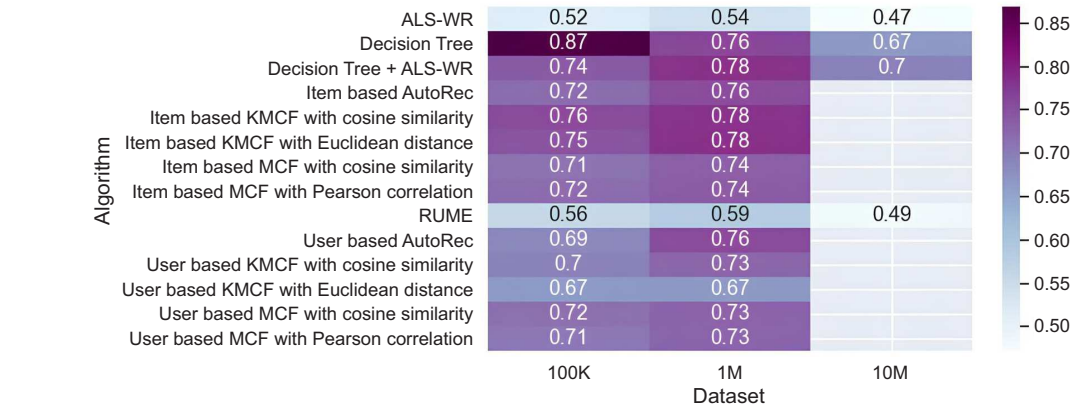
ALS-WR: Alternating least squares with weighted regularization
MCF: Memory based collaborative filtering
KMCF: K-Means + Memory based collaborative filtering
RUME: Regularized user and item effect

Fig. 4. Precision results.



ALS-WR: Alternating least squares with weighted regularization
MCF: Memory based collaborative filtering
KMCF: K-Means + Memory based collaborative filtering
RUME: Regularized user and item effect

Fig. 5. Recall results.



ALS-WR: Alternating least squares with weighted regularization
MCF: Memory based collaborative filtering
KMCF: K-Means + Memory based collaborative filtering
RUME: Regularized user and item effect

Fig. 6. F1 score results.

Compared to the decision tree algorithm, we observe that in the MovieLens 100k dataset, decision trees display high scores in RMSE, precision, recall, and F1 score. This suggests that they are capable of generating high-quality recommendations and efficiently retrieving relevant items. However, this algorithm encountered difficulties in handling larger datasets, resulting in a degradation of performance.

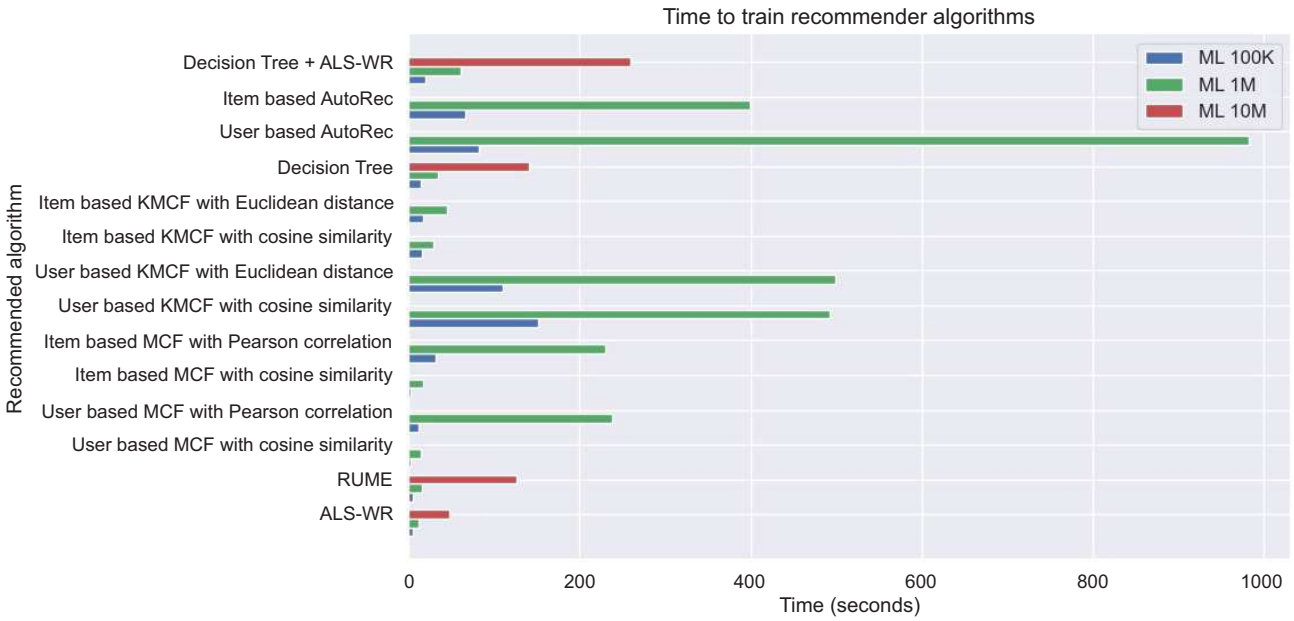


Fig. 7. Training time results.

In terms of time, decision trees are relatively fast compared to memory-based algorithms but slower than matrix factorization and RUME.

Moving on to the hybridization of decision trees with ALS-WR, in terms of results, this method shows promise. It achieves high scores in RMSE, precision, recall, and F1 score, suggesting a significant improvement compared to using ALS-WR alone. However, in terms of execution time, the hybridization led to an increase in the training time compared to ALS-WR or decision trees alone. When comparing the performance of other algorithms, we can see that these algorithms have a significant demand for RAM management, especially for larger datasets (MovieLens 10M). This increased memory demand can be considered a constraint in terms of system resource management. These algorithms require storing interaction matrices, similarity matrices, or weighting coefficients to perform recommendation calculations. This can result in a significant increase in memory consumption, especially as the dataset size increases.

AutoRec achieves reasonable performance in both the 100k and 1M datasets, with relatively good RMSE values. AutoRec based on item performs slightly better than user-based AutoRec. In general, AutoRec exhibits high precision, good recall, and competitive F1 scores compared to other algorithms.

Analyzing the results of memory based collaborative filtering, we observe higher RMSE values compared to other algorithms in the study. However, the precision, recall, and F1 scores are competitive compared to other algorithms. Specifically, collaborative filtering with Pearson correlation presents the best results. It is worth noting that this approach does not have a real learning phase but involves data transformation to prepare it for predictions.

Moving on to memory based collaborative filtering with KMeans clustering, the results reveal the highest RMSE values compared to other algorithms in the study, especially for the user-based approach. However, in terms of precision, recall, and F1 score, it achieves competitive results compared to other approaches. Specifically, user based KMCF with cosine distance achieves the best recall values for the 100k and 1M datasets.

5. Conclusion

This paper explores various collaborative filtering recommender algorithms, such as linear models, matrix factorization, neighborhood models, autoencoders, decision trees, and hybrid approaches, using the MovieLens dataset under the Hadoop/Spark framework. The study compares these algorithms using evaluation metrics like RMSE, precision, recall, and F1 score. Matrix factorization models demonstrate good prediction accuracy with faster training time and efficient resource utilization. However, they

perform worse than other algorithms in terms of precision, recall, and F1 score. Decision trees show high scores for all metrics, indicating high-quality recommendations and efficient retrieval of relevant items in the MovieLens 100k dataset. However, they struggle with larger datasets, resulting in performance degradation. Decision trees are relatively fast compared to memory-based algorithms but slower than matrix factorization. Hybridizing decision trees with ALS-WR shows promising results, but it requires more time for model training. Other algorithms like AutoRec and similarity-based collaborative filtering or KMeans clustering also achieve competitive performance in certain metrics but require significant memory management, especially for larger datasets. Ultimately, the choice of algorithm depends on specific constraints, performance requirements, and resource availability. It is crucial to consider these factors when selecting the most suitable recommender algorithm for a particular use case.

[1] Roy D., Dutta M. A Systematic Review and Research Perspective on Recommender Systems. Journal of Big Data. **9** (1), 59 (2022).

[2] Dahdouh K., Dakkak A., Oughdir L., Ibriz A. Large-Scale e-Learning Recommender System Based on Spark and Hadoop. Journal of Big Data. **6** (1), 2 (2019).

[3] Patel B., Desai P., Panchal U. Methods of Recommender System: A Review. 2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS). 1–4 (2017).

[4] Raza S., Ding C. Progress in Context-Aware Recommender Systems – An Overview. Computer Science Review. **31**, 84–97 (2019).

[5] Anwar T., Uma V. Comparative study of recommender system approaches and movie recommendation using collaborative filtering. International Journal of System Assurance Engineering and Management. **12**, 426–436 (2021).

[6] Isinkaye F. O., Folajimi Y. O., Ojokoh B. A. Recommendation Systems: Principles, Methods and Evaluation. Egyptian Informatics Journal. **16** (3), 261–273 (2015).

[7] Haruna K., Akmar I. M., Damiasih D., Sutopo J., Herawan T. A Collaborative Approach for Research Paper Recommender System. PLoS ONE. **12** (10), e0184516 (2017).

[8] Soares M., Viana P. Tuning Metadata for Better Movie Content-Based Recommendation Systems. Multimedia Tools and Applications. **74** (17), 7015–7036 (2015).

[9] Suhaim A. B., Berri J. Context-Aware Recommender Systems for Social Networks: Review, Challenges and Opportunities. IEEE Access. **9**, 57440–57463 (2021).

[10] Ben Sassi I., Mellouli S., Ben Yahia S. Context-aware recommender systems in mobile environment: On the road of future research. Information Systems. **72**, 27–61 (2017).

[11] Thaipisutikul P., et al. A Hybrid Recommender System for Personalized Point of Interest Recommendations in Online Social Networks. Information Sciences. **418**, 541–562 (2017).

[12] Zhang Y. An Introduction to Matrix factorization and Factorization Machines in Recommendation System, and Beyond. Preprint arXiv:2203.11026 (2022).

[13] Mallouk I., Abou el Majd B., Sallez Y. A generic model of the information and decisional chain using Machine Learning based assistance in a manufacturing context. Mathematical Modeling and Computing. **10** (4), 1023–1036 (2023).

[14] Aljunid M. F., Manjaiah D. H. An Improved ALS Recommendation Model Based on Apache Spark. Soft Computing Systems (ICSCS 2018). 302–311 (2018).

[15] Jannani A., Sael N., Benabbou F. Machine learning for the analysis of quality of life using the World Happiness Index and Human Development Indicators. Mathematical Modeling and Computing. **10** (2), 534–546 (2023).

[16] Sehta M. N., Patidar A. A Product Recommendation System Using HADOOP Server. International Journal of Engineering Science. **19**, 137 (2018).

[17] Bansal M., Goyal A., Choudhary A. A comparative analysis of K-Nearest Neighbour, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning. Decision Analytics Journal. **3**, 100071 (2022).

[18] Bouzaachane K., Darouichi A., El Guarmah E. M. Deep learning for photovoltaic panels segmentation. Mathematical Modeling and Computing. **10** (3), 638–650 (2023).

[19] Sedhain S., Menon A. K., Sanner S., Xie L. AutoRec: Autoencoders Meet Collaborative Filtering. WWW'15 Companion: Proceedings of the 24th International Conference on World Wide Web. 111–112 (2015).

Великомасштабні рекомендаційні системи з використанням Hadoop і спільної фільтрації: порівняльне дослідження

Чафікі М. Е.[1], Бануар О.[1], Бенсліман М.[2]

[1] *Лабораторія комп'ютерної та системної інженерії, Університет Каді Айяда, Марракеш, Марокко*
[2] *Лабораторія наук, техніки та управління, Університет Сіді Мохамеда Бен Абделлаха, Фес, Марокко*

Завдяки стрімкому розвитку інтернет-технологій за останні два десятиліття кількість інформації, доступної в Інтернеті, експоненціально зросла. Цей вибух даних призвів до розробки рекомендаційних систем, розроблених для розуміння індивідуальних уподобань і надання персоналізованих рекомендацій щодо бажаного нового вмісту. Ці системи служать корисними довідниками, допомагаючи користувачам знаходити актуальну та цікаву інформацію, яка відповідає їхнім конкретним смакам та інтересам. Основною метою цього дослідження є оцінка та порівняння найновіших методів, які використовуються в рекомендаційних системах у межах архітектури розподіленої системи, яка основана на Hadoop. Наш аналіз зосереджений на спільній фільтрації та проводитиметься з використанням великого набору даних. Реалізовано алгоритми за допомогою Python і PySpark, що дозволяє обробляти великі набори даних за допомогою Apache Hadoop і Spark. Досліджувані підходи реалізовано на наборі даних MovieLens і порівняно за такими показниками оцінки: RMSE, точність, повнота та оцінка F1. Також порівнювали час їхніх тренувань.

Ключові слова: *система рекомендацій; спільна фільтрація; Hadoop; Spark; схожість.*