International Journal of Scientific Research and Engineering Development-– Volume 6 Issue 3, May-June 2023

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Enhancing Query Recommendations using Collaborative Filtering and Locality Sensitive Hashing: A Novel Approach

Dr. J. Sharel
sharelsky@gmail.com
AssistantProfessor
Christopher Arts andScienceCollege(women), Tamil Nadu, India

Abstract

Since the inception of computing, there has been an ongoing discussion regarding the use of information systems for suggesting products to customers based on their interests. The concept of recommendation systems gained popularity after the practical application of Grundy [12], a computer-based library that suggested novels to users based on their interests. With the widespread use of the internet, recommendation systems have become an integral part of the user experience in web services and social networks. In many cases, the presence of recommendation systems is a crucial factor in choosing one service over another.

The objective of this study is to explore a novel approach to improve query recommendations. The proposed method incorporates collaborative filtering and Locality Sensitive Hashing (LSH) to enhance the system's time performance while improving the accuracy of the recommendations through a combination of collaborative filtering and content-based techniques. The results of the study indicate that the proposed approach has the potential to generate useful query recommendations, which could be implemented in real-world scenarios without compromising performance.

Keywords:LSH, DBMS, Data Mining

I. Introduction

Data has become increasingly ubiquitous over the past decade, necessitating the efficient storage of large amounts of information. Databases have emerged as the preferred method for organizing and storing information in an organized and efficient manner, with most database management systems being designed around the relational model since the 1970s. Relations, also known as tables, are the most fundamental elements that characterize the relational model, consisting of a collection of tuples, or table rows, each of which shares a set of characteristics, or table columns. Typically, data is retrieved from a DBMS by sending a query, or a structured request for a set of data.

Suppose we have access to a large number of queries that users submit to a DBMS, each of which is associated with a rating that indicates how satisfied the user is with the query's outcome. Let's assume that our DBMS is made up of a single relation that allows users to submit multiple queries to retrieve data. The question that remains is whether we can use all of this data to suggest queries to users based on previous interests.

Recommendation systems are critical in many fields, such as e-commerce websites, where these systems attempt to suggest the best products that match the user's interest to improve sales. This methodology is also used in libraries and streaming services. These systems are usually based on a Utility Matrix, which captures a user's preference for a particular item offered by the service. However, the matrix has some blank spaces because users usually do not have recorded data about each item of the system. The goal of recommendation systems is to provide meaningful values for the blank spaces in the matrix.

There are various ways to create a recommendation system, including content-based and collaborative filtering approaches. Clustering is another option that can be considered to find groups of commonalities that can suggest one thing to another member of the same group. Each of these approaches must deal with the challenge of dealing with vast amounts of data and finding the best recommendation in the shortest amount of time, taking no more than a few seconds in most cases.

As recommendation systems rely mainly on existing data, a database is necessary for the algorithm's workflow. Relational databases are typically used and organized in tables consisting of rows and columns for easy architecture and more understandable data. This paper first demonstrates how locality-sensitive hashing can be used to improve the search for similar items to provide user recommendations, then shows how the incorporation of a content-based approach to build a hybrid recommendation system can benefit recommendation accuracy.

II. Related Study

The work that has been conducted so far in the field of recommendation systems has been focused on finding ways to exploit similarities in existing data to make recommendations. This prior knowledge can take on a variety of shapes; one well-known form is provided in terms of a utility matrix. It can also be obtained from other sources of information that draw on patterns that exploits similarities, such as user behaviours [5].

Depending on the kind of prior knowledge the system has regarding the problem that is being attempted to solve, there are primarily two categories in which recommendation systems can be categorized; these methods are content-based filteringand collaborative-filtering. In addition to these two methods in the last few years

This similarity measure can be adapted to work also for binary vectors, in fact a binary vector is a common and convenient way to represent a set. Given a universe set containing all the possible elements = { 1, 2, ..., }, any subset ⊆ can be represented

emerged the necessity to combine the benefits of the two aforementioned methods into hybrid recommendation systems.

Content-based methods use a combination of the features associated with each product and the ratings given by each user to provide suggestions. This method requires the construction of user profiles that outline each user's preferences as well as item profiles that highlight an item's key features.

The collaborative-filtering method pushes the system to only consider the relationships between users and items, ignoring either the features of users or the characteristics of items: with this approach, the utility matrix's relationships are the sole thing being considered. It is possible to create collaborative-filtering recommendation systems by either locating similar items that may be of interest based on the user's past interests, this is called item-item collaborative filtering, or by utilizing user similarities to recommend products that another user has rated highly, this is user-user collaborative filtering; in both cases the similarity of items and users is determined by the similarity of the ratings given by one users to an item.

Collaborative filtering and content-based approach can be combined together to produce a hybrid recommendation system, which aim to combine the advantages of both the approaches to provide recommendations that are even more accurate. Depending on the type of problem being considered, different combining strategies may be used [5].

As was already anticipated, the fundamental component of a recommendation system is the kind of prior knowledge provided for a particular problem that enables the algorithm to make reasoning about the given information. In the case of recommendation systems, the prior knowledge is commonly embedded in a so called utility matrix. Given a set of users and a set of items, the utility matrix can be formally represented by a utility function that associates user'$su \in U$ and items $i \in I$ to a rating $r \in R$, where is a set of valid ratings, i.e.$f: U \times I \rightarrow R$.

2.1 Similarity measures

The notion of similarity holds immense significance for the recommendation system, irrespective of the chosen technique for generating recommendations. In the recent years, data mining studies have emphasized the importance of selecting suitable methods for similarity measurement as a critical and efficient factor for achieving quality outcomes. The literature [6] suggests that the choice of similarity measures is contingent upon the nature of the problem, implying that a measure that produces satisfactory results for one data structure may not perform as well for another.

2.1.1 Jaccard similarity:

The Jaccard similarity, often referred toas the Jaccard index, is a well-established method to measure thesimilarity between sets. The similarity of two given sets A and B,can be measured as the intersection of the sets divided by the unionof the sets

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \qquad (1)$$

as a n-dimensional vector$v$® where each component of the vector is 1if the ith element from the universal set is present in S, 0 otherwise.

More formally,

$$\forall 1 \leq i \leq n: x_i \in S \implies \vec{v}[i] = 1$$

From thisdefinition it's possible to derive that the Jaccard similarity for abinary vector is the number of times in which both vectors have 1in the same component, divided by the total amount of timesat least one vector has 1 in the ith component. It's clear that theJaccard similarity for vectors makes sense only in the case of vectorsmade only of 0s and 1s.

2.1.2 Cosine similarity:

The cosine similarity is a similarity measure between two n-dimensional vectors $a$and b. Let's denote $ai$the ith component of the vector a. The Cosine similarity of a and $b$corresponds to the angle between the two vectors

$$S(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \, ||\vec{b}||} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}} \qquad (2)$$

This similarity measure is defined for vectors of any form, unlikethe Jaccard similarity, which is defined on sets and consequentlyon binary vectors

2.2 Fast similarities search

The problem of identifying similarities between objects is crucial in various fields. These similarities are useful in clustering, detecting plagiarism [14], identifying almost identical web pages [8], and developing recommendation systems that suggest items based on past interests. To handle large amounts of data, such as in data mining, current research in this field relies on approximate algorithms. This study first proposes an approach that enhances similarity search for recommendation systems by using locality-sensitive hashing (LSH) technique along with two locality-sensitive functions that are suitable for Jaccard similarity and Cosine similarity - minHash [2] and simHash [10], respectively. Furthermore, the paper explores how to develop a hybrid recommendation system by integrating content-based and collaborative filtering methodologies.

III. Problem Statement

In this section, the problem is formally defined after a few fundamental concepts are formally defined.

3.1 Definition: *(Relational table).* Given a set of n domains D ={D_1, D_2, ..., D_n}, a relational table R (A: d_1, B: d_2, ...) is defined onthese n domains as a subset of the Cartesian product of D, where adomain di ∈D is the set of possible values that a data element maycontain [4].

$$R(A:d_1, B:d_2, \ldots) \subseteq \times\{D_i : i = 1, 2, \ldots, n\} \qquad (3)$$

The notion R (A: d_1, B: d_2) denotes a relational table made oftwo attributes A and B taking values respectively from domain $d1$and $d2$, i.e. A∈$d1$ and B∈$d2$.

3.2 Definition *(Query).*

To retrieve data from a relational table, a user can submit a query consisting of conditions on the attributes. A query, denoted as q, can be described as a function that takes a set of conditions C on the two attributes of a relational table R as input and produces a subset of tuples from R that satisfy the conditions in C. Typically, queries are formulated as a set of conjunctions.

$$q(C) := \bigwedge_{c_i \in C} c_i \qquad (4)$$

Throughout the entire paper, the set of all the queries will bereferred to as Q.

3.3 Definition *(Rating function).*Prior to defining the problem, it is crucial to establish the meaning of a rating function. The rating function, denoted by r, is a specialized version of the utility function discussed in the related work section. It links users from a set of users, S, and queries from a set of queries, Q, to ratings between 1 and 100. Ratings serve as a representation of a user's viewpoint on the outcome of a query. Specifically, a rating of 1 indicates an unsatisfactory result, while a rating of 100 indicates that the user is content with the result.

$$r : S \times Q \rightarrow \mathbb{N} \in [1, 100] \qquad (5)$$

A utility matrix U can be defined with the help of a rating function, where U is a matrix of size |S | × |Q|. Each cell of this matrix, denoted by U_{ij}, corresponds to a rating ranging from 1 to 100 assigned by user i ∈S to query j∈Q. In case a user has not rated a particular query, the corresponding cell U_{ij} remains empty.

The primary objective is to develop a highly advanced query recommendation system that can be seamlessly incorporated into a DBMS. This system is intended to leverage the resemblances in user ratings to provide query recommendations to other users based on the given inputs. To illustrate the process, Figure 1 clearly outlines the workflow of the recommendation system. The process involves a user submitting a query to a database through the DBMS, which then retrieves the relevant data. After the user receives the query results, they are given the option to rate it. Based on the ratings, the recommendation system generates appropriate recommendations for the user.

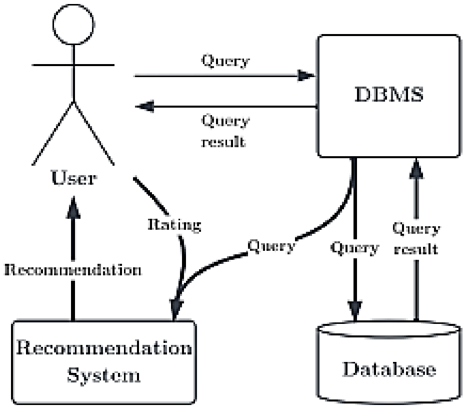


Figure 1: The workflow of the query recommendation system

IV. SOLUTIONS

This section presents several approaches to the problem, beginning with a basic naive strategy that is subsequently refined to produce a final solution. The approach involves starting with a conventional method for handling recommendation systems and gradually enhancing it by incorporating numerous data mining techniques designed to manage large quantities of data.

4.1 Naive solution

To address the aforementioned problem, the most straightforward approach is to build a recommendation system using traditional methodologies. The two primary approaches for such tasks are content-based and collaborative filtering, as discussed in the related work section. As the input of the problem is a utility matrix with partially filled ratings, the initial choice was to use collaborative filtering instead of content-based methods to avoid defining item and user profiles. Collaborative filtering allows for the creation of either an Item-Item or a User-User recommendation system, where the missing values (i.e., blank cells Uij in the utility matrix) are filled using one of the following techniques:

(1) Item-Item collaborative filtering: This method looks for the top K queries that are most similar to query j in the utility matrix. The average of the Top-K items rated by user i is then used to fill the value for Uij.

(2) User-User collaborative filtering: This method is similar to the previous one in concept but differs slightly in that it looks for the Top-K users that are similar to user i instead of the Top-K items that are similar to item j. The value Uij is then filled with the average of the K most similar users to user j that have rated item j.

Both these strategies require defining a similarity measure to locate a neighbourhood of K similar items or users. After considering several factors, including explain ability of the model and ease of finding items of the same type, the Item-Item approach was chosen over the User-User approach. This decision was based on the following reasons:

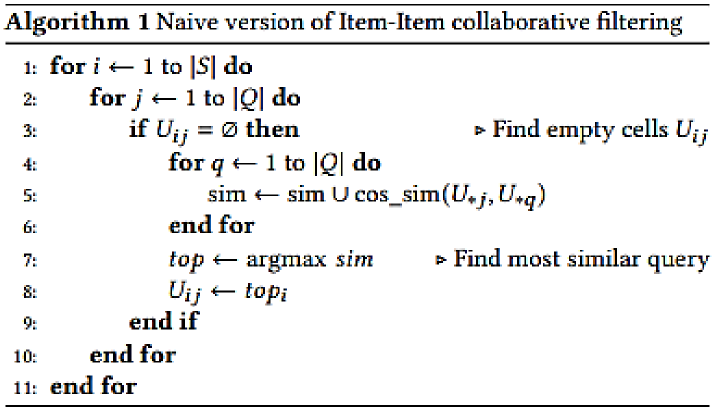
- The Item-Item approach is less likely to experience significant changes in the item's neighbourhood compared to the User-User approach. This is because the User-User approach is more effective in making recommendations to users with unique tastes.
- The Item-Item approach is more explainable. It is easier to explain why an item has been recommended to a user based on previously rated goods than to describe why a person has similar preferences to another.

- Finding items of the same type is easier than finding users who only like items of a particular type, making item-item similarity more reliable.

- The systems considered in the problem definition typically have more users than items, and the User-User approach becomes more expensive as the number of users increases.

4.1.1 Choosing a similarity measure.It is important to carefully consider several factors before selecting the optimal similarity measure for making a sound neighbourhood choice. Section 2.1 introduced two similarity measures, namely the Jaccard similarity and the Cosine similarity, for sets (including binary vectors) and vectors in Euclidean space, respectively. The Cosine similarity is deemed the most appropriate similarity measure for this problem, as it is based on item ratings that correspond to a column of the utility matrix, i.e., a vector. Although the Jaccard similarity is not ideal for handling vectors, a slightly modified version of this similarity measure is presented in the next section to address vectors of user ratings.

4.1.2 Pseudocode.The concept of Item-Item collaborative filtering involves examining all the cells in the utility matrix. When a cell Uij does not contain a value, its rating is estimated by averaging the ratings provided by user i for the K most similar queries to query j. The similarity between two queries is measured using the cos_sim method, which requires two vectors for comparison. The vector representing the ratings provided by all users for a particular query j is identified as $U{*}j$. The pseudocode assumes a neighbourhood size of K equal to 1, meaning that the rating for Uij is predicted using the rating provided by user i for the query that is most similar to query j.



Algorithm 1 Naive version of Item-Item collaborative filtering

```
for $i \leftarrow 1$ to $|S|$ do
    for $j \leftarrow 1$ to $|Q|$ do
        if $U_{ij} = \varnothing$ then                    ▷ Find empty cells $U_{ij}$
            for $q \leftarrow 1$ to $|Q|$ do
                sim ← sim ∪ cos_sim$(U_{*j}, U_{*q})$
            end for
            $top \leftarrow \operatorname{argmax} sim$          ▷ Find most similar query
            $U_{ij} \leftarrow top_i$
        end if
    end for
end for
```

To determine query similarities, this approach involves calculating the cosine similarity between possible query pairs. The expense of computing the cosine similarity is equivalent to that of computing the dot product of two queries. Each query corresponds to a column vector with $|S|$ components, and its cost is $O(|S|)$. The overall cost to forecast a single missing value in the utility matrix is $O(|S|\cdot|Q|^2)$. It is evident that forecasting all of the missing utility matrix values is expensive, particularly when dealing with a large number of users and queries.

4.2 MinHash for LSH

The algorithm's time complexity is the main challenge with the previous solution. Specifically, the computation of the similarity between all possible query combinations in the utility matrix drives the cost of identifying the top K queries that are similar to a given query. To overcome this issue, this section presents a more efficient technique that utilizes minHash [2] and LSH to enhance the speed of the similarity search while sacrificing some accuracy. LSH aims to eliminate the need for comparing items that are undoubtedly dissimilar and instead focuses only on pairs of items that are likely to be similar, which are called candidate pairs. However, finding these candidate pairs in the original utility matrix is costly. Therefore, the proposed approach applies LSH to a signature matrix rather than the original utility matrix to reduce computation expenses.

4.2.1 MinHash.Locality Sensitive Hashing (LSH) typically operates on compact signatures that are derived from a characteristic matrix, which is a common representation of data used to describe the characteristics of items. In the case of documents, the characteristic matrix can be viewed as a matrix C, where each row represents a potential word from a given vocabulary, and each column represents a document. If the element Cij is nonzero, it means that the word i is present in document j. The concept of the characteristic matrix is related to the utility matrix of recommendation systems, where each row corresponds to a user and each column corresponds to an item (a query). If the element Cij is nonzero, it means that user i has rated item j. Therefore, the MinHash technique can be used to obtain a compressed representation of the utility matrix by exploiting the correlation between the two matrices.

Working with the utility matrix directly can be impractical due to its massive size. Therefore, the characteristics matrix is replaced with a compressed version called a signature matrix. The goal is to ensure that the similarity of items in the signature matrix is as close as possible to the similarity of items in the original characteristics matrix. Although some information from the characteristics matrix may be lost during the compression process, the MinHash technique can estimate the Jaccard similarity between two items accurately by adding more signatures to the signature matrix.

The MinHash technique generates a signature matrix δ by taking a number of permutations π of the rows of the utility matrix U. For each item, it finds the first row in the permutation πt of $U{*}j$ that has a value of one. This process is repeated for a number of permutations H to create the signature matrix. It should be noted that the MinHash technique was originally designed for the Jaccard similarity, which is a measure of the similarity between two sets.

$$\delta_{tj} \leftarrow \min_{\pi}(U_{\pi_t j} = 1) \qquad (6)$$

Example. Given a utility matrix U of three items and a set of two permutations π, the signature matrix δ is produced as follows.

$$\pi = \begin{array}{|c|c|} \hline \pi_1 & \pi_2 \\ \hline 4 & 6 \\ \hline 3 & 3 \\ \hline 6 & 4 \\ \hline 2 & 1 \\ \hline 1 & 5 \\ \hline 5 & 2 \\ \hline \end{array} \qquad U = \begin{array}{|c|c|c|} \hline i_1 & i_2 & i_3 \\ \hline 1 & 1 & 1 \\ \hline 0 & 1 & 1 \\ \hline 1 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 1 & 1 \\ \hline 1 & 0 & 1 \\ \hline \end{array} \qquad \delta = \begin{array}{|c|c|c|} \hline i_1 & i_2 & i_3 \\ \hline 4 & 1 & 1 \\ \hline 1 & 3 & 2 \\ \hline \end{array}$$

4.2.2 Permutations generation.It may not be practical to generate permutations and store them in memory when dealing with utility matrices that have a large number of users. To address this issue, a potential solution is to mimic the impact of random perturbations. This can be done by selecting H hash functions, with each function associated with a permutation, and returning the index of a specific row in that permutation.

$$\pi(x) = (ax + b) \bmod c \qquad (7)$$

where a, b and c are random value less than the number or usersin the utility matrix,
i.ea, b, $c \leq |S|$.

4.2.3 Adapted MinHash for ratings.Section 2.1.1 explains that the Jaccard similarity is defined for sets and binary vectors. However, there is an issue with the MinHash formulation presented above because it assumes that the characteristics matrix consists only of binary values of 0s and 1s. This assumption creates a problem when dealing with the utility matrix U, as described in Section 3, which contains ratings in the range of [1, 100]. To address this issue, one solution is to transform the utility matrix U into a binary matrix Ub, where a value of 1 in Ub indicates that the user i has liked query j. The threshold parameter T is also introduced, which determines whether a query is liked or not based on its rating value. Since the threshold value can vary, it is important to set a value for T to determine whether a query is considered liked, where a rating greater than T indicates a liked query.

$$\widehat{U}_{ij} = 1 \iff U_{ij} \geq T \qquad (8)$$

Example.By considering a utility matrix U with ratings ranging between 1 to 100, where the absence of rating values is indicated by 0, and a positivity threshold of $T = 50$, a new utility matrix, Ub, can be derived for the ratings that are greater than or equal to T. This transformation can be achieved using Eq. 8, as presented below.

$$U = \begin{array}{|c|c|c|} \hline q_1 & q_2 & q_3 \\ \hline 80 & 5 & 12 \\ \hline 0 & 67 & 0 \\ \hline 46 & 85 & 0 \\ \hline 0 & 55 & 65 \\ \hline 35 & 10 & 90 \\ \hline 95 & 0 & 45 \\ \hline \end{array} \qquad \widehat{U} = \begin{array}{|c|c|c|} \hline q_1 & q_2 & q_3 \\ \hline 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline 0 & 0 & 1 \\ \hline 1 & 0 & 0 \\ \hline \end{array}$$

The practical computation of the modified utility matrix is expensive, particularly given $|S|$ and $|Q|$, which represent the number of users and queries in the utility matrix. The cost of converting the matrix is $(|S| \cdot |Q|)$, which is a quadratic cost proportional to the input size. One potential solution to this issue is to employ a slightly modified version of MinHash called MinHash with threshold. This approach assigns the index of the first row in the permutation with a rating above a threshold level T to the signature matrix entry δtj.

$$\delta_{tj} \leftarrow \min_{\pi}(U_{\pi_t j} \geq T) \qquad (9)$$

The signature matrix that is produced provides a useful estimation for a modified version of the Jaccard similarity measure. This version considers positive ratings as 1 if they exceed the threshold value T, while elements smaller than T are considered 0.

Algorithm 2 presents a pseudocode implementation of this approach. The method involves iterating over all queries and H permutations of the rows in the utility matrix to identify the first row in the permuted order of the matrix that has a rating greater than the threshold T. The loop on line 4 typically requires fewer iterations than the total number of users $|S|$ in the utility matrix. The index i is utilized as an input to generate the permuted index u, which increments by one with each iteration until it reaches the maximum value of $|S|$.

Algorithm 2 MinHash algorithm: signature matrix generation

```
1:  for $j \leftarrow 1$ to $|Q|$ do
2:      for $h \leftarrow 1$ to $H$ do
3:          $i \leftarrow 1$                        ▹ $i$ is the original index of the rows
4:          while $\delta_{hj} = \varnothing \cap i \leq |S|$ do
5:              $u \leftarrow \pi_h(i)$                ▹ $u$ is the permutation of row $i$
6:              if $U_{uj} \geq T$ then
7:                  $\delta_{hj} \leftarrow i$
8:              end if
9:              $i \leftarrow i + 1$
10:         end while
11:     end for
12: end for
```

4.2.4 Locality Sensitive Hashing.

After generating the signature matrix using the modified version of MinHash with a threshold of T, the next step is to utilize Locality Sensitive Hashing (LSH) to hash queries in the signature matrix multiple times. This ensures that queries with similar ratings are more likely to be hashed to the same bucket compared to dissimilar queries. The signature matrix is divided into b bands, with each band consisting of r rows. Each band is then hashed column-wise into a set of buckets specific to that band, and queries that are hashed to the same bucket are considered candidate pairs. These candidate pairs provide information that can reduce the time needed to detect similar queries. Therefore, each query i is tested for similarity with every query j that lies in the same bucket as query i for at least one band, i.e. candidate pairs.

The number of bands b and the number of rows for each band r are critical in determining the number of candidate pairs for each query. These two parameters are related, as $b \cdot r = H$, where H is the number of permutations, or the number of rows in the signature matrix δ. A larger number of bands and fewer rows per band will increase the number of candidate pairs detected by LSH. Conversely, fewer bands and a higher number of rows per band will reduce the number of detected candidate pairs. Therefore, it is essential to choose a suitable trade-off for the b parameter or equivalently the r parameter. Factors such as the size of the signature matrix and the distribution of data in the original utility matrix must be considered while selecting these parameters. If all queries are very dissimilar, even a small value of r will capture a considerable number of candidate pairs. On the other hand, if every query is very similar, a low value of r is insufficient, and the number of rows per band should be increased to prevent an excessive number of similar items from hindering the goal of finding a good compromise on the number of similar items.

4.2.5 Making recommendations.The final step involves providing recommendations through the item-item collaborative filtering technique, similar to the one used in the Naive approach, except that the iterative process for identifying the queries with the highest degree of similarity only focuses on the candidate pairs of a specific query, rather than all possible queries. In terms of time complexity, if the worst-case scenario occurs, the number of candidate pairs identified for each query may be equivalent to the set of all queries, which could cause the algorithm to revert to the Naive approach. However, this is just a limiting case, as the number of candidate pairs can be reduced by adjusting the number of bands b and the number of permutations H appropriately.

4.3 SimHash for LSH

The utilization of Jaccard similarity in conjunction with MinHash is deemed unsuitable for the specific problem being analyzed. As expounded in section 4.1.1, Jaccard similarity is not the optimal measure for vector similarity. The revised version of Jaccard similarity that considers the threshold parameter T, fails to gauge the similarity between two queries in certain instances. This issue arises when two queries share an almost identical rating, such as 49 and 51, but the threshold parameter T=50 differentiates them. The root cause of this problem is the transformation of the utility matrix into a matrix of binary values that indicate user preference. The Cosine similarity is a more appropriate similarity measure for the problem at hand, as it can be calculated on vectors of any type.

The structure of this algorithm is quite similar to the one proposed for the solution using MinHash for LSH. The algorithm aims to create a signature matrix δ that mirrors the original utility matrix. Consequently, when two queries are compared, their similarity is conserved as much as possible in the signature matrix. The computation of the signature matrix must be quick and cost-effective to ensure that the algorithm can benefit from using the signature matrix instead of the original utility matrix. Once the signature matrix is computed, the next step is to subject it to LSH to identify local similarities that will generate a set of candidate similar queries.

4.3.1 SimHash intuition.The SimHash algorithm is based on the concept of generating a signature matrix with H rows, where the aim is to partition the vector space for queries into H hyperplanes that pass through the origin. This results in each query being assigned a region either above or below the hyperplane. The signature matrix is then populated with each query's position in relation to the corresponding hyperplane, denoted by the i-th row in the signature matrix δ. The main idea is that if two queries are similar, it is more likely that they will be in the same region of a random hyperplane. Thus, on most randomly generated planes, two queries that are close to each other will belong to the same region. An illustration in Figure 2 shows that for any arbitrary number of hyperplanes and two similar queries $q1$ and $q2$, the probability of the queries falling on different sides of the hyperplanes is relatively low compared to the total number of hyperplanes. Only the red plane in this specific example separates the two queries into two different sides. SimHash has been shown to be a locality-sensitive hash function that provides an approximate measure of cosine similarity [13] [7].

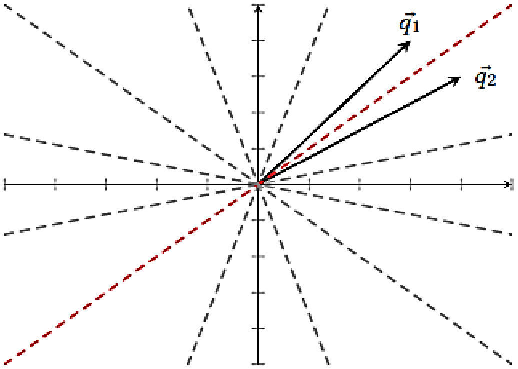


Figure 2: Two similar queries $\vec{q}_1$ and $\vec{q}_2$ are more likely to fall into the same region of any random hyperplane than when they are dissimilar.

4.3.2 Signature matrix computation.The generation of the signature matrix is a simple process based on the following intuition: create H random hyperplanes and populate the ith row of the signature matrix with the query's position relative to the hyperplane. From a formal perspective, a hyperplane is represented by an n-dimensional vector that is orthogonal to the hyperplane in an n-dimensional space. The position of a query vector qj in relation to a general hyperplane Ht, represented by its orthogonal vector □t, is determined by the sign of the projection of vector qj on □t. Since the dot product is used to calculate the projection of a vector on another one, the signature matrix cell corresponding to the ith hyperplane represented by its orthogonal vector □t and the jth query vector qj is calculated in the following manner.

$$\delta_{tj} \leftarrow \text{sign}(\vec{h_t} \cdot \vec{q_j}) \qquad (10)$$

The sign function produces a result of 1 when its input argument is positive, while it returns -1 when the input is negative. These values signify that the query vector is situated on the positive or negative side of the hyperplane, correspondingly.

$$\text{sign}(\vec{h_t} \cdot \vec{q_j}) = \begin{cases} 1 & \text{if } \vec{h_t} \cdot \vec{q_j} \geq 0. \\ 0 & \text{otherwise.} \end{cases} \qquad (11)$$

Example.In Figure 3, there is a representation of a generic hyperplane $H1$ in two-dimensional space, which is characterized by its orthogonal vector □1 ∈ R2, and there are two queries $q1$, $q2$ ∈ R2, which are rated by two different users. The queries' position with respect to the hyperplane is determined by projecting $q1$ and $q2$ onto □1, and the sign is evaluated by computing the dot product between the two vectors. In this case, both queries are found to be on the positive side of the hyperplane $H1$, and hence the corresponding cell in the signature matrix is assigned a value of 1. Using the notation of signature matrix cell δij, where the it□ row corresponds to the it□ hyperplane, and the jt□ column corresponds to the jt□ query, the signature for the first plane $H1$, and the aforementioned queries are $\delta 11 \leftarrow 1$ and $\delta 12 \leftarrow 1$.

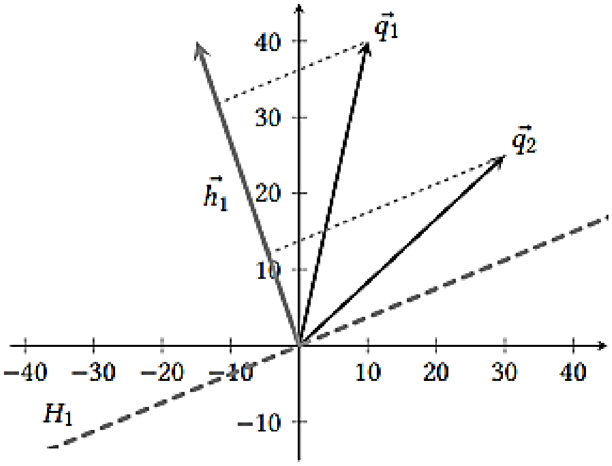


Figure 3: Projection of queries $\vec{q}_1$ and $\vec{q}_2$ onto $\vec{h}_1$ to find their position with respect to the hyperplane H_1.

Pre-processing.To avoid the issue of all query vectors being located in the positive quadrant of the space, it is necessary to normalize the ratings of the utility matrix. Currently, a query in the utility matrix is represented by a vector of positive ratings between 1 and 100, resulting in all queries being located in the upper rightmost quadrant in the 2-dimensional example depicted in Figure 3. However, this creates a problem as the projection on any hyperplane defined by an orthogonal vector falling within the quadrant will always have a positive sign, regardless of the query. To address this issue, the ratings of the utility matrix must be normalized into the range of [−50, 50]. This can be done by subtracting the average value of 50 from all the ratings in the utility matrix that are not 0 (missing rating). As a result, a new utility matrix Ub can be computed using this normalization technique.

$$\widehat{U}_{ij} \leftarrow U_{ij} - 50 \quad \forall i, j \text{ s.t. } U_{ij} \neq 0 \qquad (12)$$

In Figure 4's example, the left side displays the initial utility matrix U, while the right side shows the new utility matrix Ub that has been centered at 0.

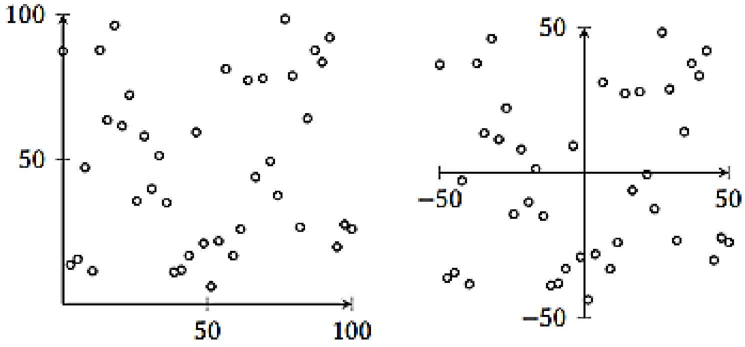
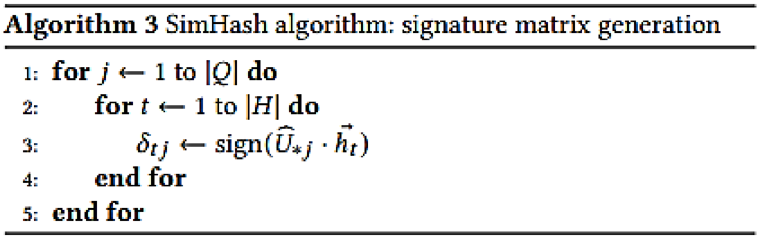


Figure 4: The rating of the queries that originally are in the range [1, 100] are centered in the space to obtain ratings in [−50, 50].

Pseudocode. The SimHash algorithm is notably simpler than the MinHash-based approach used to generate the signature matrix. To compute the signature matrix, we assume that H is a set of $|H|$ randomly generated hyperplanes, Ub is the pre-processed utility matrix, and we use the function sign defined by Equation 11.



Algorithm 3 SimHash algorithm: signature matrix generation

```
for $j \leftarrow 1$ to $|Q|$ do
    for $t \leftarrow 1$ to $|H|$ do
        $\delta_{tj} \leftarrow \text{sign}(\widehat{U}_{*j} \cdot \vec{h}_t)$
    end for
end for
```

V. Experimental Evaluation

In this section, we evaluate the proposed solutions presented in the previous sections and showcase their effectiveness in a simulated real-world scenario using realistic data. The algorithms investigated in this study were tested on a computer system that features an Intel Core i7-6700HQ CPU @ 3.5GHz with 16 GB of RAM, and their performance was assessed accordingly.

5.1 Datasets

To evaluate the proposed solutions, the algorithms were tested on several datasets that were artificially constructed to closely resemble a real-world scenario. Specifically, a synthetic dataset was generated by utilizing the Scikit-Learn library's make blobs function [11], which allows for the creation of correlated data. This approach was chosen as it reflects the observation that in a real-world scenario, relational tables representing people often contain groups of individuals who share similar characteristics, such as eye color, height, and more.

To ensure the query set was realistic, the conditions for each query were constructed by randomly selecting a number of features (including none), assigning each one a value with a 99% probability, and using the remaining probability to provide a random value (even if not an admitted value for the feature). This approach results in queries that may return a small number of rows, all rows, or none at all. It is important to note that when a query has no conditions, all rows in the relational database are returned.

The utility matrix forms the core of the dataset, and it is generated by creating categories of users who tend to share similar tastes. Furthermore, users often act following patterns, which is why the idea is to split users into three categories. Firstly, 60% of user's rate queries that return similar rows in the same way. If two queries, q1 and q2, return the majority of the rows in common, the user who rates query q1 with rating r will rate query q2 with a rating that differs from r by a small factor γ, such as $\gamma = 5$. Secondly, 30% of users grade queries proportionately with the number of rows returned. Lastly, the remaining 10% of users assign random ratings to queries.

5.1.1 Synthetic dataset characteristics. To generate a synthetic dataset, we followed the same process mentioned earlier, where a relational table with 100 attributes was created with 10000 rows, and integer values were used to populate the table. This was a logical choice since relational tables typically contain values from a specific domain, and representing cities using names or integer values is equivalent. The dataset was designed to simulate a system with 500 users who generated 2000 queries for the DBMS. As a result, the utility matrix consisted of 500 rows and 2000 columns representing the queries. Additionally, a script was developed to extract important information from the relational database and queries. The results showed that out of the 2000 queries, 721 produced at least one row, with an average of 3311 rows returned per query. Although these figures may seem small, they are adequate to assess the effectiveness of the proposed algorithms and demonstrate their performance in various scenarios. The dataset can be divided into smaller subsets for more specific experiments.

5.1.2 Real dataset.Experimental observations of real data are typically stored in relational tables. Therefore, this work evaluates proposed solutions on a dataset that replaces the synthetic table with a real one. Specifically, the relational table used in this study is taken from the 1994 Census Bureau's relational database [1], which contains 14 columns representing individual attributes such as age, sex, marital status, and income. While the original dataset had over 50,000 individuals, the study was conducted on a scaled-down version of 10,000 individuals to measure performance. The remaining components of the dataset were generated using the same methodology as the synthetic table, resulting in a total of 2,000 queries and 500 users. The dataset revealed that out of 2,000 search queries, 908 produced at least one row, and the average number of rows returned by these queries was 3,649.

5.2 Fast query similarity search with LSH

In this section, we analyze the algorithm's initial building block and compare it to the baseline method used for determining query similarity in collaborative filtering. The primary objective is to demonstrate the effectiveness of the LSH approach, which is explained in sections 4.2 and 4.3, in significantly improving the naïve solution without LSH described in section 4.1. Firstly, we show that utilizing LSH with MinHash reduces the time required to find similar queries for collaborative filtering using the Jaccard similarity as a similarity measure for the queries. Secondly, we demonstrate that SimHash with LSH outperforms its corresponding baseline solution, which involves trying all possible query combinations to identify similar ones. The focus of this section is to identify the most related queries for each query based on the utility matrix ratings given by the users. To simplify the objective, we slightly modify the goal to locate the most similar query instead of the K most similar ones, while still considering the original solutions.

5.2.1 LSH with MinHash.In this section, we conduct a comparison between the LSH MinHash algorithm and its naïve counterpart, which computes Jaccard similarity to identify the most similar query. We evaluate the time efficiency of these two algorithms in two different scenarios: one involves a varying number of queries with a constant number of users, while the other involves a varying number of users with a constant number of queries. To simulate smaller datasets, we divide the input dataset into smaller fractions and test the algorithms on those fractions. In the case of MinHash and Jaccard similarity, a dataset of up to 1000 queries and 500 users is sufficient to observe time complexity growth.
We plot the execution time of both methods in Figure 5 for a scenario with a constant number of 500 users and a variable number of queries. For the other scenario, we keep the number of queries constant at 1000, and vary the number of users, as shown in Figure 8. In both scenarios, MinHash generates a signature matrix with 200 rows, corresponding to a total of 200 random perturbations. Furthermore, we configure the LSH bands to have six rows per band, which divides the signature matrix into 33 bands.
As illustrated in Figures 5 and 6, the plots clearly demonstrate the time efficiency improvements brought about by the use of LSH over computing similarity between every pair of queries. Interestingly, both approaches perform similarly in the early stages with smaller datasets, but one algorithm outperforms the other in scenarios with larger datasets.

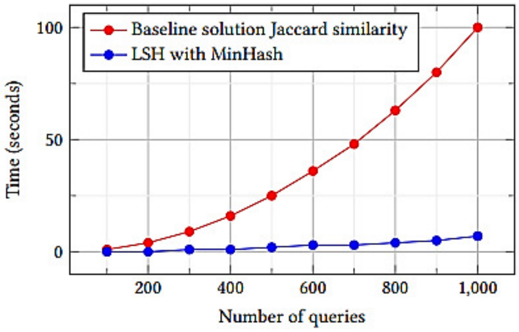


Figure 5: Time performance of the Naive algorithm compared with LSH using MinHash – Variable number of queries

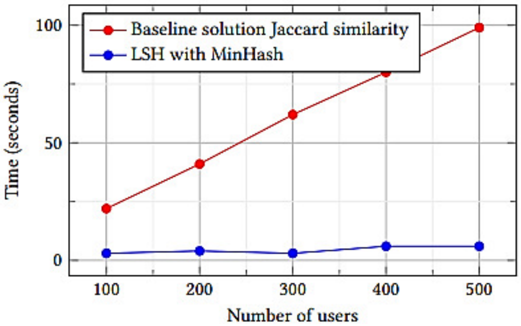


Figure 6: Time performance of the Naive algorithm compared withLSH using MinHash – Variable number of users

5.2.2 LSH with SimHash.

In this section, we compare the SimHash technique with its naïve counterpart, which computes the cosine similarity between all queries in the utility matrix. The results demonstrate that utilizing LSH to identify similar queries improves the time performance of the algorithms significantly. To further evaluate the algorithm's efficiency, we conducted experiments on a dataset of 2000 queries instead of 1000, using 200 hyperplanes created randomly for SimHash, which corresponds to 200 rows of the signature matrix. LSH divides the matrix into 13 bands, with 15 rows for each band. The decision to increase the number of rows per band is due to SimHash's signature matrix of zeros and ones, which increases the probability of two queries in the same band hashing to the same bucket. In contrast, MinHash's signature matrix comprises integers between 1 and the total number of users in the system, making it less likely that two columns in the same band would hash to the same bucket. Similar to the previous instance, the LSH-based approach outperforms the naïve technique that determines similarity between all possible query combinations. The time performance of the two algorithms as the number of queries and users increases is compared in Figures 7 and 8.

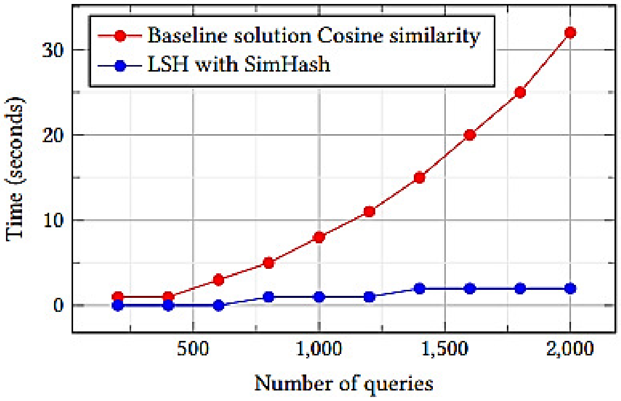


Figure 7: Time performance of the Naive algorithm compared withLSH using SimHash – Variable number of queries

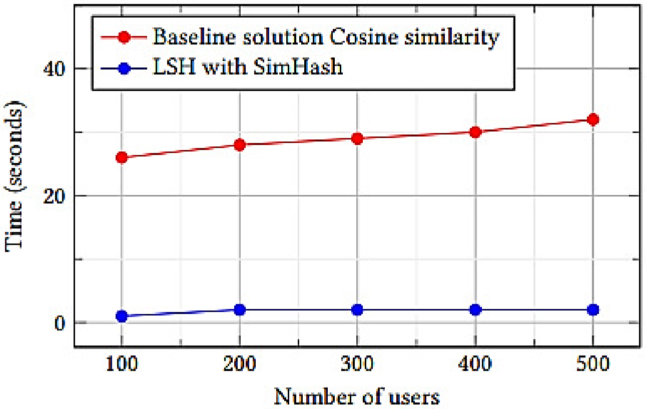


Figure 8: Time performance of the Naive algorithm comparedwith LSH using SimHash – Variable number of users

VI. Conclusion and Research Scope

The research conducted in this study explored the integration of multiple data mining approaches to develop an advanced recommendation system that generates useful recommendations. The results of the study were satisfactory, with two distinct algorithms being produced in a block-by-block manner. The first algorithm was based on collaborative filtering with LSH, which produced high accuracy and good time performance. The second solution integrated content-based approach with the previous algorithm to create a hybrid recommendation system with even higher accuracy but lower time performance. The experiments conducted demonstrated that combining content-based and collaborative filtering methods results in more accurate recommendations. The study also showed that standard methodologies, like the naïve algorithm using only collaborative filtering, are inapplicable for large datasets, and combining various strategies like LSH provides solutions that can handle massive datasets. Additionally, the study proposed a solution for scoring the importance of a new query sent to the system. However, providing useful query recommendations for DBMS is a challenging task that depends on several factors. Further work is required to implement and fine-tune the proposed system fully. Future research can focus on implementing the solution using the map-reduce framework for efficient and scalable processing of large datasets. Additionally, the second part of the algorithm, regarding query importance, requires proper evaluations, leaving scope for further research.

References

[1] Barry Becker. 1996. Census Income Data Set. https://archive.ics.uci.edu/ml/datasets/Census%2BIncome.

[2] Andrei Broder. 1997. On the Resemblance and Containment of Documents.Proceedings of the International Conference on Compression and Complexity ofSequences. https://doi.org/10.1109/SEQUEN.1997.666900

[3] Edgar F. Codd. 1970. A relational model of data for large shared data banks.Communication of the ACM 13, 6 (1970).

[4] E. F. Codd. 1979. Extending the Database Relational Model to Capture MoreMeaning. ACM Trans. Database Syst. 4, 4 (dec 1979), 397–434. https://doi.org/10.1145/320107.320109

[5] Zhiyuan Fang, Lingqi Zhang, and Kun Chen. 2016. Hybrid Recommender SystemBased on Personal Behavior Mining. https://doi.org/10.48550/ARXIV.1607.02754

[6] Wael H. Gomaa and Aly A. Fahmy. 2013. Article: A Survey of Text SimilarityApproaches. International Journal of Computer Applications 68, 13 (April 2013),13–18. Full text available.

[7] Qixia Jiang and Maosong Sun. 2011. Semi-Supervised SimHash for EfficientDocument Similarity Search., Vol. 1. 93–101.

[8] J. Prasanna Kumar and P. Govindarajulu. 2013. Near-Duplicate WebPage Detection: An Efficient Approach Using Clustering, Sentence Featureand Fingerprinting. International Journal of Computational IntelligenceSystems 6, 1 (2013), 1–13. https://doi.org/10.1080/18756891.2013.752657arXiv:https://doi.org/10.1080/18756891.2013.752657

[9] Jimmie D. Lawson and Yongdo Lim. 2001. The Geometric Mean,Matrices, Metrics, and More. The American Mathematical Monthly108, 9 (2001), 797–812. https://doi.org/10.1080/00029890.2001.11919815arXiv:https://doi.org/10.1080/00029890.2001.11919815

[10] Gurmeet Singh Manku, Arvind Jain, and Anish DasSarma. 2007. DetectingNear-Duplicates for Web Crawling. In Proceedings of the 16th International Conference on World Wide Web (Banff, Alberta, Canada) (WWW '07). Association forComputing Machinery, New York, NY, USA, 141–150. https://doi.org/10.1145/1242572.1242592

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M.Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J.Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: MachineLearning in Python. Journal of Machine Learning Research 12 (2011), 2825–2830.

[12] Elaine Rich. October 1979. User modeling via stereotypes. In Cognitive Science,Vol. 3. 329–354.

[13] SadhanSood and Dmitri Loguinov. 2011. Probabilistic Near-Duplicate Detection Using Simhash. In Proceedings of the 20th ACM International Conferenceon Information and Knowledge Management (Glasgow, Scotland, UK) (CIKM'11). Association for Computing Machinery, New York, NY, USA, 1117–1126.https://doi.org/10.1145/2063576.2063737

[14] Du Zou, Wei-jiang Long, and Zhang Ling. 2010. A cluster-based plagiarismdetection method - Lab report for PAN at CLEF 2010, Vol. 1176