Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Course recommender for FMPH CU students

Bachelor Thesis

2023
Patrícia Vnenčáková

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Course recommender for FMPH CU students

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	Mgr. Askar Gafurov, PhD.

Bratislava, 2023
Patrícia Vnenčáková

80103040

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Patrícia Vnenčáková
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Course recommender for FMPH CU students
Systém na odporúčanie predmetov pre študentov FMFI UK

Anotácia: Odporúčacie systémy sú široko obsiahnuté v súčasnom digitálnom svete. Otázka personalizácie obsahu a ponuky si vydobyla aj samostatné svetové konferencie, ako napríklad ACM Conference on Recommender Systems.

Otázka personalizácie sa vyskytuje aj vo vzdelávacom procese. Moderný systém vysokoškolského vzdelávania sa vyznačuje flexibilitou vďaka širokej ponuke povinne-voliteľných a voliteľných predmetov.

Cieľ: Cieľom tejto práce je vytvoriť systém na odporúčanie predmetov pre študentov FMFI UK. Systém bude používať anonymizované dáta z akademického informačného systému AIS2. Okrem úspešnosti odporúčaní sa bude klásť dôraz aj na bezpečnosť osobných údajov študentov.

Vedúci:	Mgr. Askar Gafurov, PhD.
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	prof. RNDr. Martin Škoviera, PhD.
Dátum zadania:	25.10.2022

Dátum schválenia: 31.10.2022

doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

....
študent vedúci práce

80103040

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Patrícia Vnenčáková
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Course recommender for FMPH CU students

Annotation: Recommender systems are widely present in the modern digital world. The task of personalization of content has its own international conferences, such as ACM Conference on Recommender Systems.

Personalization is also present in the education process. The modern system of higher education is flexible thanks to a wide offer of facultative and semi-facultative courses.

Aim: The goal of this thesis is to create a course recommender system for students of FMPH CU. The system will use anonymized data from the academic information system AIS2. Aside from the common success metrics, the thesis should also focus on the security of the personal data of the students.

Supervisor:	Mgr. Askar Gafurov, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	25.10.2022
Approved:	31.10.2022

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.... Student

.... Supervisor

Acknowledgments: The biggest thanks go to my supervisor Mgr. Askar Gafurov, PhD., who came up with the idea for this bachelor thesis. I am grateful for his willingness, time and all the knowledge he has taught me. He was always available to me during the whole process and often sacrificed his free time for me. I would also like to thank RNDr. Jaroslav Janáček, PhD. for providing the data from the information system. Without these data, we could not accomplish the thesis. Last but not least, I would like to thank my family, friends and classmates for supporting me all the time.

Abstrakt

Moderný cielený marketing využíva širokú paletu metód pre určenie čo najvhodnejšej ponuky produktov pre jednotlivých zákazníkov. Tieto metódy sa súhrne označujú ako odporúčacie systémy. Cieľom danej práce je aplikácia týchto metód na odporúčanie predmetov pre vysokoškolských študentov. Implementovali sme odporúčací systém založený na k-NN algoritme s tromi rozličnými metrikami podobnosti. Ďalej sme otestovali náš systém na anonymizovaných dátach študentov FMFI UK za akademické roky 2009/10 až 2021/22. Výsledky ukázali, že najlepšou metódou podobnosti je Jaccardov index. Podstatnou časťou práce bolo takzvané čistenie dát, ako napríklad zlúčenie viacerých verzií predmetov, ktoré však predstavovali ten istý predmet.

Kľúčové slová: odporúčací systém, model, kolaboratívne filtrovanie, metóda podobnosti, evaluátor, metrika presnosti, dáta

Abstract

Modern targeted marketing uses a wide palette of methods to determine the best offer of products to individual customers. Those methods are collectively called recommender systems. The goal of this thesis is an application of such methods in order to recommend courses to university students. In this thesis, we implemented a recommender system based on the k-NN algorithm with three similarity methods. We evaluated our recommender on anonymized data of students of FMPH CU from the academic years 2009/10 to 2021/22. The results showed that the best similarity method is the Jaccard index. A considerable part of this thesis was data cleaning, such as the merging of multiple versions of courses that nevertheless represented the same course.

Keywords: recommender system, model, collaborative filtering, similarity method, evaluator, accuracy metric, data

Contents

List of Figures

List of Tables

Introduction

Recommender systems have emerged as powerful algorithms that provide personalised suggestions to users across a wide range of platforms and applications. Found on platforms such as YouTube, Netflix, Spotify, Amazon, Facebook and Twitter, these systems play a key role in enhancing the user experience and maximising profits for businesses. By capturing and leveraging users' interests, companies strive to implement accurate recommender systems that effectively match users with relevant items.

Recommender systems are complex and sophisticated, using advanced mathematical methods to analyse user behaviour and compute optimal recommendations. These methods generally fall under the umbrella of machine learning algorithms, which enable systems to learn patterns and make predictions based on user data.

The goal of this thesis is to implement a system to recommend faculty courses for the students of FMPH CU. The users' preferences are extracted from the university's internal database of students' transcripts known as AIS2.

This thesis is structured as follows. In the first chapter, we will focus on summarizing the existing knowledge about recommender systems, which will be necessary for the reader to understand the rest of our thesis. The basic concepts from this area will be listed and explained here. In the second chapter, we describe the provided database, which we used in the implementation of our recommender system. In this chapter, we will also explain the steps we took to clean the data from unnecessary records. The third chapter will be used to describe the implementation of our recommender system. We will also define the basic concepts from the first chapter in relation to our recommender system. The fourth and also final chapter of this thesis covers our findings and results. We compared several implemented similarity methods, and evaluated them using the implemented evaluator. From the compared methods we then selected the one that showed the best results. In the last stages of this thesis, we let the recommender system, which used the winning method, recommend us the courses for our personal data and we compared the prediction with reality.

Chapter 1

Recommender systems

The main goal of this chapter is to familiarise the reader with the general concepts related to recommender systems. Then we will explain the k-NN algorithm and the basic models on which many of today's recommender systems are built. Finally, we will discuss the aspects of the model evaluation and describe several success metrics. The main source of information for this chapter is the book *Recommender Systems* by Charu C. Aggarwal [1].

1.1 Basic concepts

A **recommender system** is a type of software system that provides personalised recommendations of different items to users based on their past behaviour, preferences and interests. The **user** is an entity to which the recommendation is provided and **item** is a product being recommended. These systems are commonly used in e-commerce websites, online streaming platforms, social media networks and other digital applications. They can be based on different types of algorithms, such as collaborative filtering, content-based filtering, or hybrid approaches, and they can use different data sources. We will discuss these algorithms in more detail in section 1.3.

The key concept in this area is the **rating**. It is a method of expressing user feedback (numerical or verbal) on a given item. Part of the job of a recommender system is a loop of collecting and predicting users' ratings. A unit of feedback, i.e. a single rating of a single item j by a single user u is called **entry**. We can represent the total collected entries from all users as a single **rating matrix**, where rows correspond to individual users, columns correspond to individual items, and the cells correspond to the rating of the given item by the given user. A sample example is illustrated in Table 1.1.

In order to predict user ratings, a recommender system must have a set of items to train on. Such a set is called the **training set** and it is a selected part of the rating

	$Item_1$	$Item_2$	$\cdots$	$Item_{n-1}$	$Item_n$
$User_1$	1	-	$\cdots$	4	5
$User_2$	-	4	$\cdots$	-	2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$User_{m-1}$	2	1	$\cdots$	-	2
$User_m$	3	-	$\cdots$	2	-

Table 1.1: $m \times n$ rating matrix with ratings from 1 to 5 ('-' symbol stands for an ungiven rating)

matrix. The part of the rating matrix that we have not chosen as the training set is called the **test set**. On the test set, we evaluate the quality of the recommender system. We will discuss the evaluation of the recommender system later in section 1.4. From the machine learning's point of view, the task of rating prediction can be viewed as either regression or classification problem. A **regression problem** is a type of problem where the goal is to predict a concrete numerical value. A **classification problem** is a problem that attempts to predict to which category or class a given observation belongs.

1.2 Determination of similarity

One of the most important tasks in recommender systems is to find similar items/users and to determine how similar the items/users are. In this section, we will first describe the methods used to determine the level of similarity and then the k-Nearest Neighbours algorithm. These components are described in more detail in the book by Oliver Kramer [6], which was also the source for writing this section.

1.2.1 Methods for determining similarity

In this subsection, we describe several methods for determining the similarity between two data points, which are used in various fields such as machine learning, natural language processing, and data mining. The goal of these methods is to measure the degree of closeness between objects based on their common features or attributes. The measure of similarity can then be used for a variety of purposes such as regression, classification, or recommendation. We will cover some popular methods such as Cosine similarity, Hamming Distance, Intersection from the set theory and Jaccard Index.

Cosine similarity - It is a popular method for measuring the similarity between two vectors in high-dimensional space [9]. Cosine similarity measures the cosine of the angle between two vectors and ranges from -1 to 1, where 1 means the two vectors are identical, 0 means they are orthogonal, and -1 means they are diametrically opposed. Let $a = (a_1, a_2, \ldots, a_n)$, $b = (b_1, b_2, \ldots, b_n)$ be two n-dimensional vectors. Mathematically, the cosine similarity between vectors a and b is given by:

$$cos(a, b) = \frac{a \cdot b}{\|a\| \cdot \|b\|} = \frac{a_1 b_1 + ... + a_n b_n}{\sqrt{a_1^2 + ... + a_n^2}\sqrt{b_1^2 + ... + b_n^2}}$$

Hamming Distance - This method is used when comparing the difference between two strings of the same size and is defined as an integer that indicates the number of positions where the given two strings differ. In other words, the fewer positions in which two strings differ, the more similar they are. For example, consider the strings "000111" and "011001". The Hamming distance between them is 4 because there are four positions where the symbols differ: at positions 2, 3, 4, and 5. The Hamming distance can also be applied to vectors, which are ordered sets of numbers. In this case, the Hamming distance measures the number of positions at which the corresponding elements in two vectors are different. Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ be two n - dimensional vectors [7, 3]. Mathematically, the Hamming distance between vectors a and b is given by:

$$d(a, b) = |\{i \in \{1, \ldots, n\} : a_i \neq b_i\}|$$

Intersection - This method is used to determine the similarity between two sets and determines the number of common elements that are found in both sets. The more elements two sets have in common, the more similar they are. It is a classical operation from set theory, which is denoted as $A \cap B$. Mathematically, the intersection between the sets A and B is defined as:

$$I(A, B) = A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$

We can apply this concept to compute the Intersection between the two vectors by treating them as sets of elements.

Jaccard Index - This method, also called *Jaccard similarity coefficient*, is used to compare the similarity of two data sets. It is defined as the ratio of the size of the intersection of the given sets to the size of their union. The resulting value ranges from 0 to 1, where 0 indicates that the sets have no common elements and 1 indicates that the sets are identical [2]. Mathematically, the Jaccard index between sets A and B is given by:

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

We can apply this concept to compute the Jaccard index between the two vectors by treating them as sets of elements.

A sample and comparison of the above-described methods are in Table 1.2.

Item-Id → User-Id (i) ↓	1	2	3	4	5	*cos(i,3)*	*d(i, 3)*	*I(i, 3)*	*J(i, 3)*
1	1	-	3	4	5	0.867	2	2	0.6
2	-	4	3	2	1	0.861	4	0	0.4
3	1	2	5	4	2	1	0	5	1
4	-	4	3	-	1	0.853	3	0	0

Table 1.2: Calculating the similarity between user 3 and other users using selected methods (Cosine Similarity, Hamming Distance, Intersection and Jaccard Index)

1.2.2 k-Nearest Neighbours algorithm

The k-Nearest Neighbors (k-NN) algorithm is a type of machine learning algorithm that can be used for both classification and regression problems. It is a non-parametric algorithm, which means it does not make any assumptions about the underlying data distribution. The k-NN algorithm learns from the data itself by finding the k closest training examples in the feature space to a new data point and using those examples to make a prediction. When using the k-NN algorithm, we need to choose some suitable method to determine the distance between two data points. We have described some of these methods above in subsection 1.2.1.

In k-NN classification, the output is a class membership. The k-NN algorithm uses the majority class of the k nearest neighbours to predict the class label of a new data point. In k-NN regression, the output is the property value for the object. The algorithm computes the average (or median) of the k nearest neighbours' values to predict the value of a new data point. This is basically how the k-NN algorithm works. It looks at the k closest data points to the new data point, compares them to each other, and uses the majority vote to classify the new data point.

The fact that k-NN makes no assumptions about the distribution of the data is also its main advantage, as this makes it applicable to a wide range of applications in different domains, including recommender systems where the data may have complex distributions. Its other advantage is that it behaves in a very intuitive way, which makes it very easy to understand and also easy to implement. In addition, it does not require any training time as it simply stores the trained data and makes predictions based on it. However, k-NN also has its disadvantages. The biggest is that it can be too expensive to compute for very large and extensive datasets, and it is very sensitive to the choice of either a different size of k or a different method for determining the distance between two data points.

1.3 Models

In this section, we provide an overview of the different types of models and algorithms used in recommender systems. The goal of this section is to help the reader understand the different approaches that can be used to make recommendations and to provide a clear explanation of the strengths and weaknesses of each approach. We cover two main types of models: collaborative filtering and content-based filtering. We also discuss hybrid models that combine elements of both collaborative filtering and content-based filtering.

1.3.1 Collaborative filtering models

Collaborative filtering is one of the most popular and effective techniques used in recommender systems, which predicts a user's interests (filtering) by collecting and analysing the preferences or behaviours of many other users (collaboration). The algorithms used in such models are based on the fact that similar users have similar patterns of rating behaviour and similar items receive similar ratings. In collaborative filtering, the first step is to create a user-item rating matrix, and the algorithm then identifies similar users or items based on the similarity of the users' interaction patterns using different methods for determining similarity. There are two major types of collaborative filtering algorithms:

The first type is called **user-based** collaborative filtering. In this case, the ratings provided by similar users to a target user U are used to make recommendations for U. The predicted ratings of U are computed as the weighted average values of these "peer group" ratings for each item. For example, consider a situation where user U_1, user U_2 and user U_3 have similar ratings on selected items. So we assume that in the future these three users will give similar ratings to the items as they do now. U_1 has not yet given a rating for item I, while U_2 and U_3 have given ratings for I. To predict U_1's rating for item I, we use the average of U_2's and U_3's ratings (based on the fact that users U_1, U_2 and U_3 are similar). This example can be seen in Figure 1.1.

The second type is called **item-based** collaborative filtering. In order to make the rating predictions for target item I by user U, the first step is to determine a set S of items that are most similar to target item I. The ratings in the item set S, which are specified by U, are used to predict whether the user U will like item I. For example, consider the situation where we want to predict the rating of item I_1 for user U. Items I_2, I_3 and I_4 are considered similar to item I_1 and therefore form the aforementioned set S. User U has only rated items I_2 and I_3 from the set S. So U's rating for item I_1 is calculated from I_2 and I_3 by averaging them. This example can be seen in Figure 1.2.

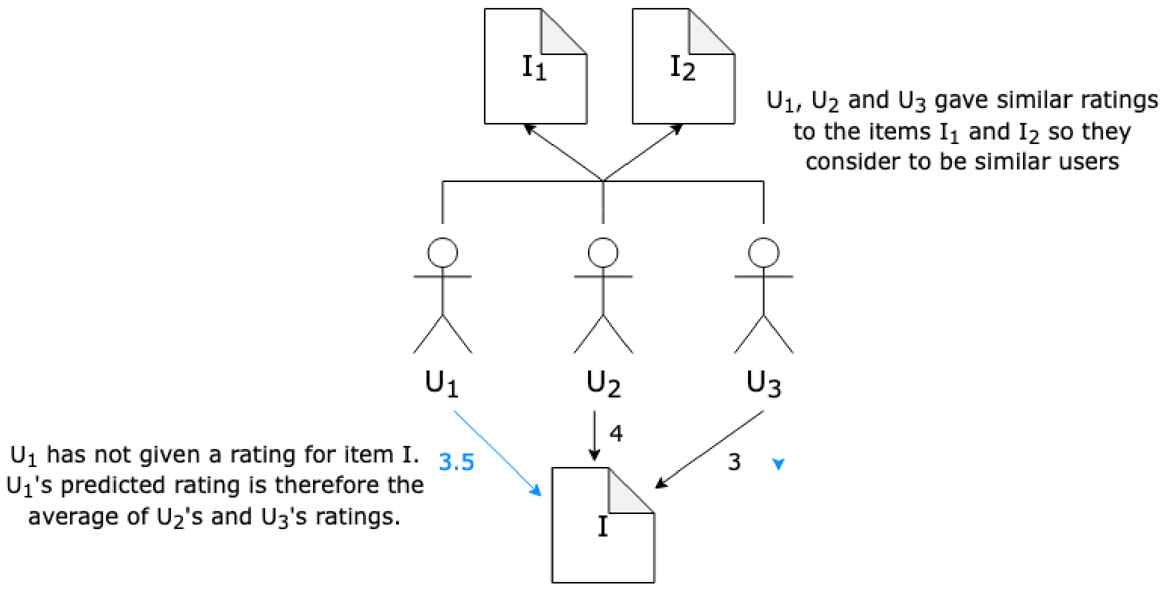

I1
I2
U1, U2 and U3 gave similar ratings to the items I1 and I2 so they consider to be similar users
U1
U2
U3
U1 has not given a rating for item I. U1's predicted rating is therefore the average of U2's and U3's ratings.
3.5
4
3
I

Figure 1.1: Example of user-based collaborative filtering

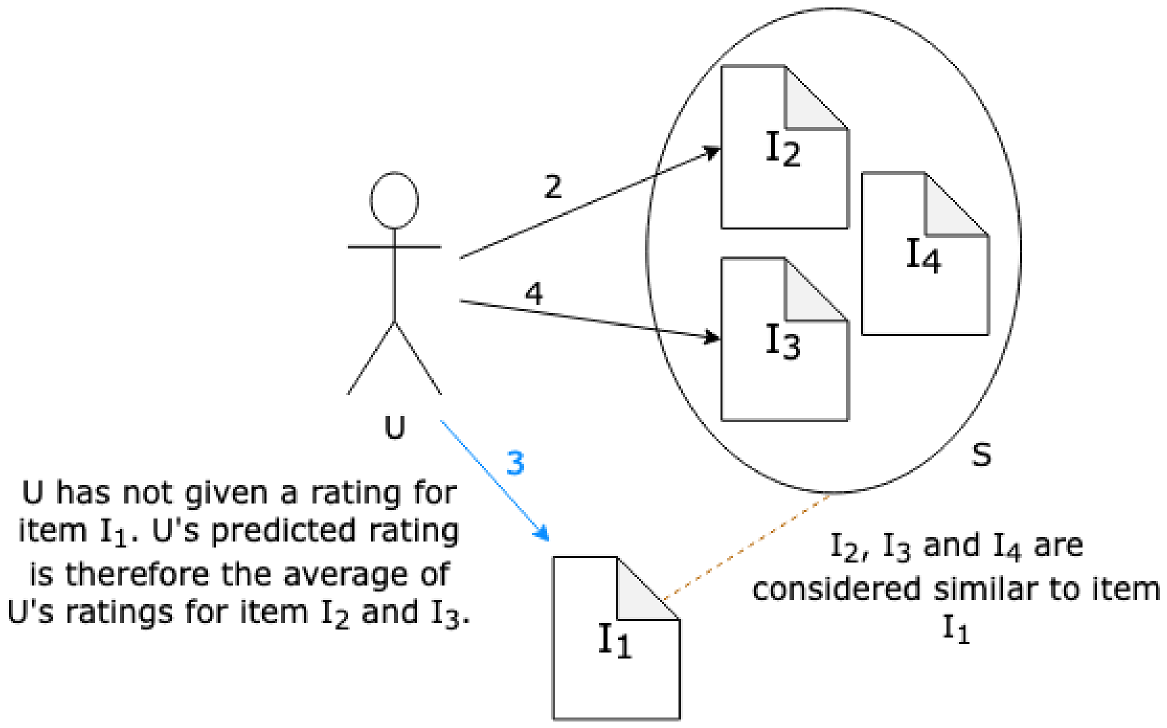

I2
I4
I3
2
4
U
3
S
U has not given a rating for item I1. U's predicted rating is therefore the average of U's ratings for item I2 and I3.
I1
I2, I3 and I4 are considered similar to item I1

Figure 1.2: Example of item-based collaborative filtering

One of the main advantages of collaborative filtering is that they do not require explicit knowledge of item characteristics or user preferences. In addition, such models behave very intuitively, are easy to understand, are not difficult to implement, and it is straightforward to justify why we recommended the items we did. However, they also have their disadvantages. On one hand, if the amount of gathered data is small, a model struggles with making correct predictions (this is called the *cold start* problem in the literature). On the other hand, a large dataset often requires a lot of computational power to produce a prediction.

1.3.2 Content-based filtering models

Other widely used recommender system models are content-based models. These models use a more detailed description of items, such as features, attributes and specifications, to recommend items to users. The main idea of these models is that items with the same description will be attractive to users with the same tastes. The description of items is called *content*. Content-based recommenders rely on a user's past interactions with items to determine their preferences, and then identify and recommend other items that share similar features or attributes. For example, consider a situation where user U has rated the film M_1 highly and we have no information about the ratings of other users. In this situation, we have no way of using collaborative filtering. However, we do know the genre of M_1, we also know that films M_2 and M_3 are of the same genre and that movie M_4 is of a different genre. When we recommend movies to user U to watch next, we will recommend movies M_2 and M_3 to him because U has rated a movie of the same genre highly, and therefore we assume that he would also like M_2 and M_3. This example can be seen in Figure 1.3.

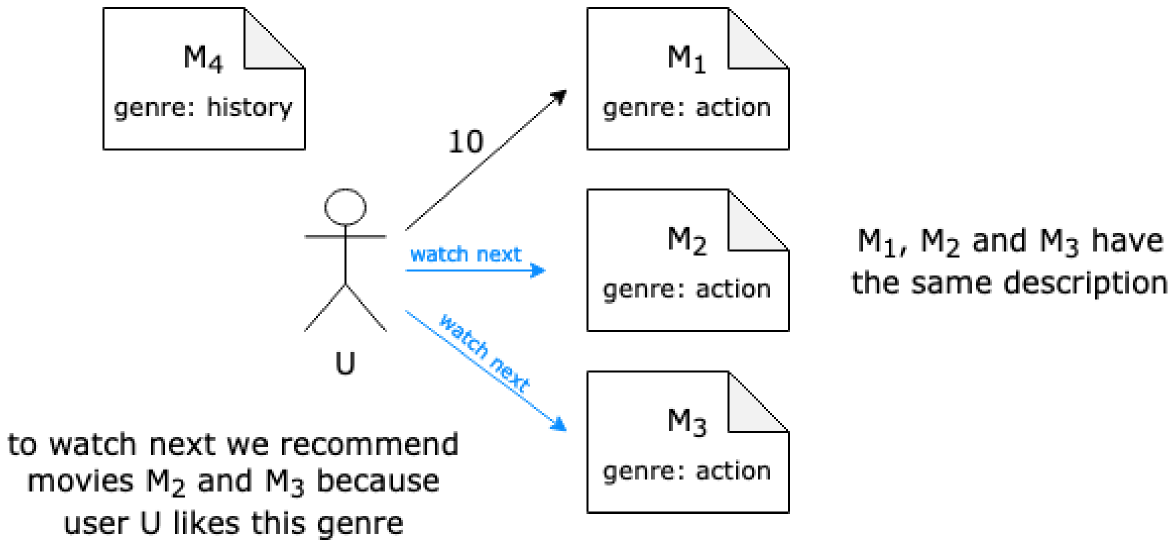

M4
genre: history
M1
genre: action
10
M2
genre: action
watch next
U
watch next
M3
genre: action
M1, M2 and M3 have the same description
to watch next we recommend movies M2 and M3 because user U likes this genre

Figure 1.3: Example of content-based filtering

Content-based recommendation models have the advantage of being able to recommend items to users without a comprehensive user-item rating matrix. The recommendation is based solely on the description of the items rated by the user to whom we are making the recommendation. This approach can help overcome the *cold start* problem, as recommendations can be made even when there is limited interaction data available. Additionally, it is easy to justify why certain items were recommended.

However, content-based models may not be as effective at making recommendations for completely new users without any interaction data. This is because the model used to train the recommendation system requires the user's past ratings to accurately predict future preferences. Furthermore, a considerable amount of ratings must be available for the target user to ensure reliable and accurate predictions although this number may not be very large.

1.3.3 Hybrid models

Hybrid recommender system models combine multiple recommendation techniques. One of the most common combinations is the combination of collaborative filtering with content-based models. Hybrid models have several advantages over single techniques. They can provide more accurate recommendations by using multiple recommendation techniques and can handle a wider range of recommendation scenarios. Hybrid models can also overcome the limitations of individual techniques, for example, they can overcome the *cold start* problem where there is little or no historical interaction data available for new items or users. On the other hand, hybrid models can have some drawbacks due to their higher complexity and computational cost, as well as their need for more resources for training and implementation compared to single techniques. These factors can increase the difficulty of implementing and maintaining hybrid models in certain situations.

1.4 Evaluation of success

Evaluation of success is a critical step in the development and deployment of recommender systems, as the quality of the system can have a significant impact on user satisfaction and belief in the system. To ensure that users are satisfied with the recommender system and use it frequently, it is essential to maximise accuracy and minimise errors. The evaluation of recommender systems involves comparing the predicted recommendations to the actual user behavior. This is typically done by dividing the user data into a training set and a test set. In this section, we will introduce and explain the main goals of the evaluation and describe its possible outcomes. In addition, we will list the metrics used to evaluate the accuracy of the system and provide the corresponding mathematical formulas for these metrics.

1.4.1 Goals of an Evaluation

When developing a recommender system, it is important to prioritise certain qualities to ensure that the system is effective. The main goal of implementing the system is to achieve these qualities, and the evaluation process is critical in continuously improving them. As mentioned above, the main goal of recommender systems is to recommend items as accurately as possible. Therefore, improving **accuracy** is one of the primary goals of the evaluation process of recommender systems. A high level of accuracy is essential for the success of a recommender system, as inaccurate or irrelevant recommendations can lead to user dissatisfaction and reduced engagement with the system.

Another important quality that recommender systems should consider is **coverage**. The aim of coverage is to ensure that the system recommends a diverse and representative set of items, rather than just a few popular ones. It should not exclude anything or anyone from the recommendation process. A high level of coverage is important because it allows users to discover new things they may not have known about, and it can improve overall user satisfaction with the system.

Novelty and **serendipity** are also important aspects that should be considered when developing and evaluating a recommender system. Novelty refers to the degree to which the recommended items are new to the user. Recommending only popular or already-known items might lead to boredom or frustration, while recommending new items can enhance user experience and engagement with the system. On the other hand, serendipity refers to the level of surprise in successful recommendations. Serendipitous recommendations can introduce users to exciting items that they may not have discovered otherwise, leading to a positive user experience and increased satisfaction with the system.

Closely related to novelty and serendipity is the **diversity** of recommender sytem. Diversity refers to the degree to which the recommended items cover a broad range of categories, genres, and attributes, rather than being limited to a few popular or similar items. A lack of diversity in recommendations can lead to user dissatisfaction and limited engagement with the system. For example, if a user frequently watches action movies, a recommender system that only recommends action movies may limit their exposure to other genres they may also enjoy, such as sci-fi or drama. The more diverse the items we recommend to the user, the better the chance that the user will choose one of the recommended items.

Last but not least, **trust** and **confidence** are important goals in the evaluation of recommender systems, as they play a crucial role in the acceptance of recommendations by users. If users do not trust the recommendations, they are unlikely to use or follow them. Similarly, if users do not trust the system, they may be unwilling to provide personal information or engage with the system at all. To evaluate trust and confidence in a recommender system, we can use various methods such as user questionnaires or monitoring user engagement data. If we find that users do not trust the recommender system over time, it is important to modify and improve it to restore trust and confidence in the system.

1.4.2 Accuracy metrics

When evaluating recommender systems, accuracy metrics are used to measure how well and how accurately the system recommended the items that users have actually rated or interacted with. The evaluation is performed on a test set. For each real user-item entry specified in the test set, the prediction of that entry is computed using the specific recommender system we want to evaluate. Then, using the chosen metric, the difference between the real user-item entry value and the prediction value is computed. In this section, we will explore some commonly used accuracy metrics for evaluating recommender systems.

Root Mean Squared Error (*RMSE*) - This metric is commonly used to assess the accuracy of a recommender system. It measures the difference between the predicted rating and the actual rating entered by the user. The *RMSE* ranges from 0 to infinity, depending on the range of data analysed. For example, if the ratings range from 0 to 100, the *RMSE* can range from 0 to 100. In general, the lower the *RMSE*, the more accurate the predictions made by the recommender. The *RMSE* is calculated by taking the square root of the average of the squared differences between the predicted and actual ratings. Let's denote E as a set of entries in the test set used for evaluation, (u, j) as a user-item index pair, corresponding to a position in the rating matrix, r_{uj} as a value of the actual rating of entry $(u, j) \in E$, r'_{uj} as a predicted rating of the entry $(u, j) \in E$ and $e_{uj} = r'_{uj} - r_{uj}$ as an entry-specific error [1]. The *RMSE* can then be expressed mathematically as:

$$RMSE = \sqrt{\frac{1}{|E|} \sum_{(u,j) \in E} e_{uj}^2}$$

F1 score - The *F1* score is an accuracy metric commonly used in binary classification tasks that combines **precision** and **recall** into a single value. Precision measures the proportion of true positive predictions out of all positive predictions made by the recommender system. In simple terms, it measures how accurate the system is at making positive predictions. Mathematically we can express precision as:

$$precision = \frac{true\ positives}{(true\ positives + false\ positives)}$$

Recall measures the proportion of true positive predictions made by the recommender system out of all actual positive cases in the dataset. In other words, it quantifies the system's ability to correctly identify all relevant elements in the data. Mathematically we can express recall as:

$$recall = \frac{true\ positives}{(true\ positives + false\ negatives)}$$

The equations use the terms *true positives*, *false negatives* and *false positives*. *True positives* represent the number of data points that are truly positive and have been correctly identified as positive by the system. On the other hand, *false negatives* refer to positive data points that were incorrectly classified as negative, while *false positives* refer to negative data points that were incorrectly classified as positive. Precision and recall are shown in Figure 1.4 for better understanding.

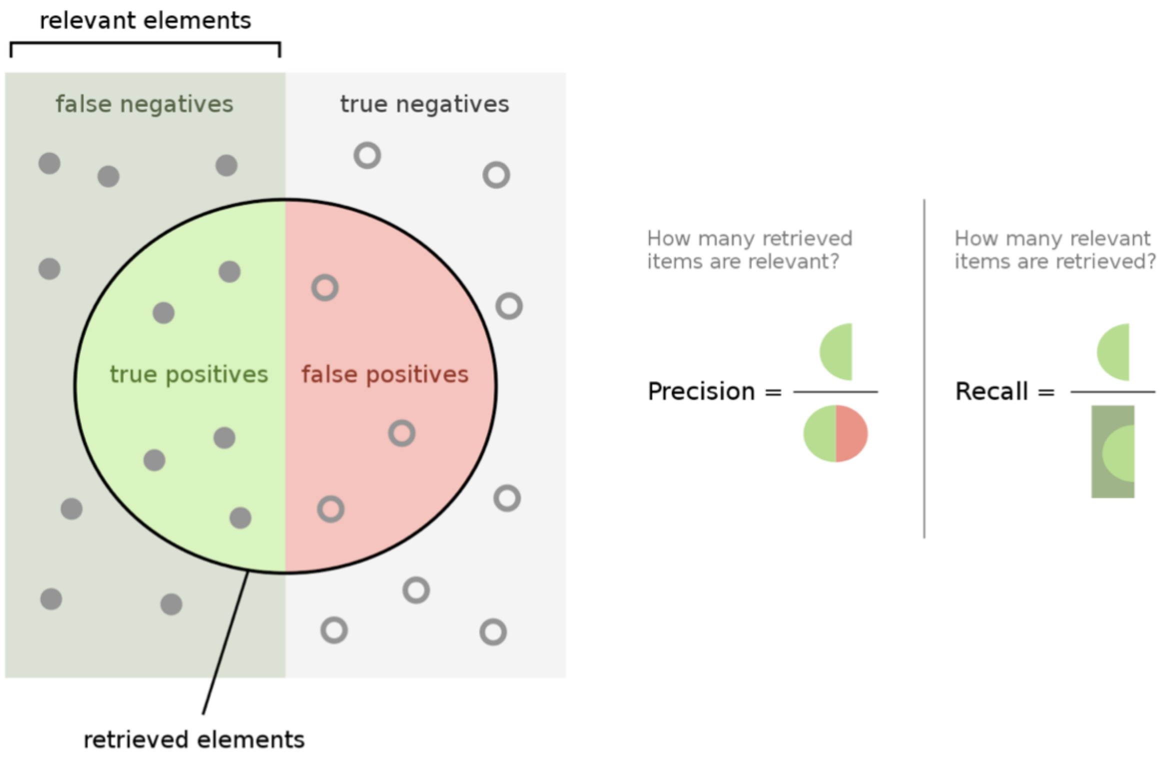


Figure 1.4: Precision and recall
Source: Wikipedia [8]

In the context of recommender systems, the *F1* score can be used when the problem is formulated as a binary classification task, such as predicting whether a user will like or dislike a particular item. The F1 score ranges from 0 to 1, where 1 is the best possible value, indicating perfect precision and recall, and 0 is the worst possible value, indicating that the model has not made any correct predictions. A higher F1 score indicates better performance of the recommender system [5]. Mathematically we can express the *F1* score as:

$$F1 = \frac{2 \cdot precision \cdot recall}{(precision + recall)}$$

Chapter 2

Data

In this chapter, our main goal is to introduce the data we used to make predictions. We will start with a description of the database. Following this, we provide basic statistics about the data. Finally, we describe the steps we took to eliminate redundant records.

2.1 Database description

In our work, we used anonymized student data available from the university's AIS system. In order to be able to work with this data, we first had to receive training on data protection according to the General Data Protection Regulation (GDPR) [4]. The data include information about Bachelor's and Master's students. Our database does not include PhD students. All provided data we used is from the academic year 2009/10 to the academic year 2021/22.

Specifically, for each student, we know his study programme, which courses he enrolled in each semester and if he passed the course, i.e. obtained at least the grade E. No other metadata about the students were provided (i.e. no name, age, sex, grade, etc.). For each course, we were provided with its full official name, official course code, a shortcut of the official course code and the department that ensures the teaching of this course. Additionally, for courses with at least 10 students enrolled in a particular academic year, we were provided with the exact total grade counts. We were also provided with information about the study programmes of our faculty such as the official name of the programme and its official shortcut. The exact description of the provided data are available in the supplementary section *Appendix A: Database description.* Table 2.1 shows the basic statistics of the provided dataset. Figure 2.1 shows a plot which indicates how many students were enrolled in at least one course per academic year from 2009/10 to 2021/22. As you can see, the number of students is decreasing while the number of different courses is increasing.

total number of enrollment records	**281255**
total number of enrolled students	**6161**
number of different programmes	**148**
number of different courses	**4688**

Table 2.1: Basic statistics about provided database

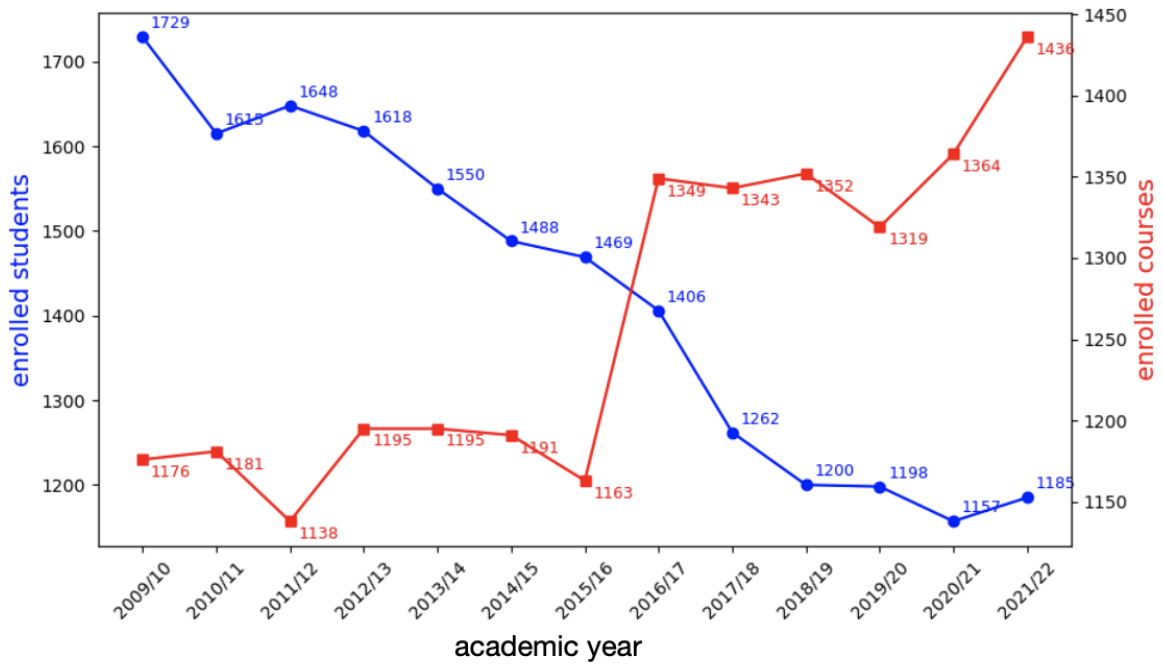


Figure 2.1: A plot showing how many students (excluding doctoral) enrolled in at least one course each academic year (blue line with round dots) and how many courses had at least one enrolled student (red line with square dots). The horizontal axis represents a particular academic year. The left vertical axis represents the number of enrolled students in a particular academic year, and the right vertical axis represents the number of courses that have at least one enrolled student that year.

2.2 Data preparation

Given the size of the provided database, it is expected to have duplicates and irrelevant records. As we strive to ensure that our recommender system provides highly accurate recommendations, it is necessary to take multiple steps to eliminate redundant data from the database. In this section, we outline the data-cleaning process.

The problem that we consider to be the most important is the **merging of courses that are considered to be identical**. An effective way of merging identical courses is to examine their official codes. The official course code format provided by our university is as follows: ***department/shortcut/year*** where ***department*** is the department that ensures the teaching of a given course, ***shortcut*** is an official shortcut of given course and ***year*** represents the year of changing the version of the course. The university stores all versions of the courses in the database and therefore merging is necessary. It is possible for two courses to have the same official shortcut, but they are not identical. Therefore, it is not possible to merge courses only according to the same official shortcut. In Table 2.2 we present a concrete example from the database[1].

course code	**shortcut**	**official name**
FMFI.KDMFI/1-UIN-350/15	1-UIN-350	Programming in C#
FMFI.KZVI/1-UIN-350/00	1-UIN-350	Programming in C++

Table 2.2: Example of non-identical courses

We merged the courses according to the following criterion: courses that have the same official shortcut and at the same time are identical in their official name are considered identical. In Table 2.3 we present a concrete example from the database[1].

course code	**shortcut**	**official name**
FMFI.KMANM/1-FYZ-120/17	1-FYZ-120	Mathematics (1)
FMFI.KMANM/1-FYZ-120/15	1-FYZ-120	Mathematics (1)
FMFI.KMANM/1-FYZ-120/00	1-FYZ-120	Mathematics (1)

Table 2.3: Example of identical courses

Courses that meet the above-described criterion are merged into one, and the one is that course, which is the most up-to-date, which means that it has the highest number in the part ***year***.

The next step in the data-cleaning process was to **remove redundant records from the database**. First of all, we removed *doctoral courses* because there are no records of doctoral students in the enrollment database. Doctoral courses can be easily detected by their official shortcut because their shortcut starts with the substring '3-'. In Table 2.4 we present a concrete example from the database[1].

course code	**shortcut**	**official name**
FMFI.KI/3-INF-803/15	3-INF-803	Pedagogical practice
FMFI.KI/3-INF-601/15	3-INF-601	Workplace seminar
FMFI.KI/3-INF-403/15	3-INF-403	Scientific activity III
FMFI.KI/3-INF-120/15	3-INF-120	Literature study

Table 2.4: Example of doctoral courses

Next, we removed *courses enrolled less than twice* in the entire period from the academic year 2009/10 to 2021/22. These courses are considered uninteresting and will no longer be recommended to students. In Table 2.5 we present a concrete example from the database[1].

course code	**shortcut**	**official name**
RKCMBF.CD.BA/K-KT61-114/20	K-KT61-114	Religious history - modern times
FiF.KAA/A-buAN-430/20	A-buAN-430	History of comics in the USA
FM.KIS/370B/19	370B	Chinese for business I

Table 2.5: Example of courses enrolled less than twice

The last step in the data-cleaning process was to remove *non-attended study programmes*. The database contains all offered study programmes and therefore also includes programmes for which no one has applied for the whole period from the academic year 2009/10 to 2021/22. In Table 2.6 we present a concrete example from the database[2].

skratkasp	**sp**
pUFY	additional pedagogical study of physics
pUXX	basic module of additional pedagogical studies
MA,EF	mathematics, specialisation: economic and financial mathematics

Table 2.6: Example of non-attended study programmes

After cleaning the data, the total number of courses and study programmes has changed significantly. Table 2.7 shows the statistics we got after the data-cleaning process.

number of courses before merging identical	4688
number of courses after merging identical	**4349**
number of doctoral courses	1032
number of courses enrolled less than twice	1641
total number of redundant courses	1649
total number of courses after removing redundant	**2700**
number of study programmes before removing non-attended	148
number of study programmes after removing non-attended	**68**

Table 2.7: Statistic after data-cleaning process

[1]The columns are extracted from the table *predmet* with appropriate English translation.
[2]The columns are extracted from the table *studprog* with appropriate English translation.

Chapter 3

Implementation

In this chapter, we describe the proposed recommenders and the technical details of their implementation and evaluation. We explain which model we have chosen when implementing the systems. Introduce basic concepts for our model such as who are the users, what are the recommended items and the rating matrix. We also present the similarity metrics we implemented and the accuracy metric we used to evaluate the systems. For development, we utilized `Python` as the primary programming language due to its flexibility and ease of use. In managing the database, we used `SQLite` for its lightweight and efficient storage capabilities.

3.1 Model selection

In the context of our course recommender system based on a binary rating matrix[1] of student enrollments, the utilization of content-based is not feasible. This limitation arises from the lack of detailed descriptions of the courses in our dataset. The absence of detailed information about the courses hinders our ability to identify relevant attributes or keywords that can effectively capture the preferences and interests of the students. Furthermore, hybrid models, which combine multiple recommendation techniques, including content-based filtering, are also impractical in this scenario. Without sufficient course descriptions, the content-based component of the hybrid models would lack the necessary data to contribute meaningfully to the recommendation process.

Considering the limitations of content-based and hybrid models in our course recommender system, we have chosen user-based recommender systems as the preferred approach. This decision is driven by the need for a more meaningful recommendation strategy that takes into account the focus on student enrollments. In our binary rating matrix of student enrollments, where the presence or absence of enrollments serves as an indicator of user preferences, user-based collaborative filtering proves to

[1]Described in more detail in section 3.2.

be well-suited.

Unlike item-based collaborative filtering, which relies solely on course similarities, user-based collaborative filtering offers the advantage of leveraging relatable students for making predictions. By considering the enrollment patterns of similar students, we can generate recommendations that align with individual preferences. This user-based approach also enhances transparency and understanding, as we can provide explanations for recommendations based on the behaviors and preferences of similar students.

User-based collaborative filtering has the potential to enhance the educational experience by providing recommendations that embrace *novelty*, *serendipity* and *diversity*. Students have diverse interests and can benefit from exploring a wide range of fields or courses. Through user-based collaborative filtering, we can identify similar students who have enrolled in different sets of courses. This approach allows us to recommend courses that not only match their existing preferences, but also expose them to a variety of options, promoting novelty and serendipity. This approach leverages the enrollment patterns of these like-minded students, allowing us to generate personalised recommendations that are both meaningful and diverse. By adopting this methodology, we ensure that students have access to a wide range of courses, enabling them to discover new areas of interest and embark on a more engaging and personalised educational journey.

3.2 Basic components of selected model

In this section, we introduce the basic concepts from section 1.1, we also list the methods we used to determine the similarity between users and also the metrics we used to implement the evaluator.

In the context of our recommender system, the recommended *items* are courses that students can enroll in. The *users* are the students, but one student is considered to be a different user in each academic year of his/her studies. We use binary *ratings* that represent the student's enrollment in the course (1 = student enrolled in the course, 0 = student did not enroll in the course). In our *rating matrix*, the columns represent the courses IDs, and the rows represent the students. As mentioned above, each student occupies multiple rows in this matrix corresponding to the number of years they have studied at the faculty. As shown in Figure 2.1, the matrix consists of 18525 rows. Additionally, Table 2.7 indicates that the matrix consists of 2814 columns. The matrix is filled in the following way: if user i has enrolled in course j in a given year, then the *entry* $(i, j) = 1$, otherwise $(i, j) = 0$. Our rating matrix, therefore, looks like the rating matrix shown in Table 3.1 (the entries are indicative only).

Course ID → **User ID ↓**	1	2	3	4	⋯	2812	2813	2814
1	1	0	0	1	⋯	1	0	1
2	0	1	1	0	⋯	0	0	0
3	1	0	1	0	⋯	0	0	1
⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
18523	1	1	1	1	⋯	0	0	0
18524	0	0	1	1	⋯	0	0	1
18525	1	0	1	1	⋯	1	1	0

Table 3.1: Our rating matrix

The rating matrix is used to search for similar users. To forecast courses for the next academic year, we require another matrix with the same dimensions as our rating matrix. Let's call this new matrix a *prediction matrix.* In prediction matrix, the users and items are the same as in the rating matrix. However, the prediction matrix provides information about the courses in which a particular user has enrolled for the upcoming academic year. For example, consider a situation where a student has studied at a faculty for three years from academic year 2019/20 to 2021/22. As a first-year student he was given a User ID 1200 and he enrolled in courses 1 and 2. As a second-year student he was given a User ID 1201 and he enrolled in courses 3 and 4. And as a third-year student he was given a User ID 1202 and he enrolled in courses 5 and 6. Table 3.2 shows the part of the rating matrix and Table 3.3 shows the part of the prediction matrix that corresponds to this example.

Course ID → **User ID ↓**	1	2	3	4	5	6	⋯	2814
1200	0	0	0	0	0	0	⋯	0
1201	1	1	0	0	0	0	⋯	0
1202	1	1	1	1	0	0	⋯	0

Table 3.2: Example of rating matrix

Since both of these matrices are extensive, for easier data manipulation we decided to represent these matrices internally in a different way by combining both matrices into one matrix. The matrix we have constructed consists of five columns. The User ID is positioned as the first column, which corresponds to the rows in both the rating and prediction matrices. The second column contains an array of the courses that the student has enrolled in so far. In the third column we observe an array capturing the courses taken in a given academic year. The fourth column denotes the academic year

Course ID → **User ID ↓**	1	2	3	4	5	6	⋯	2814
1200	1	1	0	0	0	0	⋯	0
1201	0	0	1	1	0	0	⋯	0
1202	0	0	0	0	1	1	⋯	0

Table 3.3: Example of prediction matrix

of enrolment, while the last column denotes the student's study programme. Table 3.4 shows the internal representation of the data, i.e. the union of the rating and prediction matrices from Table 3.2 and Table 3.3.

User ID	**so far**	**this year**	**academic year**	**study programme**
1200	`[]`	`[1, 2]`	2019/20	INF
1201	`[1, 2]`	`[3, 4]`	2020/21	INF
1202	`[1, 2, 3, 4]`	`[5, 6]`	2021/22	INF

Table 3.4: Internal representation of the data

We further divided the modified data into *test set* and *training set* according to the fourth column, i.e. according to academic years. For example, the test set contains records from the 2020/21 and 2021/22 academic years, and the training set contains records from all other years.

To detect similar users, we implemented the *k-NN* algorithm (see section 1.2.2) with three similarity methods, namely: **Intersection**, **Hamming distance** and **Jaccard index** (see section 1.2.1). To evaluate these methods, we implemented an evaluator that uses the **RMSE** and **F1 score** (see section 1.4.2). In our implementation, the F1 score requires transforming the prediction into a 0-1 vector by selecting a *threshold.* We set this threshold to 30%, which means that courses that have a probability of being enrolled at least 30% are assigned 1, other courses are assigned 0. In chapter 4, we conduct a comparative analysis of the implemented methods using the evaluator, evaluating their performance and effectiveness in relation to each other.

Chapter 4

Results

In this chapter, we present the results obtained in our thesis. We extensively examined and analyzed the behaviour of the implemented similarity methods across various experimental conditions. Specifically, we investigated the impact of different values of k in the k-NN algorithm, variations in the composition of training and test sets, and the effects of subsampling the data. For the evaluation, we used our implemented evaluator.

In each set of experiments we will state our expectations and compare them with the results we have obtained. During the experiments, we focused on groups of students divided by field of study: mathematicians, physicists, computer scientists and teachers. We further divided these fields into bachelor's and master's students. Most of the plots in this chapter will contain a legend, in which the following shortenings will be used:

- **MAT-b** - represents the results of the evaluator for bachelor mathematicians
- **CS-b** - represents the results of the evaluator for bachelor computer scientists
- **PHY-b** - represents the results of the evaluator for bachelor physicists
- **TEA-b** - represents the results of the evaluator for bachelor teachers
- **MAT-m** - represents the results of the evaluator for master mathematicians
- **CS-m** - represents the results of the evaluator for master computer scientists
- **PHY-m** - represents the results of the evaluator for master physicists
- **TEA-m** - represents the results of the evaluator for master teachers
- **overall score** - represents the overall score of the evaluator

4.1 Impact of k

In this experimental analysis, we expect to observe a trend where increasing the value of k in the k-NN algorithm leads to a reduction in the RMSE and an increase in the F1 score calculated by the evaluator. By sampling a larger number of similar users, there is an increased likelihood of providing more accurate course recommendations to the student. This set of experiments was performed on a training set containing data from academic years 2015/16 to 2019/20 and a test set containing data from academic years 2020/21 to 2021/22. The results of these experiments are shown in Figure 4.1 and Figure 4.2.

The results indicate that our hypothesis is confirmed in most cases. The plots in Figure 4.1 show a significant decrease in RMSE and plots in Figure 4.2 a significant increase in the F1 score almost for each field already at $k = 10$. In most cases for $k > 10$, the error decreases and the score increases only slightly.

It is interesting to observe the curves *TEA-b* and *TEA-m*, which strongly disprove our hypothesis. This can be explained by the fact that there are not many students who specialise in teaching at our faculty. In the data for our recommender system, the total number of users is 18525, of which there are 639 bachelor teachers and only 345 master teachers. These numbers are insignificant compared to the total number of users, and therefore the RMSE for these fields will increase and the F1 score will decrease with higher k.

If we notice the *CS-b* curves, the error is steadily decreasing and the score is steadily increasing for this field. Unlike students specialising in teaching, our faculty has significantly more computer science students. Of all the users, there are 4952 bachelors in computer science fields and therefore for these fields the RMSE will decrease and the F1 score will increase with higher k.

It is also clear from the results that in most cases the master's fields perform worse than the overall score. This can be explained by the fact that students at the master's level have more freedom to choose their courses and therefore it is harder to hit an accurate prediction.

Since the most pronounced decrease in RMSE and the most pronounced increase in F1 score is at k from the interval $[10, 20]$, we have chosen the average of the numbers from this interval as the best choice of k for our recommender systems, i.e. $k = 15$. We will therefore use this value of k in further experiments.

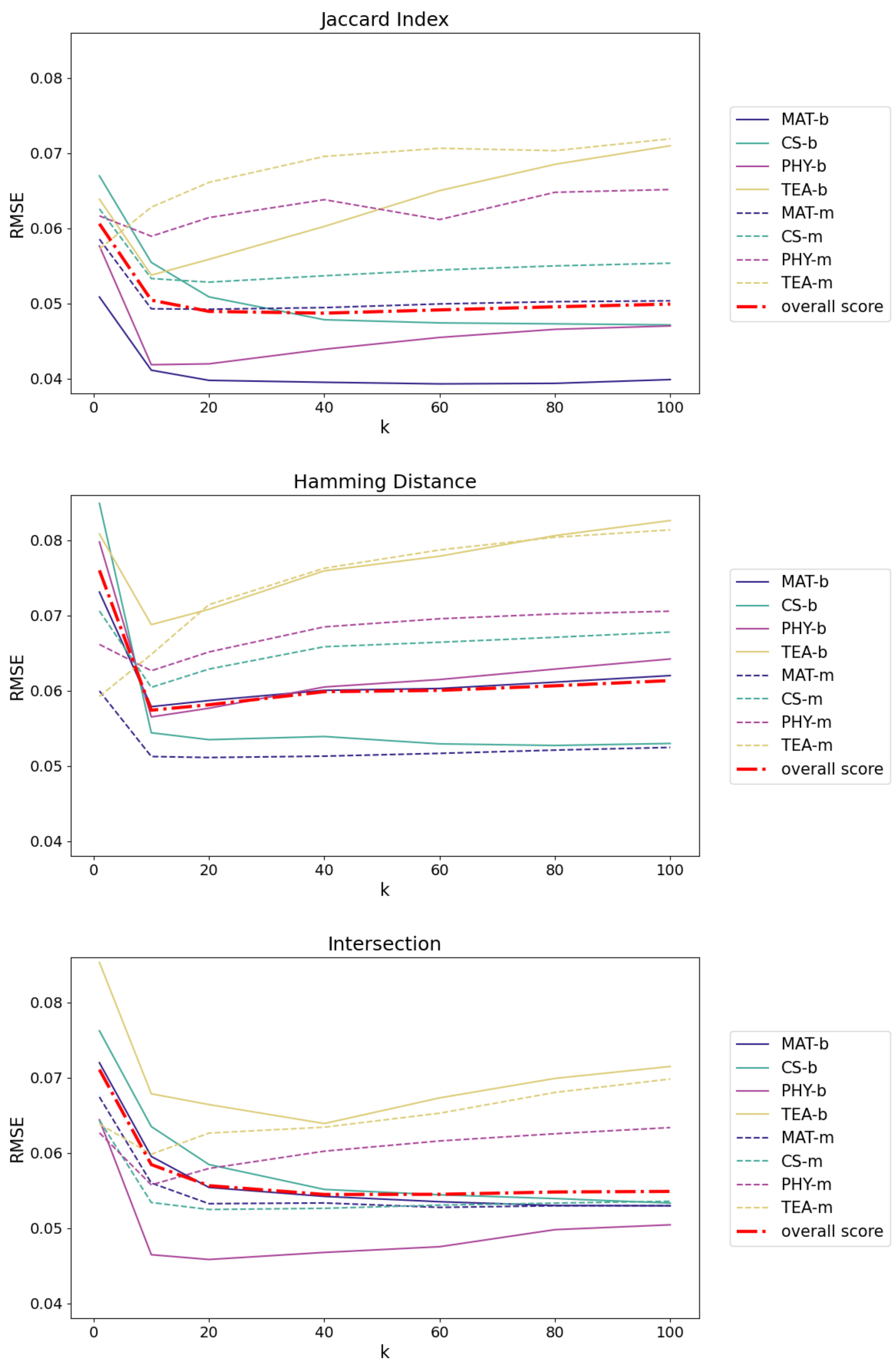


Figure 4.1: Plots showing the impact of the size of k on the implemented similarity methods using the RMSE accuracy metric. The horizontal axes represent the size of k. The vertical axes represent the calculated RMSE value.

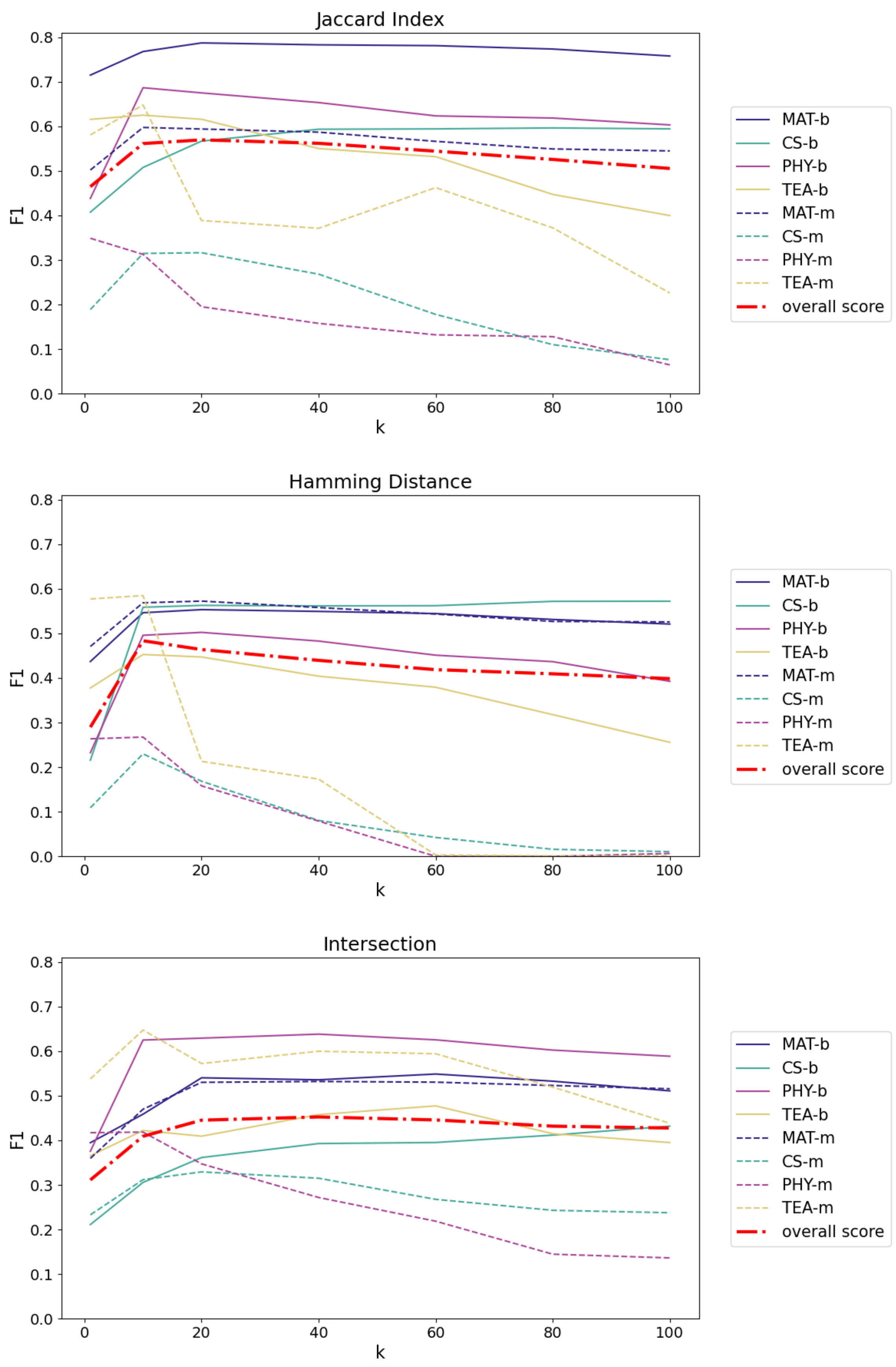


Figure 4.2: Plots showing the impact of the size of k on the implemented similarity methods using the F1 score accuracy metric. The horizontal axes represent the size of k. The vertical axes represent the calculated F1 score.

4.2 Evaluation of the academic year 2021/22

In this series of experiments, we evaluate the performance of our recommender systems in predicting courses for the most recent academic year in our database, which is 2021/22. These experiments involve training the systems with different sets of training data. By systematically increasing the training data, we aim to observe the impact on the performance of the recommender systems and analyze how their predictive abilities improve with more historical data. Our expectation is that as we increase the amount of training data, the RMSE will decrease, while the F1 score will increase.

To conduct these experiments, we will use test data from the academic year 2021/22. We will start by training the systems on data from the academic year 2009/10. Then, we will gradually expand the training data range year by year. For instance, the next step would involve training the systems on data from 2009/10 to 2010/11, followed by 2009/10 to 2011/12, and so on. We will use $k = 15$, as we found in the previous experiment in section 4.1. The results of these experiments are shown in Figure 4.3 and Figure 4.4.

The results confirm that our expectations are met: providing the model with more training data closer to the academic year 2021/22 leads to more accurate predictions for that year. As in section 4.1, the *TEA-b* and *TEA-m* curves showed the worst results. Also, in most cases, the master's fields are significantly weaker than the overall scores.

The plots indicate a noticeable pattern: a rise in RMSE and a decline in F1 score following the academic year 2016/17. This observation can be explained by the faculty's recent accreditation, which makes the previous data less relevant in predicting outcomes. But if we focus only on the curves showing the overall score, our hypothesis is confirmed in every case.

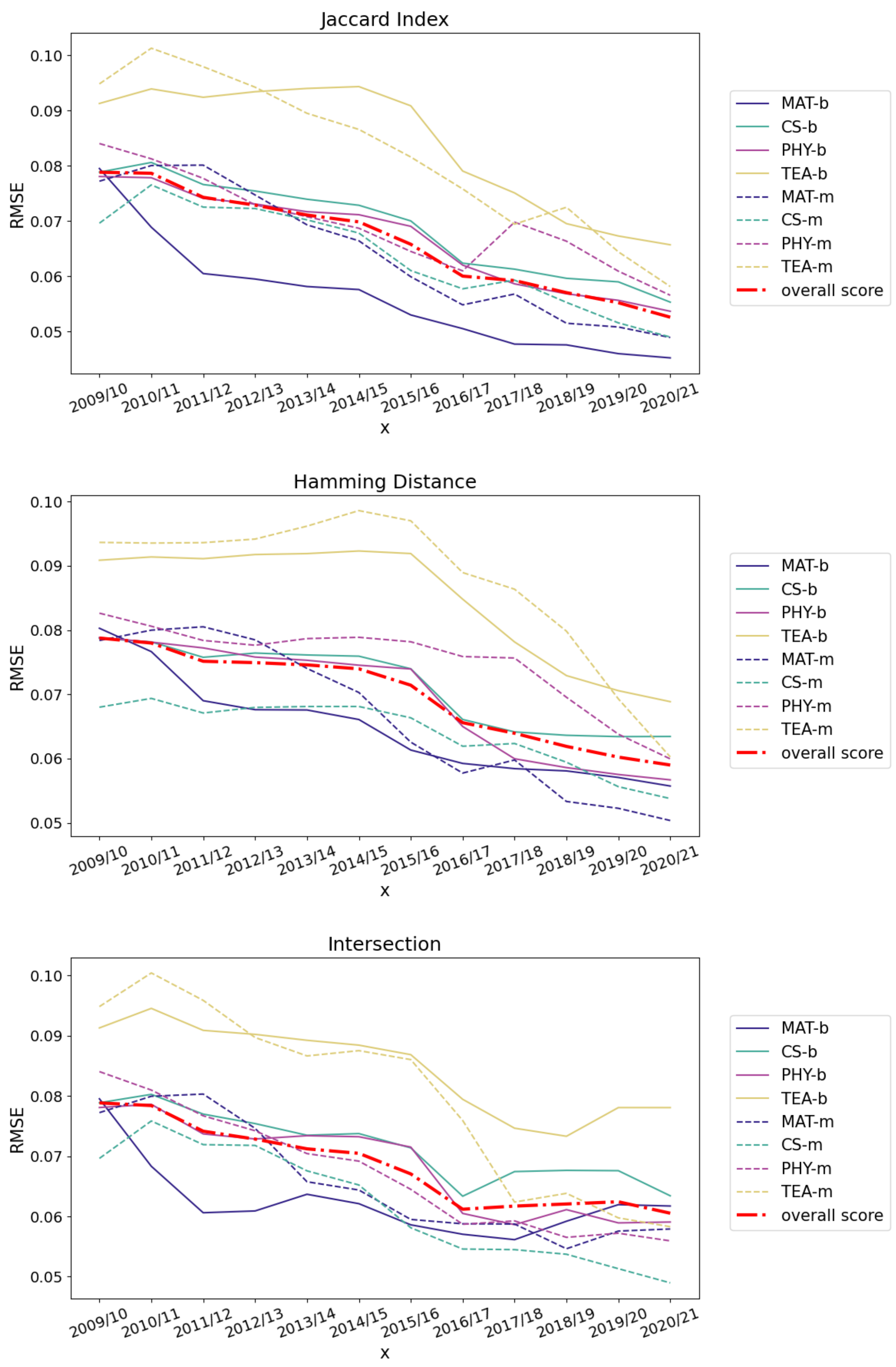


Figure 4.3: Plots showing the performance of implemented similarity methods in predicting courses for the academic year 2021/22 using the RMSE accuracy metric. The horizontal axes represent the size of the training data from 2009/10 to x. The vertical axes represent the calculated RMSE.

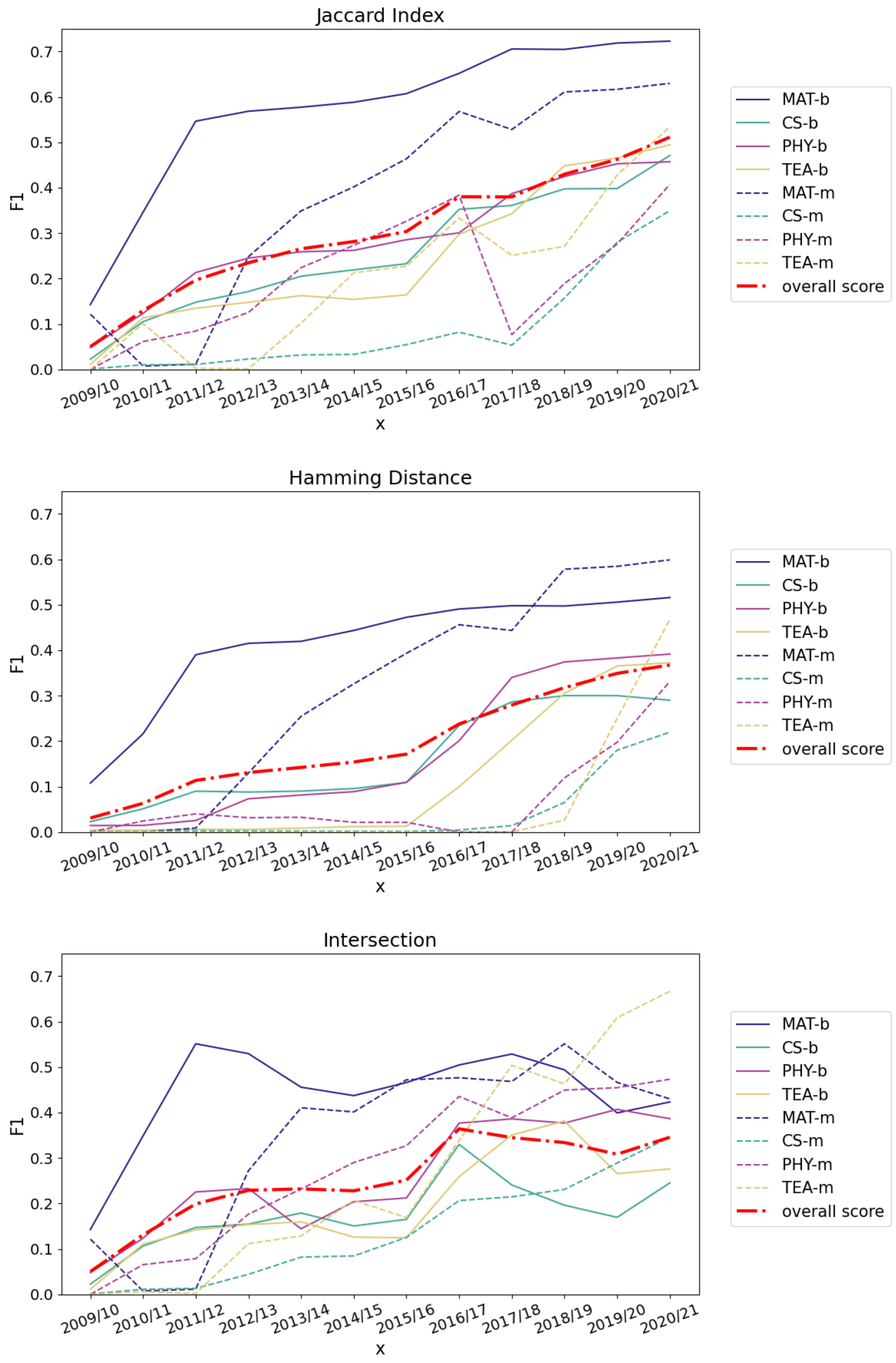
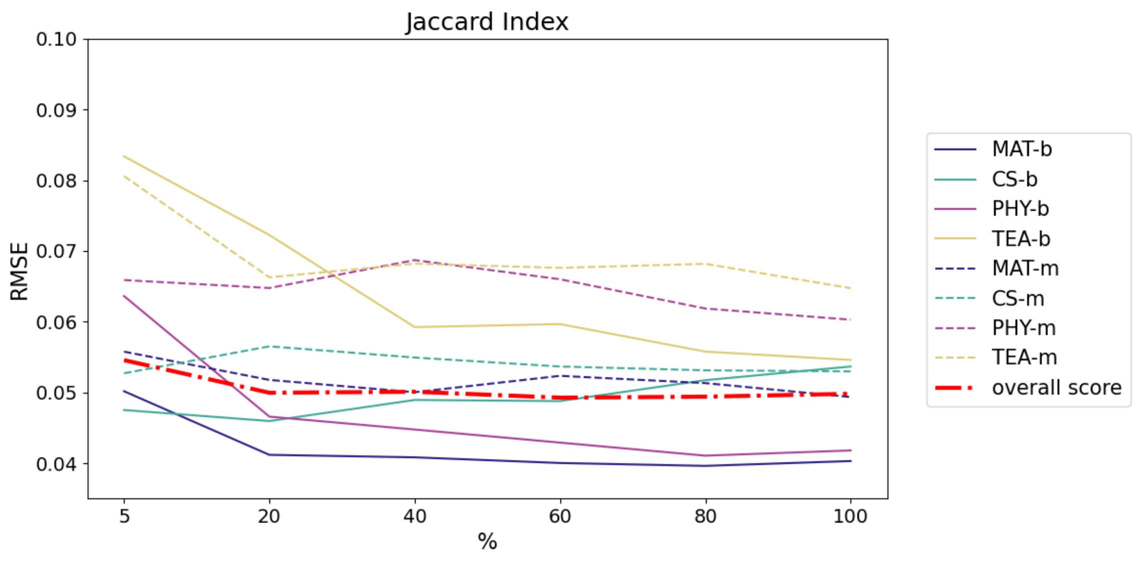


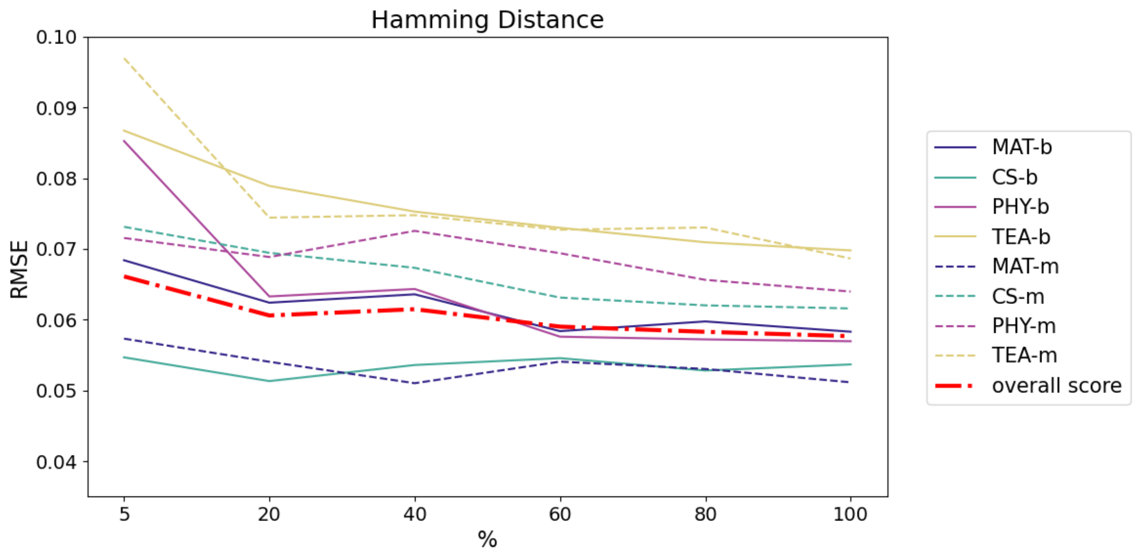
Figure 4.4: Plots showing the performance of implemented similarity methods in predicting courses for the academic year 2021/22 using the F1 score accuracy metric. The horizontal axes represent the size of the training data from 2009/10 to x. The vertical axes represent the calculated F1 score.

4.3 Impact of subsampling

In this set of experiments, we aim to determine the impact of data subsampling on the RMSE and F1 score in the implemented similarity methods. We perform these experiments using training data from the academic years 2015/16 to 2019/20. To evaluate the performance, we set the test data from the academic years 2020/21 and 2021/22. In these experiments, we set the value of k to 15 (see section 4.1), which is a parameter used in the k-NN algorithm. However, to introduce variability and analyse the impact of subsampling, we randomly select a subset from the available data. We expect that as we increase the amount of subsampled data, the RMSE will decrease. At the same time, we expect the F1 score to increase. The results of these experiments are shown in Figure 4.5 and Figure 4.6.

It is clear to see from the results of this experiment that our expectations were fulfilled and therefore the larger percentage of data we sample, the more accurate the results are. As in experiment 4.1 and experiment 4.2, the results for curves *TEA-b* and *TEA-m* were the worst. We assume that it is still for the same reason, namely that there are relatively few students specializing in teaching at our faculty and therefore it is hard to make an accurate prediction. Also, in most cases, the curves for master's fields show a much weaker score than the overall score. We expect this to be due to the same factor as in previous experiments, namely that students in master's fields have more freedom to choose their courses and therefore can differ significantly from one another. However, when we examine the curves showing the overall score in all the shown plots, they behave exactly as we expected.





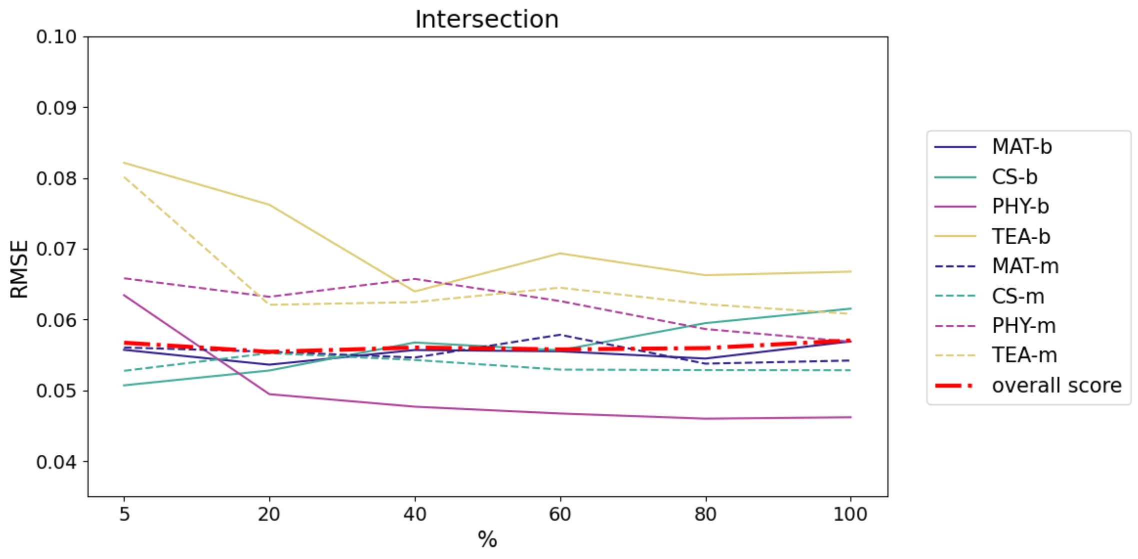


Figure 4.5: Plots showing the impact of data subsampling on the implemented similarity methods using the RMSE accuracy metric. The horizontal axes represent the percentage of subsampled data. The vertical axes represent the calculated RMSE.

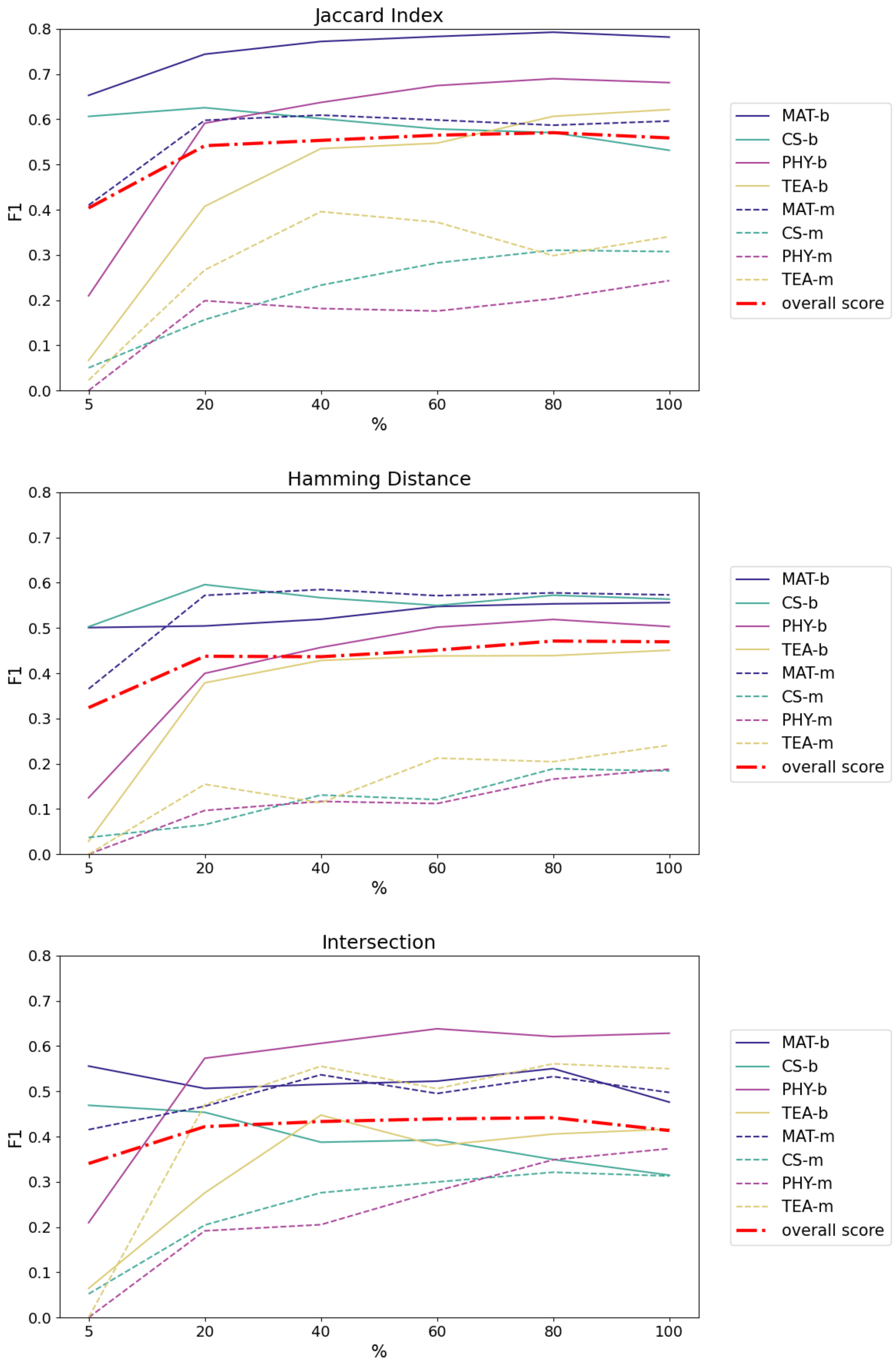


Figure 4.6: Plots showing the impact of data subsampling on the implemented similarity methods using the F1 score accuracy metric. The horizontal axes represent the percentage of subsampled data. The vertical axes represent the calculated F1 score.

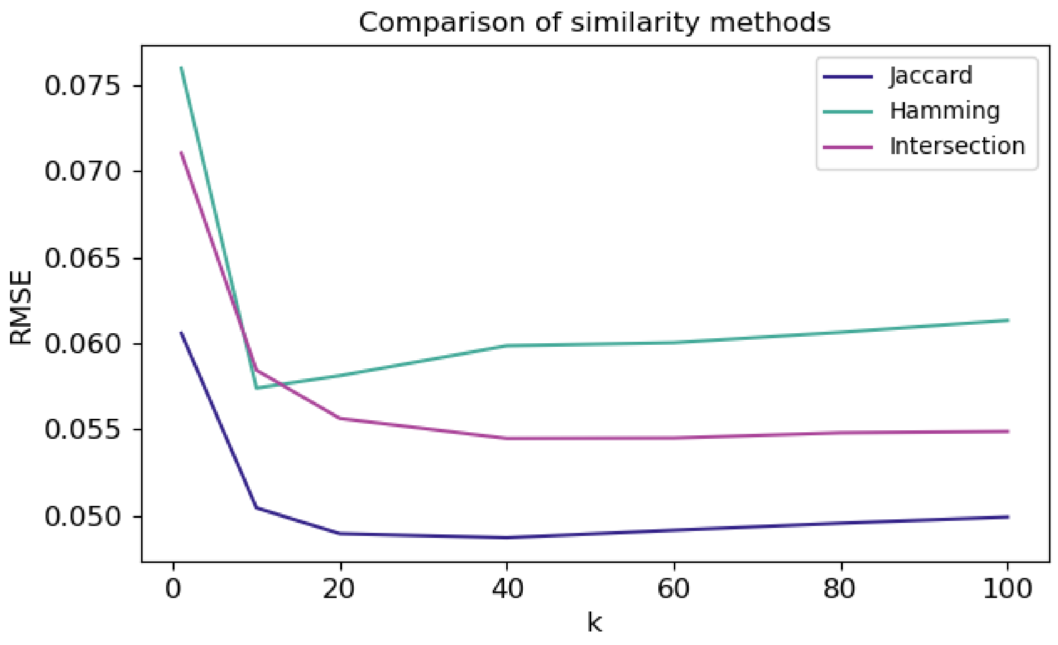
4.4 Selecting the best similarity method

In this section, our primary goal is to identify the optimal implemented similarity method for our course recommender. To achieve this goal, we perform a general analysis of the *overall score* curves obtained from experiment 4.1, experiment 4.2, and experiment 4.3. In order to determine the most accurate similarity method, we combine the curves obtained from the individual experiments. By merging these curves, we create comprehensive plots that allow us to compare and evaluate the performance of each similarity method. Our selection process for the final similarity method involves considering the results of all the experiments. We prioritise the method that performs the best in the majority of experiments, indicating its overall accuracy.

In Figure 4.7 you can see the *overall score* curves from Figure 4.1 and Figure 4.2[1]. In Figure 4.8 you can see the *overall score* curves from Figure 4.3 and Figure 4.4[2]. In Figure 4.9 you can see the *overall score* curves from Figure 4.5 and Figure 4.6[3]. The Jaccard Index similarity method consistently produces the best results in terms of both accuracy metrics. This method outperforms the other similarity methods in all three experiments. The Intersection method turns out to be the second best method. On the other hand, the Hamming Distance method gives the worst results among the implemented similarity methods. It is also interesting to note that in each pair of plots, the curves are axially symmetric along the x-axis.

Based on the results, it is clear that the **Jaccard Index** is the best similarity method among the other evaluated methods. Following this finding, we use the Jaccard Index to make predictions for our personal data. We will make a prediction for second-year course enrolments (the academic year 2021/22), using first-year course enrolments (the academic year 2020/21) for comparison with other users. As training data, we use data from the academic years 2015/16 to 2019/20 and the value of $k = 15$. The sample of courses that were actually enrolled together with the probability of enrolment predicted by the recommender is shown in Table 4.1. From this table, it can be observed that most of the courses that we have actually enrolled in have been predicted with a high probability of enrollment. Table 4.2 shows the courses that were predicted by the recommender, but we did not enrolled them. However, these courses were each predicted with low probability.

[1]experiment in section 4.1
[2]experiment in section 4.2
[3]experiment in section 4.3



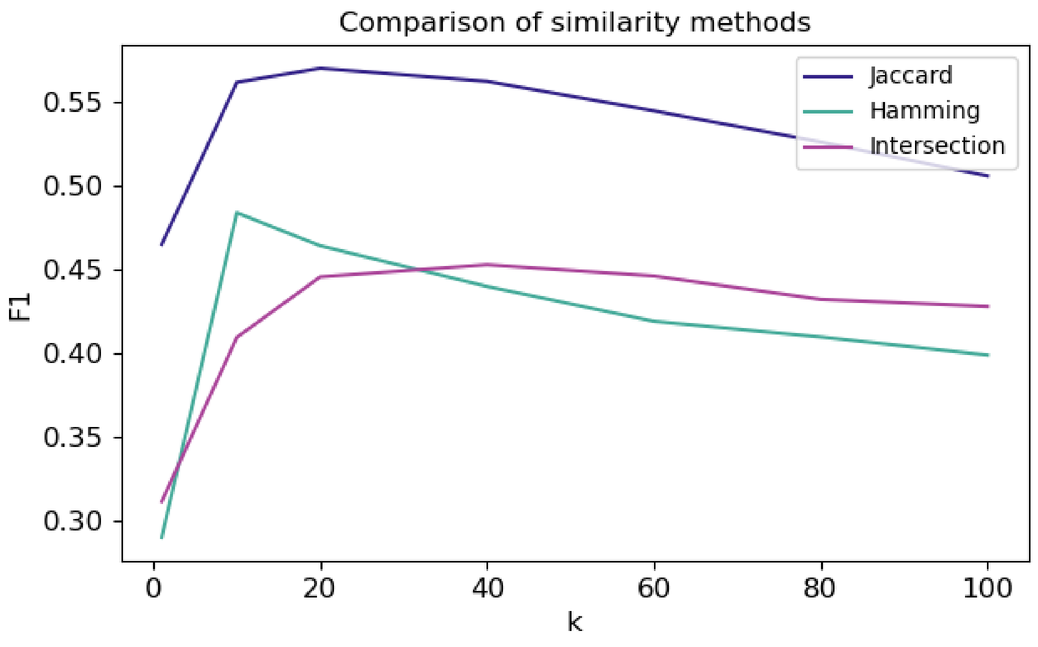
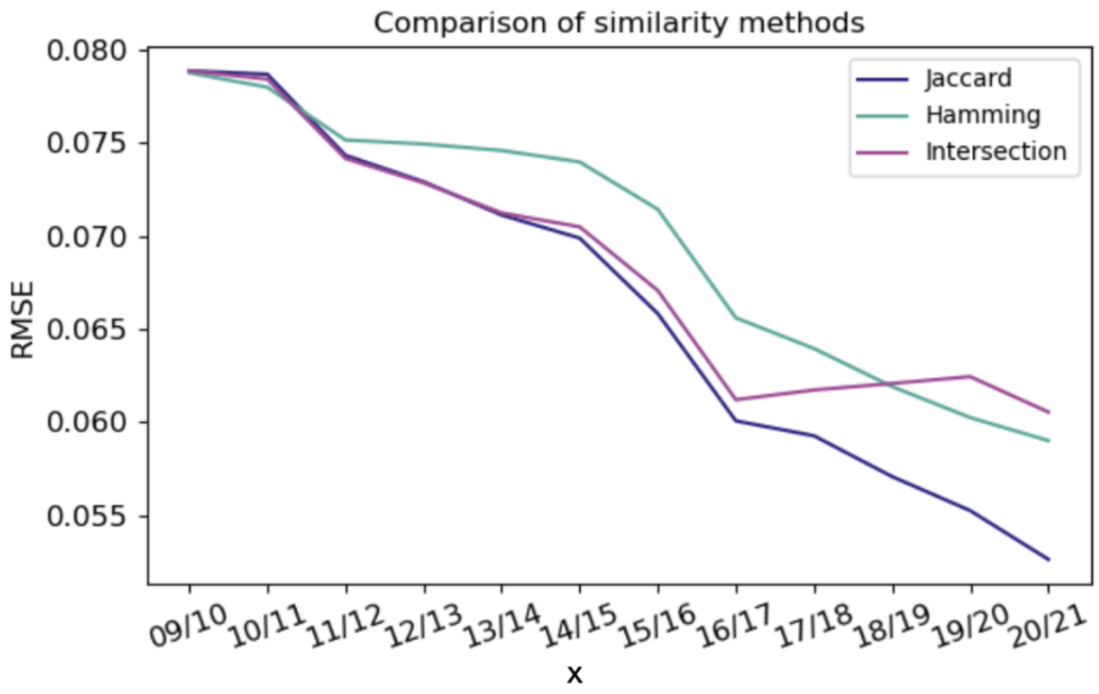


Figure 4.7: Plots showing the comparison of curves *overall score* from Figure 4.1 and Figure 4.2. The horizontal axes represent the size of k. The vertical axes represent the calculated RMSE and F1 score values.



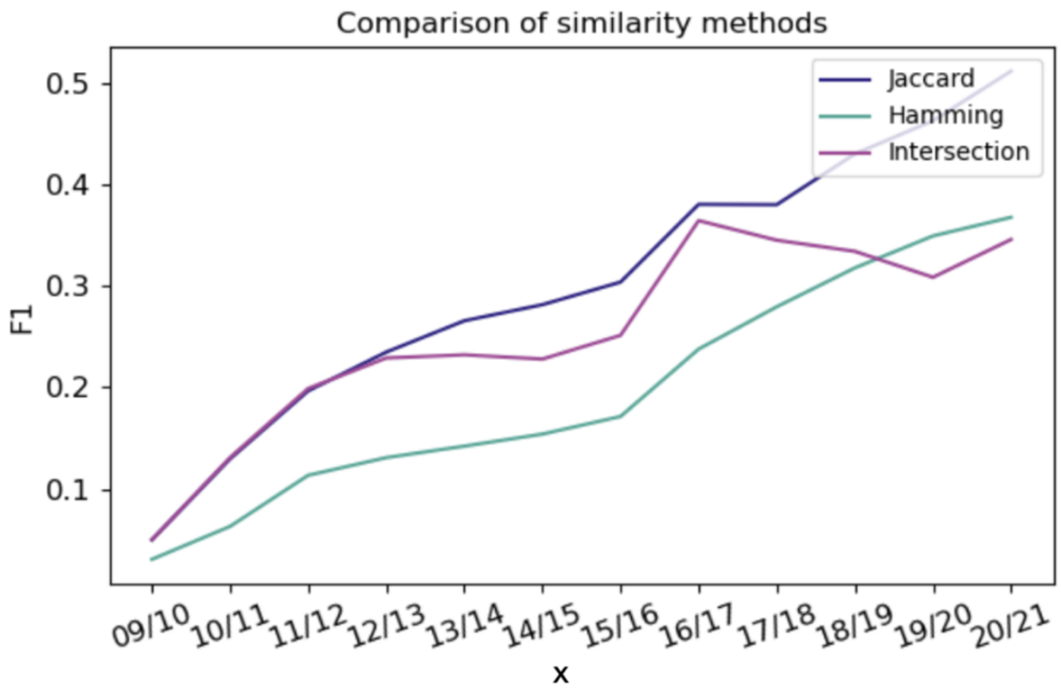
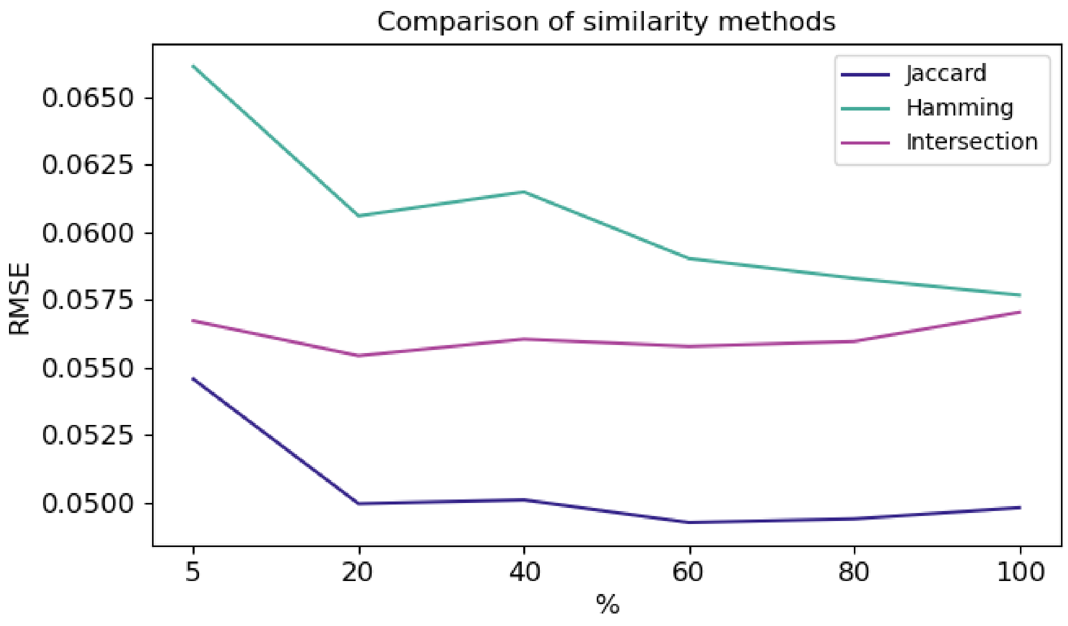


Figure 4.8: Plots showing the comparison of curves *overall score* from Figure 4.3 and Figure 4.4. The horizontal axes represent the size of the training data from 2009/10 to x. The vertical axes represent the calculated RMSE and F1 score values.



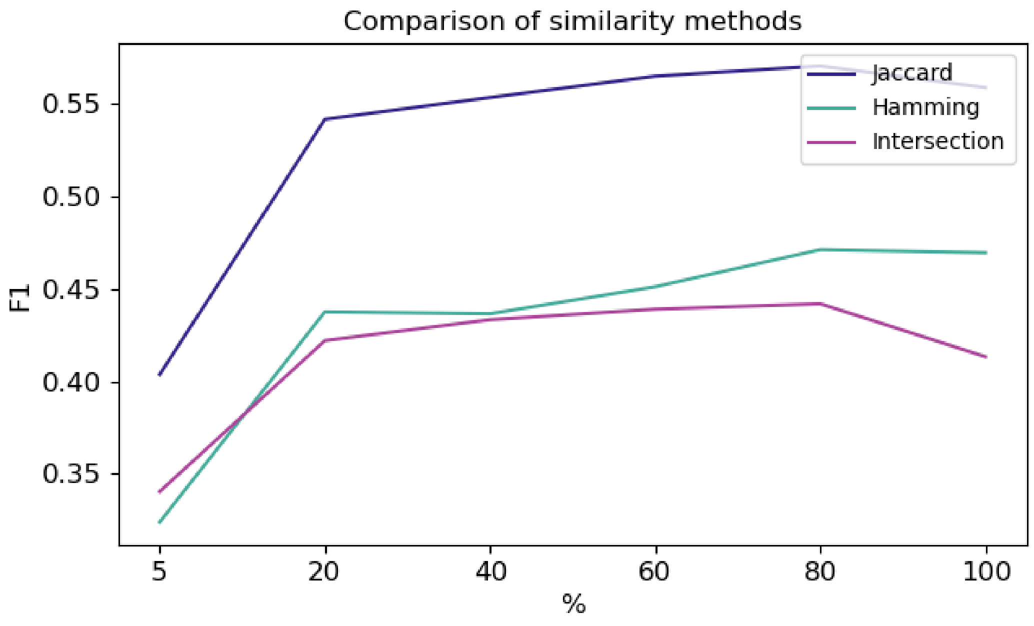


Figure 4.9: Plots showing the comparison of curves *overall score* from Figure 4.5 and Figure 4.6. The horizontal axes represent the percentage of subsampled data. The vertical axes represent the calculated RMSE and F1 score values.

enrolled course	**prediction**
Algorithms and data structures	1.0
Database practice	0.533
Formal languages and automata (1)	1.0
Linux - principles and resources	0.333
Software development principles	0.933
Programming (3)	0.933
Year project (1)	1.0
Physical Education and Sport (3)	0.666
Introduction to database systems	1.0
Operating systems	0.933
Computer Networks (1)	0.866
Year project (2)	0.933
Social aspects of computer science	0.933
Physical Education and Sport (4)	0.466
Creating efficient algorithms	0.933
Introduction to mathematical logic	1.0

Table 4.1: Courses that have been actually enrolled, together with the predicted probability of enrollment.

unenrolled course	**prediction**
Web applications (2)	0.066
Mathematical Analysis (3)	0.066
Cryptology (1)	0.066
Algebra (3)	0.133
Formal languages and automata (2)	0.333
Fundamentals of reverse engineering	0.066
Digital production technologies	0.066
Mathematics (2) - Mathematical analysis	0.066
Mobile application development	0.066
Introduction to information security	0.333
Quantum information processing	0.066
Programming (2)	0.066

Table 4.2: Courses that have not been enrolled, together with the predicted probability of enrollment.

Conclusion

In this bachelor's thesis, we focused on developing a course recommender system specifically designed for the students of *Faculty of Mathematics, Physics and Informatics.* The available data lacked detailed information about the courses themselves, so we were limited to only having access to individual students' course enrollments. Despite this limitation, we wanted to extract as much value as possible from the provided data. As a result, we implemented a *collaborative filtering model* using *user-based* techniques.

To implement our model, we used the *k-NN algorithm*, using three different similarity methods to compare users, namely: the *Jaccard Index*, the *Hamming Distance* and the *Intersection.* To evaluate the effectiveness of these methods, we developed an evaluator that uses two accuracy metrics: the *RMSE (Root Mean Square Error)* to quantify the error, and the *F1 score* to evaluate the overall accuracy.

The results of the evaluator were then analysed and compared. In our thesis, the best similarity method for the provided data was found to be the **Jaccard Index**. In the final stages of our thesis, we tested the recommender system using this similarity method on our personal data and let the courses be recommended to us. We then compared the prediction with reality, where it turned out that the recommender implemented in this way recommended the courses we actually enrolled in with high probabilities and, on the other hand, recommended the courses we did not enroll in with low probabilities.

Since our implemented model did not use a more detailed description of the recommended items, i.e. courses, one of the main improvements that could be made in future work is to add a description of the courses. If in the future we could get information about the content of the courses from the study lists and plans, we would be able to identify similar courses based on this description, which would open up the possibility to implement *content-based* or *hybrid* models as well. We would also be able to focus on recommending elective and obligatory-optional courses, potentially moving away from recommending obligatory courses.

Another possible improvement would be to obtain data from a Student Questionnaire, where courses could be recommended based on verbal reviews from students, along with the help of overall course ratings.

We would be pleased if in the future such a modified recommender system would also start to be used in reality. A web interface could be implemented for students who need help with recommending courses for future years of their studies. One of the realizable plans could also be the implementation of such a recommender system into the Votr student system. When enrolling in courses on this portal, the student could use the recommendation service.

We strongly believe that some of these plans will be implemented in the future and that they will be useful for the students of the Faculty of Mathematics, Physics and Informatics who need help with selecting the right courses.

Bibliography

[1] Charu C. Aggarwal. *Recommender Systems*. Springer, 2016.

[2] Luciano da F. Costa. Further generalizations of the jaccard index. *CoRR*, abs/2110.09619, 2021.

[3] European Mathematical Society. Hamming distance. In *Encyclopedia of Mathematics*.

[4] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council.

[5] Félix Hernández del Olmo and Elena Gaudioso. Evaluation of recommender systems: A new approach. *Expert Syst. Appl.*, 35:790–804, 10 2008.

[6] Oliver Kramer. *Dimensionality Reduction with Unsupervised Nearest Neighbors*. Springer, 2013.

[7] Yue Ruan, Xiling Xue, Heng Liu, Jianing Tan, and Xi Li. Quantum algorithm for k-nearest neighbors classification based on the metric of hamming distance. *International Journal of Theoretical Physics*, 56:3496–3507, 08 2017.

[8] Wikipedia. Precision and recall — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Precision%20and%20recall&oldid=1149017180`, 2023. [Online; accessed 04-May-2023].

[9] Dexin Yang, Chunjing Lin, and Bo Yang. A novel secure cosine similarity computation scheme with malicious adversaries. *International Journal of Network Security and Its Applications*, 5:171–178, 03 2013.

Appendix A: Database description

The data has been provided to us in the following five tables: **export**, **poctyznamok2**, **predmet**, **studprog** and **znamky2**. Below is a description of these tables and their columns.

Export - this table contains the student's course enrolment records. Table 4.3 is a description of *export* table.

column	**description**
id	unique identifier of the (anonymous) student
idpred	course identifier (key for the *predmet* table)
akrok	academic year of the enrollment
semester	value indicating the semester (Z for winter semester and L for summer semester)
pass	an indication of how the student passed the course (1 if the student received a grade of A, B, C, D or E. 0 if the student received a grade of Fx)
skratkasp	official shortcut of the student's study programme (key for the *studprog* table)

Table 4.3: Description of *export* table

Studprog - this table is a list of all study programmes provided by the faculty. Table 4.4 is a description of *studprog* table.

column	**description**
skratkasp	official shortcut of the study programme
sp	official full name of the study programme

Table 4.4: Description of *studprog* table

Predmet - this table is a list of all courses provided by the faculty. Table 4.5 is a description of *predmet* table.

column	**description**
idpred	unique identifier of the course
kodpred	official course code
stredisko	the department which provides teaching of this course
skratkapred	shortcut of the official course code
nazovpred	the full official name of the course

Table 4.5: Description of *predmet* table

Poctyznamok2 - this table describes the number of a specific grade for each grade (A/B/C/D/E/Fx) and each course where at least ten students were enrolled in a given academic year. Table 4.5 is a description of *poctyznamok2* table.

column	**description**
idpred	unique identifier of the course
akrok	academic year of the enrollment
kodhod	specific grade (A/B/C/D/E/Fx)
pocet	number of students who received a specific grade (*kodhod*) in a given academic year (*akrok*)

Table 4.6: Description of *poctyznamok2* table

Znamky2 - this table describes the number of students who graduated with each grade in a given academic year where at least 10 students were enrolled. Table 4.7 is a description of *studprog* table.

column	**description**
idpred	unique identifier of the course
akrok	academic year of the enrollment
a,b,c,d,e,fx	number of students who graduated with a grade A, B, C, D, E or Fx in a given course (*idpred*) in a given academic year (*akrok*)
bez	number of students who did not receive any of the grades described above
n	total number of students on a given course (*idpred*) in a given academic year (*akrok*)

Table 4.7: Description of *znamky2* table

Appendix B: Electronic appendix

In the electronic appendix attached to the thesis (file `main.py`), we provide the source code of the implemented recommender systems, using the mentioned similarity methods and also the evaluator. This file also contains the function `main()`, in which the situation from section 4.4 is simulated when we let the recommender predict the courses.

The source code is also published at `https://github.com/patriciavnencakova/RecommenderSystem`.

Before running the program itself, we recommend installing the *Miniconda*[4], which contains the Python packages that we have used during our work. You can install the Miniconda by running the command `conda install`. Our program expects a database at the input, which form is described in supplementary section *Appendix A: Database description*. However, the attachment does not include the database we worked with during the implementation because it contains confidential information about the enrollment of other students at our faculty. We thus consider these data to be sensitive.

[4]See more on page `https://docs.conda.io/en/latest/miniconda.html`