BACHELOR

# Developing a Versatile Hybrid Recommender System
Case Study on Programming Task Prediction

Dumitru Toader, Maria Emanuela

*Award date:*
2023

*Awarding institution:*
Tilburg University

Link to publication

Eindhoven University of Technology and Tilburg University

JBP000 Final Bachelor Project

# Developing a Versatile Hybrid Recommender System: Case Study on Programming Task Prediction

Maria Emanuela Dumitru Toader (1537156)

June 12, 2023

# Abstract

This study addresses the challenge that educators face, that of identifying suitable datasets for programming coursework. While datasets enhance students' learning by bridging theory and real-world scenarios, the search for a perfect match can be cumbersome due to the sheer diversity of available datasets. Recognizing the gap of research in this particular topic as well as the limitations of current recommender systems, which are often inflexible and domain-specific, this research introduces a hybrid recommender system that tackles the issue of matching programming tasks with relevant datasets. By evaluating several predictive methods, integrating the most effective ones, and conducting a rigorous evaluation, the project develops a versatile, adaptable system. Notably, certain machine learning techniques like decision trees and random forests demonstrated strong performance, but the collective strength of the hybrid system outperformed them. This system facilitates dataset selection for educators, allowing more focus on pedagogy, and it provides a promising foundation for future multi-purpose recommender systems.

# Contents

# 1 | Introduction

In today's data-driven educational landscape, the use of datasets in programming coursework has become more of a necessity than a mere luxury. Datasets serve as a vital bridge between theoretical knowledge and practical, real-world scenarios, thereby enhancing the overall learning experience of students. The underlying idea is to expose learners to real-world data sets early, promoting a better understanding of data analysis and interpretation.

However, many educators struggle with the challenge of finding suitable datasets for their assignments, which can be time-consuming and resource-intensive. With the wide array of datasets available, finding the perfect match for a particular assignment or programming task is like finding a needle in a haystack. Each dataset holds unique potential, but fully understanding this potential is not a straightforward process. It typically requires deep exploration and analysis, which is a time-intensive process.

The main challenge lies in fully understanding the potential of datasets without thorough exploration. Educators, especially those without extensive data science backgrounds, face difficulty in extracting insights from these datasets. Additionally, time constraints prevent educators from delving deep into each dataset, resulting in untapped potential. In this age of technology, using an automated method of identifying suitable programming tasks, such as a recommender system, makes sense.

Another challenge arises however from the inflexibility of existing recommender systems. Most of these systems are designed to serve specific purposes or deal with a particular set of topics, making them less adaptable to new subjects or areas. Furthermore, there is a noticeable absence of technology solutions that effectively address the specific need for matching programming tasks with relevant datasets, a gap that this project aims to bridge.

To address these challenges, this study takes a thorough and diverse approach. The first step of this approach is the evaluation of various individual predictive methods, including popular machine learning algorithms as well as similarity-based collaborative and content-based filtering. Each method is tested and assessed rigorously to understand its potential as well as its strengths and weaknesses fully.

The next step taken is the aggregation of the most effective methods to create an integrated system. This hybrid approach maximizes overall performance by leveraging the best techniques for each predictive task. Finally, this integrated system is subjected to rigorous evaluation processes, ensuring that it not only functions well but also effectively recommends suitable programming tasks for datasets.

The result of the chosen approach is a versatile, hybrid recommender system. This system stands out for its adaptability, being capable of handling a variety of topics, making it a valuable tool for educators across different disciplines. During the analysis of the individual predictive methods, some machine learning techniques, particularly decision trees, and random forests, exhibited strong performance.

However, when these techniques were integrated into the recommender system, the overall performance showed a marked improvement, highlighting the power of a hybrid approach. Collaborative filtering, in particular, proved to be a valuable component, providing high-quality predictions in scenarios where the system struggled to reach a consensus.

This study presents a significant advancement at the intersection of education and data science. It introduces a new recommender system that predicts suitable programming tasks for specific datasets, providing a solution to the challenge faced by educators and addressing the gap in research on this topic. The development of this hybrid, adaptable system also lays the groundwork for future research and the creation of effective systems across different domains.

In a broader context, this work contributes to the overarching objective of enhancing education through technology. By simplifying the process of selecting datasets for programming tasks, this system has the potential to save educators valuable time and effort, allowing them to dedicate more attention to teaching and interacting with students. Additionally, the developed system exhibits great potential for applicability across a wider variety of topics, paving the way for recommender systems that are designed to serve multiple purposes.

# 2 | Related Works

The objective of this project revolves around engineering a recommender system that predicts suitable programming tasks for specific CSVs. However, this necessitates a thorough understanding of the current state of research in this field. Therefore, this section presents the exploration of several key areas related to the development of recommender systems, ensuring a better understanding of the field. The first subsection will provide an overview of prevalent recommender systems and their respective application domains. Following this, the predominant types of deep learning and machine learning techniques employed within the development of the systems are presented. Lastly, the diverse types of recommender systems and their adeptness in varying scenarios will be presented. This section provides insight into the development process of recommender systems, as well as the various predictive methods employed in their development.

## 2.1 | Prevalent recommender systems and their domains

The systematic review conducted by Portugal et al., 2018 sheds light on the utilization of machine learning algorithms in the creation of recommender systems and the prevalent research trends in this area. One of the goals of their study is to assist researchers in appropriately positioning new research activity in the domain. Additionally, they explored the utilization of machine learning in the development of various recommender system types. They identified prevalent trends in the construction of three primary categories: content-based filtering, collaborative filtering, and hybrid systems. They also investigated the prevalent domains and found the most frequent to be movies, social networks, and e-commerce. Notably, the coding and algorithm domains received limited attention.

In their comparative analysis, Ravi et al., 2022 compared recommender systems, expert systems, and explainable artificial intelligence. They also emphasize the substantial data requirements for applying deep learning techniques in recommender systems and explainable artificial intelligence. The findings from both studies suggest that certain recommender systems and predictive methods tend to be more prevalent in specific topics. However, it is crucial to acknowledge that this is due to the specifics of certain topics, as the choice and utilization of a particular model are dependent on the availability of data and the specific objectives of the recommendation task.

## 2.2 | Deep learning and machine learning techniques

In recent years, the usage of deep learning in recommender systems has increased. Batmaz et al., 2019 conducted a literature review on deep learning in recommender systems, investigating challenges and prevalent domains. Deep learning techniques presented in the paper include restricted Boltzmann machines, deep belief networks, auto-encoders, recurrent neural networks, and convolutional neural networks. However, a fact highlighted in the paper by Ravi et al., 2022. is that deep learning requires a large amount of data to be effective in recommender systems. Therefore, although deep learning techniques could be a feasible method for implementing a recommender system in many domains, the implementation is dependent on the availability of large amounts of data.

In their paper, Portugal et al., 2018 had, as a first goal, identifying trends in the use and research of machine learning algorithms in recommender systems. Throughout the paper, they presented the most prevalent machine learning techniques, the main approaches used for classes of recommender systems, as well as the most commonly used evaluation metrics in the reviewed papers. The most prevalent machine learning techniques identified were ensemble, k-means, support vector machines (SVM), Bayesian, and decision trees.

The most commonly encountered recommender system in the works reviewed were collaborative filtering systems, with neighborhood-based recommenders being more common than model-based ones. The second most encountered category of recommender systems were those with content-based filtering, with the subcategory of classifier-based systems being more common than the neighbor-based ones. The least encountered type of recommender systems were the hybrid ones. When investigating the most common evaluation methods, the researchers identified precision, recall, F-measure, RMSE, and MAE as the most frequently used performance metrics.

All these findings on the use of machine learning algorithms in implementing recommender systems in different domains are useful in helping to situate new research activity appropriately. The systematic review gives insight into the most common methods implemented in the field of machine learning for recommender systems. It can provide a starting point for new implementations in a different domain of application for new recommender systems.

## 2.3 | Recommender system types and their contextualization

Recommender systems are defined as computational structures designed to present personalized suggestions to users, founding their recommendations based on users' historical behaviors and preferences (Shani and Gunawardana, 2011). The fundamental components of recommender systems are usually users and items. The users represent the component to which the recommendation is being made, while items refer to the objects that are being recommended. Depending on the type of system the recommendations are made taking into consideration a combination of user and item similarity as well as the interaction between them. These recommender systems are typically categorized into three primary types: content-based filtering, collaborative filtering, and hybrid systems. Each of the first two types employs distinct methodologies and caters to diverse scenarios while the third one embodies properties of the other two.

### Content-based filtering and collaborative filtering systems

Content-based filtering systems primarily operate by generating recommendations based on user or item profiles. These systems necessitate some form of data concerning user characteristics or information on items. Subsequently, these systems suggest items that bear a resemblance to those previously favored by the user, or items that have been liked by similar users(Shani and Gunawardana, 2011). Content-based filtering systems can be further divided into classifier-based and neighbor-based systems (Shani and Gunawardana, 2011). Classifier-based systems employ machine learning algorithms, like decision trees, to predict the relevance of items to users based on past behavior. Meanwhile, neighbor-based systems recommend items that are similar to those previously favored by the user, based on similarity in item characteristics. In such systems, the cosine distance between the dataset and programming task vectors can be used to ascertain preference (Shani and Gunawardana, 2011).

On the other hand, collaborative filtering systems base their recommendations on the similarity between users' preferences. These systems exclusively rely on historical interactions, without considering the characteristics of individual users or items(Shani and Gunawardana, 2011). As such, these systems can encounter difficulties when dealing with new users or items, a challenge known as the 'cold start' problem (Lika et al., 2014). Collaborative filtering systems can also be differentiated into neighborhood-based and model-based systems (Shani and Gunawardana, 2011). Neighborhood-based systems identify similar users based on historical behavior and generate recommendations using mathematical calculations based on items liked by these similar datasets. On the other hand, model-based systems utilize statistical models, such as matrix factorization or deep learning models, to predict users' preferences, rank items that users have not interacted with, and generate recommendations (Shani and Gunawardana, 2011).

The selection between content-based and collaborative filtering systems is heavily influenced by the structure of the data and thus, by the presence of a 'cold start' problem. The term 'cold start' problem refers to the scenario where new users or items enter the database, but no information is available regarding their interaction with the other object types (Lika et al., 2014). Collaborative filtering systems can prove to be highly effective in generating recommendations when sufficient data is available. However, if new objects enter the system without any interaction information, the performance can suffer. These types of systems are widely spread through mass on-demand services. However, these algorithms require some interaction information when initializing a new object and get better in their recommendations after gathering more interactions. Therefore, in situations where the 'cold start' problem arises for collaborative filtering, other systems might be more suitable. Content-based filtering can be a viable option if there is sufficient data about the characteristics of the users and items (Lika et al., 2014). However, for situations that are not optimal for either system type, hybrid systems can be the solution.

### Hybrid recommender systems

The field of recommender systems has seen an increase in the variety of approaches, each with its own advantages and limitations. Two prominent categories are content-based and collaborative filtering techniques. However, researchers have discovered that the most effective recommender systems often combine multiple techniques in what is known as a hybrid approach.

For instance, the study conducted by Geetha et al., 2018 exemplifies the practical implementation of a hybrid recommender system. Their movie recommendation system successfully integrates content-based and collaborative filtering techniques, leveraging the strengths of each to overcome their individual limitations (Geetha et al., 2018). The user feedback received during the evaluation of their system further validates the utility of hybrid recommender systems. The system's effectiveness is not solely due to the integration of these techniques, but also the use of clustering, similarity, and classification strategies.

These findings are consistent with previous research in the field, suggesting that combining content-based and collaborative filtering can lead to more accurate and precise recommendations (Burke, 2002). Building on these insights, future studies can explore the incorporation of additional machine learning and clustering algorithms to further enhance the performance of hybrid recommendation systems. Furthermore, the applicability of the hybrid approach extends beyond movie recommendations. The reviewed literature suggests that this approach can be extended to domains such as music, videos, news, books, and e-commerce, offering users a more personalized and accurate digital experience (Geetha et al., 2018). The positive outcomes observed within the field indicate that similar methods can potentially be explored in other domains that have received less attention in recommender systems research.

Hybrid recommender systems aim to leverage the strengths of both content-based and collaborative filtering systems, addressing the limitations inherent to each approach. These systems provide a robust solution, particularly in scenarios where the 'cold start' problem is prevalent. The works of Burke, 2002 and Geetha et al., 2018 highlight the power of hybrid systems in enhancing the overall performance of recommendation systems.

Overall, integrating multiple techniques through hybrid approaches shows promise for advancing recommender systems. By leveraging the strengths of different predictive methods and tailoring them to specific domains, researchers can develop more accurate and effective recommendation systems that cater to the diverse needs and preferences of users. These hybrid systems offer a practical solution for optimizing recommendations in various domains, including those with limited or relatively scarce research.

# 3 | Objective

Based on the comprehensive literature review and relevant studies, it has been consistently observed that hybrid systems incorporating multiple predictive methods and/or steps tend to be the most effective and promising in the field of recommender systems. Building upon these findings, the primary objective of this project is to design, develop, and evaluate a recommendation system tailored for the topic of selecting programming tasks that can be performed utilizing specific CSV files.

The methodology suggests the possibility of improving task prediction accuracy through careful selection and combination of predictive methods. These selected methods will be integrated into a unified architecture for the final system. To guide the development process, three research questions have been formulated, each corresponding to a specific phase of system development. These research questions will provide the necessary structure and framework for achieving the project objectives effectively.

*Research Question 1: How accurate is each of the prediction methods in recommending suitable programming tasks?*

In the initial phase, the spotlight will be on scrutinizing the individual performance of various prediction methodologies. The accuracy of each method will be assessed. Collaborative filtering, content-based filtering as well as six machine-learning techniques will be evaluated. The machine learning techniques that will be evaluated are: linear regression, k-nearest neighbors, support vector machines, random forest, decision trees, and Naive Bayes. Expectations anticipate a range of performance across these methodologies, with certain methods likely to demonstrate superior prediction accuracy.

*Research Question 2: How can the most effective prediction methods be integrated into a unified architecture of a recommender system?*

Following the evaluation of individual methodologies, the research will progress toward developing an architecture that seamlessly integrates the most effective methods. The anticipated approach involves multiple phases, where the predictions from various models are combined to create an optimal recommendation system. The integration phase will also consider the trade-offs between complexity and performance, striving to maximize accuracy.

*Research Question 3: How does the performance of the integrated system compare to that of the individual prediction methods?*

Once the system is developed, the focus will shift to evaluating its overall performance. The expectation is that the integrated recommender system will outperform the individual prediction methods in terms of overall accuracy, thanks to the way the collaborative nature of the integration. To better understand the integrated recommender system, performance dynamics will be explored through three sub-questions:

*Subquestion 3.1: What is the frequency of disagreements among the individual prediction methods within each layer of the recommender system?*

The first sub-question revolves around discerning instances where the system's individual components diverge in their recommendations. Such discrepancies may necessitate invoking the next layer within the system, with the hope of yielding more refined recommendations. The aim of this question is to better understand the dynamics between the layers of the architecture, as well as the need for subsequent layers. The resulting information will be useful for system optimization and refinement.

*Subquestion 3.2: What is the accuracy of the predictions when the component methods are in agreement within each layer of the recommender system?*

The second sub-question focuses on the system's performance when there is a consensus among the component methods within a layer. It is expected that a unified agreement may indicate a higher level of confidence in the generated recommendations, potentially leading to increased accuracy. Evaluating the accuracy in these cases will provide insights into the information that passes to subsequent layers.

*Subquestion 3.3: How does the overall accuracy of the final prediction compare to the accuracy of the individual prediction methods?*

The third sub-question assesses the aggregate performance of the final recommender architecture, comparing it to the individual methods it encompasses. This comparison will be used to determine if the integration of multiple prediction methods into a unified architecture yields superior accuracy, thereby validating the system's design and conception.

The goal of this research project is to create a system that predicts possible programming tasks based on an input CSV file, aiming to simplify task selection for educators. Additionally, the aim is to integrate this system into a larger database of CSV files, providing professors with a user-friendly platform to easily find suitable files and programming tasks for their courses.

# 4 | Approach

This section presents an outline of the planned approach for designing a recommender system that predicts suitable programming tasks for specific CSV files. Prior research has shown that hybrid systems that integrate multiple predictive methods have the potential for achieving high performance. It is also evident from the literature that the type and amount of data required can vary depending on the specific system chosen. In the context of this project, CSV files and programming tasks will represent the core components of users and items, respectively.

The selection and effectiveness of a recommender system depend on the quality and availability of data, specifically in this situation, data related to programming tasks and datasets. When considering the development of a recommender system for this project, three main data structures are considered: the utility matrix, dataset profiles, and programming task profiles. The utility matrix captures the interactions between datasets and programming tasks and is represented as a sparse matrix. Each row of the matrix corresponds to a dataset, and each column represents a programming task. The dataset and programming task profiles consist of vectors that describe the characteristics of these items.

The tables below outline the assumed schema for the format of the three database components: Table 4.1 presents the presumed structure of the interaction matrix, Table 4.2 exhibits the assumed structure of the dataset profiles, and Table 4.3 displays the supposed structure of the programming task profiles.

| | programming task 1 | programming task 2 | programming task 3 |
|---|---|---|---|
| dataset 1 | 1 | ? | 0 |
| dataset 2 | 0 | 1 | 1 |
| dataset 3 | 1 | 0 | ? |

**Table 4.1:** Dummy structure of interaction matrix

| | Characteristics | | | |
|---|---|---|---|---|
| | #1 | #2 | #3 | #4 |
| dataset 1 | X | | | X |
| dataset 2 | | X | X | X |
| dataset 3 | X | | X | X |

**Table 4.2:** Structure of dataset profiles

| | Characteristics | | | |
|---|---|---|---|---|
| | #1 | #2 | #3 | #4 |
| programming task 1 | | | X | X |
| programming task 2 | | X | X | |
| programming task 3 | | X | X | X |

**Table 4.3:** Structure programming task profiles

As discussed previously, the choice of using content-based or collaborative filtering depends on the provided data and the state of potential new entries. For cases with an incomplete interaction matrix, a content-based approach may be preferable. This method uses dataset and programming task profiles to generate recommendations, ensuring that any new datasets or tasks introduced won't be affected by the cold start issue. However, if the interaction matrix is more comprehensive, a collaborative filtering approach might prove more effective. With the choice of developing a hybrid system, a balance needs to be struck between the necessary data and the true status of the available data.

For the context of the project, the assumption is that after the implementation of a base system, new items will be introduced without any available past interaction. It is also assumed that the state of the interaction matrix is mostly complete at the beginning of the project. Therefore, a combination of predictive methods will be employed, each being used in various stages of the prediction with various degrees of probability based on their performance capabilities. The system will primarily utilize the utility matrix and dataset profiles due to the combination of data availability and recommendation direction.

The approach will start with the evaluation of different predictive methods, including content-based filtering, collaborative filtering, and various machine-learning techniques. Each method will be analyzed individually to measure its performance. Depending on the specific predictive method, different sections of the data will be utilized. Content-based filtering uses dataset profiles to identify similar CSVs and recommends programming tasks that are preferred by those similar CSVs. On the other hand, collaborative filtering predicts programming tasks by analyzing the similarity of preferences between datasets using the interaction matrix. Meanwhile, various machine learning algorithms will assess the suitability of a programming task based on the characteristics of a CSV.

Once the individual predictive methods are evaluated, the next phase involves the development of an integrated system. This system will be designed to harness the strengths of the most effective models, applying them to appropriate situations or specific prediction tasks. It is envisioned as an ensemble system, somewhat akin to a random forest model, capable of taking multiple model predictions as input. It will then generate a final recommendation based on the majority prediction, thus capitalizing on the highest-performing methods.

Finally, the integrated system will undergo a detailed evaluation. This evaluation stage plays a critical role in understanding the effectiveness of the hybrid recommender system. To assess the system's performance, appropriate metrics will be employed, considering the findings from the literature review regarding commonly used metrics. This strategic and iterative approach involves sequential steps, starting with individual method evaluations, followed by system integration, and concluding with a comprehensive evaluation of the system's performance. Each phase is instrumental in progressing toward the primary objective - the design, development, and evaluation of an effective recommender system.

# 5 | Methodology

This section of the project presents the step-by-step procedures employed throughout the research, showcasing the data creation, the individual model evaluation, and the development and appraisal of the final system architecture. The first subsection presents the data creation, emphasizing the state of the initial data and expansion of it, the interaction matrix, and the dataset profiles data. The following subsection explains the methodology of evaluation of distinct models, including content-based filtering, collaborative filtering, and machine learning models, while also addressing the application of grid search for parameter tuning. The final subsection outlines the structure and evaluation of the ultimate system architecture, aiming to address the sub-research questions related to the third research question. The section provides a comprehensive overview of the methodologies as well as the framework employed throughout the research process.

## 5.1 | Data Creation

### Initial data available

The initial data available for the project consisted of six past assignments that were part of the JBI010 Programming course from the Joint Bachelor of Data Science at Tilburg University and the Technical University of Eindhoven. For each of the assignments, the data consisted of the CSV dataset used for that particular assignment as well as the PDF containing the assignment tasks and instructions. Since the available data does not match the assumed format of the data that would be later provided, pre-processing of the initial data was necessary. Several methods were used to transform and generate the required data to match the desired format shown in Table 4.1 and Table 4.2.

### 5.1.1 | CSV characteristics

As mentioned in the Approach section, the CSV characteristics are utilized as predictors for the purpose of predicting suitable programming tasks in this project. The structure of the target dataset was outlined and presented in Table 4.2. A function was developed to extract the characteristics of a given CSV and automate the characterization process. This function enables easy analysis and description of future CSV files in a consistent manner. The extracted dataset characteristics included row count, column count, missing value count, integer column count, flat column count, categorical column count, string column count, boolean column count, and object column count. Once all characteristics were computed, the function saved and outputted the dataframe presented in Table 5.1.

| | crow | ccol | object | float64 | int64 | bool | cnan | categorical | string |
|---|---|---|---|---|---|---|---|---|---|
| **books** | 550 | 7 | 3 | 1 | 3 | 0 | 0 | 1 | 2 |
| **GE** | 208636 | 5 | 4 | 0 | 2 | 0 | 0 | 3 | 0 |
| **hotelsv** | 478394 | 17 | 8 | 4 | 5 | 0 | 5312 | 6 | 2 |
| **hiv** | 173 | 95 | 95 | 0 | 0 | 0 | 0 | 1 | 94 |
| **ukraine** | 66722 | 21 | 13 | 4 | 6 | 0 | 142286 | 10 | 1 |
| **videos** | 38916 | 16 | 8 | 0 | 5 | 3 | 612 | 8 | 0 |

**Table 5.1:** CSV characteristics of initial 6 files

After ensuring the desired data format, the next challenge encountered is having only six entries in the dataframe, which is insufficient for effective model training and testing. To overcome this limitation, the dataframe needs to be expanded with additional entries. The first step is the extraction of multiple CSVs from Kaggle. The dataset titles and links can be found in Appendix B.1. By sourcing datasets from Kaggle projects, we ensure the validity of the CSV entries and gain insights into their intended use, which will aid in creating the mapping dataset later. After the characterization process, the dataset will consist of 25 entries, as shown in Table A.1. Due to the dataset's relatively small size and the project's scope restricting further search for additional datasets, synthetic data will supplement the existing one.

To create a dataset with 1000 entries, synthetic data will be generated by randomizing characteristics within a specified range. The mean and deviation of the characteristics will be used to generate random values for the remaining rows. This will ensure that the data stays within the parameters of the initial characteristics. The resulting dataset will consist of 1000 entries, representing the 9 CSV characteristics, with approximately 1/40 representing authentic CSVs and the rest consisting of synthetic data.

### 5.1.2 | Mapping

For the creation of the mapping dataset, the main elements necessary are the CSV characteristics and the list of possible programming tasks. To identify possible programming tasks, three source materials were investigated. Firstly, the six given PDF assignments were broken into parts in order to identify the main chapters of programming that are addressed in each of them. Secondly, the sourcebook for the course JBI010 Programming course from the Joint Bachelor of Data Science was investigated in order to evaluate the information gained throughout the course. Lastly, the findings were compared to the guide for Python Programming for Data Science , which aims to provide an introduction to using Python in data science. Links to the Python guides can be found in Appendix B.2. As a result of these investigations, 6 main chapters and programming tasks were identified. Therefore, the CSVs will be mapped to the following programming tasks: Data Manipulation, Data Visualization, Data Cleaning and Preprocessing, Programming Concepts, Exploratory Data Analysis, and Object-Oriented Programming.

The next step in mapping is to determine the data requirements for possible questions that may be part of a programming task. The goal is to identify the CSV characteristics that could be important for each programming task. After identifying a few relevant characteristics for each task, mapping functions are created accordingly. Once the 25 authentic CSVs are mapped to the programming tasks and validated against real usage of the datasets, the mapping functions can be deemed valid. Finally, the entire list of CSVs can be mapped to the respective programming tasks. The mapping of the first 25 CSVs (original 6 and an additional 19 from Kaggle) can be found in Table B.2.

## 5.2 | Individual Model Evaluation

This project employs two primary datasets in the modeling and evaluation processes: a CSV characteristics dataset and a mapping dataset. For clarity and consistency, these datasets will be renamed as *x_full* and *y_full*, respectively, as per the structures shown in Table A.1 and Table A.2 in Appendix A. The *x_full* dataset serves as the source of predictor variables, while the *y_full* dataset contains the corresponding target variables.

In keeping with established best practices from previous research, notably Nguyen et al., 2021 and Joseph, 2022, the datasets are partitioned into training and testing subsets with a 70-30 split. This division ensures a random but proportional representation across both subsets. Consequently, four new datasets are derived: *x_train*, *x_test*, *y_train*, and *y_test*. This clarified naming scheme aids in understanding which data is utilized in different modeling scenarios as discussed in prior sections.

This split facilitates a robust evaluation of the model's performance and accuracy. The training data, consisting of *x_train* and *y_train*, forms the basis for training the model and uncovering patterns within the data. Meanwhile, the testing data, represented by *x_test* and *y_test*, serves as an independent data set that allows for objective evaluation of the model's predictions.

This partitioning approach contributes to the robustness of the evaluation process by ensuring that the model is tested on unseen data, thus reducing the risk of overfitting. It allows for an assessment of the model's generalization capabilities and its ability to make accurate predictions on new, unseen instances. Throughout the modeling, parameters will be hyper-tuned using GridSearch and thresholds will be identified by testing the prediction on the training dataset. This ensures that the validation of the models will not interfere with the testing data which will remain untouched until the evaluation phase.

In the context of the collaborative filtering and content-based filtering approaches, the literature review and especially the works of Portugal et al., 2018 and Burke, 2002, combined with the subsequent machine learning approaches, have guided the selection of similarity methods. The likeness between entries is assessed using cosine similarity for collaborative filtering and the Pearson correlation coefficient for content-based filtering. These similarity metrics offer different perspectives on the relationships between entries, capturing both directional alignment and linear dependence of variables. This approach ensures that the recommendations provided are based on a comprehensive analysis of the similarities between entries and contribute to the overall effectiveness of both methods.

### 5.2.1 | Content-based filtering

Content-based filtering is the first approach used to recommend suitable programming tasks for CSVs. For this approach, for the test CSVs, the most similar train CSVs will be identified using cosine similarity. The data used for creating the similarity matrix is dataset *x_full* (the combination of *x_train* and *x_test*). For each CSV instance in the *x_test* dataset, the correlation matrix assists in identifying the most similar CSVs present within the *x_train* dataset. This notion of similarity, derived from the matrix, enables the selection of CSVs that exhibit comparable characteristics.

First, for each CSV in the test set, the indexes of the most similar CSVs from the train set are saved. This includes the most similar CSVs in terms of cosine similarity. To generate predictions, the mean of the target variables associated with the most similar CSVs from the *y_train* dataset is assigned to *y_pred* for each entry in *x_test*. This process ensures that the predictions are based on the collective behavior of similar CSVs in the training set.

Evaluating the accuracy of the predictions is a crucial step in assessing the performance of the content-based filtering approach. If the predicted values are not 0 or 1, but values between, they are checked against a 0.5 threshold and transformed into 1 or 0. The choice of threshold is due to the final's product preference of receiving recommendations of partially unclear predictions as well. Afterward, the evaluation is conducted both at the individual programming task level and as an overall measure. By comparing the predicted values (*y_pred*) with the actual values (*y_test*), the accuracy for each programming task can be determined. Lastly, the overall accuracy, balanced accuracy, F-1 score, precision, and recall will be computed. The metrics were chosen based on research presented in the related works section.

### 5.2.2 | Collaborative filtering

Collaborative filtering is the next approach used to recommend suitable programming tasks for CSVs. This approach entails constructing correlation matrices using an adapted version of the *y_full* dataset. Specifically, the combination of *y_train* and *y_drop*, which represents a modified variant of *y_test*, is used. In *y_drop*, 1 to 3 values are randomly dropped from every entry of the dataset and later replaced with 0.5, representing a midpoint between suitability and unsuitability in terms of programming tasks.

The correlation matrix is then created based on the mix of *y_train* and *y_drop*, incorporating the Pearson correlation coefficient. The matrix effectively captures the likenesses among different entries, allowing for the estimation of missing values. For every entry in *y_drop*, any absent values are substituted with the mean derived from the most similar entries in *y_train*, based on the similarity matrix.

After assigning the mean prediction, a threshold is applied to classify the values as recommended or unrecommended programming tasks. Values above 0.5 are considered recommended, while values below 0.5 are categorized as unrecommended. This transformation process ultimately converts the predicted dataset into a binary format, specifically denoted as 1 and 0. To evaluate the accuracy of this approach, a comparison is made between the predictions and the actual values contained within the initial *y_test* dataset. The evaluation consists of individual prediction tasks and an overall appraisal of balanced accuracy, accuracy, F1-score, precision, and recall (same metrics used for content-based filtering).

Similar to the content-based filtering approach, the collaborative filtering method aims to deliver robust recommendations by leveraging the correlation matrix and considering the similarities between different entries. By utilizing the modified *y_full* dataset and employing statistical measures, the collaborative filtering approach aims to provide accurate and reliable recommendations of programming tasks for cases when some of the programming task preferences are known.

### 5.2.3 | Machine learning

The next approaches that will be evaluated are six types of machine learning models. Both classifier and regressor options will be tested for the following algorithms: linear regression, k-nearest neighbors, support vector machines, decision trees, random forest, and Naive Bayes. These will be modeled and tested using the training and testing data described previously, following a 70-30 split ratio. Thus, the *x_train* and *y_train* datasets will be employed for model fitting, while *x_test* will serve as the basis for prediction, with subsequent comparison against the *y_test* dataset.

#### Threshold tuning

Considering the nature of the regressor models, their output will not fall within the binary classes of 0 or 1 but instead represent a continuous variable within this range. Consequently, direct calculation of accuracy and balanced accuracy measures becomes unfeasible after predicting the test set. However, with the objective of delivering a binary target prediction and assessing the performance, the focus lies on investigating the optimal threshold value. Although the conventional initial threshold is commonly set at 0.5 for values between 0 and 1, tuning the threshold will take into account model variations and potential imbalances across different programming tasks. Thus, the goal is to identify the optimal threshold for each model and programming task combination.

To determine the threshold value without involving the test set, the method will be developed using the training set exclusively. The same model employed for prediction on the test set will be used to predict the training set, on which the model was initially fitted. A method similar to that in the article Brownlee, 2020 will be employed. Thus, in increments of 0.01, a range of threshold values will be examined to identify the optimal F1 score for the model's predictions on the training set. Subsequently, the best-performing threshold value will be employed for making predictions on the test set. This methodology ensures that the mapping of predictions to binary values maximizes the efficiency of the model specifically for each individual programming task.

### GridSearch for Parameter Tuning

Optimizing model performance necessitates careful consideration of parameters in support vector machines, k-nearest neighbors, decision trees, and random forest models. Choosing the optimal combination of parameters is important for these four types of machine learning. To ensure parameter tuning remains independent of the test set, the process is exclusively conducted on the training set. GridSearch, with a 5-fold cross-validation, is employed to identify the optimal parameters from a predefined set for each model. Implementing GridSearch guarantees that the models are fine-tuned solely using the training data. The 5-fold cross-validation further enhances the process by continuously splitting the data, providing a validation dataset at each step of the algorithm. In this manner, model performance can be effectively assessed while avoiding any data leakage from the test set.

This method is employed for machine-learning algorithms where parameters can be fine-tuned. Therefore, it is applied to support vector machines, k-nearest neighbors, decision trees, and random forests. By utilizing GridSearch with cross-validation, the most suitable parameter configurations can be determined for each machine-learning model. This systematic approach ensures robust and reliable model performance by optimizing the parameters based on the training data, thus enhancing the predictive capabilities of the four predictive approaches.

### Evaluation

Each of the 12 machine-learning models is evaluated against the $y\_test$ dataset. The evaluation follows five measurements, namely accuracy, balanced accuracy, F1 score, precision, and recall, chosen based on the literature reviewed in the related works section. The results are stored in a dataset, along with the evaluation scores of the content-based and collaborative filtering methods.

## 5.3 | Architecture creation and evaluation

The review of relevant literature highlights the benefits of integrating diverse models within the recommender system's architecture. Thus, the aim of this part of the project is to merge the highest-performing models into a unified architecture, which will then generate the final predictions. The system's design ensures that these models are used to their full potential, with careful consideration given to both the overall system performance and the performance on individual programming tasks.

With the purpose of seamlessly integrating these models, the system will be structured into layers. The layers work together in the sense that each layer tries to predict the unfinished predictions of the previous layer. Within each layer, the models independently generate predictions, which are then aggregated. In cases where disagreements arise within a layer, the situation is escalated to the subsequent layer, where a new set of models collaboratively strives to reach a consensus on the recommendation. This ensures that the system incorporates a diverse range of perspectives and factors, leading to predictions that are more accurate and dependable.

In the upcoming sections, the methodology employed to integrate these models as well as the processes utilized to establish consensus within a layer will be explained. The evaluation process of the model's performance will also be presented. The emphasis throughout is on developing a recommender system that is both robust and adaptable, and that takes advantage of the multitude of models used in order to increase its performance.

### 5.3.1 | Data

The data used will be the previously introduced ' *x_full*' and ' *y_full*' datasets, with a 70-30 split ratio for the training and testing. To maintain the integrity of the evaluation process, the test data remains untouched until the final evaluation phase. As a result, only the models that show good performance on the training data are considered for integration into the system.

In order to steer clear of using the model evaluation done on the test data, as presented in the previous section, cross-validation is applied to the training data. This way, the performance of different models can be measured on the same scale. The evaluation method will be the same as the one done previously, but the training and testing data will differ such that the true test data will not be used. Known as k-fold cross-validation, this technique is a strong way to choose models, and it lessens the chance of overfitting on the test data.

When it comes to evaluating models, a 5-fold cross-validation is used. This method splits the training dataset into five subsets of equal size. Each round of the validation process has the model fitted on four of these subsets and then tested on the remaining subset. This cycle occurs five times in total, each time with a different part serving as the test set. The models are then evaluated based on their average performance across these five iterations.

Figure 5.1 presents the overall structure of the K-fold evaluation technique, while Table 5.2 presents the structure of the evaluation DataFrame.

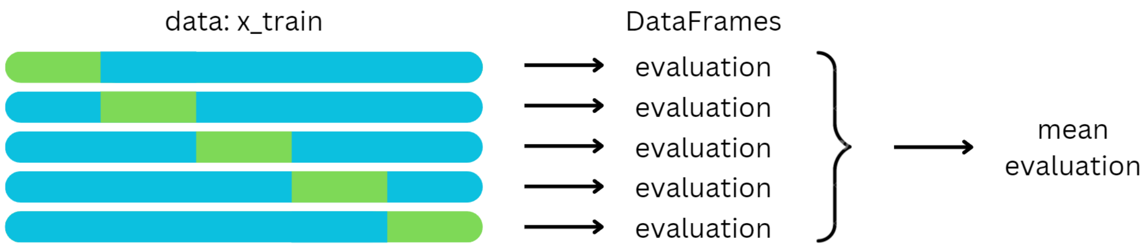

**Figure 5.1:** K-fold evaluation of individual prediction methods

| **models** | **accuracy** | | **balance acc.** | | **F-1 score** | | **precision** | | **recall** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **all** | **pt i** | **all** | **pt i** | **all** | **pt i** | **all** | **pt i** | **all** | **pt i** |
| CBF | | | | | | | | | | |
| CF | | | | | | | | | | |
| ML j regression | | | | | | | | | | |
| ML j classification | | | | | | | | | | |

**Legend: i:** index indicating programming task **j:** index indicating machine learning

**Table 5.2:** Structure of evaluation dataframe

This methodology of evaluating using validation ensures a comprehensive assessment of each model's performance. It also helps reduce bias that might come from the specific way the training data is arranged. The process of cross-validation provides a fair look at the models' skill in applying their training to data they haven't seen before. As model performance will be the key factor in choosing models for the system, an evaluation that takes into account the need for good prediction power on unseen data is important. The goal is to prevent overfitting on the test data, making sure that the models chosen will give solid and dependable predictions.

### 5.3.2 | Structure

The design of the system will follow a layered approach, with three unique layers. This approach is intended to make the system work better by improving the prediction accuracy with each layer.

Firstly, part of the input for the system will be the evaluation of individual predictive model approaches. The system will necessitate these such that it can pick the models that will be employed in each layer. This evaluation is done using the 5-fold cross-validation method on the training dataset, as mentioned before. The models used by the system are chosen based on their average performance within the cross-validation. However, it's worth noting that the collaborative filtering model is reserved for the final stage of the system because it needs some previous information regarding programming task preferences. For each of the layers, the best models will be selected by means of thresholds. Therefore, a specific threshold will be set for each layer, and the chosen metric (balanced accuracy for layer 1 and normal accuracy for the other layers) must surpass this threshold for the model to be integrated. While different layers will be looking at different performances, the performance scores will be compared to preset thresholds of 0.75 for the first layer, 0.85 for the second layer, and 0.95 for the third layer. The chosen thresholds strike a balance between allowing for diverse model options while maintaining a minimum performance standard. However, the user can choose to change these preset thresholds. In case the threshold filtering yields too few models, the safeguard implemented is that for each layer, a minimum number of models need to be selected. For the first layer, the minimum number of models is 7, for the second layer the minimum number is 5 and for the last one, the minimum number is 3. This ensures that if within a layer, the final prediction is agreed upon, this prediction has a very high chance of being an accurate recommendation. This is important as predictions that are agreed upon in one layer will not be double-checked in the next one. The input data for the selection process consists of the evaluation dataframe generated from the K-fold cross-validation, along with a list specifying the minimum number of models and the corresponding threshold values. Using this data, the selection process identifies and returns a list of models that meet the defined standards for each predictive task within each layer.

Figure 5.2 presents the overall design of the system, showcasing the connections and communication between the different layers. On the other hand, Figure 5.3 provides a visual representation of the prediction process within each layer, illustrating how predictions are generated and flow through the system. Together, these figures provide a comprehensive overview of the system's architecture and the sequential flow of predictions within it.

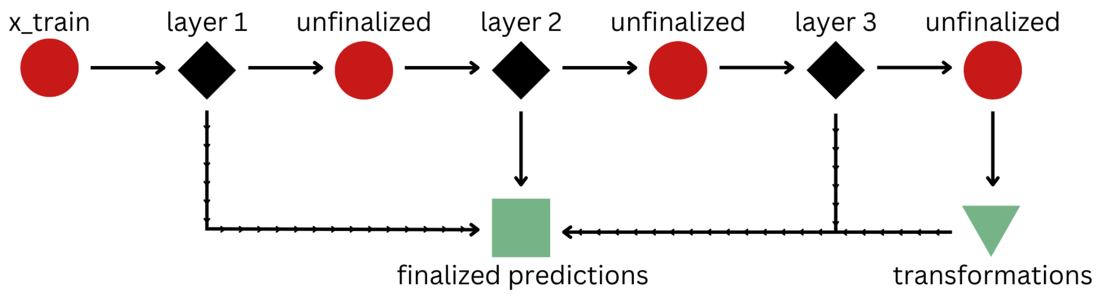

**Figure 5.2:** Overall structure of the architecture

The first layer of the architecture is populated by the models with the best overall performance. Within this layer, the predictions made by the models are averaged, creating a consensus prediction. Any predictions in which all models agree, resulting in an average of either 1 (indicating suitability) or 0 (indicating unsuitability), are regarded as finalized. These finalized predictions are not subject to further assessment in subsequent layers.

For those tasks not finalized in the first layer, they are forwarded to the second layer. The models used in this layer are selected based on their performance on each individual programming task prediction. Therefore, this layer enables a more granular, task-specific optimization of the prediction process.

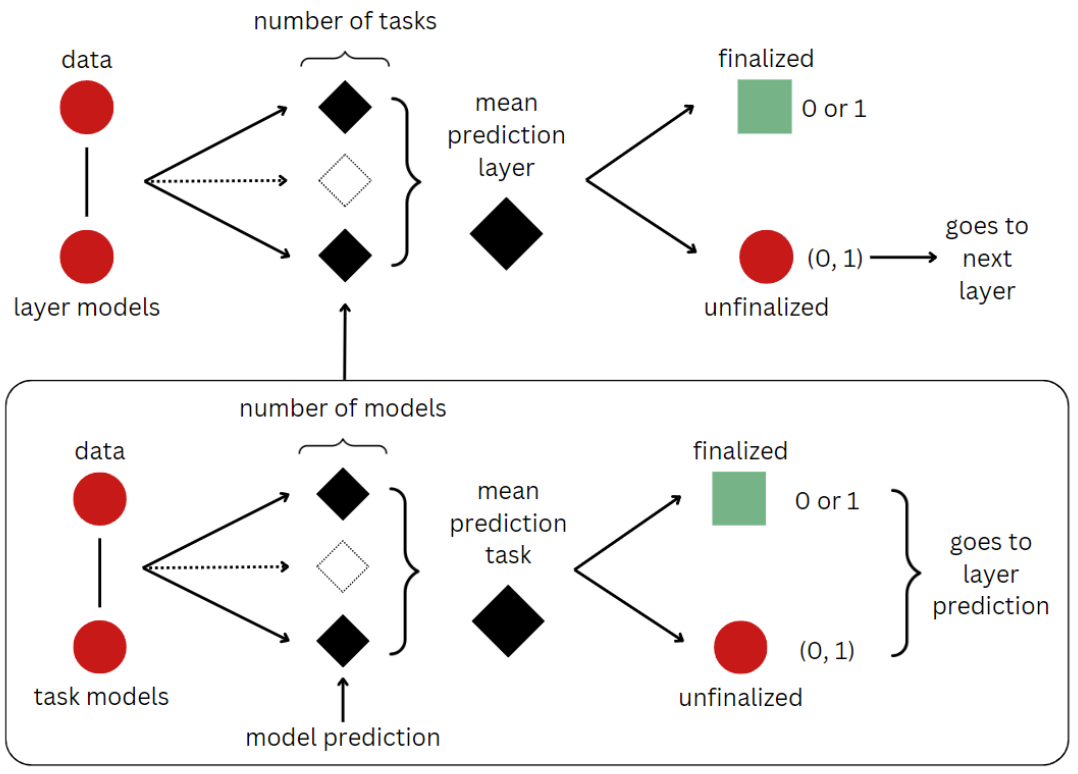

**Figure 5.3:** In-depth view of general layer prediction

Once again, the prediction process within this layer involves averaging the predictions of the models. This average prediction is then integrated back into the broader matrix and passed on to the third layer. As with the first layer, any unanimous predictions (with an average of 1 or 0) are deemed final and are not subject to further prediction in the third layer.

The architecture's third and final layer presents a more stringent set of criteria for model selection. This layer focuses on the performance of the models on individual programming tasks, requiring a higher standard than the preceding layers. Furthermore, this layer introduces a range of options for users, providing flexibility and customization in the system's output. The choice of stricter criteria ensures that the best top-performing models are the only ones predicting the recommendation.

This layer of the system is where the collaborative filtering model has a chance to come into play. This model is uniquely suited to this stage due to its reliance on pre-existing finalized predictions, which are now available from the previous layers. Therefore, the user has the option to choose whether to include the collaborative filtering model in the prediction process. In such cases, the unfinalized predictions are subsequently finalized using the collaborative filtering model, which takes advantage of the pre-existing predictions from previous layers. Additionally, users have the option to choose from 8 formats for the non-finalized predictions.

The 8 transformation options are a combination of individual and combined methods. One set of options involves the use of a confidence interval transformation. Half of the output formats utilize this confidence interval transformation, while the other half do not. When a confidence interval is applied, values that are within the specified distance (pre-loaded as 0.2) of 0 or 1 are mapped to them accordingly. As mentioned before, one of the options for transformations is collaborative filtering. This option is available both with and without a confidence interval. If applied with a confidence interval, the values that are still not 0 or 1 after applying the interval transformation, are mapped to 0 or 1 with collaborative filtering. Another option for transformation is using the top model. This option is similar to the collaborative filtering option and its only difference is that it applies the best-performing model for that specific task instead of collaborative filtering. The top model transformation can also be done both with and without a confidence interval transformation. Another option is to deliver percentage transformations. This option can also be employed both with and without a confidence interval. This option simply delivers any remaining undecided predictions as percentages. The last confidence interval option is to deliver undecided predictions as 0.5. The last transformation option is of using a threshold mapping (with a pre-loaded value of 0.5). Therefore, all undecided predictions larger than the threshold are mapped to 1, while the others are mapped to 0.

### 5.3.3 | Evaluation of the architecture

The evaluation of the system's architecture aims to find a better understanding of the input each layer has on the final performance, as well as the relation between layers and models within a layer. This assessment will also help in answering the last research question.

Firstly, the focus will be on evaluating the disagreements within each layer. Of interest on this topic are both the overall ratio of agreements to disagreements as well as the distribution of these. For this task, the amount and position of finalized decisions will be assessed within each layer. This will yield the proportion of overall disagreement and the distribution over the individual programming tasks. These results will help in elevating the understanding of the system's performance and help identify possible faults in the layers or programming tasks where the prediction power is decidedly lower.

The next aspect of the system performance that will be investigated is the accuracy of the finalized predictions. Of interest in this case are once again, both the overall accuracy and the accuracy of each of the programming tasks. This aspect of the system's performance is very important due to the fact that finalized predictions are no longer touched in the subsequent layers. Therefore, if the predictions are not accurate in the first layer, then they will not be accurate in the final result either and will make the overall accuracy of the system lower. Thus, this analysis is an important part of a comprehensive understanding of the final performance of the system.

Lastly, the accuracy of the final results will be compared with that of the individual model approaches. For this step, similar to the evaluation of individual models, measures of accuracy and balanced accuracy will be calculated for each of the programming tasks, as well as for the overall performance. For the final system, the various options of output will also be compared in order to find if there is an increase in performance in one of the choices of output for the non-finalized predictions.

# 6 | Results

The following section presents the results obtained from the final evaluation of the implemented system. The results will specifically address the research questions and subsequent sub-research questions that have guided the project. It is important to emphasize that the results presented in this section are derived from a single, uninterrupted run of the implementation process, encompassing the system's complete execution from start to finish. Although the implementation allows for adjustments in threshold values at certain stages, it is crucial to note that running the system again, for instance, by resplitting and regenerating the data, may yield marginally different results. Furthermore, modifying the threshold values can potentially lead to the utilization of different models, thereby introducing variations in the outcomes.

It is worth mentioning that prior to the architectural implementation, individual methods were also evaluated at an intermediate point. However, the results presented in this section refer only to the final evaluation, which occurs after the execution of the implemented system. By adopting this approach, it is ensured that the results encompass the full extent of the system's performance, incorporating all modifications and refinements implemented throughout its execution as well as portraying the final run in cases where a model might have been called multiple times throughout the process.

Throughout this section, each research question and sub-research question will be presented individually, providing an in-depth analysis of the system's effectiveness in addressing the specific objectives outlined in this study. This structured approach enables a comprehensive analysis of the system's efficacy in addressing the specific objectives of the project as well as a deeper understanding of the effects some of the decisions have on the final results. By adopting this approach, the aim is to present a comprehensive evaluation of the implemented system's efficacy and its fulfillment of the research goals.

## 6.1 | Research question 1

In accordance with the methodology outlined in the previous section, a total of 14 individual methods were evaluated. Each method underwent evaluation for task prediction as well as an overall assessment based on five different metrics: accuracy score, balanced accuracy score, F1 score, precision, and recall. Some of the rounded results for each method can be found in the appendix of this paper (accuracy scores in Table A.3). Additionally, a CSV format of the results is provided in the repository linked in Appendix B.4, allowing for further exploration and analysis.

Upon examining the final results, it becomes evident that except for the Multinomial Naive Bayes model, all other 13 models predict with an accuracy score of over 0.7. Furthermore, 10 of these models (excluding Support Vector Classification, Support Vector Regression, and K-Neighbors Regressor) also exhibit a balanced accuracy score above 0.7. From these findings, it can also be inferred that Random Forest and Decision Trees emerge as the highest-performing models, followed by collaborative filtering. However, it is crucial to note that the prediction and evaluation of collaborative filtering are heavily dependent on the specific dropped values used for testing. Consequently, the performance of collaborative filtering may not be consistently maintained across all cases.

| Model | Overall | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---:|---|---|---|---|---|---|---|
| LogisticRegression | 0.871 | 0.753 | 0.891 | 0.883 | 0.830 | 0.925 | 0.918 |
| LinearRegression | 0.849 | 0.805 | 0.943 | 0.860 | 0.864 | 0.781 | 0.840 |
| MultinomialNB | 0.666 | 0.610 | 0.618 | 0.757 | 0.549 | 0.920 | 0.542 |
| GaussianNB | 0.894 | 0.806 | 0.912 | 0.889 | 0.924 | 0.933 | 0.910 |
| KNeighborsClassifier | 0.705 | 0.785 | 0.589 | 0.708 | 0.549 | 0.919 | 0.560 |
| KNeighborsRegressor | 0.674 | 0.770 | 0.570 | 0.663 | 0.551 | 0.850 | 0.563 |
| DecisionTreeClassifier | 0.974 | 0.996 | 0.928 | 0.972 | 0.984 | 0.996 | 0.972 |
| DecisionTreeRegressor | 0.979 | 0.996 | 0.957 | 0.972 | 0.984 | 0.996 | 0.972 |
| RandomForestClassifier | 0.982 | 1.000 | 0.950 | 0.981 | 0.982 | 1.000 | 0.977 |
| RandomForestRegressor | 0.972 | 0.996 | 0.925 | 0.966 | 0.984 | 0.996 | 0.972 |
| SVC | 0.641 | 0.658 | 0.603 | 0.657 | 0.567 | 0.751 | 0.576 |
| SVR | 0.636 | 0.646 | 0.606 | 0.646 | 0.553 | 0.751 | 0.573 |
| CBF | 0.787 | 0.775 | 0.691 | 0.844 | 0.683 | 0.895 | 0.769 |
| CF | 0.835 | 0.832 | 0.856 | 0.815 | 0.825 | 0.840 | 0.831 |

**Table 6.1:** Balanced Accuracy Scores Individual Prediction Methods

Table 6.1 presents the balanced accuracy scores for both the overall evaluation and each individual programming task. These results reveal that the Random Forest Classifier achieves perfect predictions for Task 1 and Task 5, while also exhibiting the highest performance for Task 3 and Task 6. On the other hand, the Decision Tree Regressor emerges as the highest-performing model for Task 2. In the case of Task 4, the Random Forest Regressor demonstrates the highest balanced accuracy. Additionally, it is noteworthy that certain models, such as Logistic Regression and Multinomial Naive Bayes, display notable differences in their prediction capabilities depending on the specific task being evaluated.

In conclusion, the evaluation of the 14 individual methods reveals varying levels of performance across different metrics. The results indicate that Random Forest and Decision Trees emerge as the top-performing models. Collaborative filtering, although dependent on the iteration of dropped variables also exhibits high accuracy. The findings also highlight that lower overall accuracy doesn't necessarily mean a lower accuracy in all programming tasks but might also represent a big fluctuation in prediction capabilities depending on the task being predicted. These findings facilitate the understanding of certain models as well as highlight the potential some of them have for specific predictive tasks.

## 6.2 | Research question 2

The second research question delves into the incorporation of individual methods into a system. In pursuit of this objective, several combinations were tested, aiming to identify an architecture that guarantees both the highest level of certainty and comprehensive range across all predictive tasks. The architecture presented in the methodology section emerged as the most suitable choice, meeting the desired criteria.

Analyzing the results from the previous subsection, it becomes evident that certain models exhibit relatively strong prediction capabilities across all tasks, while others excel only in specific prediction tasks. Additionally, the collaborative filtering model necessitates pre-existing task predictions to be available. Thus, a simple overall average would provide a rudimentary solution but lacks comprehensiveness.

The concept behind implementing layers is to employ increasingly specialized groups of models. Initially, models with an overall accuracy above a predetermined threshold contribute to the baseline prediction. This is a simple yet effective way to integrate several methods into the prediction process. The subsequent layer, layer 2, focuses on the prediction capabilities for each specific task. Lastly, only the most effective models for each individual predicting task are utilized in the last layer.

The approach adopted for addressing this research question involved exploring various methods of generating predictions that incorporate the opinions of different models. The resulting structure allows for the inclusion of both models that perform well across all tasks and models that shine on particular tasks. By combining these different models in a systematic manner, the resulting predictions offer an understanding of each prediction task and ensure a higher certainty of prediction.

## 6.3 | Research question 3

The final research question focuses on evaluating the implemented system, encompassing both the assessment of its content and comparison with simpler, individual implementations. The system's output is contingent upon the approach chosen to handle values for which a unanimous agreement was not reached. There are eight options of output format and they were all evaluated using binary mapping thresholds set at 0.5 and confidence interval mapping thresholds at 0.2. For output options involving values other than 0 and 1, such values were considered incorrect for the accuracy evaluation. Assessing the effectiveness of the architecture in comparison to individual implementations serves as both a benchmark and validation of the initial premise that integrating multiple models can enhance performance. This evaluation directly contributes to the ongoing enhancement and fine-tuning of the system.

The models used in each layer of the system were carefully selected to ensure an unbiased decision-making process. Employing the same evaluation tactic as before, combined with a 5-fold cross-validation approach on the training data, the models were chosen based on their performance and accuracy. Specific balanced accuracy thresholds were set at 0.75, 0.85, and 0.95 for each layer, along with minimum model requirements of 7, 5, and 3, respectively. These thresholds and requirements ensured that only models meeting the desired level of accuracy and relevance were incorporated into each layer of the system. The resulting models used are as follows:

- Layer 1: Logistic Regression, Linear Regression, Gaussian Naive Bayes, Decision Tree Classifier and Regressor, Random Forest Classifier and Regressor, Content-Based Filtering (CBF)
- Layer 2:
  - Programming task 1: Linear Regression, Decision Tree Classifier and Regressor, Random Forest Classifier and Regressor
  - Programming tasks 2-4 and 6: Logistic Regression, Linear Regression, Gaussian Naive Bayes, Decision Tree Classifier and Regressor, Random Forest Classifier and Regressor
  - Programming task 5: Logistic Regression, Multinomial Naive Bayes, Gaussian Naive Bayes, K-Nearest Neighbors Classifier, K-Nearest Neighbors Regressor, Decision Tree Classifier and Regressor, Random Forest Classifier and Regressor, Content-Based Filtering (CBF)
- Layer 3:
  - Programming tasks 1 and 3-6: Decision Tree Classifier and Regressor, Random Forest Classifier and Regressor
  - Programming task 2: Random Forest Regressor and Classifier, Logistic Regression

- Top models:
  - Programming tasks 1, 3, and 4: Random Forest Classifier
  - Programming task 2: Logistic Regression
  - Programming tasks 5 and 6: Random Forest Regressor

.

### 6.3.1 | Subquestion 3.1

In addressing subquestion 3.1, the focus is on examining the agreement-disagreement ratio within each layer of the implemented system. The agreement is defined as a scenario where all models predict either 0 or 1, resulting in a mean prediction value of either 0 or 1 (excluding values in between). To evaluate this, the percentage of disagreements was calculated for each prediction task individually and overall. These percentages were determined for each layer, along with the percentage of disagreed-upon cases. A case registers as disagreed if there exists at least one disagreement within its associated prediction tasks. The results can be found in Table 6.2.

| Name | Case | Total | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---|---|---|---|---|---|---|---|---|
| Layer 1 | 0.920 | 0.385 | 0.490 | 0.467 | 0.310 | 0.343 | 0.293 | 0.407 |
| Layer 2 | 0.723 | 0.224 | 0.193 | 0.263 | 0.227 | 0.237 | 0.180 | 0.247 |
| Layer 3 | 0.173 | 0.030 | 0.003 | 0.117 | 0.020 | 0.027 | 0.003 | 0.010 |

Table 6.2: Cumulative Disagreement Proportions For Each Layer

| Name | Case | Total | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---|---|---|---|---|---|---|---|---|
| Layer 1 | 0.920 | 0.385 | 0.490 | 0.467 | 0.310 | 0.343 | 0.293 | 0.407 |
| Layer 2 | 0.786 | 0.583 | 0.395 | 0.564 | 0.731 | 0.689 | 0.614 | 0.607 |
| Layer 3 | 0.240 | 0.134 | 0.017 | 0.443 | 0.088 | 0.113 | 0.019 | 0.041 |

Table 6.3: Proportionate Disagreement Proportions For Each Layer

It is important to note that as the disagreements are calculated based on the entire layer after prediction, each subsequent layer can have at most the same number of disagreements as the previous layer, if not fewer. This implies that the results in Table 6.2 demonstrate how the proportion of disagreements decreases with each layer throughout the prediction process. From the table, it can be observed that after the first layer, 92% of cases had at least one disagreement. However, before the final transformation in the third layer, only 17% of cases exhibited at least one disagreement. Additionally, after the final layer, it is clear that prediction task 2 has the highest disagreement rate at 11.67%, while the remaining tasks demonstrate disagreements ranging from 3% to 1%, with prediction tasks 1 and 5 having disagreements in only 0.3% of cases.

To provide a more comprehensive understanding of the disagreement ratio within each layer, Table 6.3 is derived from the data in Table 6.2. This table specifically focuses on the predictions made within a single layer, enabling a clearer assessment of the disagreement ratio for that particular layer. Compared to Table 6.2, Table 6.3 exclusively considers the predictions made within the specific layer. Therefore, these results give a better understanding of the comparison of the performance of the predictions by not taking into account the cumulative performance of all previous layers. As the first layer does not have any previous layers, the values in Tables 6.2 and 6.3 are similar in this case. These results indicate that each layer does improve on the previous layer's predictive capabilities. Each subsequent layer predicts with a higher rate of agreement on the cases not agreed upon from the previous layer. The final layer has its highest disagreement rates of 44.3% in prediction task 2 and 11.3% in prediction task 4, while all the other prediction tasks have disagreements in under 10% of the cases (going as low as 1.7% for prediction task 1).

### 6.3.2 | Subquestion 3.2

Subquestion 3.2 focuses on assessing the accuracy of the agreed predictions within the three layers of the implemented system. In the previous subquestion, the emphasis was on the agreement ratio, with subsequent layers exhibiting a higher percentage of agreements. The logical next step is to evaluate the accuracy of these agreed predictions. It is crucial to ensure that the agreed predictions are correct, as they remain unchanged in subsequent layers. The system design prioritizes the idea of fewer but reliable agreements. Alongside the three layers, the transformed final layer, where values are mapped, is also evaluated. The detailed results, including all measurements, can be found in the corresponding CSV file in the provided repository while the accuracy score can be found in Appendix A.4. The balanced accuracy results are presented in Table 6.4.

| Name | Overall | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---|---|---|---|---|---|---|---|
| Layer 1 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Layer 2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Layer 3 | 0.988 | 1.000 | 0.964 | 0.988 | 1.000 | 1.000 | 0.980 |
| ci0_choice0 | 0.988 | 1.000 | 0.964 | 0.988 | 1.000 | 1.000 | 0.980 |
| ci0_choice1_thr0.5 | 0.977 | 0.996 | 0.951 | 0.966 | 0.982 | 0.996 | 0.972 |
| ci0_choice2 | 0.968 | 1.000 | 0.891 | 0.981 | 0.982 | 0.996 | 0.972 |
| ci0_choice3 | 0.989 | 1.000 | 0.970 | 0.989 | 1.000 | 1.000 | 0.980 |
| ci1_choice0_thr0.2 | 0.988 | 1.000 | 0.964 | 0.988 | 1.000 | 1.000 | 0.980 |
| ci1_choice1_thr0.2 | 0.988 | 1.000 | 0.964 | 0.988 | 1.000 | 1.000 | 0.980 |
| ci1_choice2_thr0.2 | 0.968 | 1.000 | 0.891 | 0.981 | 0.982 | 0.996 | 0.972 |
| ci1_choice3_thr0.2 | 0.989 | 1.000 | 0.970 | 0.989 | 1.000 | 1.000 | 0.980 |

**Table 6.4:** Agreement Balanced Accuracy Scores

From the presented results, it is clear that both Layer 1 and Layer 2 achieve a perfect accuracy score. This is highly desirable, as it indicates that inaccurate predictions are not propagated through the system from one layer to the next. The final layer has an overall balanced accuracy score of 98%. Task 1, Task 4, and Task 5 achieve perfect accuracy scores, demonstrating the system's proficiency in accurately predicting these tasks. The remaining tasks also exhibit high accuracy, with balanced accuracy scores exceeding 96%.

Notably, the only transformation that enhances the overall accuracy of the agreed predictions is the collaborative filtering transformation. It is also worth mentioning that transformations involving confidence intervals show either no significant difference or an enhanced accuracy in the agreed predictions compared to those without confidence intervals. Since these scores are calculated solely using predictions of 1 or 0, it implies that the values that were initially disagreed upon and subsequently mapped using the confidence interval approach have been accurately assigned in the final predictions. These findings highlight the system's ability to generate accurate predictions. The perfect accuracy of the first two layers backs up the reliability of the system. While the final layer does not produce the same perfect metrics for all prediction tasks, it still maintains very high accuracy in the other cases as well.

### 6.3.3 | Subquestion 3.3

Subquestion 3.3 centers around investigating the final results of the system after the transformation in the last layer. During this evaluation, transformations containing values other than 0 and 1 are deemed as incorrect predictions. These results are subsequently compared to the findings in Table 6.1 from the results of Research Question 1. Table 6.5 presents the outcomes of the eight different types of transformations. Upon comparing these two tables, it becomes apparent that certain transformations achieve a higher balanced accuracy for the overall prediction compared to the top-performing individual methods. Notably, employing the collaborative filtering method as the final transformation enables the perfect prediction of an additional task that could not be achieved solely by relying on the predictions of individual methods. However, it is crucial to note that the system's focus extends beyond balanced accuracy alone, as this metric is insufficient for a comprehensive evaluation of success. Alongside the improvement in prediction accuracy, an additional advantage of the system lies in its enhanced reliability of performance.

By comparing the results of the system's final transformation with the performance of individual methods, it is clear that certain transformations exhibit a superior balanced accuracy for the overall prediction. This signifies the effectiveness of the system in combining the strengths of multiple individual methods to achieve higher accuracy in the predictions. Furthermore, the inclusion of the collaborative filtering method in the final transformation allows for the successful prediction of an additional task that could not be accomplished solely through individual methods. This highlights the system's ability to leverage complementary information from different models to enhance its predictive capabilities.

| Model | Overall | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---|---|---|---|---|---|---|---|
| Layer 1 | 0.593 | 0.503 | 0.530 | 0.685 | 0.509 | 0.681 | 0.587 |
| Layer 2 | 0.763 | 0.801 | 0.733 | 0.767 | 0.664 | 0.812 | 0.748 |
| Layer 3 | 0.952 | 0.996 | 0.844 | 0.958 | 0.960 | 0.996 | 0.969 |
| CI0_Choice0 | 0.952 | 0.996 | 0.844 | 0.958 | 0.960 | 0.996 | 0.969 |
| CI0_Choice1_thr0.5 | 0.977 | 0.996 | 0.951 | 0.966 | 0.982 | 0.996 | 0.972 |
| CI0_Choice2 | 0.968 | 1.000 | 0.891 | 0.981 | 0.982 | 0.996 | 0.972 |
| CI0_Choice3 | 0.989 | 1.000 | 0.970 | 0.989 | 1.000 | 1.000 | 0.980 |
| CI1_Choice0_thr0.2 | 0.952 | 0.996 | 0.844 | 0.958 | 0.960 | 0.996 | 0.969 |
| CI1_Choice1_thr0.2 | 0.952 | 0.996 | 0.844 | 0.958 | 0.960 | 0.996 | 0.969 |
| CI1_Choice2_thr0.2 | 0.968 | 1.000 | 0.891 | 0.981 | 0.982 | 0.996 | 0.972 |
| CI1_Choice3_thr0.2 | 0.989 | 1.000 | 0.970 | 0.989 | 1.000 | 1.000 | 0.980 |

**Table 6.5:** Balanced Accuracy Scores Individual Prediction Methods

The evaluation of the system extends beyond the final results, encompassing both the underlying structure and the individual layer results. The results of individual layers provide valuable insights into the system's performance. The evaluation of the agreement-disagreement ratio, accuracy of the agreed predictions, and transformation effects in each layer explain the contribution of each layer to the overall system performance. Through this comprehensive analysis, it becomes evident that the systematic combination of models and the iterative refinement process result in improved prediction accuracy and reliability.

# 7 | Discussion

The results of the implemented system for programming task prediction have been analyzed, addressing the research questions and sub-research questions that guided this study. The discussion will now provide a comprehensive analysis of the findings, highlighting the key points and implications of the results. The system's performance, as indicated by its final model accuracy, exceeded the best individual model. This result lends credence to the system's methodology, combining multiple models to leverage their individual strengths, mitigate their weaknesses and increase reliability.

A unique feature of this system, compared to traditional models, is the enhanced reliability of its predictions. The results reveal that the predictions out of layer 2 were near-perfect, showing a remarkable accuracy of 99.98% after 1000 iterations (starting from the splitting of the data). This high level of accuracy boosts the system's reliability, making it a promising alternative to individual models. The consistency in accurate predictions observed in this study suggests the system's efficacy in providing reliable solutions, thus highlighting its potential for further application in similar prediction tasks. The robustness and generalizability of the system set it apart. Although applied to programming tasks in this study, the system demonstrated potential for broader application in other prediction tasks that utilize machine learning. The strong performance of the system, coupled with its adaptability, suggests its capacity to be incorporated into various prediction tasks beyond the programming domain.

Upon evaluating the agreement-disagreement ratio out of the predictions within each layer of the implemented system, it was observed that the second layer had the largest proportion of disagreements. This finding suggests that there is room for further optimization within the architecture. The introduction of an additional layer between the second and third layers could potentially lead to a decrease in the number of disagreements before reaching the final layer. Such an addition may further improve the reliability and accuracy of the system, giving it a chance to achieve better results.

The results of this study underscore the importance of model integration in enhancing the accuracy and reliability of predictions. The incorporation of multiple models into a layered system led to superior results compared to any of the individual models. This lends support to the approach of combining models to form a system, suggesting that a coordinated effort of multiple models may be more effective than relying on single models. Moreover, the layered structure of the system allows for flexibility and adaptability, enabling the system to perform reliably across different prediction tasks. The system's final models' balanced accuracy scores are a testament to its high performance. While some tasks, such as Task 2, exhibited higher rates of disagreement, the overall high accuracy of the final models demonstrates the system's proficiency in managing a diverse range of tasks.

While the layered system architecture exhibits considerable robustness and reliability, it is crucial to remember that the predictive power of the system is fundamentally dependent on the individual prediction methods, which form the bedrock of the system's predictions. There will inevitably be cases where the system's predictive capacity might falter, particularly when the individual models themselves are in disagreement. A clear illustration of this phenomenon can be seen in the example of Task 2, where the individual methods consistently struggled to reach a consensus, resulting in the system having the highest number of disagreements. However, despite this inherent challenge, the layered system exhibits its true merit by the end of layer 3, where the predictions that were agreed upon achieved an impressive accuracy rate of 97%. This performance surpasses the accuracy of any individual prediction method, further reinforcing the system's potential for increased reliability and robustness.

The validation of the system's performance with 1000 iterations, taking the mean of the results, revealed consistent outcomes. Additionally, the system's versatility and applicability in other domains were examined using a Diabetes classification dataset from Kaggle (found in Appendix B.3). While the performance results were not as high as with programming tasks in absolute terms, the system outperformed the individual methods. The detailed performance results can be found in one of the data folders within the code repository (linked in Appendix B.4).

# 8 | System Delivery

The code for delivering the system presented in the paper consists of five Python files, each containing different functions and classes. These files are named ”data_creation.py,” ”methods_pipeline.py,” ”system_architecture.py,” ”delivery_ui.py,” and ”main.py”. Each file uses some elements imported from previous files. To run the tool, users can simply execute the ”main.py” file, which acts as the entry point and calls the necessary components.

Running the ”main.py” file will call the necessary class and prompt the user interface. When the tool is run, a pop-up appears, prompting the user to make a choice between using the tool for prediction or testing. For testing, two options are available: using programming task prediction data or using other datasets. In the latter case, a preloaded diabetes prediction dataset with 4000 entries is included. Users can also provide their own dataset by following the instructions in the README file, ensuring the data is appropriately named and separated into predictor and target variable files. For using the tool to predict programming tasks, two CSV files are preloaded in the correct file location. Users can change or add additional CSVs in the correct file location as well. The preloaded CSVs are the full Diabetes dataset and a fruit prices dataset. Both were retrieved from Kaggle and the links can be found in Appendix B.3.

After selecting the purpose and dataset, users are prompted to choose from eight transformation options. They can select any number of transformations according to their needs. The choices are then double-confirmed with the users through the means of a pop-up that gives the option of going back and reselecting. Once the choices are confirmed, the user is asked whether they would like to retrain or not. Retraining involves running grid search and k-fold cross-validation again. While retraining is not required for predicting suitable tasks for specific CSVs, it is recommended when testing in order to ensure maximal performance. The tool notifies the user about this and mentions that training is mandatory when using a new dataset as errors can occur otherwise.

Depending on the transformations chosen, there are two pop-ups that may appear. These pop-ups ask the user whether they want to change the threshold value or the confidence interval value. Users can also choose to input multiple options for thresholds and confidence interval values if desired. The pop-ups prompt users to specify the number of values they want to input for each variable, and then the corresponding number of prompts appear, with the default variable values pre-loaded as options. During this step, if the value provided by the user is not within the range of 0 to 1 for the threshold or 0 to 0.5 for the confidence interval, an error message appears, requesting the user to input a valid value. The final values are double-checked with the user to ensure accuracy.

After the tool completes its execution, the results are displayed in a pop-up window and are also recorded in a text file, which includes the timestamp of the execution. For testing purposes, the pop-up window presents only the overall balanced accuracy to avoid overcrowding the interface, while the full results are available in the text file. This allows users to have a comprehensive view of the system's performance.

In summary, the code delivery includes a set of organized Python files that facilitate the execution and testing of the tool. User interaction is handled through pop-up prompts, allowing users to make choices and provide input for various parameters. Results are presented in a clear and accessible manner, providing users with the necessary information about the system's performance. The modular structure of the code ensures flexibility and ease of use, enabling users to utilize the tool for prediction and testing purposes efficiently.

# 9 | Conclusion & future work

This project aimed to answer several questions related to the accuracy of prediction approaches in recommending suitable programming tasks, the effective integration of these methods into a unified recommender system architecture, and a comparative analysis of the performance of the integrated system against the individual prediction methods. By addressing three research questions and three sub-questions, this project sheds light on the potential and effectiveness of the proposed system in delivering reliable predictions.

Research Question 1 centered around the accuracy of individual prediction approaches. The findings from our analysis revealed that decision trees and random forest algorithms consistently outperformed other methods in terms of precision. However, it's important to note that other models also performed well in specific tasks, underscoring the necessity of utilizing a broad palette of prediction methodologies. The localized high performance of some models supports the theory of necessitating specific model choices for each prediction task. Thus, applying only the overall best model to everything might not be the best solution as the one-size-fits-all approach doesn't always yield optimal results in all predictive tasks.

Research Question 2 was dedicated to integrating various prediction methods into a unified, effective system. The result was a layered system that aimed for consensus at each level, with disagreements leading to further consideration at the next layer. This methodology efficiently combined the strengths of each prediction method while mitigating their individual limitations.

For Research Question 3, a comparison was conducted between the integrated system and the individual methods. This comparison was broken down into three sub-questions to provide an in-depth understanding of the system's performance. Subquestion 3.1 evaluated the agreement-disagreement ratio within each layer, finding that disagreements were significantly diminished by the third layer. Through this analysis, it was also revealed that the second layer has the highest ratio of disagreements out of the predicted values. This finding could prompt further investigation into the balancing of the layers and the potential addition of further layers. Through the results of subquestion 3.2, near-perfect accuracy was discovered for the predictions on which component methods were under agreement within the first 2 layers. This finding further supports the efficacy of collective decision-making. Furthermore, the results indicated that the collaborative filtering method as a transformation of disagreed predictions shows the most promising results in terms of overall accuracy. Subquestion 3.3 compared the metrics of the integrated system to those of the individual methods. The results showed that some of the transformations of the system outperform the individual prediction methods, vouching for the merit of the integrated, layered approach.

In conclusion, the study highlighted the benefits of an integrated, layered system for programming task recommendations. The goal was not exclusively to achieve superior accuracy, but also to produce more reliable predictions across a broader variety of tasks. The aim was not just identifying the best predictive method, but the creation of a system that consistently delivers high-quality recommendations across a wider variety of prediction tasks.

The findings from this project offer valuable insights into the field of recommendation systems, emphasizing the significance of a diverse range of prediction methods, the strength of integration, and the essential role that a layered architecture can play in boosting prediction performance. Looking ahead, there is potential for this research to prompt more advanced, nuanced, and adaptable recommendation systems both on the topic of programming task prediction and beyond. The system leverages the strengths of multiple models, promoting a collaborative decision-making process, thereby aiming for more accurate and reliable predictions, facilitating improved decision-making.

The prospects for future work, based on the outcomes and findings of the current study, are abundant. The system shows significant promise beyond just programming task prediction, indicating a vast potential for further exploration and development. For instance, the layered architecture could be expanded upon, with the introduction of additional layers and nuanced adjustments to the decision thresholds within each layer. Furthermore, an in-depth investigation of the variety and quantity of models incorporated within each layer could potentially enhance the system's robustness and adaptability.

The choice of initial individual models offers another intriguing avenue for research. By integrating models that have demonstrated high efficacy in other prediction contexts, the system could potentially exhibit enhanced performance and generalizability. Moreover, applying this system to diverse prediction tasks is crucial for validating the structure of further research.

# 10 | Acknowledgments

This paper is written as part of the Final Bachelor Project and is submitted in partial fulfillment of the requirements of the degree of Joint Bachelor of Science in Data Science from the Eindhoven University of Technology and Tilburg University under the supervision of Prof.Dr. M. van den Brand, Dr. M. Seraj, and Dr. L. Ochoa Venegas. Online tools were used for grammar checking and rephrasing.

## References

Batmaz, Z., Yurekli, A., Bilge, A., & Kaleli, C. (2019). A review on deep learning for recommender systems: Challenges and remedies. *Artificial Intelligence Review*, *52*, 1–37.

Brownlee, J. (2020). A gentle introduction to threshold-moving for imbalanced classification. *Machine Learning Mastery*.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, *12*, 331–370.

Geetha, G., Safa, M., Fancy, C., & Saranya, D. (2018). A hybrid approach using collaborative filtering and content based filtering for recommender system. *Journal of Physics: Conference Series*, *1000*(1), 012101.

Joseph, V. R. (2022). Optimal ratio for data splitting. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *15*(4), 531–538.

Lika, B., Kolomvatsos, K., & Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert systems with applications*, *41*(4), 2065–2073.

Nguyen, Q. H., Ly, H.-B., Ho, L. S., Al-Ansari, N., Le, H. V., Tran, V. Q., Prakash, I., & Pham, B. T. (2021). Influence of data splitting on performance of machine learning models in prediction of shear strength of soil. *Mathematical Problems in Engineering*, *2021*, 1–15.

Portugal, I., Alencar, P., & Cowan, D. (2018). The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, *97*, 205–227.

Ravi, M., Negi, A., & Chitnis, S. (2022). A comparative review of expert systems, recommender systems, and explainable ai. *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, 1–8.

Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. *Recommender systems handbook*.

# A | Additional data tables and results

| | crow | ccol | object | float64 | int64 | bool | cnan | categorical | string |
|---|---|---|---|---|---|---|---|---|---|
| **books** | 550 | 7 | 3 | 1 | 3 | 0 | 0 | 1 | 2 |
| **GE** | 208636 | 5 | 4 | 0 | 2 | 0 | 0 | 3 | 0 |
| **hotels** | 478394 | 17 | 8 | 4 | 5 | 0 | 5312 | 6 | 2 |
| **hiv** | 173 | 95 | 95 | 0 | 0 | 0 | 0 | 1 | 94 |
| **ukraine** | 66722 | 21 | 13 | 4 | 6 | 0 | 142286 | 10 | 1 |
| **videos.csv** | 38916 | 16 | 8 | 0 | 5 | 3 | 612 | 8 | 0 |
| **admissions** | 1914 | 16 | 2 | 8 | 6 | 0 | 0 | 2 | 0 |
| **air** | 7323 | 21 | 2 | 18 | 1 | 0 | 19171 | 1 | 1 |
| **AQI** | 16695 | 14 | 7 | 2 | 5 | 0 | 302 | 6 | 1 |
| **tennis** | 60101 | 17 | 10 | 2 | 5 | 0 | 0 | 10 | 0 |
| **homicide** | 195 | 6 | 3 | 1 | 2 | 0 | 0 | 2 | 1 |
| **suicide** | 916 | 8 | 8 | 0 | 0 | 0 | 0 | 0 | 1 |
| **GOOGLE** | 4717 | 6 | 1 | 4 | 1 | 0 | 0 | 0 | 1 |
| **abortion** | 64814 | 19 | 3 | 13 | 3 | 0 | 276577 | 3 | 0 |
| **penguins** | 344 | 18 | 10 | 7 | 2 | 0 | 336 | 8 | 1 |
| **piano** | 917 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| **population** | 234 | 18 | 4 | 3 | 11 | 0 | 0 | 1 | 3 |
| **killers** | 1531 | 6 | 6 | 0 | 0 | 0 | 0 | 1 | 3 |
| **spotify** | 300 | 11 | 6 | 0 | 4 | 1 | 0 | 6 | 0 |
| **travel** | 1303 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| **housing** | 5000 | 7 | 1 | 6 | 0 | 0 | 0 | 0 | 1 |
| **births** | 5496 | 9 | 4 | 2 | 3 | 0 | 0 | 4 | 0 |
| **wdi_wide** | 217 | 16 | 5 | 11 | 0 | 0 | 140 | 4 | 1 |
| **avg_temp** | 465 | 16 | 16 | 0 | 0 | 0 | 2 | 0 | 16 |
| **world_pop** | 216 | 63 | 1 | 42 | 20 | 0 | 42 | 0 | 1 |

**Table A.1:** CSV characteristics of first 25 files

| | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---|---|---|---|---|---|---|
| **books** | 1 | 1 | 0 | 1 | 0 | 1 |
| **GE** | 0 | 1 | 0 | 1 | 0 | 1 |
| **hotels** | 1 | 1 | 1 | 1 | 1 | 1 |
| **hiv** | 0 | 0 | 1 | 1 | 0 | 1 |
| **ukraine** | 1 | 1 | 1 | 1 | 1 | 1 |
| **videos.csv** | 1 | 1 | 1 | 1 | 1 | 1 |
| **admissions** | 1 | 1 | 0 | 0 | 0 | 0 |
| **air** | 1 | 1 | 1 | 0 | 1 | 0 |
| **AQI** | 1 | 1 | 1 | 1 | 0 | 1 |
| **tennis** | 1 | 1 | 1 | 1 | 0 | 1 |
| **homicide** | 0 | 1 | 0 | 1 | 0 | 1 |
| **suicide** | 0 | 0 | 1 | 1 | 0 | 1 |
| **GOOGLE** | 1 | 1 | 1 | 1 | 1 | 0 |
| **abortion** | 1 | 1 | 1 | 1 | 0 | 1 |
| **penguins** | 0 | 0 | 0 | 0 | 0 | 0 |
| **piano** | 1 | 1 | 0 | 1 | 0 | 1 |
| **population** | 1 | 0 | 1 | 1 | 0 | 1 |
| **killers** | 1 | 1 | 1 | 1 | 1 | 1 |
| **spotify** | 0 | 0 | 0 | 0 | 0 | 0 |
| **travel** | 0 | 1 | 0 | 0 | 0 | 0 |
| **housing** | 1 | 1 | 0 | 1 | 0 | 1 |
| **births** | 0 | 1 | 1 | 1 | 0 | 1 |
| **wdi_wide** | 1 | 0 | 1 | 1 | 0 | 1 |
| **avg_temp** | 1 | 1 | 1 | 1 | 1 | 1 |
| **world_pop** | 1 | 0 | 1 | 1 | 0 | 1 |

**Legend: pt1:** Data Manipulation **pt2:** Data Visualization **pt3:** Data Cleaning and Preprocessing **pt4:** Programming Concepts **pt5:** Exploratory Data Analysis **pt6:** Object-Oriented Programming

**Table A.2:** CSV characteristics of first 25 files

| Model | Overall | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---:|---|---|---|---|---|---|---|
| LogisticRegression | 0.888 | 0.777 | 0.900 | 0.900 | 0.907 | 0.920 | 0.923 |
| LinearRegression | 0.868 | 0.810 | 0.947 | 0.877 | 0.913 | 0.810 | 0.853 |
| MultinomialNB | 0.666 | 0.630 | 0.610 | 0.690 | 0.600 | 0.907 | 0.560 |
| GaussianNB | 0.878 | 0.797 | 0.907 | 0.867 | 0.880 | 0.923 | 0.897 |
| KNeighborsClassifier | 0.743 | 0.787 | 0.607 | 0.753 | 0.793 | 0.907 | 0.610 |
| KNeighborsRegressor | 0.747 | 0.780 | 0.620 | 0.787 | 0.797 | 0.860 | 0.637 |
| DecisionTreeClassifier | 0.978 | 0.997 | 0.930 | 0.983 | 0.983 | 0.997 | 0.977 |
| DecisionTreeRegressor | 0.983 | 0.997 | 0.960 | 0.983 | 0.983 | 0.997 | 0.977 |
| RandomForestClassifier | 0.985 | 1.000 | 0.953 | 0.987 | 0.990 | 1.000 | 0.980 |
| RandomForestRegressor | 0.979 | 0.997 | 0.933 | 0.980 | 0.993 | 0.997 | 0.977 |
| SVC | 0.736 | 0.710 | 0.650 | 0.797 | 0.813 | 0.797 | 0.647 |
| SVR | 0.733 | 0.703 | 0.653 | 0.790 | 0.810 | 0.797 | 0.643 |
| CBF | 0.797 | 0.773 | 0.693 | 0.853 | 0.793 | 0.897 | 0.770 |
| CF | 0.880 | 0.860 | 0.873 | 0.890 | 0.927 | 0.870 | 0.860 |

**Table A.3:** Accuracy Scores Individual Prediction Methods

| Name | Overall | pt1 | pt2 | pt3 | pt4 | pt5 | pt6 |
|---:|---|---|---|---|---|---|---|
| Layer 1 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Layer 2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Layer 3 | 0.991 | 1.000 | 0.970 | 0.993 | 1.000 | 1.000 | 0.983 |
| ci0_choice0 | 0.991 | 1.000 | 0.970 | 0.993 | 1.000 | 1.000 | 0.983 |
| ci0_choice1_thr0.5 | 0.983 | 0.997 | 0.957 | 0.980 | 0.990 | 0.997 | 0.977 |
| ci0_choice2 | 0.975 | 1.000 | 0.900 | 0.987 | 0.990 | 0.997 | 0.977 |
| ci0_choice3 | 0.992 | 1.000 | 0.973 | 0.993 | 1.000 | 1.000 | 0.983 |
| ci1_choice0_thr0.2 | 0.991 | 1.000 | 0.970 | 0.993 | 1.000 | 1.000 | 0.983 |
| ci1_choice1_thr0.2 | 0.991 | 1.000 | 0.970 | 0.993 | 1.000 | 1.000 | 0.983 |
| ci1_choice2_thr0.2 | 0.975 | 1.000 | 0.900 | 0.987 | 0.990 | 0.997 | 0.977 |
| ci1_choice3_thr0.2 | 0.992 | 1.000 | 0.973 | 0.993 | 1.000 | 1.000 | 0.983 |

**Table A.4:** Agreement Accuracy Scores

# B | Links pertaining to data, materials, and code

## B.1 | Kaggle datasets

- Travel Company Insurance Prediction → Travel Company New Clients.csv: https://www.kaggle.com/datasets/sellingstories/travel-company-insurance-prediction
- ATP Tennis 2000 - 2023 Daily update → atp_tennis.csv: https://www.kaggle.com/datasets/dissfya/atp-tennis-2000-2023daily-pull
- World Air Quality Index by City and Coordinates → AQI and Lat Long of Countries.csv: https://www.kaggle.com/datasets/adityaramachandran27/world-air-quality-index-by-city-and-coordinates
- US Births by Year, State, and Education Level → us_births_2016_2021.csv: https://www.kaggle.com/datasets/danbraswell/temporary-us-births
- Countries by Intentional Homicide Rate → countries-by-intentional-homicide-rate.csv: https://www.kaggle.com/datasets/bilalwaseer/countries-by-intentional-homicide-rate
- Google Stocks Complete → GOOGLE.csv: https://www.kaggle.com/datasets/bilalwaseer/google-stocks-complete
- Serial Killers Wiki → serial_killers.csv: https://www.kaggle.com/datasets/abdullahsamiir/serial-killers-wiki
- Air Index of world's All Cities 2017 to 2022 → air_index.csv: https://www.kaggle.com/datasets/bilalwaseer/air-index-of-worlds-all-cities-2017-to-2022
- Prison Population in the US → admissions_releases_states.csv: https://www.kaggle.com/datasets/konradb/prison-population-in-the-us
- Abortion Opinions in the General Social Survey → gss_abortion.csv: https://www.kaggle.com/datasets/utkarshx27/abortion-opinions-in-the-general-social-survey
- World's Cities with their Average Temperature → worlds all cities with their avg temp - Sheet1.csv: https://www.kaggle.com/datasets/bilalwaseer/worlds-cities-with-their-average-temperature
- Global Suicide, Mental Health, Substance Use → crude suicide rates.csv: https://www.kaggle.com/datasets/thedevastator/global-suicide-mental-health-substance-use-disor
- World Demographic Indicators Extract → wdi_wide.csv: https://www.kaggle.com/datasets/mathsian/world-demographic-indicators-extract

- World Population Insights: 1970-2022 → population.csv: https://www.kaggle.com/datasets/gyaswanth297/world-population-insights-1970-2022
- World Population Data 1960-2020 → world_pop.csv: https://www.kaggle.com/datasets/utkarshx27/world-population-data-1960-2020
- Housing Dataset of 5000 People Staying in USA → USA_Housing.csv: https://www.kaggle.com/datasets/darshanprabhu09/housing-dataset-of-5000-people-staying-in-usa
- Spotify Top 50 Playlist Songs | @anxods → spotify-streaming-top-50-world.csv: https://www.kaggle.com/datasets/anxods/spotify-top-50-playlist-songs-anxods
- International Piano Competitions → piano_comp.csv: https://www.kaggle.com/datasets/leesstephanie/international-piano-competitions
- Penguin Size, Clutch, and Blood Isotope Dataset → penguins_raw.csv: https://www.kaggle.com/datasets/utkarshx27/penguin-size-clutch-and-blood-isotope-data

## B.2 | Python Programming source books

- Learning Python by doing (2022) by Mark van den Brand, Mauricio Verano Merino, Lina María Ochoa, Mazyar Seraj, Tom Verhoeff and Gijs Walravens
  https://programming-pybook.github.io/introProgramming/intro.html#learning-python-by-doing
- Python Programming for Data Science (2021) by Tomas Beuzen
  https://www.tomasbeuzen.com/python-programming-for-data-science/README.html

## B.3 | Additional datasets

- Diabetes 50-50 split: https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset
- Fruit prices: https://www.kaggle.com/datasets/vstacknocopyright/fruit-and-vegetable-prices

## B.4 | Source code for the project

Dumitru Toader, Maria Emanuela. (2023). Developing a Versatile Hybrid Recommender System: Case Study on Programming Task Prediction. Zenodo. https://doi.org/10.5281/zenodo.8030182

GitHub Repository: https://github.com/emadumitru/FBP_2023_Versatile_Hybrid_RS