PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
الجمهـــوريـــــة الجزائـريـــــة الديـمقـراطيـــــة الشـعـبيـــــة
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
وزارة التعليـم العالـــي والبحـــث العلمـي
HIGHER SCHOOL OF COMPUTER SCIENCE AND DIGITAL TECHNOLOGIES-BEJAIA
المـــدرسة العـــليا في علـــوم وتكنـولوجيـات الاعـــلام الالـــي بجـــاية

**Dissertation Submitted to the Department Of Computer Science in Partial Fulfillment of the Requirements for Engineer's Degree in Computer Science**

**Specialty: Artificial Intelligence and Data Sciences**

Submitted By:

**TOUATI Yanis**

# Exploring the Integration of Visual Data for Enhancing the Accuracy and Reliability of Recommender Systems

Supervised by:

**Dr.** BERBAGUE Chemseddine
LITAN Laboratory
HIGHER SCHOOL OF COMPUTER SCIENCE AND DIGITAL TECHNOLOGIES-BEJAIA (ESTIN)

**Members of jury:**

| | | |
|---|---|---|
| ▪ **Dr.** ISSAADI Badredine | President | LITAN Laboratory - ESTIN |
| ▪ **Mrs.** BOUGLIMINA Ouahiba | Examiner | LITAN Laboratory - ESTIN |
| ▪ **Mr.** KADRI Belkacem | Examiner | LITAN Laboratory - ESTIN |
| ▪ **Mrs.** KHENNOUCHE Feriel | Examiner | LITAN Laboratory – ESTIN |

Academic year: 2023/2024

# Acknowledgements

I would like to express my deepest gratitude to all those who have supported me throughout my academic journey and made this thesis possible.

First and foremost, I thank Allah for providing me with the strength, patience, and wisdom to pursue my studies and complete this thesis. Without His guidance and blessings, none of this would have been possible.

I am profoundly thankful to my parents, whose unwavering support, encouragement, and love have been the cornerstone of my education. Their belief in me has been a constant source of motivation.

To my sister and my dear friend Mohammed, I extend my heartfelt thanks for always being there for me, offering advice, and cheering me on through every step of this journey.

I am deeply indebted to my supervisor, whose guidance, insights, and patience have been invaluable. Your mentorship has significantly shaped my academic path and contributed greatly to the completion of this thesis.

I also wish to acknowledge every teacher and supervisor I have had since the beginning of my university education. This work is a testament to the knowledge and skills I have acquired under your tutelage. Your dedication to teaching and your commitment to my learning have been instrumental in my academic growth.

**Yanis**

# Abstract

This dissertation presents a multi-model hybrid recommendation approach aimed at enhancing recommendation systems' performance. The approach integrates content-based recommendation, collaborative filtering, and clustering techniques to exploit the strengths of each method. We discuss the architecture and motivation behind our approach, along with its objectives, expectations, limitations, and potential challenges. The proposed model incorporates visual data processing through deep feature extraction techniques, as well as textual-based models utilizing word and sequence embedding learning. We also detail collaborative filtering recommendation using similarity metrics, alongside clustering techniques including deep contrastive clustering. for further enhancement, we explore the hybridization of these methods to further improve recommendation accuracy. Experimental results and discussions are also displayed at the end of our paper, offering insights for future research in recommendation system development.

**Key-words**: *Artficial Intelligence, Collaborative Filtering, Visual Data Integration, Recommender Systems, Content-Based Recommendation, Feature Extraction.*

# Résumé

Cette dissertation présente une approche de recommandation hybride multimodèle visant à améliorer les performances des systèmes de recommandation. L'approche intègre des techniques de recommandation basées sur le contenu, de filtrage collaboratif et de clustering pour exploiter les atouts de chaque méthode. Nous discutons de l'architecture et de la motivation derrière notre approche, ainsi que de ses objectifs, attentes, limites et défis potentiels. Le modèle proposé intègre le traitement des données visuelles grâce à des techniques d'extraction de caractéristiques approfondies, ainsi que des modèles textuels utilisant l'apprentissage par intégration de mots et de séquences. Nous détaillons également les recommandations de filtrage collaboratif utilisant des métriques de similarité, ainsi que des techniques de clustering, notamment le clustering contrastif profond. pour une amélioration supplémentaire, nous explorons l'hybridation de ces méthodes pour améliorer encore la précision des recommandations. Les résultats expérimentaux et les discussions sont également présentés à la fin de notre article, offrant des perspectives pour les recherches futures sur le développement de systèmes de recommandation.

**Mots-clés** : *Intelligence Artificielle, Filtrage Collaboratif, Intégration de Données Visuelles, Système de Recommendations, recommandation basée sur le contenu, extraction de fonctionnalités.*

## ملخص

تقدم هذا المذكرة نهج توصية هجين متعدد النماذج يهدف إلى تعزيز أداء أنظمة التوصية. يدمج هذا النهج التوصية القائمة على المحتوى، والتصفية التعاونية، وتقنيات التجميع لاستغلال نقاط القوة في كل طريقة. نناقش البنية والدوافع وراء نهجنا، إلى جانب أهدافه وتوقعاته وقيوده وتحدياته المحتملة. يتضمن النموذج المقترح معالجة البيانات المرئية من خلال تقنيات استخراج الميزات العميقة، بالإضافة إلى النماذج النصية التي تستخدم تعلم تضمين الكلمات والتسلسل. نقوم أيضًا بتفصيل توصيات التصفية التعاونية باستخدام مقاييس التشابه، جنبًا إلى جنب مع تقنيات التجميع بما في ذلك التجميع المتباين العميق. لمزيد من التعزيز، نستكشف تهجين هذه الأساليب لزيادة تحسين دقة التوصية. يتم أيضًا عرض النتائج والمناقشات التجريبية في نهاية ورقتنا، مما يوفر رؤى للأبحاث المستقبلية في تطوير نظام التوصيات.

**الكلمات المفتاحية**:الذكاء الاصطناعي، التصفية التعاونية، تكامل البيانات المرئية، أنظمة التوصية، التوصية المستندة إلى المحتوى، استخلاص الميزات.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**AI** Artificial Intelligence

**ML** Machine Learning

**DL** Deep Learning

**RS** Recommender System

**CF** Collaborative Filtering

**CBF** Content-based Filtering

**SVD** Singular Value Decomposition

**IbCF** Item-based Collaborative Filtering

**UbCF** User-based Collaborative Filtering

**MBM** Memory-Based Methods

**SVM** Support Vector Machines

**GMM** Gaussian Mixture Model

**SOM** Self Organizing Map

**NLP** Natural Language Processing

**kNN** k-Nearest Neighbors

**ANN** Artificial Neural Networks

**RNN** Recurrent Neural Networks

**BERT** Bidirectional Encoder Representations from Transformer

**LSTM** Long Short-Term Memory

**PCA** Principal Component Analysis

**RMSE** Root Mean Squared Error

**MAE** Mean Absolute Error

$CB\text{-}CF_{SW}$ Cluster-Based Collaborative Filtering with Similarity Weighting

# Introduction

## General Presentation of The Thematic

In the realm of data analytics, a wide variety of data sources has emerged, including user behavior data such as click patterns, browsing histories, and purchase behaviors. One of the significant challenges is recommending items for new users, often referred to as cold users, who have limited interaction history. The main challenge lies in providing accurate recommendations for these cold users due to the sparse data available. To address this, advanced algorithms that combine various types of data are increasingly employed to enhance recommendation accuracy and relevance.

In addition to behavioral data, additional data sources like textual reviews and images further enrich the potential for comprehensive data analysis. Textual reviews offer deeper insights into user sentiments and preferences, while images can reveal aesthetic preferences and contextual usage of products. Integrating these additional data sources with traditional user behavior data through advanced algorithms significantly enhances the robustness of predictive models. Moreover, grouping users with similar behaviors or preferences can help address the sparsity problem for cold users. By leveraging all these data types, our goal is to develop a reliable and accurate recommender system.

## Problematic

The state-of-art analysis made prior to the making of this study has showed us that recommender systems have significantly evolved, with advancements in collaborative filtering, content-based filtering, and hybrid approaches, each offering unique strengths and challenges. The integration of visual data is emerging as a pivotal frontier in recommendation technology, promising to enhance recommendation accuracy and user satisfaction. Thus, the potential benefits are substantial, with deep learning architectures and hybrid approaches paving the way for more effective recommendations.

Incorporating visual information into recommender systems is a complex task that aims to improve their precision and dependability. A popular strategy involves merging User-Based Collaborative Filtering (UBCF) with Content-Based Filtering (CBF) methods. This fusion allows for the utilization of user tastes and item characteristics to create tailored and pertinent suggestions. Numerous experts have explored this challenging concept, employing various strategies to enhance the effectiveness of these systems.

The motivation behind exploring this integration lies in the limitations of traditional recommendation approaches. While UBCF relies on user-item interactions to make recommendations, it often suffers from the cold start problem, where new users or items lack sufficient interaction data. On the other hand, CBF leverages item features to make recommendations, but it may overlook the dynamic nature of user preferences and interactions.

Our goal is to improve recommender systems by including visual data in the recommendation process. Images connected to items can offer valuable context that goes beyond typical user-item interactions. This extra information allows us to better understand user preferences and item traits, resulting in more precise and individualized recommendations.

The core problem lies in how to seamlessly integrate visual data with existing recommendation techniques to improve recommendation accuracy and user satisfaction. How can we effectively combine user-item interaction data, item attributes, and visual features to create hybrid recommendation models that outperform traditional methods? Moreover, how can we ensure that these models remain scalable, robust, and adaptable to evolving user preferences and changing item landscapes? These are the questions that drive our work.

Our work will focus on exploring modern techniques and methodologies for incorporating visual data into recommendation algorithms. By leveraging modern machine learning and data mining techniques, we seek to develop a hybrid recommendation model that seamlessly integrate visual cues with traditional user and item features. Through this approach, we aim to improve the accuracy, relevance, and robustness of recommender systems in various domains.

## Multi-Model Hybrid Recommendation Approach

In this section, we provide a comprehensive overview of the specific approach adopted in this research to address the challenges and objectives outlined in the preceding sections. The chosen approach represents a fundamental aspect of our methodology and serves as the guiding framework for our investigation into enhancing the accuracy and reliability of recommender systems through the integration of visual data as well as textual data to increase the effectiveness of our approach, and to augment the chances of exploiting our visual data at its peak potential.

At the heart of our research is the integration of visual data into recommendation systems, a process with great promise for improving the accuracy and relevance of recommendations. The approach we selected involves leveraging User-Based Collaborative Filtering (UBCF) and Content-Based Filtering (CBF) techniques, combined with innovative methodologies to incorporate visual cues extracted from images of elements. This hybrid approach capitalizes on the strengths of UBCF and CBF while mitigating their respective limitations, providing a holistic solution to the challenges of recommendation in the era of multimedia-rich content.

## Description

Our approach involves a multi-faceted methodology that integrates user-item interaction data (ratings), item attributes (textual descriptions), and visual features to create hybrid recommendation models. These models aim to provide personalized and contextually relevant recommendations by combining collaborative filtering, content-based filtering and visual data analysis techniques. Through a systematic process of data collection, preprocessing, feature extraction, and algorithm development. We will obviously begin with sampling as well as organizing the data, followed by preprocessing of this data, after that we will take care of recovering insights from the visual and textual data of our dataset which we will use in our CBF algorithm, then the UBCF algorithm will be implemented subsequently as well as some clustering techniques that will help us group similar users and improve the UBCF results. We will try to evaluate the performance of the algorithms under different configurations, to finally try to combine the two approaches through hybrid techniques.

## Motivation

Our approach should be supported by the increasing amount of multimedia content available online, which offers a chance to improve the accuracy of recommendations. By utilizing a range of data sources, such as user interactions, item attributes, and visual clues, we can gain a thorough understanding of user preferences and item qualities. Adding visual features enhances our recommendation models by offering more context and relevance. Our structured approach guarantees reliable and adaptable recommendation models, with assessments focused on improving performance across different setups. Ultimately, our goal is to create recommendation models that are both effective and scalable, and our approach will definitely help increase our chances in achieving that, as the structure and the sequential organization of our approach will help us control and evaluate, and adjust each part of the process.

## Objectives and Expectations

We aim to achieve several key objectives:

- **Enhance recommendation accuracy and relevance**: The primary objective of this project is to improve the accuracy of recommender systems by incorporating visual data into the recommendation process

- **Ensure Scalability and Robustness**: It is essential to ensure that the hybrid recommendation models

developed in this project remain scalable, robust, and adaptable to evolving user preferences and changing item landscapes.

- **Develop Innovative Methodologies**: Our project will focus on exploring modern techniques and methodologies for incorporating visual data into recommendation algorithms.

- **Evaluate Performance**: A critical aspect of this project is to evaluate the performance of the developed recommendation models in terms of accuracy, relevance. and dive into the characteristics of our data, in order to find accurate ways to measure the right metrics.

Our expectations for this project are straightforward. We expect that by incorporating visual data into recommender systems, we will see improvements in recommendation accuracy and relevance. This project serves as an exploration into modernizing recommendation techniques, aiming to lay a basic foundation for further understanding and implementation in this area. Our goal is to gain insights into how visual data can enhance traditional recommendation methods, with the hope of creating more effective recommendation systems in the future.

## Limitations and Potential Challenges

Despite the potential benefits of our proposed approach, it is not without significant challenges that necessitate careful consideration. We anticipate several limitations and potential obstacles, including the complexity of integrating diverse data sources, the scalability of recommendation algorithms, and the interpretability of visual features. Addressing these challenges will require careful consideration and innovative solutions to ensure the success of our research, some of the issues we might face along the process are :

- **Amount of Code Required**: Integrating visual data into a recommender system might require heavy amounts of code on preprocessing, feature extraction, and implementation of the recommendation algorithms. Managing and maintaining this codebase can get complicated, especially in larger-scale projects.

- **Lack of Data**: Obtaining sufficiently large and diverse visual datasets is often a hard task. A lack of data might hinder the full capture of user preferences and characteristics of items by recommendation models.

- **Sparse Data**: Interaction data are often sparse, meaning many items are not rated by the majority of users, which makes it difficult to generate accurate recommendations, especially for new users or unpopular items.

- **Inaccurate Data**: Visual data can sometimes show errors or impreciseness, affecting the validity of the generated recommendations. Cleansing and validating visual data is needed to ensure the reliability of recommendation models.

- **Data Not Meeting the 4 Vs**: It's not always true that visual data follows the "4 Vs" of the dataset being processed. It is not easy to find databases whose set of items is large and their users are varied, being regularly updated, and relevant for making recommendations.

- **Resource Constraints**: This includes the necessity of hardware resources required for processing and analyzing large amounts of visual data. Inadequate hardware resources may include high storage and computational capabilities, limiting efficient processing and training of accurate recommendation models.

- **Algorithm Complexity**: Combining the effects of user-based collaborative filtering, content-based filtering, and visual data analysis techniques to develop effective hybrid recommendation algorithms is challenging. Balancing the complexity of the algorithm with performance and scalability requires careful optimization and testing.

- **Evaluation Metrics**: The choice of proper metrics to evaluate the performance of recommendation models is of prime importance. The choice of which metrics would best reflect quality recommendations,

particularly in the context of visual data, requires exhaustive experimentation and analysis.

## Applications

Our approach is modular and can be used in several domains, starting with e-commerce and multimedia content, moving to social media, personalized advertising, and culinary platforms. In the context of our work, our hybrid recommendation model will provide recommendations on recipes according to user preferences. Through the analysis of users' interactions, product attributes, and recipes' visuals, we seek to provide personalized and visually appealing suggestions. From a quick weeknight dinner to an indulgent dessert, our models offer several items to improve user culinary experiences.

## Conclusion

In this introductory chapter, we have elucidated the relevance of our approach to further improve recommendation accuracy and relevance. We outlined our research motivation, which shows the limitations of the traditional approaches to recommendations and the potential of infusing visual information into recommendation procedures.

Future chapters will delve into the methodology, conception of our approach. Chapter 3 will cover our approach in the implementation of the recommendation models, detailing the technical aspects and challenges encountered in the process, we will also present the results of the experiments and evaluations conducted to judge the performance of the recommendation models developed.

This concludes the document in Chapter 4 with discussion of the findings, implications, and future directions for the research in the field.

# Chapter 1

# Conception of Our Approach

## 1.1 Introduction

In this chapter, we will delve into the process of turning the approach described in last section, into life. Our motivations, and objectives were clear, and the conception will follow that approach in an optimal and concise way so we can maximize the chances of reaching the desire results and objectives. The next sections, will be written to detail each step of the process and the proposed model, illustrating each specificity that our system incorporate.

## 1.2 Initial Proposed Model

Our proposed model consists of a basic hybrid recommendation model composed by two baseline recommendation algorithms (a) a user-based collaborative algorithm, and (b) a content based collaborative filtering algorithm. This basic conception is designed to establish the essential foundations of our approach by integrating simple user-item interactions (ratings), and employing a simple collaborative filtering.

As we progresses in this chapter, we introduce more specific enhancements to the model. We gradually add additional improvements and more sophisticated components, such as the integration of item textual descriptions, visual feature analysis, and advanced natural language processing methods. Each enhancement is carefully justified and explained in the following subsections, demonstrating how it contributes to the overall improvement of the model's performance and our conception. Ultimately, these successive enhancements culminate in a detailed and optimized model capable of providing personalized and contextually relevant recommendations. This final model is comprehensively presented at the end of this section, with a clear explanation of each component and its role in the overall system.

The general approach taken for this initial phase was a simple one to help us introduce the tools and technologies, that fits best later on. Our approach revoloves around main axes that are :

- **Data Collection and Organization:** The first step involves sampling and organizing the data. The data will be stored in a structured format suitable for further processing and analysis.
- **Preprocessing:** it is crucial to ensure the quality and usability of the data. This phase includes cleaning the data, handling missing values, normalizing ratings, and extracting relevant features from textual and visual data.

- **Feature Extraction:** From the preprocessed data, we will extract features that will be used in our recommendation algorithms. These features include user preferences, item attributes, and visual characteristics of the items. This phase ensures that the data is transformed into a suitable format for the recommendation algorithms.
- **Algorithm Development:** We will implement two primary algorithms: User-Based Collaborative Filtering (UBCF) and Content-Based Filtering (CBF). UBCF will utilize user-item interaction data to find similarities between users and recommend items based on similar user preferences. CBF will leverage item attributes and visual features to recommend items similar to those the user has interacted with in the past.
- **Hybrid Recommendation Model:** After developing the UBCF and CBF algorithms, we will evaluate their performance under different configurations. The final step involves combining the two approaches to create a hybrid recommendation model that leverages the strengths of both collaborative and content-based filtering.

As an initial step, this approach appears promising and has the potential to provide favorable results. However, to align with our goal of achieving the illustrated approach described earlier, we will examine each process in detail to observe its development towards the final concept presented in the last section. Ultimately, these iterative enhancements will result in a comprehensive and optimized model, which will be thoroughly discussed at the end of this chapter. The final architecture will deliver personalized and contextually relevant recommendations, demonstrating the evolution from a basic initial model to an advanced hybrid recommendation system.

### 1.2.1 Basic Architecture

Building on the foundational elements discussed in the initial proposed model, we now delve into the initial architecture of our recommendation system. This architecture serves as the starting point for our iterative process of enhancements and optimizations. It's designed to be straightforward yet relevant, providing a clear framework for integrating the various components of our recommendation system. As illustrated in Figure 1.1, The core of the initial architecture comprises two primary recommendation algorithms: User-Based Collaborative Filtering (UBCF) and Content-Based Filtering (CBF). UBCF analyzes user-item interaction data to identify similarities between users and generate recommendations based on shared preferences. In contrast, CBF leverages item attributes (textual data) and visual features to recommend items that are similar to those the user has previously interacted with. Combining these two techniques, will then lead into the final step that is hybridization.

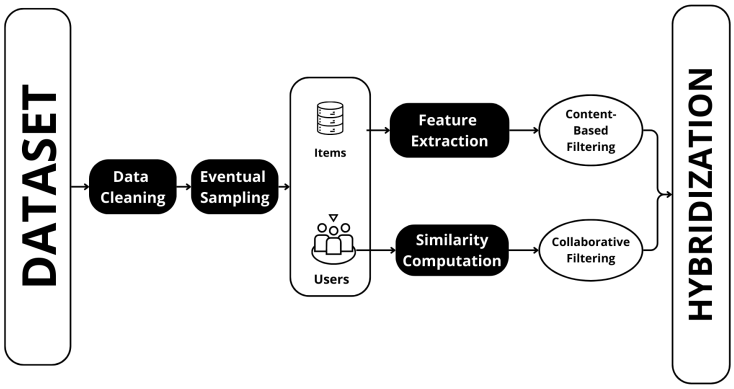



Figure 1.1: Our Approach's Architecture - Basic Model

At this stage, the architecture is designed to allow for seamless integration and interaction between these components. Data flows from collection and preprocessing to feature extraction, and finally to the recommendation algorithms. This modular approach ensures that each component can be individually enhanced and optimized without disrupting the overall system.

The model's ability to adapt is fundamental to its effectiveness, and this adaptability relies on various factors, such as the availability of diverse data types. Understanding the variations of the available data is the first step in effectively modifying the model. Additionally, assessing the preprocessing required for the data highlights the importance of ensuring data quality and compatibility with the model before implementation.

The core of our discussion centers on the necessity of employing suitable models during the feature extraction phase to effectively process and extract features and insights from the data at hand. This choice is related t the nature of the data under consideration. To expound upon this further, we will introduce the subsequent chapter, which will delve into this topic with greater depth and specificity.

## 1.3 Content-Based Recommendation

We now delve into the core aspects of content-based recommendation. This section focuses on leveraging item attributes, specifically textual descriptions and visual features, to generate personalized recommendations.

Content-based recommendation systems analyze the characteristics of items that a user has previously interacted with to recommend similar items. This approach involves two main types of data: textual and visual. Each type requires distinct preprocessing and feature extraction techniques to ensure the data is in a suitable format for the recommendation algorithms.

In this chapter, we will systematically explore the methodologies and techniques used to process and extract meaningful features from textual and visual data. The process begins with data organization, followed by preprocessing to clean and prepare the data. This sets the stage for the crucial step of feature extraction, where advanced models and techniques are employed to distill relevant features from the raw data.

**Textual data**, such as item descriptions, will be processed using natural language processing (NLP) techniques. We will discuss various methods, starting with traditional approaches and advancing to more sophisticated models like GloVe and BERT. These techniques help in capturing the semantic meaning and contextual nuances of the text, which are critical for accurate recommendations.

For **visual data**, we will explore the use of deep learning models such as VGG16 for feature extraction. Visual features can provide valuable insights, especially in domains like fashion, art, and retail, where the appearance of an item plays a significant role in user preferences. We will delve into the specifics of how these models are fine-tuned and optimized for our recommendation tasks.

### 1.3.1 Visual Data

In the following, we explore techniques for extracting valuable insights from visual data, including data cleaning, feature extraction, and optimization of computational workflows. By leveraging these methodologies, we aim to enhance the utility and relevance of visual data for content-based recommendations. Through a series of case studies, we demonstrate the application of these techniques in real-world scenarios, highlighting their effectiveness in improving recommendation outcomes.

#### 1.3.1.1 Image Selection

Sometimes, when dealing with an abundance of visual data related to a specific item, it becomes necessary to manage this data effectively through various methods such as combination, merging, and selection. These techniques help in reducing redundancy, ensuring quality, and enhancing the utility of the images for further processing or analysis [5].

- **Combination**: This involves creating a composite image from multiple images of the same item. Techniques such as image stitching and blending can be used to combine images, providing a more comprehensive view of the item.
- **Merging**: This method entails fusing the most relevant parts of different images into a single image. Merging can be beneficial when different images contain unique information about the item, and combining these details results in a richer representation.
- **Selection**: This approach focuses on choosing the best image from a set based on specific criteria. Selection helps in identifying the most representative and high-quality image, which is crucial for applications where a single image needs to be displayed or processed further.

In our case, we focus on image selection, where we aim to choose the best image from a given set based on the concept of entropy. Entropy, in this context, measures the amount of information or randomness in an image. Higher entropy indicates a more detailed and informative image. Therefore, by calculating the entropy of each image, we can select the one with the highest entropy as the best representation.

The first algorithm we employ is designed to calculate the entropy to an image. The process begins by attempting to retrieve the binary data of the image. If the data is successfully retrieved, the image is opened and converted to grayscale to simplify the entropy calculation. The grayscale levels' histogram is then computed and normalized. Entropy is calculated by summing the product of the normalized histogram values and their logarithms, using the formula:

$$\text{entropy} = -\sum \left(\text{normalized\_histogram} \times \log_2(\text{normalized\_histogram} + 1 \times 10^{-10})\right) \tag{2,1}$$

The calculated entropy is returned, or `None` is returned if an error occurs during the process.

**Algorithm 1** Calculate Entropy from an Image

**Require:** An image
**Ensure:** Entropy value or `None` if an error occurs
1: Retrieve the binary data of the image from the URL
2: **if** data is successfully retrieved **then**
3: Open the image
4: Convert the image to grayscale
5: Compute the histogram of the grayscale levels
6: Normalize the histogram
7: Calculate the entropy using the formula:

$$\text{entropy} = -\sum \left(\text{normalized\_histogram} \times \log_2(\text{normalized\_histogram} + 1 \times 10^{-10})\right)$$

8: **return** Entropy value
9: **else**
10: **return** `None`
11: **end if**

The second algorithm focuses on selecting the best image from a list of images. It initializes two variables: `best_img` to keep track of the image with the highest entropy and `max_entropy` to store the highest entropy value found. The algorithm iterates through each image in the provided list, calculating the entropy of the image at each index. If the calculated entropy is higher than the current maximum entropy, `max_entropy` and `best_img` are updated accordingly. After processing all images, the image with the highest entropy is returned.

**Algorithm 2** Choose the Best Image

**Require:** List of images
**Ensure:** Image with the highest entropy
Initialize `best_img` to `None`
Initialize `max_entropy` to `-1`
**for** each image in the list of images **do**
  Calculate the entropy of the image using Algorithm 1
  **if** entropy is not `None` and entropy > `max_entropy` **then**
    Update `max_entropy` with the current entropy
    Update `best_image` with the current image
  **end if**
**end for**
**return** `best_img`

By implementing these algorithms, we can efficiently handle large sets of visual data, ensuring that the most suitable images are selected for content-based recommendations. This enhances the quality and relevance of the recommendations provided to the users.

#### 1.3.1.2 Deep-Based Feature Extraction Techniques

Now, we explore key feature extraction methods that are vital for enhancing content-based recommendation systems. We focus on Convolutional Neural Networks (CNNs), Transfer Learning for extracting meaningful visual features. By understanding these methods, we can improve recommendation accuracy and user satisfaction [13] [6] [2] :

- **Convolutional Neural Networks (CNNs)** are particularly useful in content-based recommendation systems for their ability to automatically extract hierarchical features from raw pixel data. By analyzing the visual content of images, CNNs can capture intricate patterns and characteristics that are relevant for recommendation tasks. This allows them to effectively identify similarities and relationships between different items based on their visual attributes. Additionally, CNNs can adapt to various input sizes and aspect ratios, making them versatile for processing diverse types of visual data [13].
  - ResNET : it introduces residual learning, which uses shortcut connections to address the vanishing gradient problem in deep networks by facilitating gradient flow. ResNet architectures comprise residual blocks, each containing multiple convolutional layers with skip connections. These connections allow the network to learn residual mappings, easing training of deep networks with hundreds of layers [13].
  - VGG (Visual Geometry Group)Net : VGGNet, designed by the Visual Geometry Group at Oxford University, features a straightforward architecture with small convolutional kernels and deep stacks. Widely adopted for its simplicity and performance, VGGNet serves as a baseline in many computer vision applications [13].
- **Transfer Learning** offers a practical solution for leveraging pre-trained CNN models to extract features from visual data in content-based recommendation systems. By using models that have been trained

on large-scale datasets like ImageNet, transfer learning allows for the transfer of knowledge and learned representations to new tasks with limited labeled data. This significantly reduces the need for extensive training data and computational resources, while still providing effective feature extraction capabilities [10].

- VGG16 and VGG19 are specific configurations of the VGGNet architecture, named according to the number of weight layers they contain. VGG16 consists of 13 convolutional layers and 3 fully connected layers, while VGG19 consists of 16 convolutional layers and 3 fully connected layers. These architectures have a fixed input size of 224x224 pixels and are pretrained on the ImageNet dataset for tasks such as image classification. VGG16 and VGG19 have been widely used as feature extractors and baseline models in various deep learning applications, providing a strong foundation for further experimentation and customization [10].

#### 1.3.1.3 The Use of VGG16 in Our Approach

While there exists a big amount of CNN architectures, including ResNet, VGG, and VGG16, VGG19, selecting the most beneficial model is a critical task. Among these options, models like VGG16 and VGG19 offer notable advantages over simpler models like VGG and more complex ones like ResNet. This selection is based on the fact that they offer improved performance and efficiency. In this section, we aim to elucidate the advantages of utilizing models like VGG16 and VGG19 by providing a comparative analysis between VGG16 and VGG19 to justify our use of VGG16 as a feature extractor model in our approach [7].

| **Factor** | **Justification** |
|---|---|
| Simplicity and Efficiency | VGG16 has a slightly simpler architecture compared to VGG19, with 16 weight layers compared to VGG19's 19 layers. This simplicity translates to faster training and inference times, making VGG16 more efficient for applications where computational resources are limited or time is a critical factor. |
| Trade-off Between Complexity and Performance | While VGG19 may offer slightly higher model capacity due to its additional layers, the performance improvement may not always justify the increased complexity and computational cost. In many cases, the incremental gain in accuracy achieved by VGG19 may not outweigh the benefits of simplicity and efficiency provided by VGG16. |
| Ease of Implementation and Tuning | VGG16's architecture is straightforward and easier to implement compared to VGG19. Additionally, tuning hyperparameters and optimizing the model architecture is simpler for VGG16 due to its fewer layers. This makes VGG16 a more practical choice for users who prioritize ease of implementation and experimentation. |
| Pretrained Models Availability | Pretrained models for VGG16 are more widely available and extensively used in the deep learning community compared to VGG19. This availability of pretrained weights for VGG16 facilitates transfer learning, allowing users to leverage learned representations for various tasks with minimal effort. |

Table 1.1: Comparison between VGG16 and VGG19

When an image is passed to VGG16 for processing, it requires preprocessing to ensure compatibility with the model's input requirements. The following pseudocode algorithm, `load_and_preprocess_image`, accomplishes this task:

This algorithm resizes the input image to match the size expected by VGG16 (typically 224x224 pixels).

**Algorithm 3** Load and Preprocess Image for VGG16

**Function** LoadAndPreprocessImage(*image*)
**Input**: *image* - The input image to be preprocessed
**Output**: *preprocessed_image* - The preprocessed image ready for VGG16

*resized_image* ← resize(*image*, (224, 224))
**if** *resized_image.mode* ≠ 'RGB' **then**
    *resized_image* ← convert_to_rgb(*resized_image*)
**end if**
*preprocessed_image* ← preprocess_input(*resized_image*)
**return** *preprocessed_image*

It then checks if the image is in RGB format, and if not, converts it to RGB. Finally, the image is preprocessed to ensure compatibility with the preprocessing used by the VGG16 model.

- **Architecture of VGG16:**

  VGG16 is characterized by its simplicity and uniformity in architecture. it consists of 16 layers: 13 convolutional layers and 3 fully connected layers, organized into blocks with max-pooling layers for downsampling. Here's a concise breakdown of its architecture [22]:

  – Input Layer:
    Input dimensions: (224, 224, 3)
  – Convolutional Layers:
    Two layers with 64 filters (3×3, same padding)
    Max Pooling (2×2, stride 2)
    Two layers with 128 filters (3×3, same padding)
    Max Pooling (2×2, stride 2)
    Three layers with 256 filters (3×3, same padding)
    Max Pooling (2×2, stride 2)
    Three layers with 512 filters (3×3, same padding)
    Max Pooling (2×2, stride 2)
    Three layers with 512 filters (3×3, same padding)
    Max Pooling (2×2, stride 2)
  – Flattening:
    Flatten output feature map (7x7x512) into a vector (25088).
  – Fully Connected Layers:
    Three layers with ReLU activation. First: 4096 neurons Second: 4096 neurons Third: 1000 neurons (corresponding to ILSVRC classes)
  – Output Layer:
    Softmax activation for classification.

  This structure represented in Figure 1.2, characterized by its depth and the use of small 3x3 filters, is effective for various image classification tasks, providing high accuracy and ease of implementation.

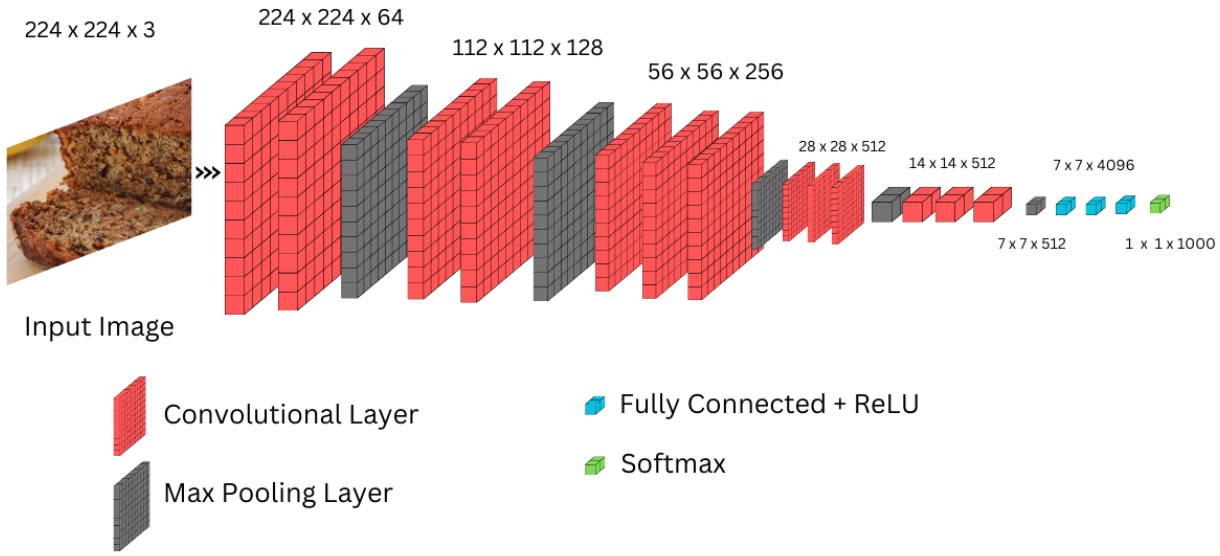



Figure 1.2: VGG16 Architecture

To utilize VGG16 for feature extraction, we modify its structure by removing the fully connected layers at the top of the model and adding a Global Average Pooling (GAP) layer. This approach leverages the pre-trained convolutional layers of VGG16, which are effective at capturing spatial hierarchies in images. By excluding the fully connected layers, we focus on the convolutional features. The GAP layer is then added to condense these features into a single vector per feature map, effectively reducing the spatial dimensions while preserving important information. This modified architecture, combining VGG16's deep convolutional capabilities with the simplicity of the GAP layer, enables efficient feature extraction for image classification and other tasks in our content-based recommendation system.

- **Applications of VGG16**:

  - Image Classification: VGG16 is widely used for image classification tasks, where it excels at extracting high-level features from input images and making predictions about their contents. Its pretrained weights on ImageNet make it particularly effective for this task.

  - Feature Extraction: VGG16 can be used as a feature extractor in transfer learning scenarios. By removing the fully connected layers and using the output of one of the convolutional layers as features, VGG16 can extract meaningful representations of input images for downstream tasks like object detection and image retrieval.

  - Object Detection: VGG16 features are commonly used as a backbone network in object detection frameworks like Faster R-CNN and YOLO (You Only Look Once). Its ability to capture detailed visual features makes it suitable for detecting objects in images with high accuracy.

  - Image Generation and Style Transfer: VGG16's pretrained weights can be used in style transfer algorithms to generate artistic images by transferring the style of one image to the content of another. Additionally, VGG16 features can be used in generative models like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) for image generation tasks.

The use of VGG16 will allow us to effectively extract meaningful features from our image data. This capability is invaluable for our content-based recommendation system, as it enables us to analyze and understand the visual content of items more comprehensively. With VGG16 as our feature extractor, we can enhance the quality and relevance of our recommendations by leveraging the rich information captured in

the visual domain. Figure 1.3, represent the the enhancement of our initial model using what we described in this section.

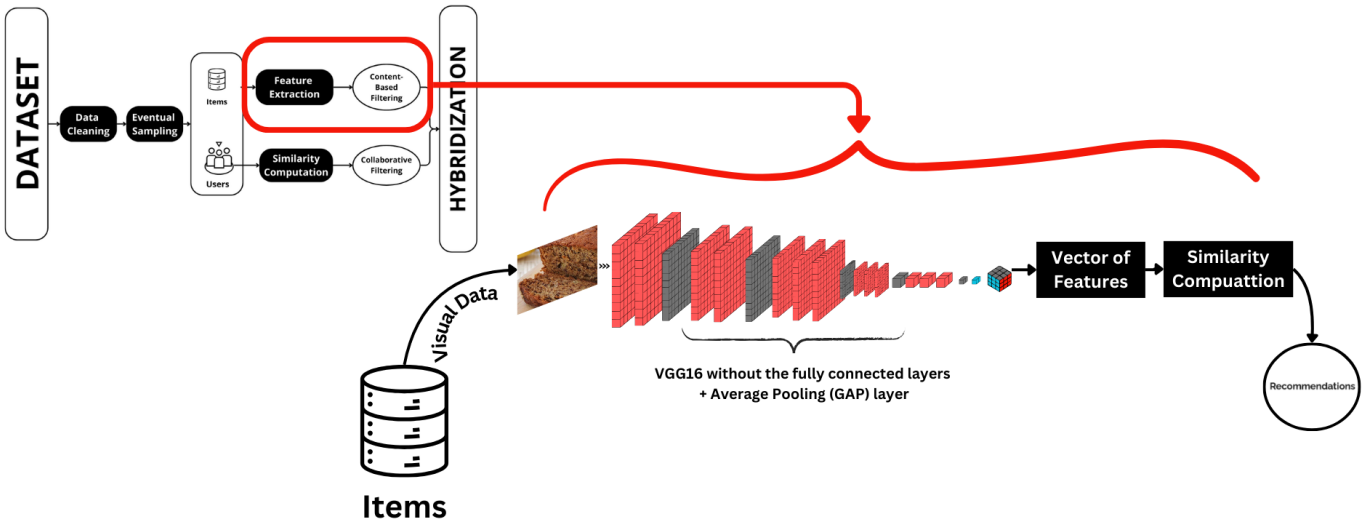



Figure 1.3: Improved Version of Our Approach's Architecture Using VGG16 in the Context of CBF

### 1.3.2 Textual-Based Models

Now we will explore the practical strategies for utilizing textual data effectively in content-based recommendation systems. We explore techniques for extracting valuable insights from textual data, including data preprocessing, feature extraction, and optimization of computational workflows. By leveraging these methodologies, we aim to enhance the utility and relevance of textual data for content-based recommendations.

#### 1.3.2.1 Word Embedding Learning

**GloVe** is a technique used for word embedding learning, where words with similar meanings have similar numerical representations. Using word embeddings like GloVe (Global Vectors for Word Representation) in content-based recommendation systems offers substantial benefits. GloVe embeddings capture semantic relationships between words, providing a deeper understanding of context and meaning, which enhances recommendation accuracy. By representing words in a dense, lower-dimensional space, GloVe reduces computational complexity and improves efficiency. Additionally, pre-trained GloVe embeddings bring rich linguistic knowledge, improving performance even with smaller datasets. These embeddings enable better similarity measures between items based on their descriptions, facilitating more relevant and context-aware recommendations. Integrating GloVe embeddings ultimately leads to a more effective and scalable recommendation system [16].

#### 1.3.2.2 Sequence Embedding Learning

To effectively utilize textual data in content-based recommendation systems, one of the powerful techniques is employing Long Short-Term Memory (LSTM) networks.

**LSTM** is a type of recurrent neural network (RNN), excels in processing and predicting sequences of

data, making it particularly well-suited for handling text. these networks are designed to capture long-range dependencies in sequential data. They achieve this through a unique architecture that includes memory cells and gating mechanisms, which help the network retain important information over long periods while mitigating the vanishing gradient problem commonly faced by traditional RNNs [8] [11].

In the context of content-based recommendation systems, LSTM networks can be used to analyze and extract meaningful features from textual descriptions, user reviews, and other text-based inputs. By embedding these textual features into a high-dimensional space, LSTMs enable the recommendation system to understand the semantic relationships between different pieces of text, leading to more accurate and relevant recommendations [18].

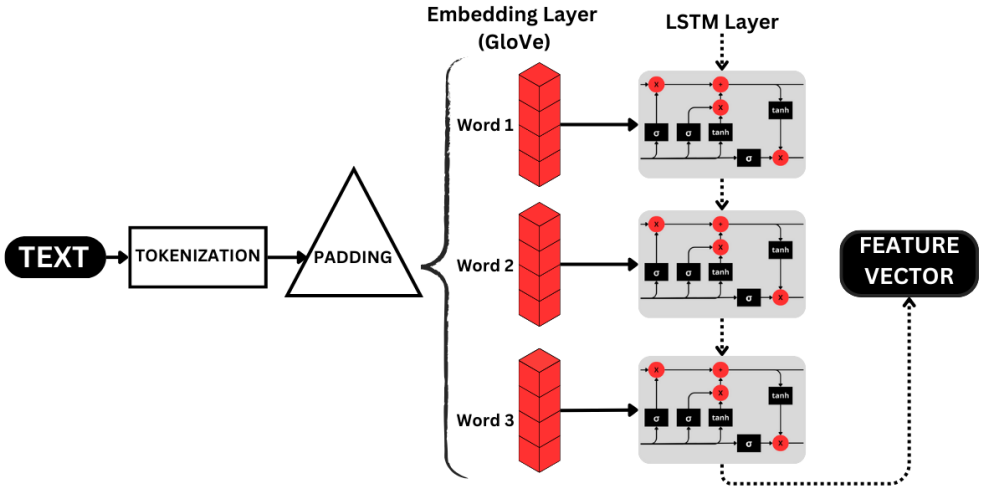



Figure 1.4: Combination of GloVe & LSTM

As illustrated is Figure 1.4 The pipeline begins with raw text input, specifically recipe descriptions. These descriptions are tokenized using a tokenizer, which converts the text into sequences of integers,each representing a unique word from the vocabulary.

To ensure uniform sequence length, padding is applied to these sequences. The padded sequences are then fed into an embedding layer, which uses pre-trained GloVe embeddings to map each word index to a dense vector representation. Following this, an LSTM layer processes these sequences of embedding vectors, capturing temporal dependencies and contextual information inherent in the text. The output of the LSTM layer is a fixed-size feature vector that encapsulates the semantic meaning of the input text.

The integration of this new component is illustrated in Figure 1.5.

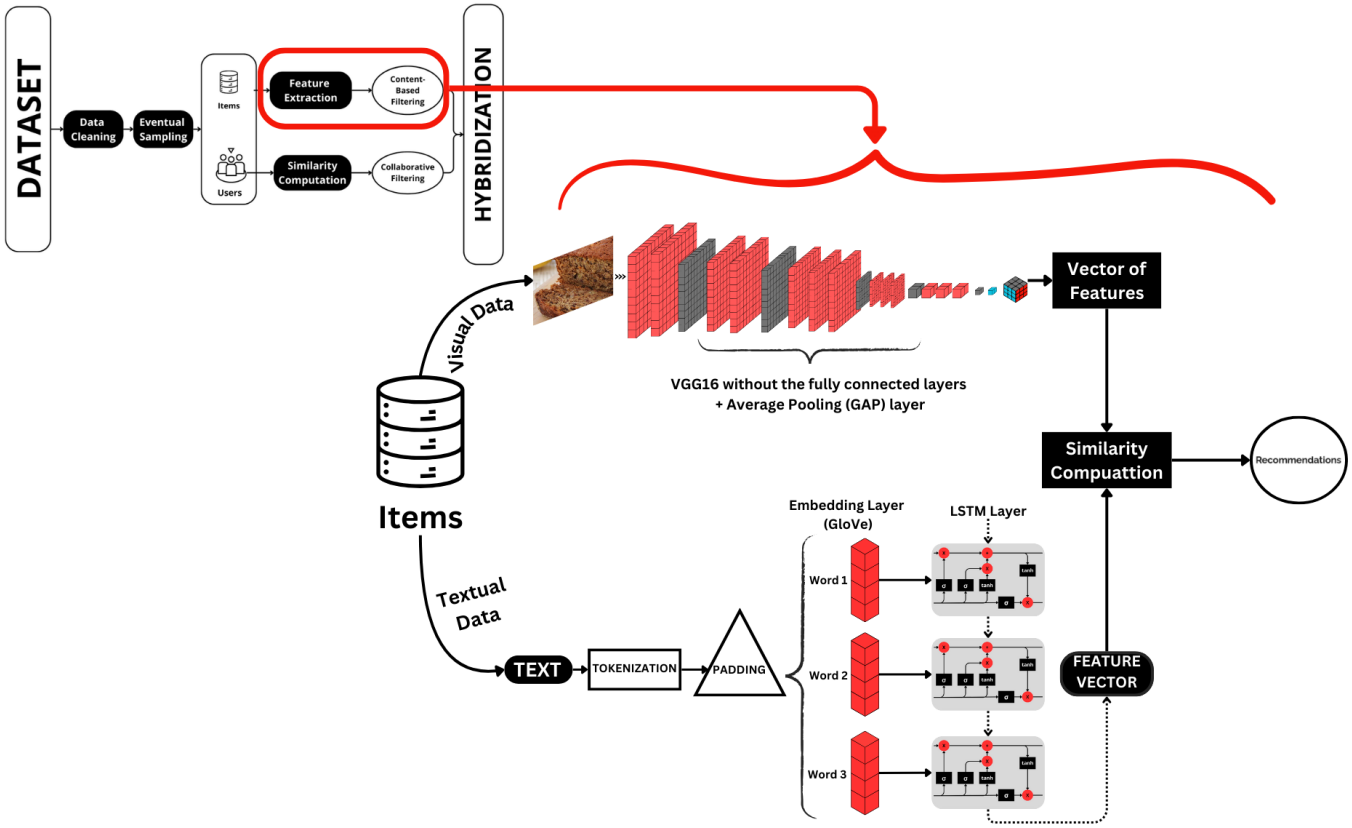



Figure 1.5: Improved Version of Our Approach's Architecture Using LSTM & GloVe in the Context of CBF

#### 1.3.2.3 Bidirectional Encoder Representations from Transformer (BERT)

It is important to acknowledge the existence of more advanced text modeling techniques such as BERT (Bidirectional Encoder Representations from Transformers). BERT represents a significant advancement in natural language processing by providing deep bidirectional understanding of text. Unlike traditional models that read text sequentially, BERT considers the context from both directions, leading to a more nuanced understanding of word meanings and relationships. This superior contextual awareness allows BERT to excel in various NLP tasks, including text classification and sentiment analysis. Although our approach does not currently incorporate BERT, recognizing its potential can inform future improvements and iterations of our recommendation system, ensuring it remains at the cutting edge of technological advancements [3].

### 1.3.3 Features Vectors Combination

At this level of the process, we already have the feature vector of our visual data as well as the Textual-Based Model feature output, so in this part, we delve into the critical aspect of combining feature vectors derived from both textual and visual data to enhance the effectiveness of content-based recommendation systems. By integrating these two types of feature vectors, we can leverage the strengths of each data modality to create a more comprehensive and accurate representation of the items in our dataset. The integration of textual and visual feature vectors is pivotal in addressing the limitations inherent in using a single data modality. Textual data provides rich, descriptive information that can capture the nuances of content, while visual data offers an immediate and intuitive understanding of the item's appearance. Combining these two allows us to

harness a more holistic view of the data, leading to more precise and relevant recommendations [20] [4].

In the process of combining textual and visual feature vectors, several techniques can be employed to fuse the information effectively. Here are some prominent methods:

- **Fusion** : Fusion techniques combine features by applying mathematical operations such as element-wise addition, multiplication, or averaging. these methods enable more flexible integration of features by allowing for the combination of specific aspects from each modality. Unfortunately, The effectiveness of fusion techniques heavily depends on the choice of operation and may require careful tuning to achieve optimal results.
- **Weighted Average Technique:** it computes a linear combination of feature vectors, where the contribution of each modality is weighted by its relevance or importance. It allows for the incorporation of domain knowledge or learned weights to prioritize certain modalities over others. The eventual issue that might get face is determining the optimal weights can be challenging and may require extensive experimentation or training.
- **Concatenation Technique:** it involves simply stacking the feature vectors from different modalities into a single vector. This straightforward approach preserves all the information from both sources. Although, it assumes equal importance of features from each modality, which may not always hold true, and also lead to a significant increase in the dimensionality of the feature space. It helps maintains the original dimensionality of the feature vectors, allowing the model to access all available information directly.

In our exploration of feature combination, we will focus on two primary methods: concatenation and weighted average. These techniques offer contrasting approaches to combining textual and visual features, each with its unique advantages and considerations. By evaluating both approaches, we aim to determine the most suitable fusion strategy for our specific recommendation task. Through experimentation and analysis, we seek to identify the method that maximizes the effectiveness and relevance of our content-based recommendation syste. Plus, since our main goal from this approach is to also give a special focus to Visual Data, the weighted average technique will allow us to give a heavier weight to features we'll get from the visual data, and the concatenation results will help us compare the results.

### 1.3.4 Similarity Computation

In the context of refining our recommendation system, having extracted feature vectors that encapsulate the essence of items, we now venture into the realm of "Similarity Computation". Here, we delve into the mechanics of cosine similarity. In a dataset, where we assume the availability of test data for each user, we use cosine similarity to get similar items to those which are considered as "test" data (this will be further discussed and explained in chapter 3 of our dissertation). Cosine similarity enables us to quantify the alignment between these vectors in high-dimensional spaces. As a natural extension of our feature extraction process, cosine similarity emerges as a vital tool, empowering our system to identify items with shared characteristics or attributes. Through its application, we unlock the potential to deliver tailored recommendations by leveraging the inherent similarities encoded within item feature vectors. With the formula:

$$\text{similarity}_{\text{Cosine}}(i_1, i_2) = \frac{\sum_{u \in U_{i_1} \cap U_{i_2}} r_{u,i_1} \cdot r_{u,i_2}}{\sqrt{\sum_{u \in U_{i_1}} r_{u,i_1}^2} \cdot \sqrt{\sum_{u \in U_{i_2}} r_{u,i_2}^2}} \tag{2,2}$$

Where:

- $\text{similarity}_{\text{Cosine}}(i_1, i_2)$ : Cosine similarity between items $i_1$ and $i_2$.
- $U_{i_1}$ : Set of users who have rated item $i_1$.
- $U_{i_2}$ : Set of users who have rated item $i_2$.
- $r_{u,i_1}$ : Rating given by user $u$ to item $i_1$.
- $r_{u,i_2}$ : Rating given by user $u$ to item $i_2$.

Following the meticulous computation of similar items, we continue with the imperative task of recalculating the new rating of an item based on these identified similarities. This will be more detailed on the Chapter 3 of this dissertation, as how it's going to be used in our CBF process.

For the recalcul we use the formula :

$$\hat{r}_{ui} = \frac{\sum_{j=1}^{k} \text{Sim}(u, n_j) \cdot r_{n_j,i}}{\sum_{j=1}^{k} |\text{Sim}(u, n_j)|} \tag{2,3}$$

Where:

- $\hat{r}_{ui}$ is the predicted rating for user $u$ and item $i$.
- $k$ is the number of nearest neighbors.
- $\text{Sim}(u, n_j)$ is the similarity score between the target user $u$ and the $j$-th neighbor $n_j$.
- $r_{n_j i}$ is the rating given by the $j$-th nearest neighbor to the item $i$.

The Figure 1.6, showcases our improved model, enhanced with the use of the Vector Combination module, at this level of our conception, the Content-Based module is ready to operate.

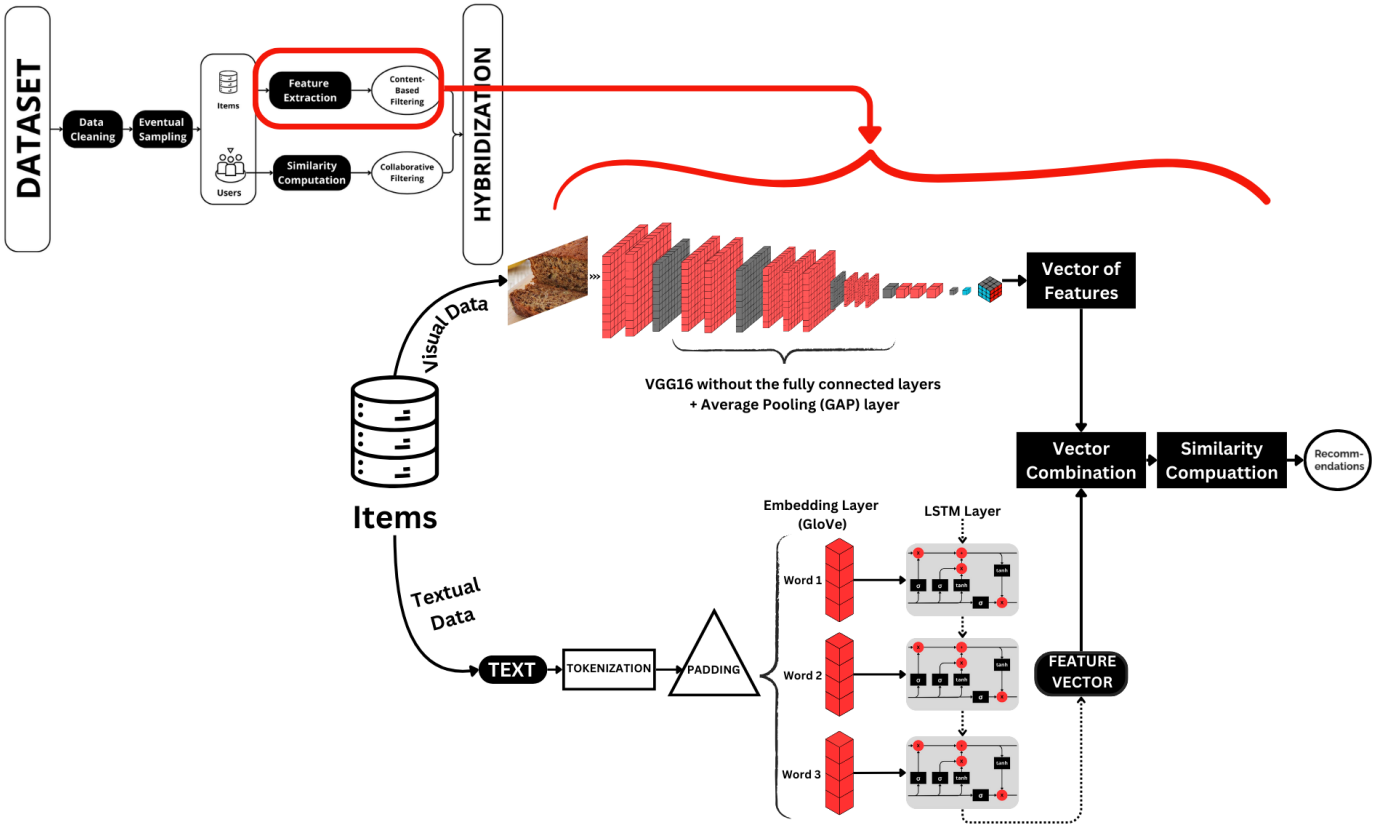



Figure 1.6: Improved Version of Our Approach's Architecture Specifying the 'Vector Combination' Module

## 1.4 Collaborative Filtering Recommendation

Expanding upon the elements introduced in the initial proposed model, we now turn our attention to the principles of collaborative filtering recommendation. This section aims to use user-item interaction data to deliver personalized recommendations tailored to individual preferences.

Collaborative filtering recommendation systems operate on the premise of analyzing historical interactions between users and items to infer user preferences. Unlike content-based approaches that focus on item attributes, collaborative filtering leverages the collective wisdom of user behavior to generate recommendations.

In this section, we'll explore the methodologies and techniques employed in collaborative filtering recommendation. The journey begins with the acquisition and organization of user-item interaction data, laying the groundwork for subsequent preprocessing and feature extraction stages.

As we progress, we will delve into advanced techniques which play a pivotal role in enhancing the performance and accuracy of collaborative filtering algorithms.

### 1.4.1 Similarity Metrics

In the realm of collaborative filtering recommendation, the concept of similarity plays a crucial role in identifying patterns and relationships within user-item interaction data. One prevalent similarity metric widely used in recommender systems is cosine similarity. Cosine similarity calculates the cosine of the angle between two vectors, representing the direction and magnitude of their similarity. It operates effectively in high-dimensional spaces, making it particularly suitable for analyzing user-item interaction data characterized by sparse vectors [9].

The utilization of cosine similarity in recommender systems offers several distinct advantages. Firstly, it provides a robust measure of similarity that is resilient to variations in magnitude, focusing solely on the orientation of vectors. This characteristic makes cosine similarity well-suited for scenarios where the absolute magnitude of user preferences or item attributes is less relevant than their relative alignment [14].

Thus, cosine similarity facilitates efficient computation and scalability in large-scale recommendation tasks. By operating directly on vector representations of users or items, cosine similarity enables rapid similarity calculations across vast datasets, ensuring real-time responsiveness and scalability.

By computing the cosine similarity between user vectors representing their preferences or behaviors, collaborative filtering algorithms can effectively discern users with comparable tastes and preferences. This enables the generation of personalized recommendations by leveraging the preferences of similar users to infer potential item affinities for the target user. The formula used for the calculation of cosine similarity among users is :

$$\text{similarity}_{\text{Cosine}}(u_1, u_2) = \frac{\sum_{i \in I_{u_1} \cap I_{u_2}} r_{u_1,i} \cdot r_{u_2,i}}{\sqrt{\sum_{i \in I_{u_1}} r_{u_1,i}^2} \cdot \sqrt{\sum_{i \in I_{u_2}} r_{u_2,i}^2}} \tag{2,4}$$

Where:

- $\text{similarity}_{\text{Cosine}}(u_1, u_2)$ is the cosine similarity between users $u_1$ and $u_2$.
- $I_{u_1}$ is the set of items rated by user $u_1$.
- $I_{u_2}$ is the set of items rated by user $u_2$.
- $r_{u_1,i}$ represents the rating of user $u_1$ for item $i \in I_{u_1}$.
- $r_{u_2,i}$ represents the rating of user $u_2$ for item $i \in I_{u_2}$.
- $\sum_{i \in I_{u_1} \cap I_{u_2}} r_{u_1,i} \cdot r_{u_2,i}$ computes the dot product of ratings of users $u_1$ and $u_2$ for the items they have both rated.
- $\sqrt{\sum_{i \in I_{u_1}} r_{u_1,i}^2}$ and $\sqrt{\sum_{i \in I_{u_2}} r_{u_2,i}^2}$ calculate the Euclidean norms (magnitude) of the rating vectors for users $u_1$ and $u_2$, respectively.

### 1.4.2 KNN Algorithm

Another cornerstone of collaborative filtering recommendation systems is the K-Nearest Neighbors (KNN) algorithm, which operates on the principle of similarity among users or items. In KNN, recommendations are generated by identifying the K most similar users to the target user, based on a predefined similarity

metric such as cosine similarity. Once these nearest neighbors are identified, their preferences are aggregated to make recommendations for the target user. The KNN algorithm offers flexibility in the choice of similarity metric and the number of neighbors considered, allowing for fine-tuning based on the specific characteristics of the dataset and the desired level of personalization. However, one challenge with KNN is its computational complexity, especially with large datasets, as it requires computing pairwise similarities between all users or items. Nevertheless, with efficient implementation strategies and optimizations, KNN remains a powerful and widely used technique in collaborative filtering recommendation systems, particularly for its simplicity and interpretability [19] [17].

**Algorithm 4** K-Nearest Neighbors Algorithm

**Require:** Target user $u$, dataset $D$, similarity metric $sim()$, number of neighbors $K$
**Ensure:** Set of K closest neighbors $N$

Initialize an empty set $N$
**for** each user $v$ in $D$ **do**
  Compute similarity $s$ between $u$ and $v$ using $sim()$
  **if** $|N| < K$ **then**
    Add $v$ to $N$
  **else**
    Find the neighbor $w$ in $N$ with the lowest similarity $s_w$
    **if** $s > s_w$ **then**
      Remove $w$ from $N$
      Add $v$ to $N$
    **end if**
  **end if**
**end for**
**return** $N$

The KNN algorithm iterates through each user in the dataset, computing the similarity between the target user and each user in the dataset using a specified similarity metric (in our case, cosine similarity). It selects the K users with the highest similarity as the nearest neighbors and returns this set of neighbors.

**Algorithm 5** Weighted Average Rating Calculation

**Require:** Target user $u$, item $i$, set of nearest neighbors $N$, similarity weights $W$
**Ensure:** Predicted rating $\hat{r}_{ui}$

Set $\hat{r}_{ui} \leftarrow 0$
Set $W_{\text{total}} \leftarrow 0$
**if** $N$ is not empty **then**
  **for** each neighbor $n$ in $N$ **do**
    Compute similarity weight $w_n$ for neighbor $n$
    Fetch rating $r_{ni}$ of neighbor $n$ for item $i$
    Set $\hat{r}_{ui} \leftarrow \hat{r}_{ui} + w_n \times r_{ni}$
    Set $W_{\text{total}} \leftarrow W_{\text{total}} + w_n$
  **end for**
  Set $\hat{r}_{ui} \leftarrow \frac{\hat{r}_{ui}}{W_{\text{total}}}$
  **return** $\hat{r}_{ui}$
**else**
  **return** `None`
**end if**

This algorithm calculates the predicted rating $\hat{r}_{ui}$ for a target user-item pair $u/i$ based on the ratings of its nearest neighbors in set $N$. It computes a weighted average of the ratings given by the neighbors, where the weights are determined by the similarity between the target user and each neighbor. If $N$ is not empty, the algorithm returns the predicted rating; otherwise, it returns `None`.

The formula used to calculate the predicted rating $\hat{r}_{ui}$ is equation (3) expressed prior in the last chapter.

Now, to conclude this chapter, we present an enhanced version of our Model, illustrating how Collaborative Filtering is going to operate within our system.

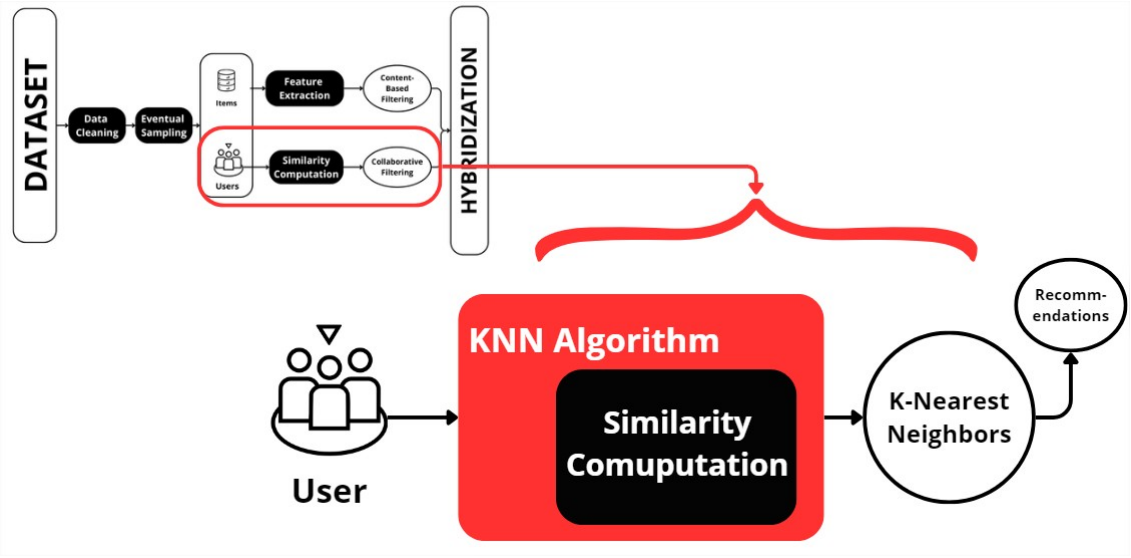



Figure 1.7: Improved Version of Our Approach's Architecture Using the KNN Algorithm in the Context of CF

Figure 1.7, illustrates the core functionality of the K-Nearest Neighbors (KNN) algorithm within a collaborative filtering recommender system. The system employs KNN to identify the k most similar users (k-nearest neighbors) to a target user. This similarity is calculated using a metric like cosine similarity, which compares the user's preferences represented as vectors. Based on the preferences of these similar users, the recommender system generates suggestions for items the target user might be interested in.

## 1.5 Clustering

Clustering algorithms enable the segmentation of data into distinct groups based on similarities in their features or attributes. By identifying patterns and trends within the data, clustering techniques provide valuable insights that can drive various applications, ranging from customer segmentation and recommendation systems to anomaly detection and resource allocation.

In the context of enhancing systems, clustering techniques play a pivotal role in optimizing processes, improving efficiency, and enhancing overall performance. By organizing complex data into meaningful clusters, these techniques facilitate targeted analysis and decision-making, leading to more informed strategies and actions.

In this approach, we explore the application of clustering techniques to enhance systems across various domains. We delve into the principles, methodologies, and applications of clustering algorithms, highlighting their potential to unlock hidden patterns and drive actionable insights for system optimization and improvement. Through practical examples and case studies, we demonstrate how clustering techniques can be leveraged to tackle real-world challenges and elevate system performance to new heights.

### 1.5.1 Dimensionality Reduction

In designing our approach, we recognized the need to address the challenges posed by high dimensionality and sparsity in real-world datasets. Our goal was to develop a robust methodology that not only reduces dimensionality but also effectively handles sparse data to extract meaningful insights.

To achieve this, we employed dimensionality reduction techniques, which is important in preprocessing complex datasets. By reducing the number of features while preserving essential information, these techniques streamline subsequent analysis, enhance interpretability, and facilitate the discovery of underlying patterns in the data [1].

Given the sparse nature of our dataset, we opted for Sparse Principal Components Analysis (SparsePCA). This method is particularly well-suited for sparse data, as it reduces the dimensionality of the data while retaining its essential structure. This approach not only reduces computational complexity but also helps uncover latent relationships within the data.

Following dimensionality reduction, we applied normalization techniques to ensure consistency in the scale of the resulting components. Normalization plays a crucial role in standardizing the magnitude of features, thereby facilitating fair comparison and interpretation of their contributions to the overall variance.

Our approach to dimensionality reduction and normalization is seamlessly integrated into our broader methodology, ensuring a holistic and efficient data preprocessing pipeline. By incorporating these techniques early in the analysis workflow, we lay a solid foundation for subsequent tasks such as clustering, classification, and regression.

### 1.5.2 Deep Contrastive Clustering

Integrating contrastive learning techniques into collaborative filtering and recommender systems can offer significant benefits in enhancing their performance and effectiveness. The use of contrastive learning, can help these systems better understand the underlying structure and relationships within the user-item interaction data.

Introducing deep learning in the contrastive clustering have created a new clustering family denoted Deep Constrative clustering (Deep-CC), which allows the system to learn rich representations of users and items, capturing subtle nuances in user preferences and item characteristics. This enables more accurate modeling of user preferences and item similarities, leading to improved recommendation quality. Furthormore, contrastive learning facilitates the discovery of latent features and patterns in the data, even in scenarios with sparse or incomplete information. This can help address common challenges in recommender systems, such as cold-start problems and data sparsity issues. By incorporating contrastive learning techniques, collaborative filtering and recommender systems can provide more personalized and relevant recommendations, ultimately enhancing user satisfaction and engagement [15].

#### 1.5.2.1 Basic Architecture

The proposed architecture for the contrastive clustering model encompasses several interconnected components designed to effectively learn representations of input data and predict meaningful cluster assignments. Each component plays a crucial role in extracting features, generating instance embeddings, and predicting cluster memberships.

As Illustrated in the Figure 1.8: The figure depicts the detailed structure of the contrastive clustering model, showcasing the flow of information and interactions between different components. The architecture comprises distinct layers, each contributing to the overall functionality of the model.

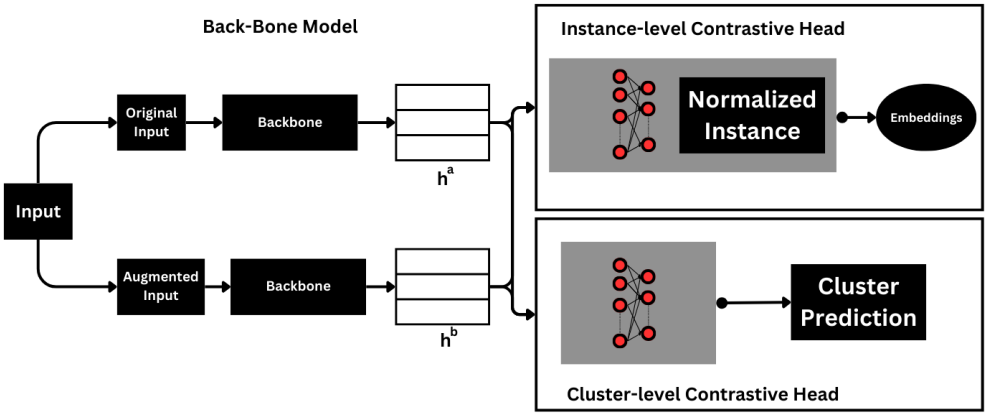



Figure 1.8: Deep-CC Architecture

- Inputs : This represents the original input to the model, serving as the data input for the first branch of the contrastive model. Additionally, the model takes another input , which represents an augmented version of the original input, providing diversity to the training process.
- Backbone Model: The backbone model, which takes both inputs (Original and Augmented one), extracts features from the data. It consists of multiple layers of neurons, such as dense layers and dropout layers, enabling it to extract meaningful features from both the original and augmented data inputs.
- Instance-Level Contrastive Head: Following the backbone model, this part of the network takes the features extracted from both inputs A and B and generates instance embeddings for each. These embeddings represent the unique characteristics of each instance in the context of both the original and augmented data.
- Normalized Instance Embeddings: The instance embeddings are then normalized using the L2 normalization function to ensure consistency in scale across different instances and inputs.
- Cluster-Level Contrastive Head: Once the normalized instance embeddings are obtained, this part of the network predicts the clusters to which the data belongs. It generates softmax outputs representing the probability of each instance belonging to different clusters.
- Cluster Predictions: Finally, the cluster predictions are obtained by taking the argmax of the softmax outputs from the Cluster-Level Contrastive Head. These predictions provide insights into the clustering structure of the data and can be used for various downstream tasks such as recommendation and classification.

The contrastive model takes two instances of data as input, extracts their features using a backbone model, generates normalized instance embeddings considering both original and augmented inputs, predicts the clusters to which the data belongs, and returns the cluster predictions. This approach enhances the model's ability to capture diverse representations of the data and effectively learn meaningful cluster boundaries. What's also worth precising is that in contrastive clustering, the number of clusters is not explicitly predefined within the model architecture. Instead, the model learns embeddings for each input instance and predicts the clusters based on these embeddings. [15].

#### 1.5.2.2 Simultaneous Learning of Deep-CC

Simultaneous learning of assignments and embeddings is a critical aspect of the proposed architecture, facilitating the model's ability to jointly optimize both cluster assignments and instance embeddings. Unlike

traditional clustering algorithms where assignments and embeddings are typically learned sequentially or through separate optimization steps, this simultaneous learning approach allows for mutual reinforcement and improved convergence of the model. By integrating the assignment and embedding learning processes within a unified framework, the model can leverage the interdependencies between them to enhance clustering performance. This simultaneous learning strategy not only accelerates convergence but also enables the model to capture complex patterns and relationships in the data more effectively. Additionally, by jointly optimizing assignments and embeddings, the model can adapt dynamically to changes in the data distribution and exhibit greater robustness in diverse real-world scenarios. Overall, simultaneous learning of assignments and embeddings represents a key advancement in contrastive clustering methodologies, contributing to more efficient and accurate clustering solutions [15].

#### 1.5.2.3 Alternate Learning of Deep-CC

Deep-CC embeddings serve as valuable feature representations that can enhance various techniques, including recommendation systems. These embeddings capture rich semantic information about the data, facilitating more effective similarity computations and clustering. Moreover, Deep-CC is often combined with other clustering algorithms like KMeans to refine clustering results. By initializing KMeans centroids with Deep-CC embeddings, we leverage their semantic richness to guide the clustering process, resulting in more meaningful and coherent clusters [21].

### 1.5.3 Adjusted Deep-CC Based Recommendation Model

Now, our focus shifts to a detailed exploration of the operational adaptability of the Adjusted Deep-CC Based Recommendation Model across various configurations. We will analyze its functionality and uncover any noteworthy enhancements that arise from these alternative setups.

#### 1.5.3.1 User Input Augmentation

As the concept of contrastive clustering stipulate it, it's often beneficial to generate variants of the same user to enrich the dataset and improve the system's ability to capture diverse user preferences and behaviors. This process involves creating multiple representations or instances of a user, each reflecting different aspects or interactions. These variants can provide a more comprehensive view of user preferences, leading to more accurate recommendations and better user satisfaction [21] [15].

- Approach 1: Generate Pairs Using Original Data, algorithm assumes that each example in the dataset represents a positive example with itself. In this approach, the data is used as is, with each user serving as their own variant. While this method is simple and straightforward, it may not capture the full diversity of user behaviors, especially if the dataset is limited or if users exhibit complex and varied preferences.

**Algorithm 6** Generate Pairs Using Original Data

**Require:** Data $D$
**Ensure:** Pairs of data $(x_i, x_i)$
1: **for** each example $x_i$ in $D$ **do**
2: Add pair $(x_i, x_i)$ to output pairs
3: **end for**
4: **return** Output pairs

- Approach 2: Generate Random Pairs algorithm randomly generates pairs of data instances. It first selects indices randomly from the dataset to create pairs of instances. By introducing randomness into the pairing process, this approach can potentially capture a wider range of user interactions and preferences. However, since pairs are chosen randomly, there's a risk of generating redundant or uninformative pairs, particularly if the dataset is large or if certain user-item interactions are rare.

**Algorithm 7** Generate Random Pairs

**Require:** Data $D$, Number of samples $n$
**Ensure:** Random pairs of data
1: **for** $i = 1$ to $n$ **do**
2: Randomly select indices $i_a$ and $i_b$ from 1 to $|D|$
3: Add pair $(D[i_a], D[i_b])$ to output pairs
4: **end for**
5: **return** Output pairs

- Approach 3: Generate All Possible Pairs algorithm creates all possible pairs of data instances by systematically combining each example with every other example in the dataset. This exhaustive approach ensures that every possible combination of user variants is considered. While this method guarantees comprehensive coverage of user interactions, it can be computationally expensive, especially for large datasets, as it generates a quadratic number of pairs. Additionally, the resulting pairs may include many redundant or similar instances, leading to potential overfitting or inefficiency in the learning process.

**Algorithm 8** Generate All Possible Pairs

**Require:** Data $D$
**Ensure:** All possible pairs of data
1: **for** $i = 1$ to $|D|$ **do**
2: **for** $j = i + 1$ to $|D|$ **do**
3: Add pair $(D[i], D[j])$ to output pairs
4: **end for**
5: **end for**
6: **return** Output pairs

#### 1.5.3.2 Contrastive Clustering Embeddings with K-Means

Contrastive clustering embeddings in K-Means is an approach that leverages embeddings generated by a contrastive clustering model to improve the performance of traditional K-Means clustering. As illustrated in Figure 1.9 Here's how the process unfolds:

- **Embedding Generation**: After training the contrastive clustering model, embeddings are generated for each data instance. These embeddings represent the unique characteristics of the data instances learned during the contrastive learning process.
- **Encoding of Input Data (Step 1)** : The input data is encoded into embeddings using the trained contrastive clustering model. This encoding process captures the distinctive features and characteristics of each data instance in a lower-dimensional space.
- **Clustering with K-Means (Step 3):** The embeddings, generated without explicit normalization in this implementation, serve as input to the K-Means clustering algorithm. K-Means partitions the data into clusters by iteratively assigning data points to the nearest centroid and updating the centroids based on the mean of the assigned points.

- **Evaluation and Refinement (Step 4):** The quality of the resulting clusters is evaluated. If necessary, the clustering process can be refined by adjusting hyperparameters or retraining the contrastive clustering model.

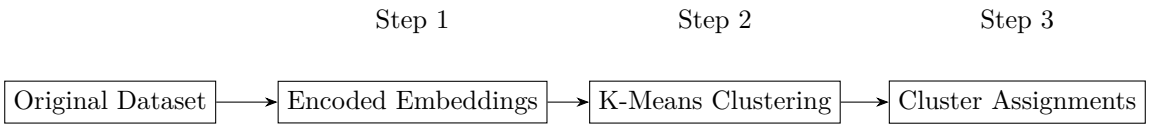



Figure 1.9: Process of Encoding and Clustering

By utilizing embeddings generated by a contrastive clustering model, this approach enhances the performance of K-Means clustering by providing more meaningful representations of the data. The contrastive embeddings capture complex relationships and patterns in the data, leading to improved cluster assignments and better overall clustering results.

### 1.5.4 Cluster-Based Collaborative Filtering with Similarity Weighting ($CB$-$CF_{SW}$)

The Cluster-Based Collaborative Filtering with Similarity Weighting approach ($CB$-$CF_{SW}$) enhances the traditional collaborative filtering technique by incorporating cluster-based probabilities and similarity measures to improve the accuracy of recommendations. The model leverages the clustering probabilities to adjust the weights of user similarities, ultimately refining the predicted ratings for items.

After obtaining the probabilities of each user belonging to different clusters. those are then incorporated in the computation of the new rating values of item, that are susceptible to be recommended for user. The new rating for an item is calculated by considering the ratings from similar users and adjusting them based on the cluster probabilities and similarity scores. The formula used is:

$$R_{\text{new}}(i,u) = \frac{\sum_{v \in S(u)} (R(v,i) \cdot W_{uv} \cdot C_u \cdot C_v)}{\sum_{v \in S(u)} (W_{uv} \cdot C_u \cdot C_v)} \tag{2,5}$$

where:

- $R_{\text{new}}(i,u)$ is the new rating for item $i$ by user $u$.
- $S(u)$ is the set of similar users to $u$.
- $R(v,i)$ is the rating of item $i$ by user $v$.
- $W_{uv}$ is the similarity weight between users $u$ and $v$.
- $C_u$ and $C_v$ are the cluster probabilities for users $u$ and $v$, respectively.

The cluster probabilities ($C_u$ and $C_v$) represent the degree to which a user belongs to a particular cluster. This is critical in adjusting the influence of each user's rating in the final recommendation.

This adjusted formula ensures that the cluster probabilities and user similarities are properly accounted for, leading to more accurate and personalized recommendations. The $CB$-$CF_{SW}$ model thus represents an effective blend of clustering techniques and collaborative filtering, used to show the specific strengths of each approach.

### 1.5.5 Comparison Clustering Algorithms

- **KMeans clustering** is a widely-used clustering algorithm known for its simplicity and efficiency. It partitions the data into K clusters by iteratively assigning data points to the nearest centroid and updating the centroids based on the mean of the assigned points. In the context of collaborative filtering (CF), KMeans clustering can be used to group similar users or items based on their features or preferences. It serves as a straightforward baseline method for evaluating the performance of more complex clustering techniques like contrastive clustering. The number of clusters, denoted by $K$, is typically predefined by the user. Finding the optimal $K$ can be done using techniques like the elbow method or silhouette score analysis. However, for simplicity, a reasonable starting point is often chosen based on domain knowledge or trial and error

**Algorithm 9** KMeans Clustering

**Input**: Data $X$, Number of clusters $K$
**Output**: Cluster assignments
Randomly initialize $K$ centroids
**while** Not converged **do**
    Assign each data point to the nearest centroid
    Update centroids as the mean of data points in each cluster
**end while**
**return** Cluster assignments

The KMeans clustering algorithm iteratively assigns data points to the nearest centroid and updates the centroids as the mean of the data points in each cluster until convergence. It aims to partition the data into K clusters based on minimizing the within-cluster sum of squares.

- **Agglomerative clustering** is a hierarchical clustering algorithm that starts with each data point as a separate cluster and iteratively merges the closest clusters until a stopping criterion is met. This method is advantageous for its ability to capture hierarchical structures in the data. In CF, agglomerative clustering can be applied to uncover clusters of users or items with similar preferences or behavior patterns. It offers a different perspective on clustering compared to KMeans and can provide insights into the underlying data structure. Unlike KMeans, agglomerative clustering does not require the number of clusters to be predefined. Instead, it starts with each data point as its own cluster and merges them iteratively based on a distance metric until a predefined number of clusters is reached or until a stopping criterion is met.

**Algorithm 10** Agglomerative Clustering

**Input**: Data $X$, Number of clusters $K$
**Output**: Cluster assignments
Compute the pairwise distance matrix between data points
Initialize each data point as a singleton cluster
**while** Number of clusters $> K$ **do**
    Merge the two closest clusters
    Update the pairwise distance matrix
**end while**
**return** Cluster assignments

Agglomerative clustering iteratively merges the closest clusters until the desired number of clusters is reached. It forms a hierarchical clustering structure by iteratively combining clusters based on a chosen linkage criterion, such as single linkage, complete linkage, or average linkage. GMM clustering also does not require the number of clusters to be predefined. It fits a mixture of Gaussian distributions to the data, and the number of components (clusters) can be determined using techniques like the Bayesian

Information Criterion (BIC) or the Akaike Information Criterion (AIC), which penalize models with more parameters.

- **Gaussian Mixture Models (GMM)** clustering assumes that the data points are generated from a mixture of several Gaussian distributions. It assigns probabilities to each point belonging to each cluster and iteratively adjusts the parameters of the Gaussian distributions to maximize the likelihood of the observed data. GMM clustering is beneficial for its flexibility in capturing complex data distributions. In CF, GMM clustering can be useful for identifying latent clusters of users or items with overlapping preferences or characteristics.

**Algorithm 11** Gaussian Mixture Models Clustering (GMM)

**Input**: Data $X$, Number of clusters $K$
**Output**: Cluster assignments
Initialize the parameters of $K$ Gaussian distributions
**while** Not converged **do**
    E-step: Compute the probability of each data point belonging to each cluster
    M-step: Update the parameters of the Gaussian distributions based on the data points assigned to each cluster
**end while**
**return** Cluster assignments

GMM Clustering initializes the parameters of $K$ Gaussian distributions and iteratively updates them until convergence. In the E-step, it computes the probability of each data point belonging to each cluster, and in the M-step, it updates the parameters of the Gaussian distributions based on the data points assigned to each cluster. Finally, it returns the cluster assignments.

- **Self-Organizing Maps (SOM)** clustering is a type of neural network-based clustering algorithm that maps high-dimensional data onto a lower-dimensional grid while preserving the topological properties of the input space. It organizes similar data points into nearby regions of the map, making it effective for visualizing and exploring high-dimensional data. In CF, SOM clustering can aid in understanding the underlying structure of user-item interactions and discovering clusters of users or items with similar preferences. The number of clusters in SOM is determined by the size and topology of the SOM grid, which is typically predefined by the user based on the problem domain and desired granularity of clustering. Larger grids may lead to more clusters, while smaller grids may result in fewer clusters.

**Algorithm 12** Self-Organizing Maps Clustering (SOM)

**Input**: Data $X$, Number of clusters $K$, SOM grid dimensions
**Output**: Cluster assignments
Initialize the SOM grid with random weights
**for** each data point $x_i$ **do**
    Find the Best Matching Unit (BMU) in the SOM grid
    Update the weights of the BMU and its neighboring units
**end for**
**return** Cluster assignments based on the SOM grid

SOM Clustering initializes a SOM grid with random weights and iteratively updates them based on the input data points. For each data point, it finds the Best Matching Unit (BMU) in the SOM grid and updates the weights of the BMU and its neighboring units. Finally, it returns cluster assignments based on the SOM grid.

We will utilize the resulted clusters to improve the collaborative filtering process. By incorporating the cluster assignments obtained from clustering, we aim to enhance the collaborative filtering process by grouping

users or items with similar characteristics into clusters. This approach enables the recommendation system to capture finer-grained patterns and preferences, leading to more accurate and relevant recommendations for users. Additionally, manipulating the clusters allows for a more interpretable and actionable understanding of user preferences, facilitating insights into user behavior and preferences beyond traditional collaborative filtering methods.

The forthcoming diagram Figure 1.10 illustrates the sequential execution of collaborative filtering within each cluster generated by the clustering models. This iterative process enables us to delve deeper into the dynamics of user-item interactions within distinct clusters, facilitating a more nuanced understanding of user preferences and item affinities

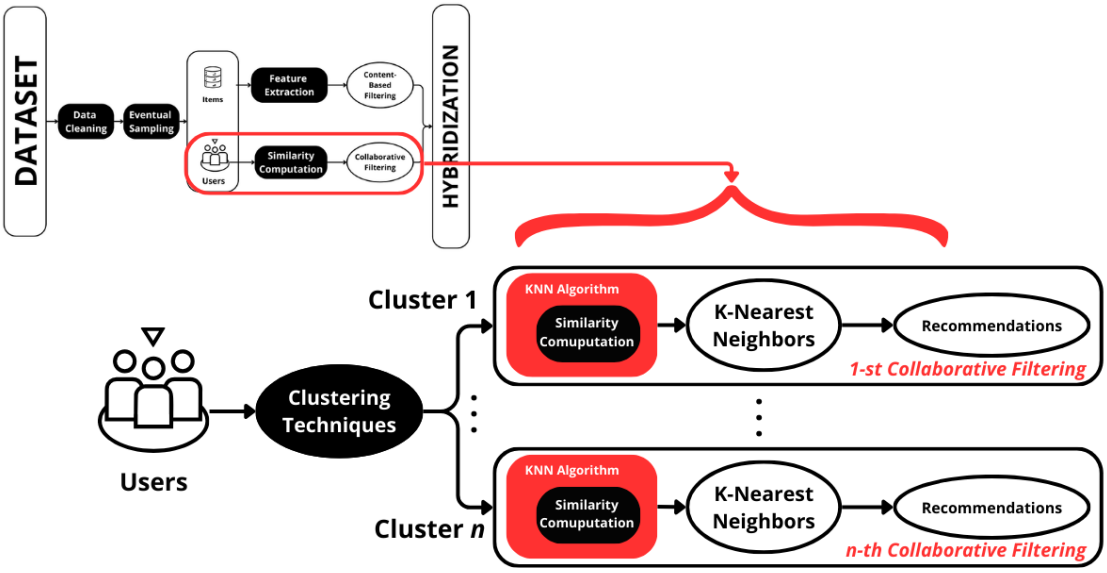



Figure 1.10: Improved Version of Our Approach's Architecture Using Clustering Techniques

## 1.6 Hybridization

In our continuous quest to refine and augment our recommendation framework, we transition to the pivotal phase of hybridization. Hybridization serves as a foundation in the evolution of recommender systems, combining the strengths of multiple recommendation techniques to overcome individual limitations and deliver more accurate and personalized recommendations to users.

This chapter delves into the fusion of Content-Based Filtering with traditional collaborative filtering methods, marking a significant advancement in our approach. By integrating Content-Based Filtering insights with collaborative filtering algorithms, we aim to leverage the rich hierarchical structure captured during clustering to enhance the recommendation process. This hybridization strategy promises to provide recommendations that are not only more precise but also more diverse and customized to the unique preferences of each user.

Throughout this chapter, we will explore various hybrid recommendation strategies, ranging from simple ensemble approaches to more sophisticated blending techniques. Each strategy aims to capitalize on the complementary nature of collaborative clustering and collaborative filtering, thereby elevating the performance and utility of our recommendation system.

- **Multi-level Hybridization**

  Multi-level hybridization in recommender systems involves integrating multiple recommendation techniques at different levels of the recommendation process. This approach aims to leverage the strengths of diverse recommendation algorithms and strategies to enhance the accuracy, diversity, and coverage of recommendations. It is used to combine the strengths of different recommendation techniques to improve the overall performance and accuracy of the recommender system. By integrating methods such as collaborative filtering, content-based filtering, and others at multiple levels, it aims to leverage the unique advantages of each approach [12].

  Combining various techniques enhances the accuracy of the system's recommendations by leveraging the strengths of each approach. Additionally, it addresses the sparsity issue commonly encountered in collaborative filtering, thus ensuring more comprehensive and reliable recommendations. Moreover, this integration boosts the system's robustness, enabling it to effectively handle cold-start problems and noisy data, ultimately improving the overall performance and user experience.

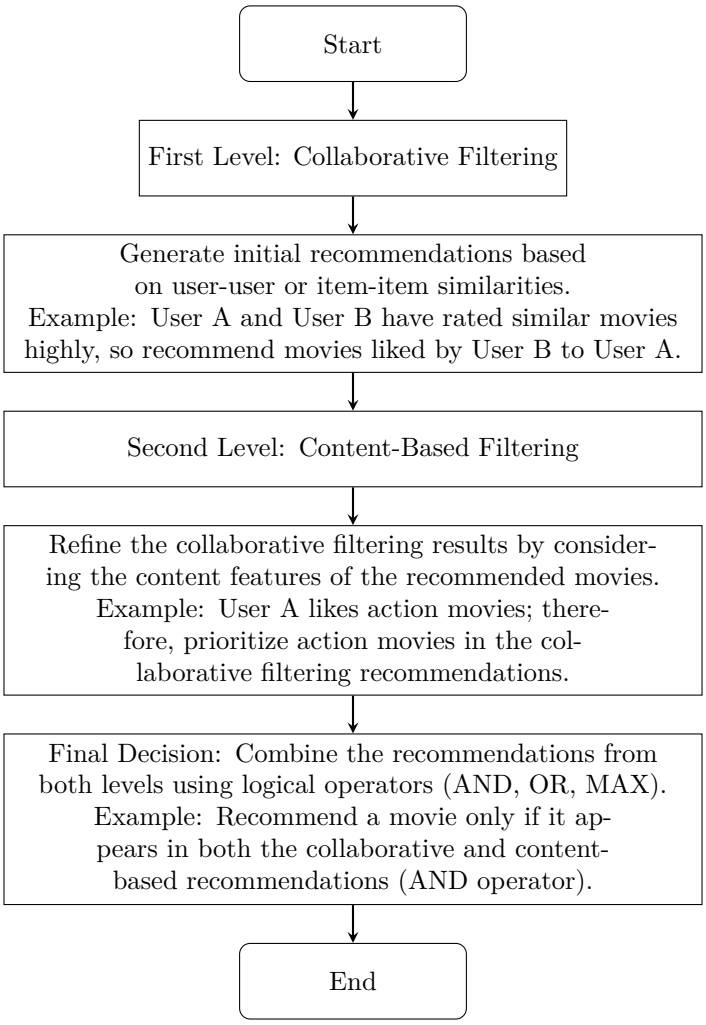



Figure 1.11: Working Process of Multi-level Hybridization

Figure 1.11 illustrates the working process of a hybrid recommendation system using Multi-level Hybridization, combining collaborative filtering and content-based filtering techniques. The process begins with collaborative filtering generating initial recommendations based on user-item similarities. Subsequently, content-based filtering refines these recommendations by considering item content features.

Finally, recommendations from both levels are combined using logical operators to make the final decision. This process ensures the generation of personalized recommendations for users.

- **Multi-level Hybridization On Clusters**

  In attempt to improve the multi-level hybridization approach defined earlier, our focus extends to the clusters formed through the clustering techniques deliberated upon in the preceding sections of this chapter. By applying this method to these clusters, we aim to enhance the sophistication of our recommendation system. Through clustering, we can segment users into distinct groups based on similarities, enabling a more nuanced understanding of user preferences and item characteristics. This approach not only enhances recommendation accuracy but also addresses challenges such as data sparsity and cold-start problems, thereby advancing the efficacy of our recommendation system in delivering personalized and relevant suggestions to users.

- **Unified Collaborative-Content Filtering Hybridization (UCCFH)**

  Incorporating content-based filtering into collaborative filtering creates a unified model that can exploit both user-item interactions and item content features, leading to more comprehensive and accurate recommendations.

  Figure 1.12 showcases the workflow of the Unified Collaborative-Content Filtering Hybridization involved in recommending songs within a music streaming service, for instance. The process commences with Collaborative Filtering, where recommendations are generated based on user listening history and similarities. Subsequently, in Content-Based Filtering, song attributes are integrated to refine the recommendations, prioritizing songs aligned with user preferences. Following this, a Unified Model is developed, incorporating both user-item interaction data and item content features, enhancing recommendation accuracy. Finally, in the Final Recommendations stage, a balanced list of recommended songs is provided, drawing insights from both collaborative and content-based approaches. This workflow ensures the delivery of personalized and relevant song suggestions to users, facilitating an enriched music streaming experience.

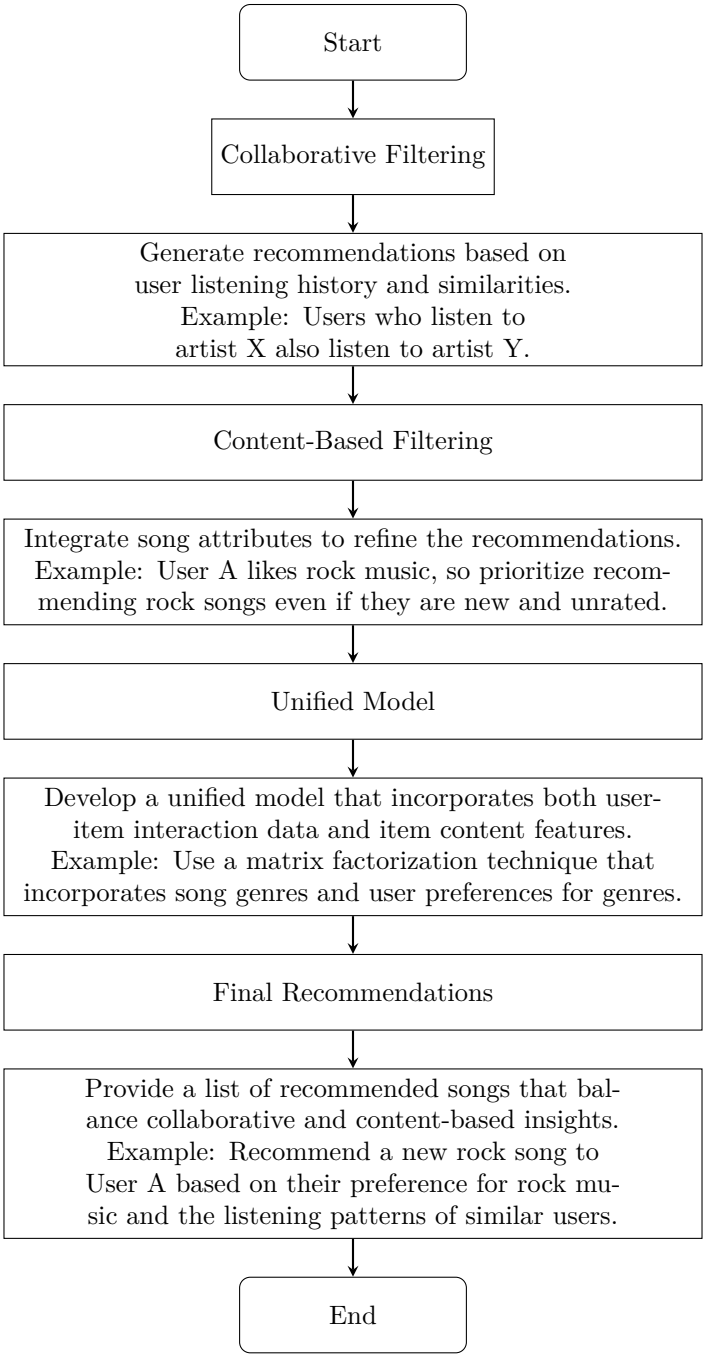



Figure 1.12: Workflow of Unified Collaborative-Content Filtering Hybridization

### 1.6.1• Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting

In our proposed *Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting (Hybrid* $CB\text{-}CF_{SW}$*)* approach, we enhance the recommendation system by integrating the content-based filtering (CBF) ratings directly into the cluster-based collaborative filtering with similarity weighting ($CB\text{-}CF_{SW}$) framework. This hybrid method aims to improve the accuracy and personalization of recommendations by leveraging the strengths of both collaborative and content-based approaches.

The Hybrid $CB\text{-}CF_{SW}$ method starts by calculating the ratings from the $CB\text{-}CF_{SW}$ technique. $CB\text{-}CF_{SW}$ clusters users based on their preferences and computes similarity scores among users within the same cluster. These similarity scores are then used to weight the influence of neighboring users' ratings on the target user's predicted ratings.

Simultaneously, we obtain ratings from a content-based filtering system, referred to as CBF rating. These ratings are determined based on the similarity of item features, providing a personalized rating that reflects the content characteristics of the items.

The hybrid approach combines these two sets of ratings, those derived from $CB\text{-}CF_{SW}$ and those from CBF, by using a weighted average formula. The integration process involves two parameters, $\alpha$ and $\beta$, which determine the contribution of each method to the final rating. The parameter $\alpha$ controls the influence of the $CB\text{-}CF_{SW}$ ratings, while $\beta$ controls the influence of the CBF ratings.

The final rating $a$ for a given item is computed using the following formula:

$$a = \frac{\sum \text{ of weighted } CB\text{-}CF_{SW} \text{ ratings}}{\sum \text{ of weights}} \times \alpha + \text{CBF rating} \times \beta \tag{2,6}$$

Where:

- Sum of weighted $CB\text{-}CF_{SW}$ ratings is calculated based on the similarity between the target user and other users within the same cluster, taking into account the cluster affiliation of the users.
- CBF rating is the rating predicted by the content-based filtering system, reflecting the item's content features.
- Sum of weights is the sum of all the similarity weights used in the $CB\text{-}CF_{SW}$ approach.

By incorporating the CBF ratings into the $CB\text{-}CF_{SW}$ framework, the Hybrid $CB\text{-}CF_{SW}$ method effectively balances the collaborative filtering and content-based approaches. This combination enhances the overall accuracy and relevance of the recommendations, providing users with more personalized and contextually appropriate suggestions. The use of $\alpha$ and $\beta$ parameters allows for flexible adjustment, ensuring that the system can optimally integrate the strengths of both techniques to meet the specific needs and preferences of users.

## 1.7 Final Proposed Model

As a summary, of the technologies we described prior to this section, we propose the schema to the final proposed model as illustrated in 1.13, it integrates the various techniques we've explained before to deliver a robust and effective recommendation system. The process begins with dataset preparation, including data cleaning and sampling. The model utilizes both Content-Based Filtering (CBF) and Collaborative Filtering (CF). On the content-based side, it processes visual data using deep learning technique which is VGG16 for feature extraction, and textual data through embedding learning and transformer-based models. These features are then combined and their similarities computed. For collaborative filtering, the model employs similarity computation and the K-Nearest Neighbors (KNN) algorithm to generate recommendations based on user similarities. Clustering techniques are applied to group users, with dimensionality reduction enhancing clustering efficiency. Users are distributed across clusters, and the model leverages these clusters to refine recommendations through multi-level fusion. The final hybrid techniques combine content-based and collaborative filtering recommendations, employing a cluster-based collaborative filtering with similarity weighting $CB\text{-}CF_{SW}$ for an enriched recommendation output. This integrated approach ensures that it's able to deliver highly personalized and accurate recommendations by exploiting the strengths of both content and collaborative methodologies, combined with advanced clustering and hybridization techniques.

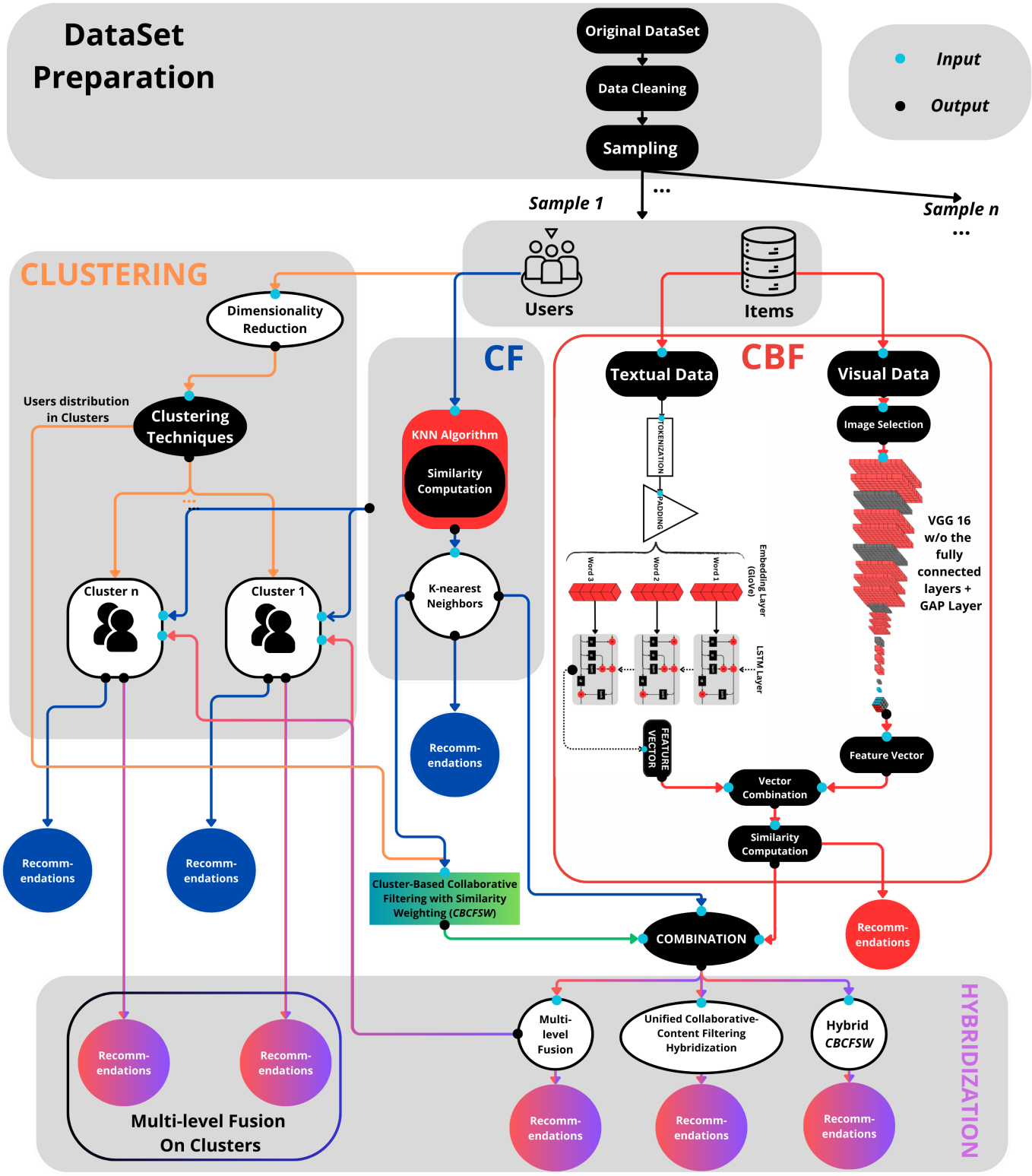



Figure 1.13: Final Proposed Architecture

## 1.8 Conclusion

Our exploration into the conception of our approach commenced with the establishment of an initial model, serving as the architectural blueprint for our recommendation system. This framework evolved systematically as new components of complexity were added.

Incorporating content-based recommendation techniques, we delved into the nuances of visual data processing and textual-based models, leveraging advancements in feature extraction and semantic understanding. These enhancements enriched our model's capacity for accurate and personalized recommendations.

Simultaneously, collaborative filtering methodologies were integrated, employing user interactions to augment recommendation diversity and relevance. Complementary clustering techniques, particularly Deep Contrastive Clustering (Deep-CC), facilitated efficient user segmentation and preference adaptation.

The culmination of these techniques provided a hybrid recommendation model, seamlessly combining collaborative filtering, content-based recommendation, and clustering methodologies. This Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting model represents a robust solution to recommendation challenges, promising enhanced user experiences and satisfaction.

In the forthcoming chapter, we shall delve into the implementation and results of this conceptual framework, elucidating the practical implications of our approach.

# Chapter 2

# Experiments and Results

## 2.1 Work Environement

Our work environment is equipped with hardware resources comprising 8 GB of RAM and an Intel® Core™ i5-8350U CPU @ 1.70GHz, providing ample computational power for our tasks. Operating on a 64-bit operating system, x64-based processor, our setup ensures smooth execution and efficient performance. Python, a versatile and widely-used scripting language, serves as the backbone of our implementation. Python's high-level, portable, dynamic, and open-source nature facilitates rapid prototyping and development of our recommendation algorithms. Our experimentation and analysis are conducted within the Jupyter Notebook environment, leveraging its interactive and collaborative features for seamless workflow management and documentation.

## 2.2 Dataset Description

The dataset we chose to conduct our approach is a Kaagle dataset, called Recipes and Reviews with Data on over 500,000 recipes and 1,400,000 reviews from Food.com, that were scrapped using a web scrapping technique. The project is composed of two dataset *recipes.csv* and *reviews.csv*. The recipes dataset "*recipes.csv*" contains 522,517 recipes from 312 different categories. This dataset provides information about each recipe like cooking times, servings, ingredients, nutrition, instructions, and more. The reviews dataset "*reviews.csv*" contains 1,401,982 reviews from 271,907 different users. This dataset provides information about the author, rating, review text, and more.

For further understanding of the datasets we have done some data analysis, on each one to explore the characteristics of each dataset, and gather useful informations about its purpose and eventual utility in our work :

- **First DataSet : "***recipes.csv*"

| **Overview of Dataset 1** | |
|---|---|
| Number of rows | 522,517 |
| Number of columns | 28 |
| Total missing values | 1,132,236 |
| Percentage of missing values | 7.74% |
| Duplicated rows | 0 |
| **Data Types** | **Count** |
| object | 14 |
| float64 | 12 |
| int64 | 2 |

Table 2.1: Overview of Dataset 1

The dataset contains 28 columns, and each stands for a specific feature related to the recipe :

| **Column Name** | **Description** | **Data Type** | **Missing Values** | **Unique Values** |
|---|---|---|---|---|
| RecipeId | A unique identifier for each recipe. | int64 | 0 | 522517 |
| Name | The name of the recipe. | object | 0 | 438188 |
| AuthorId | A unique identifier for the author of the recipe. | int64 | 0 | 57178 |
| AuthorName | The name of the author of the recipe. | object | 0 | 56793 |
| CookTime | The cooking time required for the recipe. | object | 82545 | 490 |
| PrepTime | The preparation time required for the recipe. | object | 0 | 318 |
| TotalTime | The total time required for the recipe, including preparation and cooking. | object | 0 | 1240 |
| DatePublished | The date when the recipe was published. | object | 0 | 245540 |
| Description | A description of the recipe. | object | 5 | 492838 |
| Images | Links or identifiers of images associated with the recipe. | object | 1 | 165889 |
| RecipeCategory | The category of the recipe (e.g., dessert, main course). | object | 751 | 311 |
| Keywords | Keywords associated with the recipe for easier search. | object | 17237 | 216569 |
| RecipeIngredientQuantities | Quantities of the ingredients needed for the recipe. | object | 3 | 459571 |
| RecipeIngredientParts | Names of the ingredients needed for the recipe. | object | 0 | 497120 |
| AggregatedRating | The average aggregated rating of the recipe. | float64 | 253223 | 9 |
| ReviewCount | The number of reviews or ratings left for the recipe. | float64 | 247489 | 420 |
| Calories | The number of calories per serving of the recipe. | float64 | 0 | 30138 |
| FatContent | The fat content per serving of the recipe. | float64 | 0 | 4523 |
| SaturatedFatContent | The saturated fat content per serving of the recipe. | float64 | 0 | 2533 |
| CholesterolContent | The cholesterol content per serving of the recipe. | float64 | 0 | 9803 |
| SodiumContent | The sodium content per serving of the recipe. | float64 | 0 | 40455 |
| CarbohydrateContent | The carbohydrate content per serving of the recipe. | float64 | 0 | 8102 |
| FiberContent | The fiber content per serving of the recipe. | float64 | 0 | 1067 |
| SugarContent | The sugar content per serving of the recipe. | float64 | 0 | 6008 |
| ProteinContent | The protein content per serving of the recipe. | float64 | 0 | 2581 |
| RecipeServings | The number of servings the recipe yields. | float64 | 182911 | 171 |
| RecipeYield | The yield of the recipe, often similar to RecipeServings but sometimes expressed differently. | object | 348071 | 34043 |
| RecipeInstructions | Step-by-step instructions to prepare the recipe. | object | 0 | 519993 |

Table 2.2: Overview of Dataset Columns

To gain insights into the relationships between different variables in our dataset, we calculated the correlation matrix. This matrix provides a numerical representation of the strength and direction of linear relationships between pairs of variables. A correlation value close to 1 indicates a strong positive correlation, while a value close to -1 indicates a strong negative correlation. A correlation value around 0 suggests no linear relationship between the variables. Analyzing the correlation matrix helps us identify potential patterns and dependencies that may influence our recommendation model. Below is the correlation matrix visualized as a heatmap.

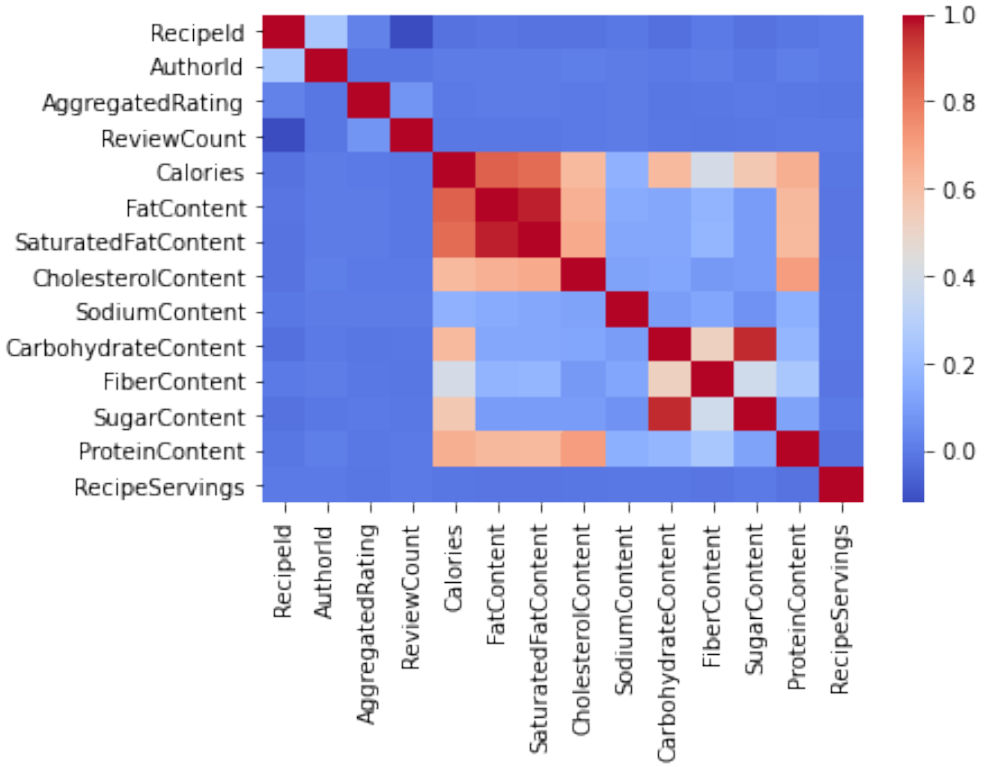



Figure 2.1: Correlation Matrix Heatmap for recipes.csv

In our recommender systems research focusing on visual data, one of the most crucial columns in our dataset is the "Images" column. This column serves as a vital component in enhancing the recommendation process by providing visual representations of the recipes. Images play a significant role in capturing users' attention and conveying important information about the recipes, such as their appearance, presentation, and key ingredients. Incorporating images into our recommendation system enables users to make more informed decisions and enhances their overall browsing experience. Additionally, visual content has been shown to increase engagement and satisfaction among users, leading to improved user satisfaction and retention rates. Therefore, exploiting the images column in our dataset allows us to develop more effective and visually appealing recommendation algorithms that understands users' preferences and enhance their culinary exploration journey.

| Statistic | Value | Explanation |
|---|---|---|
| Count | 522516 | Total number of entries in the column |
| Unique | 165889 | Number of unique images |
| Top | character(0) | Most frequent image |
| Frequency | 356620 | Frequency of the top image |
| Number of Unique Images | 165889 | Number of unique images |
| Maximum Images per Row | 1609 | Maximum number of images in a single row |
| Minimum Images per Row | 0 | Minimum number of images in a single row |
| Average Images per Row | 4.54 | Average number of images per row |

Table 2.3: Statistics of the Images Column

To refine our model, we also incorporated textual data in our approach alongside visual informations, emphasizing the significance of exploring the "Description" column. Utilizing Python and libraries like pandas and NLTK, we extracted key statistics. By analyzing metrics such as description lengths and vocabulary sizes, we gained valuable insights into the textual characteristics of our dataset. This analysis enables us to effectively leverage textual information alongside visual data, enhancing the accuracy and relevance of our recommender system.

| Statistic | Value | Explanation |
|---|---|---|
| Number of Descriptions | 522517 | Total number of descriptions in the dataset |
| Number of Unique Descriptions | 492839 | Number of unique descriptions in the dataset |
| Maximum Description Length | 6325 characters | Maximum length of a description in characters |
| Minimum Description Length | 3 characters | Minimum length of a description in characters |
| Average Description Length | 186.80 characters | Average length of descriptions in characters |
| Maximum Vocabulary Size | 499 words | Maximum vocabulary size (unique words) in a description |
| Minimum Vocabulary Size | 1 word | Minimum vocabulary size (unique words) in a description |
| Average Vocabulary Size | 30.79 words | Average vocabulary size (unique words) in descriptions |

Table 2.4: Textual Statistics for Description Column

In our content-based filtering approach, we use both image and text data from the "Images" and "Description" columns, respectively.

- **Limitations** :

One limitation of our dataset is the variability in description lengths and vocabulary sizes, which can complicate natural language processing tasks. Additionally, the structure of image links, presented in concatenated format, poses challenges for direct access and manipulation. To address these issues, preprocessing steps are required, such as text normalization for uniformity in textual data and custom parsing algorithms for extracting individual image links. These measures ensure data consistency and facilitate efficient processing for improved model performance.

- **Second Dataset** : "*reviews.csv*"

The "reviews" dataset serves as a complementary resource to the "recipes" dataset, providing insights into user interactions and opinions on the recipes. By analyzing reviews, we gain valuable feedback and sentiment regarding recipe quality and user experiences, enriching our understanding of recipe popularity and user preferences. Here are the statistics for the "reviews" DataFrame:

| Statistic | Value |
|---|---|
| Number of rows | 1,401,982 |
| Number of columns | 8 |
| Total missing values | 214 |
| Percentage of missing values | 0.00% |
| Duplicated rows | 0 |
| Data Types | int64: 4, object: 4 |

Table 2.5: Statistics for the ”reviews” DataFrame

The dataset contains 8 columns, and each stands for a specific feature related to the review :

| Column Name | Description | Data Type | Missing Values | Unique Values |
|---|---|---|---|---|
| ReviewId | Unique identifier for each review. | int64 | 0 | 1,401,982 |
| RecipeId | Unique identifier for the associated recipe. | int64 | 0 | 271,678 |
| AuthorId | Unique identifier for the author of the review. | int64 | 0 | 271,907 |
| AuthorName | Name of the author of the review. | object | 0 | 241,365 |
| Rating | Rating given to the recipe by the author. | int64 | 0 | 6 |
| Review | Textual review provided by the author. | object | 214 | 1,392,745 |
| DateSubmitted | Date when the review was submitted. | object | 0 | 1,384,268 |
| DateModified | Date when the review was last modified. | object | 0 | 1,384,268 |

Table 2.6: Overview of the ”reviews” DataFrame columns

In our approach, the ”AuthorId” column within the ”reviews” dataset plays a pivotal role as it identifies the unique authors or users who submitted reviews. Analyzing this column provides valuable insights into user engagement, preferences, and contribution patterns. We delve into a detailed analysis of the ”AuthorId” column to understand user behavior and its implications for our recommendation system.

| Analysis | Result |
|---|---|
| Number of unique authors | 271,907 |

Table 2.7: Number of Unique Authors

| Author Contribution | |
|---|---|
| Top 10 Authors | Count |
| 424680 | 8,842 |
| 37449 | 6,605 |
| 383346 | 5,438 |
| 128473 | 4,693 |
| 169430 | 4,586 |
| 89831 | 3,963 |
| 58104 | 3,743 |
| 199848 | 3,688 |
| 133174 | 3,590 |
| 305531 | 3,543 |

Table 2.8: Author Contribution

| Rating Distribution | |
|---|---|
| Rating | Count |
| 0 | 76,248 |
| 1 | 16,559 |
| 2 | 17,597 |
| 3 | 50,279 |
| 4 | 229,217 |
| 5 | 1,012,082 |

Table 2.9: Rating Distribution

| Temporal Analysis | |
|---|---|
| Year | Count |
| 2000 | 156 |
| 2001 | 3,904 |
| 2002 | 27,572 |
| 2003 | 43,820 |
| 2004 | 59,549 |
| 2005 | 84,913 |
| 2006 | 111,991 |
| 2007 | 173,397 |
| 2008 | 202,979 |
| 2009 | 191,722 |
| 2010 | 129,605 |
| 2011 | 86,178 |
| 2012 | 66,192 |
| 2013 | 57,623 |
| 2014 | 35,434 |
| 2015 | 27,636 |
| 2016 | 21,453 |
| 2017 | 32,266 |
| 2018 | 24,825 |
| 2019 | 9,475 |
| 2020 | 11,292 |

Table 2.10: Temporal Analysis

| Correlation with RecipeId | | |
|---|---|---|
| | AuthorId | RecipeId |
| AuthorId | 1.000 | 0.099 |
| RecipeId | 0.099 | 1.000 |

Table 2.11: Correlation with RecipeId

| Missing Values | |
|---|---|
| AuthorId | 0 |

Table 2.12: Missing Values

For further analysis, we generated a heatmap to visualize the correlation between different columns in the reviews dataset. This heatmap allows us to quickly identify any patterns or relationships between variables, which can be valuable for our analysis.


Figure 2.2: Correlation Matrix Heatmap for reviews.csv

## 2.3 Data Preprocessing

As discussed prior, the *recipes.csv* presented some limitations that need to be fixed and cleaned. Cleaning our data goes through two different step. One before the sampling, and the second afterwards.

- **reviews.csv :**

  We decided to create samples of our reviews datasets due to the substantial size of the original dataset and the limited computational resources available. The volume of data in the reviews dataset poses a significant challenge for processing and analysis, requiring extensive computational power and memory. By creating samples, we can reduce the dataset size to a manageable level while still preserving its essential characteristics. Additionally, sampling allows us to conduct preliminary analyses and experiments efficiently. Moreover, sampled datasets can serve as eventual valuable resources for testing and validating models and algorithms.

  The cleaning process involved several steps to ensure data integrity and quality. Initially, we identified and removed any entries with missing or invalid image links in the recipes dataset. This step aimed to filter out irrelevant or corrupted data that could potentially impact our analysis. Subsequently, we eliminated any rows with missing image information from the dataset to further refine the data quality. The cleaning process described can be expressed algebraically as follows:

  Let $D$ represent the original recipes dataset.

  - *Step 1*: Identify and remove entries with missing or invalid image links.

    $$D_1 = \{d \in D : \text{image link in } d \text{ is not missing or invalid}\} \tag{3,1}$$

  - *Step 2*: Eliminate rows with missing image information.

    $$D_2 = \{d \in D_1 : \text{image information in } d \text{ is not missing}\} \tag{3,2}$$

  The resulting dataset after these cleaning steps is $D_2$, which contains only entries with valid and complete image information, ensuring data integrity and quality for further analysis.

  Following the cleaning process for the recipes dataset, we focused on aligning the reviews dataset with the refined recipes dataset. To achieve this, we filtered the reviews dataset to include only those entries corresponding to recipes present in the cleaned recipes dataset. This alignment ensured that our reviews dataset remained consistent and relevant to the recipes under consideration.

  After cleaning the recipes dataset, the alignment process with the reviews dataset can be represented as follows:

  Let $R$ represent the original reviews dataset. Let $D_2$ represent the cleaned recipes dataset.

  Filter the reviews dataset to include only entries corresponding to recipes present in the cleaned recipes dataset.

  $$R_1 = \{r \in R : \text{recipe id in } r \text{ exists in } D_2\} \tag{3,3}$$

  The resulting dataset after this alignment process is $R_1$, which contains reviews consistent and relevant to the recipes in the cleaned recipes dataset.

  Then, we performed data sampling on the reviews dataset. This involved selecting a subset of users who had evaluated a significant number of recipes, ensuring a diverse representation of user interactions. By sampling users with substantial engagement, we obtained a manageable subset of data that retained essential characteristics of the original dataset.

  The first sample, labeled as 'sample1_df', was generated by selecting 1000 users who had evaluated more than 50 recipes each. This sample aimed to capture a broad spectrum of user interactions while maintaining a manageable dataset size. The resulting sample dataset served as a foundational dataset for subsequent analysis and model development, providing a representative subset for testing

and validation purposes." Data Preprocessing: Discuss any preprocessing steps that were performed on the dataset, such as cleaning missing values, handling outliers, or transforming variables.

Sample 2, denoted as 'sample2_df', was derived from the previously generated 'sample1_df' dataset through a randomized selection process. Initially, we shuffled the rows of 'sample1_df' to ensure a random order using the 'sample' function with the parameter 'frac=1'. This operation allowed us to maintain the integrity of the dataset while obtaining a randomized sample.

Next, we extracted a subset of unique RecipeId values from the shuffled dataset, limiting the selection to the first 2000 unique RecipeId entries. This step aimed to focus on a specific subset of recipes while ensuring diversity and representation across different recipes in the sample.

Subsequently, we filtered the original reviews dataset to retain only those rows corresponding to the selected RecipeId values from the shuffled sample. This process ensured that 'sample2_df' contained reviews associated with the chosen subset of recipes, enabling targeted analysis and investigation.

Sample 3, denoted as 'sample3_df', was created by filtering the 'reviews' dataset to include only the reviews that were modified in the most recent year.

Initially, the 'DateModified' column was converted to datetime format to facilitate temporal analysis. Subsequently, the most recent modification date was identified using the 'max()' function, providing a reference point for selecting data from the latest year.

Next, the dataset was filtered to retain only the reviews modified in the most recent year, based on the extracted year from the maximum modification date. This process ensured that 'sample3_df' contained reviews associated with the most recent updates, allowing for focused analysis and exploration of recent trends.

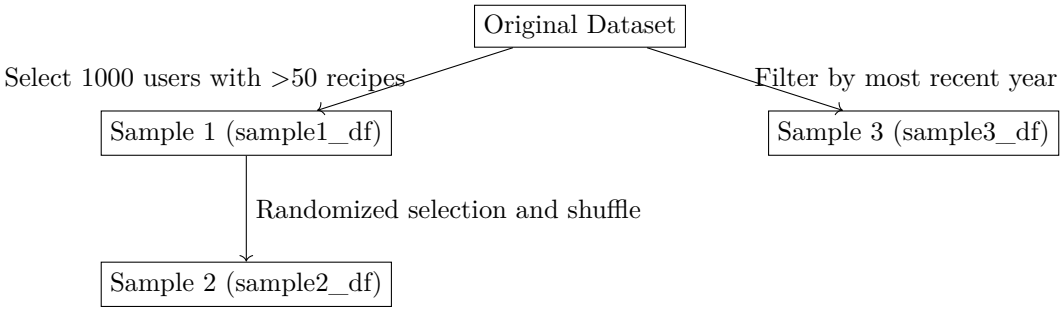



Figure 2.3: Sampling Process

- **recipes.csv :**

| Step | Description |
|---|---|
| Initial Filtering | The dataset was filtered to remove rows with missing or invalid image links. This was done by excluding rows where the 'Images' column contained the string "character(0)" or NaN. |
| Filtering Based on Reviews | The dataset was further filtered based on the RecipeIds present in the 'sample1_df' and 'sample3_df' subsets of the reviews dataset ('reviews'). This step ensured that only recipes associated with reviews in the sampled datasets were retained. |
| Entropy Calculation and Image Selection | – A function was defined to calculate the entropy of images from their URLs. (see Algorithm 1)<br>– URLs of images associated with each recipe were extracted using a regular expression.<br>– The dataset was divided into two subsets based on the number of image URLs per recipe.<br>– For recipes with multiple image URLs, the function for choosing the best image URL was applied to select the URL with the highest entropy. (see Algorithm 2)<br>– The final dataset, containing cleaned and preprocessed recipe data, including the selected best image URL for each recipe, was saved to a CSV file. |

Table 2.13: Summary of Data Cleaning and Preprocessing Process

Following the sampling process, we have decided to merge the sample datasets, namely *Sample 2* and *Sample 3*, into a single dataset to consolidate our data for further analysis. The merging operation was performed by concatenating the two datasets while retaining all rows. Subsequently, duplicate entries based on the 'ReviewId' column were removed to ensure data integrity and avoid redundancy. This merged dataset, denoted as '*merged_df*', provides a comprehensive collection of reviews while eliminating duplicate records. Additionally, considering the size of Sample 1, which may pose challenges in processing and analysis due to its large volume, we have opted to exclude it from the merged dataset for the current analysis.

**Data Analysis**: In this section, we present key visualizations to provide insights into the reviews dataset:

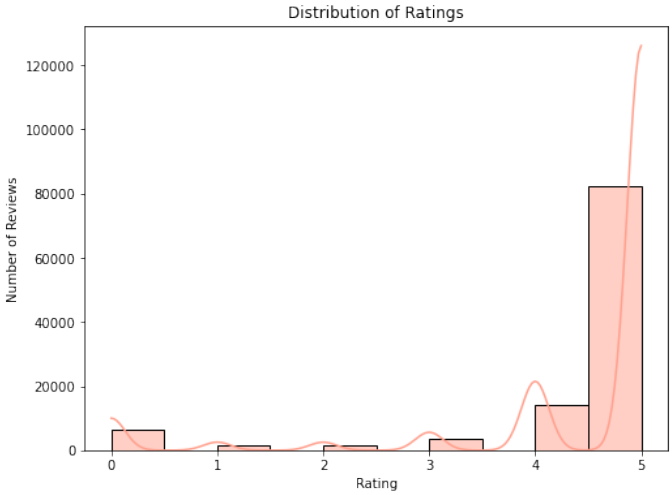
1. **Distribution of Ratings:** This histogram illustrates the distribution of ratings given by users. The x-axis represents the rating scale, while the y-axis shows the number of reviews. The distribution provides an overview of how users perceive the recipes, indicating whether they tend to give higher or lower ratings.

2. **Top 10 Authors (AuthorId) by Number of Reviews:** This bar plot displays the top 10 authors based on the number of reviews they have contributed. The x-axis represents the AuthorId, and the y-axis shows the number of reviews. It highlights the authors who have contributed the most reviews to the dataset, indicating their level of engagement or activity.

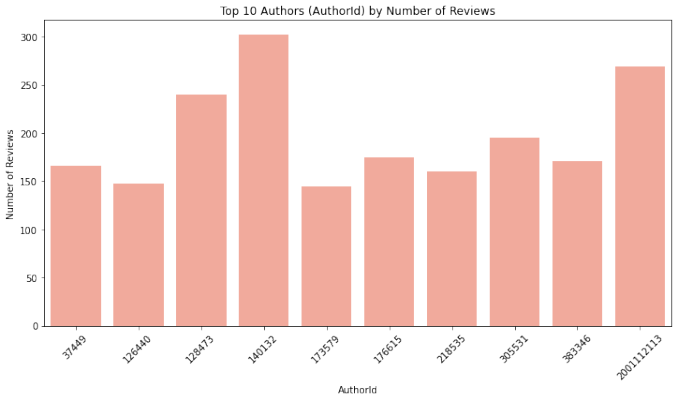
3. **Top 10 Authors (AuthorName) by Number of Reviews:** Similar to the previous plot, this bar plot shows the top 10 authors based on the number of reviews they have contributed, but using the AuthorName instead of the AuthorId. It provides insights into the most active authors by their names.

4. **Top 10 Authors (AuthorId) with Highest Mean Rating:** This bar plot depicts the top 10 authors with the highest mean ratings for their reviews. The x-axis represents the AuthorId, while the y-axis

shows the mean rating. It offers insights into authors who consistently provide highly rated reviews, indicating their influence or expertise in evaluating recipes.

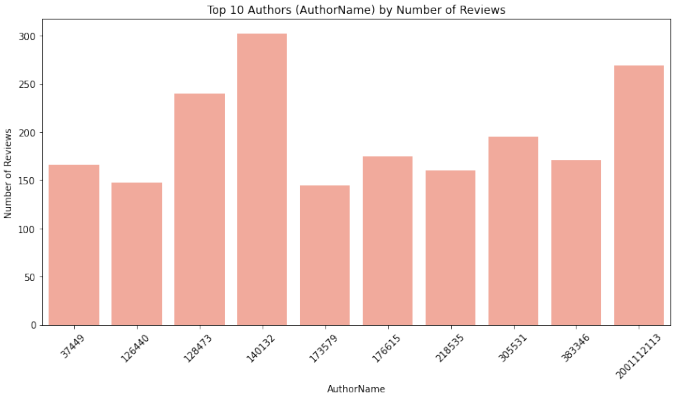



(a) Distribution of Ratings

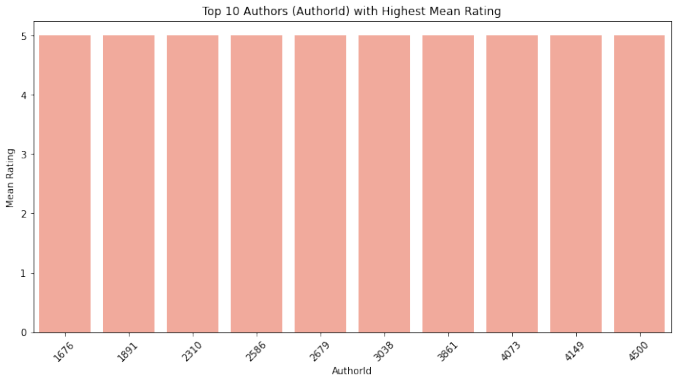



(b) Top 10 Authors (AuthorId) by Number of Reviews

Figure 2.4: Analysis of Rating Distributions and Author Review Counts




(a) Top 10 Authors (AuthorName) by Number of Reviews

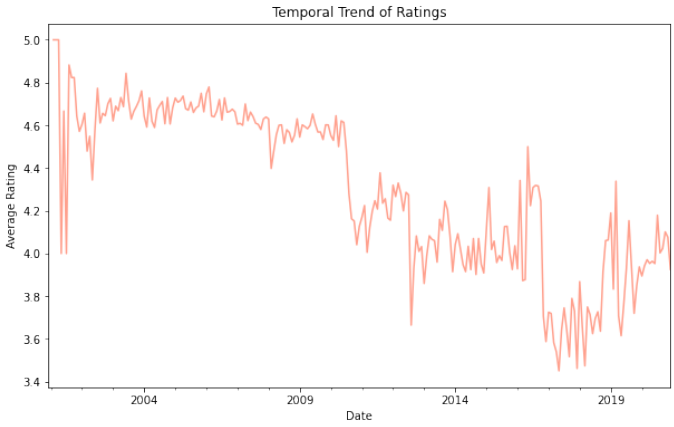



(b) Top 10 Authors (AuthorId) with Highest Mean Rating

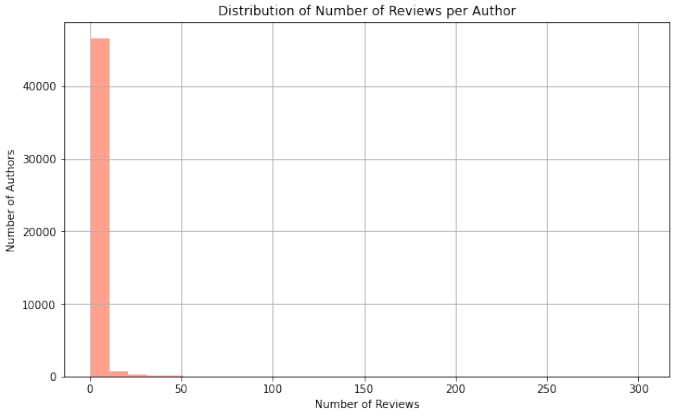
Figure 2.5: Top Authors by Review Count and Mean Rating

In addition to the previous visualizations:

5. **Temporal Trend of Ratings**: This plot illustrates the average ratings over time, revealing trends or fluctuations in user sentiment towards recipes.
6. **Author Review Distribution**: This histogram shows the distribution of reviews contributed by each author, offering insights into user engagement levels within the dataset.




(a) Temporal Trend of Ratings




(b) Distribution of Number of Reviews per Author

Figure 2.6: Rating Trends Over Time and Review Distribution per Author

To complete the analysis, we delve into sentiment analysis of the reviews dataset:

7. **Distribution of Sentiments in Review**s: Utilizing Natural Language Toolkit (NLTK), sentiment analysis is performed on each review to discern its sentiment polarity : positive, negative, or neutral. The sentiment analyzer assigns a sentiment label to each review based on its compound score. This visualization presents the distribution of sentiments across the dataset, illustrating the proportion of reviews categorized as positive, negative, or neutral. The bar plot offers insights into the overall sentiment landscape of the reviews, aiding in understanding user perceptions and sentiments towards the recipes.

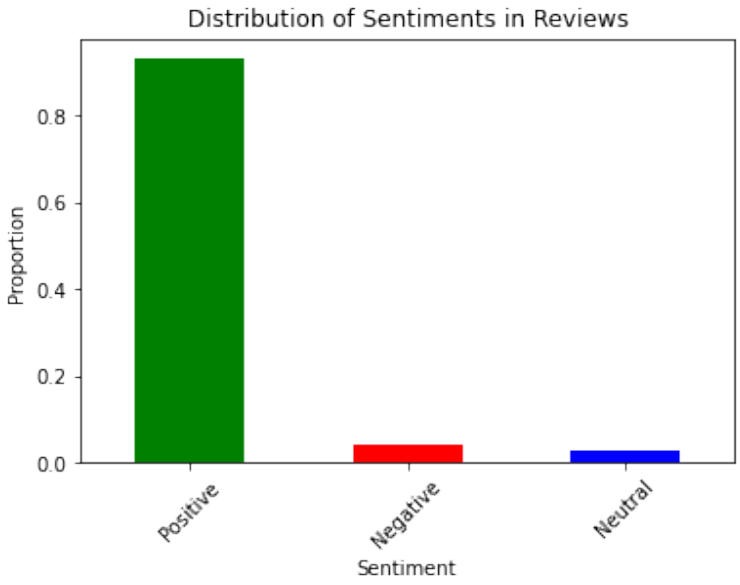



Figure 2.7: Graph Exposing the Distribution of Sentiments in reviews.csv

These visualization have managed offer an exploration of the reviews dataset, uncovering key insights into user behavior, sentiments, and trends. Through visualizations and analyses, it sheds light on various

aspects of user interactions, providing valuable knowledge for understanding recipe reviews and enhancing user experiences.

## 2.4 Experimental Scenario

1. **Content-Based Filtering**

   (a) **Extraction of Visual Data Features Using VGG16**
   **Methodology:**

   - **Model Loading:**
     The pre-trained VGG16 model is loaded with the 'imagenet' weights, excluding the fully connected layers at the top.
   - **Model Modification:**
     Global Average Pooling 2D layer is added on top of the VGG16 model to extract features from images efficiently.
   - **Freezing Layers:**
     The layers of the VGG16 model are frozen to prevent their weights from being updated during training.

   **Dataset Organization:**

   - **Image Preprocessing:**
     Images from the dataset are loaded, resized to the VGG16 input dimensions (224x224), and preprocessed to align with VGG16 input requirements.
   - **Feature Extraction:**
     A function is defined to extract image features using the modified VGG16 model, which outputs a flattened vector representing the image features.

   **Experiment Execution:**

   - **Feature Extraction Process:**
     The feature extraction function is applied to each image in the cleaned dataset : *recipes.csv*, generating a feature vector for every image that is then saved in a new column.
   - **Error Handling:**
     Exception handling is implemented to address any issues that may arise during image loading or feature extraction.

   (b) **Extraction of Textual Data Features Using Glove & LSTM**

   - **Tokenization**:
     The dataset descriptions are tokenized using the Tokenizer class provided by the Keras library. The tokenizer is fitted on the description texts to convert them into sequences of numerical tokens.
   - **Vocabulary Size**:
     The total number of unique words in the vocabulary is determined by adding 1 to the length of the tokenizer's word index.
   - **Sequence Padding**:
     To ensure uniform sequence lengths, the sequences of tokens are padded with zeros to match the length of the longest sequence.
   - **GloVe Embeddings**:
     GloVe (Global Vectors for Word Representation) word embeddings are used to represent the textual descriptions. The pre-trained GloVe embeddings are downloaded and parsed from the 'glove.840B.300d.txt' file. Using the file that contains pre-trained GloVe embeddings, specifically trained on 840 billion tokens with a 300-dimensional vector space, will provide us comprehensive and high-quality word representations

- E**mbedding Matrix Initialization**:
  An embedding matrix is initialized with zeros, with dimensions corresponding to the vocabulary size and embedding dimension.
- **Embedding Matrix Population**: Each word in the vocabulary is iterated over, and if its corresponding GloVe embedding is available, it is inserted into the embedding matrix.
- **LSTM Model Creation**:
  A sequential LSTM (Long Short-Term Memory) model is constructed for embedding the textual descriptions using the pre-trained GloVe embeddings. The embedding layer is initialized with the embedding matrix, which is set to non-trainable.
- **Feature Extraction**:
  The LSTM model is used to obtain feature vectors for the textual descriptions. The sequences of padded tokens are fed into the model, and the resulting embedded descriptions are extracted as feature vectors.
- **Feature Assignment**:
  The obtained feature vectors are assigned a new column in the DataFrame.

(c) **Vector Combination**

- **Data Transformation**:
  The Image Feature Vector and Text Feature Vector columns in the DataFrame are transformed from string representations to lists of decimal elements using the 'string_to_list' function. This process involves splitting the string, converting each element to a float, and excluding 'NaN' values.

**Algorithm 13** string_to_list($s$)

**Function** string_to_list($s$)
**Input**: $s$ - The input string
**Output**: *elements* - List of elements extracted from $s$

**if** pd.isna($s$) **then**
  Return an empty list for NaN values
  **return** []
**else**
  $sub\_chains \leftarrow s[1:-1].\text{split}()$
  $elements \leftarrow []$
  **for each** *element* **in** *sub_chains* **do**
    **if** $element \neq' NaN'$ **then**
      **append** $float(element)$ **to** *elements*
    **end if**
  **end for**
  **return** *elements*
**end if**

- **Weight Assignment**:
  Weight values are assigned to the image and text vectors, with a weight of 0.6 for image vectors and 0.4 for text vectors, we gave more weight to the image vectors, as our main goal is to use these visual datas to enhance our recemmender systems.
- **Vector Combination**:
  For each row in the DataFrame, the image and text vectors are combined using the assigned weights to create a combined vector representation. The combined vector is computed element-wise as the weighted sum of the corresponding elements in the image and text vectors.
- **Combined Vector Storage**:
  The combined vectors are stored in a list, and subsequently added as a new column, to the DataFrame.

- **Vector Concatenation**:
  Additionally, a basic function, 'concatenate_vectors', is defined to concatenate the image and text vectors for each row in the DataFrame. This function is applied to create a new column, 'Concatenated Vectors ', containing the concatenated vectors.

(d) **Data Organization**

Rigorous data organization sets the stage for a systematic exploration of author-product interactions. The initial step involves computing the number of evaluations per author within the dataset. This computation provides valuable insights into the distribution of author engagement, serving as a foundational metric for subsequent analysis.

Authors exhibiting a prolific engagement, characterized by evaluations surpassing a predefined threshold (number of distinct rated items) ($T_e$) which in our case is $T > 22$, are selectively retained for subsequent analysis. These discerning criteria yield a subset of authors ($A_{>T_e}$) contributing significantly to the dataset's richness and relevance.

Furthermore, the identification of unique authors within the refined dataset provides crucial context regarding the breadth of authorship present. This metric, determined through the count of distinct author identities, underpins subsequent analyses, facilitating a comprehensive understanding of the dataset's composition.

As the dataset is structured to encapsulate author-specific vectors detailing item ratings, a pivotal step involves assembling pairs of users/ratings. This process entails grouping evaluations by author identity. These vectors serve as fundamental building blocks for subsequent machine learning tasks, facilitating the development and assessment of predictive models aimed at unraveling intricate author-product dynamics. the dataset is structured into author-specific vectors ($V_{A_i}$), encapsulating item ratings ($R_{A_i}$) associated with each author ($A_i$). This process involves meticulous grouping of evaluations by author identity, culminating in the formulation of comprehensive vectors elucidating individual author preferences:

$$V_{A_i} = [(P_j, R_{A_i,j}) \quad \text{for} \quad P_j \in \text{Products}, R_{A_i,j} \in \text{Ratings}] \tag{3,4}$$

Simultaneously, the dataset is partitioned into training and testing subsets, ensuring the integrity of subsequent analyses. This partitioning strategy involves an 80-20 split, allocating 80% of the data for training and 20% for testing. This allocation enables robust model training on representative data while facilitating rigorous evaluation against unseen instances.

Out[17]:

| | AuthorId | List of items/ratings | List of items/ratings Training 80% | List of items/ratings Test 20% |
|---|---|---|---|---|
| 0 | 1535 | [(50022, 5), (87689, 5), (27208, 4), (28603, 5... | [(50022, 5), (87689, 5), (27208, 4), (28603, 5... | [(52177, 5), (33671, 4), (28659, 3), (112683, ... |
| 1 | 3288 | [(16902, 4), (25885, 5), (90442, 5), (78814, 4... | [(16902, 4), (25885, 5), (90442, 5), (78814, 4... | [(173175, 4), (16531, 4), (117031, 3), (155290... |
| 2 | 4439 | [(56103, 5), (24051, 5), (77493, 5), (13285, 5... | [(56103, 5), (24051, 5), (77493, 5), (13285, 5... | [(49786, 5), (158326, 5), (348802, 5), (54351,... |
| 3 | 4470 | [(10422, 5), (834, 5), (21761, 5), (14784, 5),... | [(10422, 5), (834, 5), (21761, 5), (14784, 5),... | [(68955, 5), (184023, 5), (109734, 5), (61363,... |
| 4 | 5060 | [(12365, 5), (12591, 5), (20616, 5), (29570, 5... | [(12365, 5), (12591, 5), (20616, 5), (29570, 5... | [(37554, 5), (8845, 5), (26516, 5), (83770, 4)... |
| ... | ... | ... | ... | ... |
| 505 | 2001112113 | [(47606, 5), (139518, 5), (79116, 5), (211485,... | [(47606, 5), (139518, 5), (79116, 5), (211485,... | [(51935, 5), (238239, 5), (43585, 5), (123800,... |
| 506 | 2001330613 | [(49964, 5), (32614, 5), (20616, 5), (32204, 5... | [(49964, 5), (32614, 5), (20616, 5), (32204, 5... | [(16887, 5), (135350, 5), (97674, 5), (100005,... |
| 507 | 2001379957 | [(69173, 5), (13269, 5), (137575, 5), (33919, ... | [(69173, 5), (13269, 5), (137575, 5), (33919, ... | [(106499, 5), (335360, 5), (428528, 5), (35411... |
| 508 | 2001625595 | [(121897, 5), (76616, 5), (69173, 5), (82102, ... | [(121897, 5), (76616, 5), (69173, 5), (82102, ... | [(16682, 5), (181870, 5), (536311, 5), (23821,... |
| 509 | 2002754832 | [(33665, 5), (45809, 5), (99423, 5), (42641, 5... | [(33665, 5), (45809, 5), (99423, 5), (42641, 5... | [(62662, 5), (3474, 5), (97889, 5), (246186, 5... |

510 rows × 4 columns

Figure 2.8: Data Splitting

(e) **CBF Process**

For each item (recipe) in the test set (20% of the data), we aim to predict its rating based on the ratings of similar items in the training set (80% of the data). This is achieved through the following steps:

1. **Similarity Calculation**:
   - For a test item represented by a vector $\mathbf{v}_{\text{test}}$, we calculate its similarity to each item in the training set using cosine similarity:

$$\text{sim}(\mathbf{v}_{\text{test}}, \mathbf{v}_{\text{train}_i})$$

   - Here, $\text{sim}(\mathbf{v}_{\text{test}}, \mathbf{v}_{\text{train}_i})$ denotes the cosine similarity between the test item vector $\mathbf{v}_{\text{test}}$ and each training item vector $\mathbf{v}_{\text{train}_i}$.
   - The vector $\mathbf{v}$ of items is the vector resulting from the combination of image feature vectors and description feature vectors from the recipe dataset. We use the RecipeId (item) to access this combined vector.
   - In this specific similarity computation, we use concatenated vectors, and the weighted average is used to compare the results after, and we'll select the one that provides the best results for further processes.
2. **Selecting Most Similar Items**:
   - Once the similarities are calculated, the training items are sorted in descending order of similarity to the test item.
   - We select the top $K$ most similar items. Let the indices of these $K$ items be denoted by $\{i_1, i_2, \ldots, i_K\}$.
3. **Weighted Rating Calculation**:
   - For each selected training item $i_j$, we have its rating $R_{\text{train}_{i_j}}$ and its similarity $\text{sim}(\mathbf{v}_{\text{test}}, \mathbf{v}_{\text{train}_{i_j}})$.
   - The new predicted rating $R_{\text{test}}$ for the test item is calculated using a weighted average of the ratings of the top $K$ similar training items:

$$R_{\text{test}} = \frac{\sum_{j=1}^{K} \text{sim}(\mathbf{v}_{\text{test}}, \mathbf{v}_{\text{train}_{i_j}}) \cdot R_{\text{train}_{i_j}}}{\sum_{j=1}^{K} \text{sim}(\mathbf{v}_{\text{test}}, \mathbf{v}_{\text{train}_{i_j}})} \tag{3,5}$$

   - This formula ensures that the influence of each training item's rating is proportional to its similarity to the test item.
4. **Handling Edge Cases**:
   - If no similar items are found (i.e., $K = 0$), the new rating $R_{\text{test}}$ is set to NaN (not a number) to indicate that a prediction could not be made.

By following these steps, we predict the ratings for items in the test set based on the known ratings of similar items in the training set, thereby utilizing the patterns and preferences exhibited by the users in the training data.

2. **Collaborative Filtering**

- **Data Organization**

  In the collaborative filtering (CF) approach, we maintain the same data organization structure as used in the content-based filtering (CBF) method. Specifically, the dataset is partitioned into training and testing subsets, following an 80-20 split, where 80% of the data is used for training and 20% for testing. This consistent partitioning strategy enables robust model training and rigorous evaluation, ensuring a coherent comparison between CF and CBF techniques.

- **Similarity Function**
  - This function computes the similarity between a specified user and other users in the dataset based on the number of similar recipes they have rated, utilizing the K-Nearest Neighbors (KNN) algorithm.
  - For a specified user with AuthorId $u$, let $L_u$ be the list of items/ratings of user $u$.
  - For each other user $v$ (excluding user $u$), the similarity score $S_{uv}$ is calculated as the maximum number of similar recipes between $L_u$ and $L_v$, using the KNN approach.

- Let $k$ be the number of similar users to retrieve (K in KNN).
- The top $k$ similar users are selected based on the similarity scores.
- The similarity percentage $P_{uv}$ for each similar user $v$ is calculated as the ratio of $S_{uv}$ to the total number of similar recipes among all selected users.
- If there are no similar users (or an exception occurs), a default similarity percentage is assigned.

- **CF Process**

  1. **Compute Similarity between Vectors**:
     - Given two feature vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, the similarity between them can be calculated using a similarity measure such as cosine similarity:

$$\text{similarity}(\mathbf{v}_1, \mathbf{v}_2)$$

  2. **Calculate New Rating**:
     - For each test item $\mathbf{t}$ in the 20% test column:
       * Retrieve the k-nearest neighbors (similar users) of the targeted user.
       * For each neighbor $\mathbf{n}$, pick the rating of the same $\mathbf{t}$ that is in the training data of the neighbors, if the neighbor haven't rated it we'll just ignore the neighbor.
       * Compute a weighted average of the ratings of the same item by the neighbors, where the weights are the similarities between users:

$$\text{new\_rating}(\mathbf{t}) = \frac{\sum_{i=1}^{k} \text{rating}(\mathbf{t}, \mathbf{n}_i)}{k} \tag{3,6}$$

       * Assign the new rating to $\mathbf{t}$ of the traget user.

     This approach leverages collaborative filtering to predict new ratings based on the preferences of similar users.

3. **Clustering**

- **Dimensinality Reduction**

  **Data Preparation**

  - **Dictionary Formation:** Initially, a structured dictionary is created to store rating data. Each dictionary entry corresponds to a specific item (e.g., a recipe) and maps item identifiers to sub-dictionaries. These sub-dictionaries map user identifiers to their respective ratings for the item.
  - **Data Grouping:** The dataset is grouped by user identifiers to facilitate systematic population of the dictionary. Each user's ratings are iterated over and assigned to the appropriate entries in the dictionary.
  - **DataFrame Construction:** The dictionary is then converted into a DataFrame, providing a tabular representation of the data where rows represent items, columns represent users, and cell values represent ratings. Missing ratings are filled with zeroes to maintain a consistent matrix structure.

  **Sparse Principal Component Analysis (SparsePCA):**

  - **Scaling:** Standard scaling is applied to normalize the data, ensuring that each feature contributes equally to the analysis.
  - **SparsePCA Application:** SparsePCA is selected for dimensionality reduction. Unlike standard PCA, SparsePCA introduces sparsity in the principal components, which can be beneficial for interpretability and efficiency. The method retains the most significant components that capture the largest variance in the data while promoting sparsity.
  - **Component Selection:** The number of components to retain is chosen to balance the trade-off between dimensionality reduction and information preservation. An alpha parameter controls the sparsity of the components, with higher values leading to sparser components.

**Outcomes and Storage**

- **Reduced-Dimensional Data:** The result of applying SparsePCA is a new matrix with significantly fewer dimensions. This reduced-dimensional data retains the core characteristics and patterns of the original high-dimensional dataset, making it suitable for further machine learning tasks and analyses.
- **Storage and Usage:** The reduced-dimensional feature vectors are stored systematically for easy retrieval and use in subsequent predictive modeling and clustering tasks. This efficient representation facilitates faster computation and helps mitigate issues related to high dimensionality, such as overfitting and increased computational complexity.

**Process Execution**

- **Standard Scaling:** The data is first normalized using standard scaling, which standardizes features by removing the mean and scaling to unit variance.
- **SparsePCA Transformation:** SparsePCA is then applied to the scaled data. The algorithm computes principal components with added sparsity, reducing the data to a specified number of dimensions while maintaining key information and enhancing interpretability.
  Then, the resulted data will be used to create effective users clusters.

- **Deep-CC**
  - **Data Pair Generation:**
    Data pairs were generated using three methods as previously described in Chapter 2. These methods include identical pairing, random pair generation, and sequential pair generation. Each method introduces a different level of variability and complexity to the training process.
  - **Model Architecture:**
    The deep contrastive clustering (DeepCC) architecture was utilized, as detailed in Chapter 2. This architecture includes a backbone for feature extraction and two heads for contrastive learning at the instance and cluster levels.
  - **Training Procedure:**
    The model was compiled with the Adam optimizer and trained using a combination of instance-level and cluster-level contrastive loss functions. Instance-level contrastive loss aims to maximize the cosine similarity between pairs of data points, while cluster-level contrastive loss encourages similar cluster predictions for the same instances.
  - **Implementation Steps:**
    (a) **Generate Data Pairs:** Generate pairs of data points using the methods described in Chapter 2.
    (b) **Model Compilation:** Compile the model with the Adam optimizer and appropriate loss functions.
    (c) **Training:** Train the model on the generated pairs for a set number of epochs.
    (d) **Prediction:** Use the trained model to predict cluster assignments for the data points.
  - **Evaluation:**
    The performance of the model was evaluated by analyzing the predicted clusters. The argmax of the cluster head's output was taken to determine the cluster assignment for each data point, providing insight into the clustering quality and distribution.
    Afterwards, we'll execute the CF Process on each one of the clusters generated.

- **Cluster-Based Collaborative Filtering with Similarity Weighting ($CB\text{-}CF_{SW}$)**
  - **Cluster Prediction and Probability Calculation:**
    Cluster predictions were obtained from the trained model, as detailed in Chapter 2. The softmax function was applied to these predictions to calculate the probabilities of each data point belonging to each cluster. This resulted in a matrix of cluster probabilities.
  - **DataFrame Creation:**
    A DataFrame was created to store the cluster probabilities for each data point. This DataFrame included a column for the primary key (AuthorId) to link the cluster probabilities back to the

original data points. The mean divergence of cluster probabilities was calculated to provide an overall measure of cluster distribution.

- **Rating Adjustment Process:**
  (a) **Similarity Calculation:** For each data point in the test set, similar users were identified based on their cluster probabilities. The similarity between users was calculated using a function defined earlier.
  (b) **Cluster Analysis:** For each similar user, the cluster with the highest probability was identified. The corresponding probability value was used to weigh the influence of that user's ratings.
  (c) **Rating Calculation:** New ratings for test items were calculated by combining the ratings of similar users, weighted by their cluster probabilities and the similarity scores. A threshold was applied to ensure balanced clusters, enhancing the reliability of the ratings.
  (d) **Threshold Adjustment:** A threshold was set to exclude low-probability clusters from influencing the new ratings significantly. This adjustment was made to balance the clusters effectively.
  (e) **Rating Aggregation:** The aggregated ratings were assigned to the test items, replacing the original ratings. The list of items and their new ratings were then sorted and stored back in the DataFrame.
  (f) **Evaluation:** The updated ratings were compared to the original ratings to evaluate the effectiveness of the new rating calculation process.

Afterwards, we'll execute the CF Process on each one of the clusters generated.

- **KMeans with Contrastive Clustering Embeddings**
  - **Pair Generation and Labeling:**
    * Positive and negative pairs were generated using two different data generation techniques, as explained in Chapter 2.
    * The pairs were created from the reduced dataset obtained after preprocessing and dimensionality reduction.
    * Labels were assigned to these pairs: 1 for positive pairs and 0 for negative pairs.
    * The pairs and labels were combined and shuffled to prepare the training dataset.
  - **Model Training:**
    * The same contrastive model architecture used in DeepCC was employed.
    * The model was trained on the combined pairs and labels for 100 epochs with a batch size of 128.
  - **Embeddings and Clustering:**
    * After training, embeddings for the original dataset were generated using the same model architecture as in DeepCC.
    * These embeddings were then clustered using the KMeans algorithm with 5 clusters (justified by the elbow method).

Afterwards, we'll execute the CF Process on each one of the clusters generated.

- **KMeans Clustering**
  - **Determining Optimal Number of Clusters:**
    * The sum of squared distances was calculated for different numbers of clusters ranging from 2 to 10 using the KMeans algorithm.
    * This process helps in identifying the optimal number of clusters by observing the "elbow" point in the plot.
    * The Elbow Method plot was generated to visualize the relationship between the number of clusters and the inertia, representing the sum of squared distances.

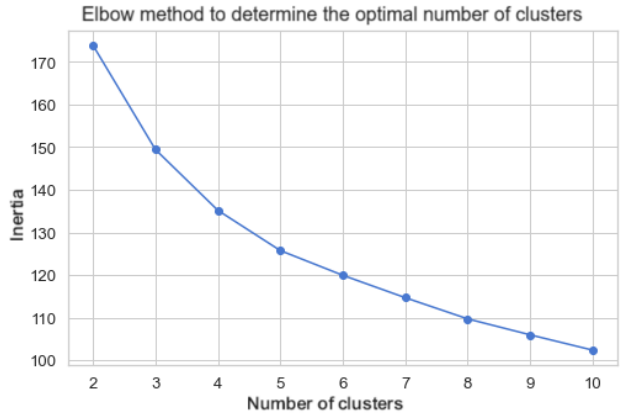



Figure 2.9: Elbow plot to determine the optimal number of clusters

* Additionally, the silhouette score was computed for each number of clusters to evaluate the quality of the clustering.

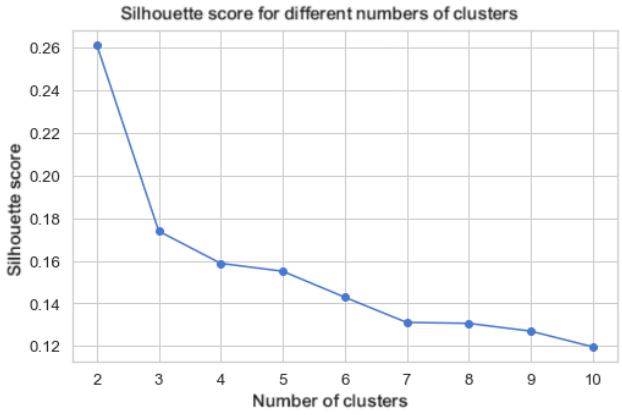



Figure 2.10: Silhouette plot to determine adequate number of clusters to use

* Another plot, Silhouette Score vs. Number of Clusters, was created to determine the optimal number of clusters based on silhouette scores.

– **Cluster Assignment:**
  * After determining the optimal number of clusters, a KMeans model with the chosen number of clusters (5 in this case) was created and fitted to the reduced dataset.
  * Cluster labels were assigned to each data point based on the fitted model.
  * The cluster labels were retrieved for further analysis or exploration.

Afterwards, we'll execute the CF Process on each one of the clusters generated.

- **Agglomerative Clustering**
  – The Agglomerative Clustering algorithm was employed to perform hierarchical clustering on the reduced dataset.
  – The number of clusters was predefined (in this case, set to 5) based on the experimental requirements.
  – The model was created and fitted to the reduced dataset, and cluster labels were assigned to each data point.

- **Gaussian Mixture Models Clustering (GMM)**
  - The Gaussian Mixture Model (GMM) was utilized to perform clustering on the reduced dataset.
  - The number of components (clusters) was predefined (in this case, set to 5) based on the experimental requirements.
  - The GMM model was created and fitted to the reduced dataset using the `GaussianMixture` class from `sklearn.mixture`.
  - Cluster labels were predicted for each data point using the `predict` method.

  Afterwards, we'll execute the CF Process on each one of the clusters generated.
- **Self-Organizing Maps Clustering (SOM)**
  - The Self-Organizing Map (SOM) was employed to visualize and cluster the reduced dataset.
  - The dimensions of the SOM grid were specified, with a width of 5 and a height of 5.
  - Initialization and training of the SOM were performed using the `MiniSom` class from the `minisom` library.
  - The SOM was trained with 10,000 iterations using random training data.
  - Additional steps may be required for interpreting the neuron grids for visualization results.
  - The Unified Distance Matrix (U-Matrix) was computed to visualize the distance between neurons.

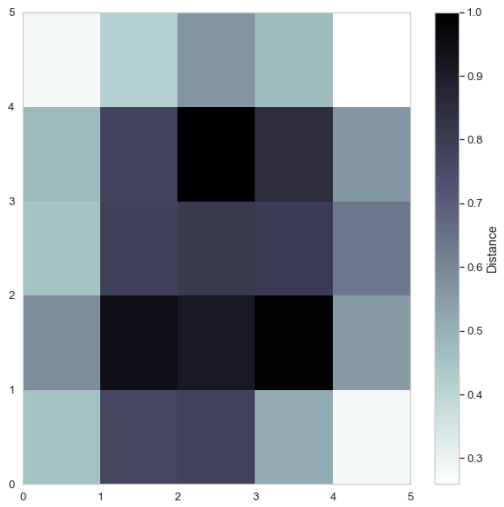



Figure 2.11: U-Matrix Visualization of the Self-Organizing Map (SOM)

Figure 2.11 illustrates the U-Matrix of the Self-Organizing Map (SOM) trained on the reduced dataset. The U-Matrix visualizes the distance between the neurons in the SOM, with each cell representing a node (neuron) in the 5x5 SOM grid. The color of each cell corresponds to the average distance between the respective neuron and its neighboring neurons, providing insights into the structure of the data.

  - Neuron weights were obtained from the trained SOM and reshaped into a suitable format for clustering.
  - KMeans clustering was applied to cluster the neuron weights into a predefined number of clusters (in this case, 5 clusters).
  - The function `convert_to_linear_index` was defined to calculate the linear index from SOM grid coordinates.
  - An example of its usage was provided to obtain data labels based on the clustering of neuron weights.

Afterwards, we'll execute the CF Process on each one of the clusters generated.

4. **Hybridization**

- **Multi-level Hybiridization**
  - Merging CBF and CF Results:
    - The CBF (Content-Based Filtering) and CF (Collaborative Filtering) results were merged to create a unified dataset.
    - A function was applied to merge the lists of recipes and their corresponding ratings from both CBF and CF results. In cases where a recipe appeared in both sources, the rating from the CBF result was prioritized.
  - Recalculating Test Set Ratings:
    - The test set ratings were recalculated based on the merged ratings from CBF and CF.
    - For each recipe in the test set, the recalculated rating was determined by selecting the maximum value between the ratings from CBF and CF.
  - Creation of New Dataset:
    - A new dataset was formed to store the original training set, the merged test set, and the recalculated test set ratings.
    - This dataset facilitated the comparison and evaluation of the merged and recalculated ratings.
- **Multi-level Hybiridization On Clusters**
  - Utilizing DeepCC Clusters:
    - The clusters generated by the DeepCC model were used as a basis for applying the multi-fusion technique.
    - Each cluster represented a distinct group of similar items or users identified by the DeepCC model.
  - Application of Multi-Fusion:
    - The multi-fusion technique was applied to each cluster individually to combine the results of various recommendation algorithms or models.
    - For each cluster, the ratings or recommendations from different models or algorithms were fused together using a predefined fusion strategy.
- **Unified Collaborative-Content Filtering Hybridization**

  1. **Compute Similarity between Vectors**:
     - Given two feature vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, the similarity between them can be calculated using a similarity measure such as cosine similarity:

$$\text{similarity}(\mathbf{v}_1, \mathbf{v}_2)$$

  2. **Calculate New Rating**:
     - For each test item $\mathbf{t}$ in the 20% test column:
       - Retrieve the k-nearest neighbors (similar users) of the targeted user.
       - For each neighbor $\mathbf{n}$, calculate the similarity between $\mathbf{t}$ and the $k_1$-most similar items in the training data of the neighbors ($t_{ni}$).
       - Compute a weighted average of the ratings of similar items, where the weights are the similarities between users:

$$\text{new\_rating}(\mathbf{t}) = \frac{\sum_{i=1}^{k} \text{similarity}(\mathbf{t}, \mathbf{t}_{ni}) \times \text{rating}(\mathbf{t}_{ni}, \mathbf{n}_i)}{\sum_{i=1}^{k} \text{similarity}(\mathbf{t}, \mathbf{t}_{ni})} \tag{3,7}$$

       - Assign the new rating to $\mathbf{t}$ of the traget user.

- **Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting Integration Methodology:**
  - $CB$-$CF_{SW}$ **Ratings Calculation:**
    * $CB$-$CF_{SW}$ clusters users based on preferences and computes similarity scores among users within the same cluster.
    * Similarity scores weigh the influence of neighboring users' ratings on the target user's predicted ratings.
  - **CBF Ratings Calculation:**
    * CBF ratings are determined based on item feature similarity, offering personalized ratings reflecting item content characteristics.
  - **Fusion Process:**
    * $CB$-$CF_{SW}$ and CBF ratings are combined using a weighted average formula controlled by parameters $\alpha = 0{,}5$ and $\beta = 0{,}5$.
    * Parameter $\alpha$ influences $CB$-$CF_{SW}$ ratings, while $\beta$ influences CBF ratings.

  **Experiment Execution:**
  - **Data Preparation:**
    * Dataframes representing user data, CBF ratings, and cluster information are prepared.
  - **Fusion Algorithm:**
    * The fusion algorithm iterates through users and computes weighted $CB$-$CF_{SW}$ ratings and CBF ratings for each test item.
    * Final ratings are calculated as a combination of weighted $CB$-$CF_{SW}$ ratings and CBF ratings using parameters $\alpha$ and $\beta$.
  - **Evaluation and Analysis:**
    * Recommendations produced by *Hybrid* $CB$-$CF_{SW}$ are evaluated using metrics such as accuracy, personalization, and relevance.
    * Comparative analysis with baseline models assesses the performance of *Hybrid* $CB$-$CF_{SW}$.

### 2.4.1 Validators

The process begins with sorting the ratings of test items based on their new ratings, calculated using the CBF, CF, hybrid recommendation approaches. After sorting, the test items are rearranged according to the order of their new ratings.

Next, the mean divergence between the original ratings and the new ratings is calculated. This involves comparing each test item's original rating with its corresponding new rating, computing the absolute difference, and then averaging these differences across all test items.

After computing the mean divergence, it is transformed into a mean difference on a scale of 5. This transformation ensures that the divergence values are consistent with the rating scale, making them easier to interpret.

Following this, the comparison between the original and new ratings is performed. For each test item, a binary value of 0 or 1 is assigned based on whether the absolute difference between the original and new ratings exceeds a predefined threshold (in our case, it's the mean divergence). If the difference exceeds the threshold, a value of 0 is assigned, indicating a mismatch between the ratings. Otherwise, a value of 1 is assigned, indicating a match.

#### 2.4.1.1 Recommendation Set Validators

After assigning binary values to the comparisons, the evaluation metrics are computed. These metrics include precision, accuracy, and recall, which provide insights into the performance of the recommendation system. The precision measures the proportion of recommended items that are relevant to the user, while accuracy measures the proportion of correctly recommended items. Recall measures the proportion of relevant items that are successfully recommended. Using Precision and recall, we'll also calculate the F1 score as it is the harmonic mean of precision and recall, and it balances both metrics.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \tag{3,8}$$

The formula for the F1 score is:

$$F1 = 2 \times \frac{Presion \times Recall}{Precision + Recall} \tag{3,9}$$

$TP$ is the number of true positives, where the original and new ratings match (i.e., the binary comparison value is 1)

$FN$ is the number of false negatives, where the original rating is relevant (i.e., the binary comparison value is 1), but the new rating does not match (i.e., the binary comparison value is 0).

$FP$ is the number of false positives, where the original rating does not match (i.e., the binary comparison value is 0), but the new rating is relevant (i.e., the binary comparison value is 1).

$TN$: The number of cases where the original rating does not match (binary comparison value is 0), and the new rating also does not match (binary comparison value is 0)

#### 2.4.1.2 Prediction Validators

This process evaluates the performance of a recommendation system by comparing its predictions to the actual ratings provided by users. It focuses on two key metrics: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

MAE measures the average difference between predicted and actual ratings without considering their direction. It provides insight into the overall accuracy of the system's predictions.

RMSE, on the other hand, penalizes larger prediction errors more heavily due to its squared nature. It provides a measure of how well the system's predictions match the actual ratings while taking into account the magnitude of the errors. To conduct the evaluation, the process iterates over each user's predictions and actual ratings. For each user:

1. It extracts the predicted ratings and actual ratings for the top $N$ items.
2. It calculates the absolute differences between the predicted and actual ratings.
3. It computes the MAE by taking the mean of the absolute differences.
4. It computes the RMSE by taking the square root of the mean of the squared differences.
5. It repeats this process for all users and calculates the average MAE and RMSE across all users.

In this study, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics were computed to assess the accuracy of our recommender system predictions. To align these error measures with the system's 0-5 rating scale, all MAE and RMSE values were multiplied by 5. This scaling facilitates direct comparison and interpretation of the errors in relation to the original rating scale, ensuring clarity in evaluating the system's performance.

Scaled MAE:

$$\text{Scaled MAE} = 5 \times \text{Original MAE} \tag{3,10}$$

Scaled RMSE:

$$\text{Scaled RMSE} = 5 \times \text{Original RMSE} \tag{3,11}$$

These metrics provide valuable insights into the accuracy and performance of the recommendation system, helping researchers and developers understand its strengths and weaknesses. Lower values of MAE and RMSE indicate better alignment between predicted and actual ratings, signifying higher accuracy in the recommendations.

### 2.4.2 Results

- **Content-Based Filtering (Using Concatenated Vectors) - CBF1**

  In this section, we present the performance metrics of content-based filtering using five different values of k in the k-Nearest Neighbors (k-NN) algorithm to retrieve the most similar images (feature vectors are combiened using the concatenation technique). The primary focus is on Accuracy, with Precision and F1 Score also considered.

Table 2.14: Performance Metrics for Different Values of $k$ (CBF1)

| $k$ | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|---|
| 30 | 0.5958 | 0.6913 | 0.8261 | 0.7521 |
| **25** | **0.5993** | **0.6947** | **0.8340** | **0.7579** |
| 15 | 0.5946 | 0.6890 | 0.8591 | 0.7653 |
| 10 | 0.5826 | 0.6719 | 0.8744 | 0.7590 |
| 5 | 0.5983 | 0.6925 | 0.8866 | 0.7751 |

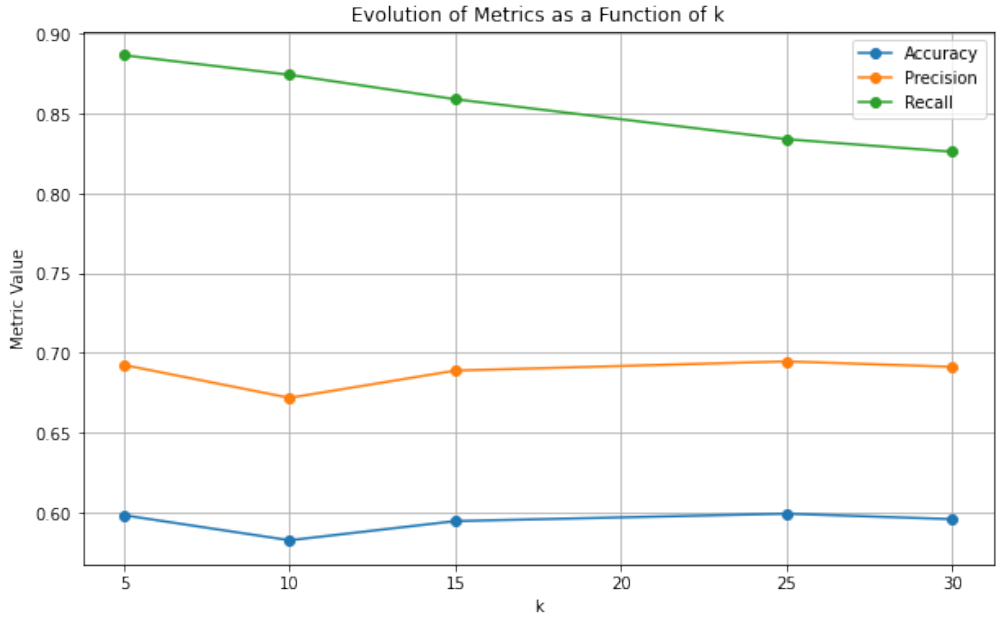



Figure 2.12: Evolution of CBF Metrics (Concatenation Method) as a Function of k

The results indicate that the highest Accuracy is achieved with k=25 (0.5993) and k=5 (0.5983). Although the Recall increases as k decreases, the Accuracy remains relatively stable, with a slight dip at k=10. This suggests that intermediate k values might be optimal for balancing accuracy and other metrics in content-based filtering. Despite variations in Accuracy, the highest Precision (0.6947) and F1 Score (0.7751) are observed at k=5

Table 2.15: Performance Metrics for $k = 25$ (Content-Based with Concatenated Vectors)

| **Metric** | **Value** |
|---|---|
| MAE (*5) | 0.7951 |
| RMSE (*5) | 0.7980 |

- **Content-Based Filtering (Using Weighted Vectors) - CBF2**

  Now, we present the performance metrics of content-based filtering using five different values of k in the k-Nearest Neighbors (k-NN) algorithm to retrieve the most similar images (feature vectors are combiened using the weighted average technique). The primary focus is on Accuracy, with Precision and F1 Score also considered.

Table 2.16: Performance Metrics for Different Values of $k$ (CBF2)

| $k$ | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|---|
| 5 | 0.5988 | 0.6930 | 0.8747 | 0.7731 |
| 10 | 0.5882 | 0.6802 | 0.8701 | 0.7637 |
| 15 | 0.6029 | 0.6988 | 0.8544 | 0.7689 |
| **25** | **0.6066** | **0.7036** | **0.8496** | **0.7707** |
| 30 | 0.6056 | 0.7026 | 0.8450 | 0.7690 |

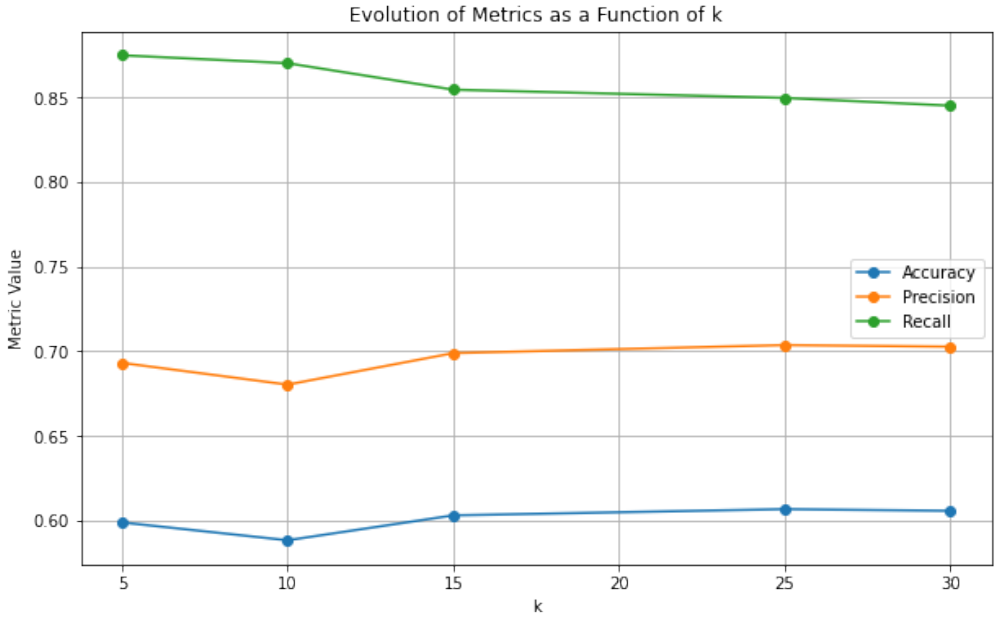



Figure 2.13: Evolution of CBF Metrics (Weighted Average Method) as a Function of k

The results indicate that the highest Accuracy is achieved with k=25 (0.6066) and k=30 (0.6056). Although the Recall increases as k decreases, the Accuracy remains relatively stable, with a slight dip at k=10. This suggests that intermediate k values might be optimal for balancing accuracy and other metrics in content-based filtering.

Table 2.17: Performance Metrics for $k = 25$ (Content-Based with Weighted Vectors)

| **Metric** | **Value** |
|---|---|
| MAE (*5) | 0.7939 |
| RMSE (*5) | 0.7968 |

- **Collaborative Filtering - CF**

  This section discusses the outcomes of collaborative filtering using five different values of k in the k-NN algorithm to find the most similar users. The primary focus is on Accuracy, with Precision and F1 Score also considered.

Table 2.18: Performance Metrics for Different Values of $k$ (CF)

| $k$ | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|---|
| 5 | 0.6549 | 0.7597 | 0.9095 | 0.8284 |
| 10 | 0.6424 | 0.7456 | 0.9118 | 0.8203 |
| 15 | 0.6498 | 0.7527 | 0.9036 | 0.8218 |
| 25 | 0.6623 | 0.7682 | 0.9031 | 0.8297 |
| **30** | **0.6630** | **0.7684** | **0.9025** | **0.8296** |

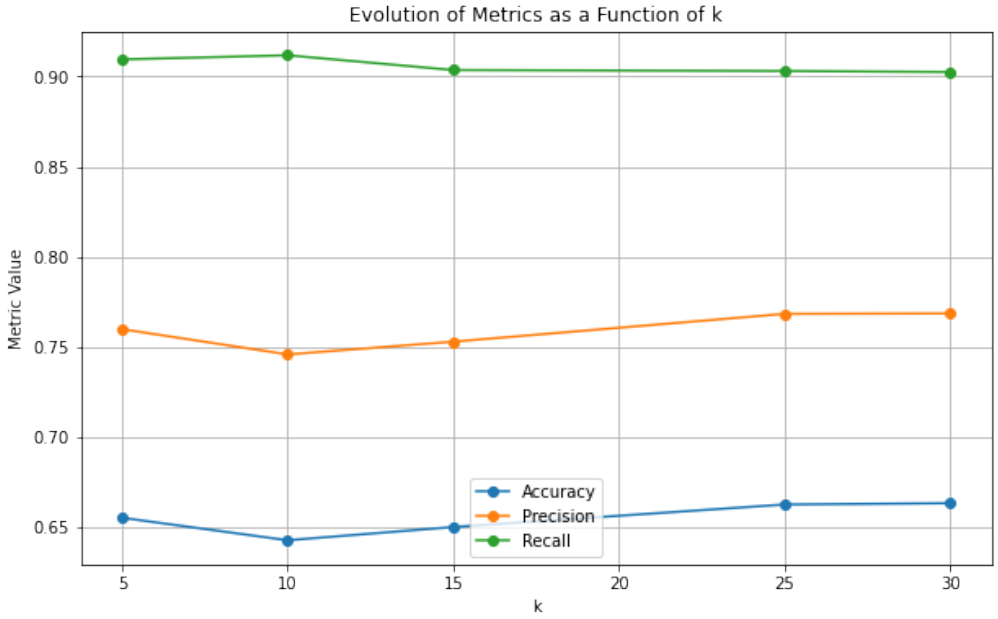



Figure 2.14: Evolution of CF Metrics as a Function of k

The collaborative filtering results show that the highest Accuracy is achieved with k=30 (0.6630) and k=25 (0.6623). These findings suggest that higher k values generally offer better overall accuracy, with Precision and F1 Score also showing improvements at these k values. Higher k values not only improve Accuracy but also enhance Precision (0.7684) and F1 Score (0.8297).

Table 2.19: Performance Metrics for $k = 30$ (User-Based)

| **Metric** | **Value** |
|---|---|
| MAE (*5) | 0.8019 |
| RMSE (*5) | 0.8046 |

- **CBF vs. CF**

  we compare the performance of content-based filtering and collaborative filtering based on Accuracy, Precision, and F1 Score. The table below summarizes the comparison.

Table 2.20: Performance Metrics for Different Approaches

| **Approach** | **Accuracy** | **Precision** | **Recall** | **RMSE (*5)** | **MAE (*5)** |
|---|---|---|---|---|---|
| Content Based - Concatenation (CBF1) | 0.598284 | 0.692474 | 0.886616 | 0.798009 | 0.795149 |
| Content Based - Weighted Vectors (CBF2) | 0.606618 | 0.703550 | 0.849565 | 0.796754 | 0.793866 |
| **User Based (CF)** | **0.663000** | **0.768447** | **0.902534** | **0.829644** | **0.801931** |

Now, we execute the same algorithms for different configurations of N (Top-N recommendations)

| Approach | Precision | Accuracy | Recall |
|---|---|---|---|
| Content-Based - Concatenated | 0.7088 | 0.7088 | 0.5267 |
| Content-Based - Weighted Average | 0.7088 | 0.7088 | 0.5265 |
| User-Based | 0.8529 | 0.8529 | 0.6349 |

Table 2.21: Performance Metrics for Top-4 Recommendations

| Approach | Precision | Accuracy | Recall |
|---|---|---|---|
| Content-Based - Concatenated | 0.7017 | 0.6804 | 0.7383 |
| Content-Based - Weighted Average | 0.7050 | 0.6833 | 0.7416 |
| User-Based | 0.7988 | 0.7755 | 0.8247 |

Table 2.22: Performance Metrics for Top-6 Recommendations

| Approach | Precision | Accuracy | Recall |
|---|---|---|---|
| Content-Based - Concatenated | 0.6947 | 0.5993 | 0.8340 |
| Content-Based - Weighted Average | 0.7036 | 0.6066 | 0.8496 |
| User-Based | 0.7684 | 0.6630 | 0.9025 |

Table 2.23: Performance Metrics for Top-8 Recommendations

| Approach | Precision | Accuracy | Recall |
|---|---|---|---|
| Content-Based - Concatenated | 0.6909 | 0.5220 | 0.8799 |
| Content-Based - Weighted Average | 0.6989 | 0.5275 | 0.8914 |
| User-Based | 0.7575 | 0.5708 | 0.9407 |

Table 2.24: Performance Metrics for Top-10 Recommendations

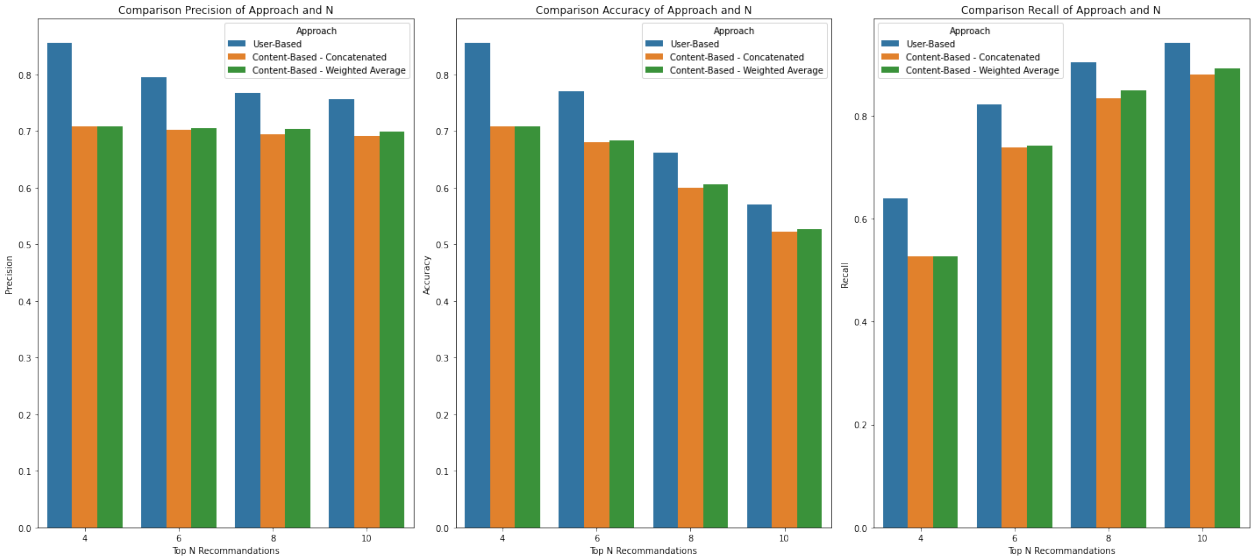



Figure 2.15: Comparison between CBF's and CF's Accuracy, Recall and Precision Scores with different N configurations

The comparison reveals that user-based collaborative filtering outperforms content-based filtering approaches in terms of Accuracy, with the highest Accuracy observed for the user-based approach (0.6623). Content-based filtering with weighted vectors shows a competitive accuracy of 0.6066, suggesting its potential effectiveness despite being slightly lower than collaborative filtering.

| Approach | Top-4 | | Top-6 | | Top-8 | | Top-10 | |
|---|---|---|---|---|---|---|---|---|
| | MAE (*5) | RMSE (*5) | MAE (*5) | RMSE (*5) | MAE (*5) | RMSE (*5) | MAE (*5) | RMSE (*5) |
| Content-Based - Concatenated | 0.796450 | 0.798914 | 0.795340 | 0.798098 | 0.795149 | 0.798009 | 0.795001 | 0.797913 |
| Content-Based - Weighted Average | 0.797585 | 0.800074 | 0.795351 | 0.798130 | 0.793866 | 0.796754 | 0.793785 | 0.796689 |
| User-Based | 0.798442 | 0.800246 | 0.799694 | 0.802090 | 0.801360 | 0.804031 | 0.801604 | 0.804368 |

Table 2.25: Performance Metrics: MAE and RMSE for different N configurations

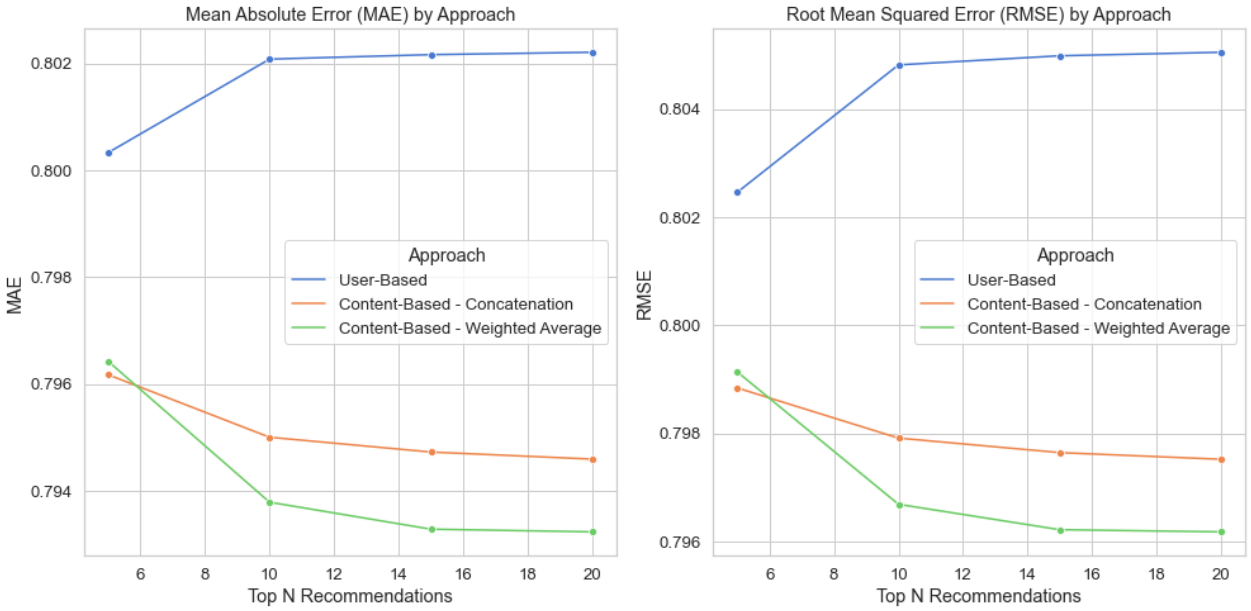



Figure 2.16: Comparison between CBF's and CF's RMSE, MAE Scores with different N configurations

Collaborative filtering achieves higher Precision (0.7682) and F1 Score (0.8297) compared to content-based methods. content-based approaches show lower RMSE (0.7968 for weighted vectors) and MAE (0.7939), collaborative filtering has slightly higher values for RMSE (0.8046) and MAE (0.8019), indicating a trade-off between accuracy and error metrics.

- **Clustering Results Comparison**

  This section evaluates the performance of collaborative filtering when applied to various clustering methods, focusing primarily on Accuracy.

Table 2.26: Performance Metrics for the CF Algorithm Using Different Clustering Techniques

| Algorithm | Accuracy | Precision | Recall | RMSE (*5) | MAE (*5) |
|---|---|---|---|---|---|
| SOM | 0.666148 | 0.742378 | 0.911424 | 0.811325 | 0.808323 |
| GMM | 0.648199 | 0.723407 | 0.912624 | 0.812568 | 0.809424 |
| AC | 0.653298 | 0.731059 | 0.915634 | 0.810894 | 0.807820 |
| KMeans with PCA | 0.653680 | 0.729157 | 0.912261 | 0.811913 | 0.808722 |
| Kmeans with CC Embeddings - Cosine Similarity-based Pair Generation | 0.660209 | 0.734570 | 0.902427 | 0.812696 | 0.809808 |
| Kmeans with CC Embeddings - Balanced Pair Generation with Cosine Similarity | 0.668772 | 0.741778 | 0.910806 | 0.812391 | 0.809590 |
| CC with PCA - Self-Paired Generation | 0.680069 | 0.758558 | 0.911450 | 0.811091 | 0.808191 |
| CC with PCA - Random Pair Generation | 0.687476 | 0.752287 | 0.911886 | 0.813277 | 0.807334 |
| **CC with PCA - Sequential Pair Generation** | **0.692811** | **0.764891** | **0.888478** | **0.815319** | **0.812591** |
| **Cluster-Based Collaborative Filtering with Similarity Weighting** | **0.746717** | **0.844472** | **0.902209** | **0.794464** | **0.790087** |

The results indicate that the "Cluster-Based Collaborative Filtering with Similarity Weighting" method achieves the highest Accuracy (0.7467), followed by "CC with PCA - Sequential Pair Generation" (0.6928). "Cluster-Based Collaborative Filtering with Similarity Weighting" also achieves the highest Precision (0.8445) and notably lower RMSE (0.7945). These clustering-based methods significantly outperform traditional collaborative filtering, highlighting the effectiveness of incorporating clustering techniques.

- **Hybridization Results**

  This section outlines the outcomes of different hybridization approaches combining collaborative and content-based filtering methods, with a primary focus on Accuracy.

Table 2.27: Performance Metrics for Different Hybridization Approaches

| Approach | Accuracy | Precision | Recall | RMSE (*5) | MAE (*5) |
|---|---|---|---|---|---|
| Multi-level Hybridization | 0.679412 | 0.790824 | 0.891270 | 0.813253 | 0.810635 |
| Multi-level Hybridization On Clusters | 0.717128 | 0.820643 | 0.900902 | 0.809954 | 0.807707 |
| Unified Collaborative-Content Filtering Hybridization | 0.680147 | 0.791050 | 0.884566 | 0.789026 | 0.786218 |
| **Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting** | **0.839950** | **0.973037** | **0.888073** | **0.764234** | **0.756818** |

The "Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting" approach demonstrates the highest overall performance, achieving remarkable Accuracy (0.8400). The "Multi-level Hybridization On Clusters" also shows a significant improvement in accuracy (0.7171) compared to other methods. The hybrid method with similarity weighting excels in Precision (0.9730) and demonstrates the lowest RMSE (0.7642). These results suggest that hybrid approaches, especially those leveraging clustering techniques, can substantially enhance recommendation accuracy and user satisfaction.

- **Summarization**

  We presented an analysis of the performance metrics for content-based filtering, collaborative filtering, and their hybrid approaches. We evaluated each method using various k values in the k-Nearest Neighbors (k-NN) algorithm and assessed their accuracy, precision, recall, F1 score, RMSE, and MAE. In

content-based filtering, the highest accuracy was observed at k=25 (0.5993), with the highest precision (0.6947) and F1 score (0.7751) at k=5. For collaborative filtering, the best accuracy was achieved at k=30 (0.6630), with precision and F1 scores peaking at 0.7684 and 0.8297, respectively. Comparing these approaches, collaborative filtering outperformed content-based methods in accuracy and precision but had slightly higher RMSE (0.8046) and MAE (0.8019) compared to content-based filtering. Clustering methods further enhanced performance, with "Cluster-Based Collaborative Filtering with Similarity Weighting" achieving the highest accuracy (0.7467) and precision (0.8445). The hybrid methods showed substantial improvements, particularly the "Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting," which achieved the highest accuracy (0.8400) and precision (0.9730) while demonstrating the lowest RMSE (0.7642). These findings are visually summarized in the accompanying graph, providing a clear comparison of all methods across key performance metrics.

The following figure presents a comparative analysis of different recommendation algorithms across multiple performance metrics: Accuracy, Precision, Recall, RMSE (Root Mean Square Error), and MAE (Mean Absolute Error). The algorithms are categorized into three main groups: **Classical Algorithms** (*Content-Based Filtering (Using Concatenated Vectors), Content-Based Filtering (Using Weighted Vectors), Collaborative Filtering*) , **Clustering Algorithms** (*SOM, GMM, AC, KMeans with PCA, Kmeans with CC Embeddings - Cosine Similarity-based Pair Generation, Kmeans with CC Embeddings - Balanced Pair Generation with Cosine Similarity, CC with PCA - Self-Paired Generation, CC with PCA - Sequential Pair Generation, Cluster-Based Collaborative Filtering with Similarity Weighting*) and **Hybridization Approaches** (*Multi-level Hybridization, Multi-level Hybridization On Clusters, Unified Collaborative-Content Filtering Hybridization, Hybrid Cluster-Based Collaborative Filtering with Similarity Weighting*). Each bar in the plots represents a specific algorithm within these categories, in order, respectively.


Figure 2.17: Complete Summary of Each Approach's Performance Metrics

# Chapter 3

# Discussion and Conclusion

We reflect on the transformative potential of our study, which has introduced a paradigm by integrating visual and textual data into recommendation systems. This promising approach, augmented by contrastive clustering and cluster ratio distribution for rating calculation, has provided remarkable insights and outcomes.

The incorporation of visual data alongside textual information has significantly enriched the recommendation process, allowing for a more holistic understanding of user preferences and content characteristics. Contrastive clustering emerges as a foundation of this advancement, enabling the extraction of intricate patterns and semantic relationships from the combined dataset. Through the utilization of cluster ratio distribution in rating calculation, our model has demonstrated enhanced accuracy.

Throughout our investigation, we've unearthed several notable implications. Firstly, the utilization of contrastive clustering has facilitated a deeper understanding of the underlying data structure, enabling more nuanced recommendation strategies. Additionally, the incorporation of cluster-based rating distributions has enhanced the granularity and precision of our recommendation outcomes, leading to better metrics results.

The hybridization of these techniques marks a significant step forward in recommendation system development. Its potential for industry applications is immense, promising heightened user engagement, improved decision-making processes, and ultimately, enhanced business outcomes. This innovative approach not only addresses existing challenges in recommendation systems but also opens up new avenues for exploration and innovation.

However, it's imperative to acknowledge the limitations and challenges encountered in our research. Despite the promising results, our study is not without its constraints. We recognize the need for further exploration into optimizing the hybridization process and addressing potential biases or limitations in the clustering algorithms utilized.

As we conclude this thesis, it is clear that the integration of visual data, textual information, and advanced clustering techniques represents a promising direction for future research and industry practice. The potential for extending this approach to diverse domains and applications underscores its significance and relevance in the evolving landscape of data-driven decision-making. Indeed, the journey does not end here; rather, it invites further exploration, collaboration, and refinement to realize its full potential in shaping the future of recommendation systems.

# Bibliography


[1] Farzana Anowar, Samira Sadaoui, and Bassant Selim. Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne). *Computer Science Review*, 40:100378, 2021.

[2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, page 4171–4186, 2018.

[4] Xiaoxu Ding, Jiawei Han, and Xiang Zhang. Cross-modal fusion for recommendation systems: A comprehensive survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(6):1–37, 2020.

[5] Xuejie Gao, Bin Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 23(1):113–129, 2020.

[6] Jiawei Han, M. Kamber, and J. Pei. Data mining: Concepts and techniques. pages 585–631, 01 2006.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[8] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[9] Bahrudin Hrnjica, Denis Music, and Selver Softic. Model-based recommender systems. *Trends in Cloud-based IoT*, pages 125–146, 2020.

[10] Jason Iyosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[11] Alexandros Karatzoglou and Balázs Hidasi. Deep learning for recommender systems. pages 396–397, 08 2017.

[12] Zahra Koutanaei, Mohammad Komeili Akbari, and Mohsen Bahrami. Multi-level hybridization of recommender systems: A comprehensive review. *Journal of Ambient Intelligence and Humanized Computing*, 12(6):6297–6313, 2021.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[14] Ji Soo Lee, Young Geun Kim, and Jong Won Kim. Personalized recommendation system based on cosine similarity for fashion products. *Sustainability*, 13(1):240, 2021.

[15] Yunfan Li, Peng Hu, Zitao Liu, Dezhong Peng, Joey Tianyi Zhou, and Xi Peng. Contrastive clustering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 8547–8555, 2021.

[16] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[17] Iqbal Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2, 03 2021.

[18] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.

[19] V Subramaniyaswamy and R Logesh. Adaptive knn based recommender system through mining of user preferences. *Wireless Personal Communications*, 97:2229–2247, 2017.

[20] Xiaojun Wang and Avi Gupta. Text and image fusion using deep neural networks for sentiment analysis. *IEEE Transactions on Multimedia*, 20(1):169–179, 2018.

[21] Ziqiang Wang, Long Chen, and Xiaolin Hu. A survey on deep clustering: Towards robustness and scalability. *Neural Networks*, 135:65–82, 2021.

[22] Website : GeeksforGeeks. Vgg-16 cnn model, 2023. Accessed: 2024-05-31.