2017

# Building a Course Recommender System for The College of Wooster

Nan Jiang
*The College of Wooster*, njiang17@wooster.edu

# Building a Course Recommender System for The College of Wooster

Independent Study Thesis

Presented in Partial Fulfillment of the Requirements for
the Degree Bachelor of Arts in the
Department of Computer Science and Mathematics at The
College of Wooster

by
Nan Jiang

The College of Wooster
2017

**Advised by:**

Dr. Sofia Visa (Computer Science)

Dr. Robert Kelvey (Mathematics)

THE COLLEGE OF WOOSTER

## Abstract

The goal of this project is to investigate the approaches for building recommender systems and to apply them to implement a course recommender system for the College of Wooster. There are three main objectives of this project. The first is to understand the mathematics and computer science aspects behind it. The mathematic concepts built into this project include probability, statistics and linear algebra. The final product is consist of two components: a collection of Python scripts containing the implementation code of the course recommender system, and a simple user interface allowing people to use the recommender system without typing commands. The second goal is to analyze the pros and cons of different approaches by comparing their performance on the same training data set which have information about students and courses at the college in the last seven years. The final goal is to apply the best model to build the course recommender system that can provide helpful and personalized course recommendations to students.

# Acknowledgments

I would like to express my gratitude to my advisors for their patience and feedback in overcoming numerous obstacles I have been facing through my research. I would like to thank my girlfriend Freya for all her love and support throughout writing this thesis. Last but not the least, I would like to thank my mom and aunt for supporting me spiritually and financially throughout not just the past four years but also my life in general.

# Contents

# List of Figures

# List of Tables

CHAPTER 1

# Introduction

The College of Wooster is a private liberal arts college located in Wooster, Ohio, United States. It is primarily known for its mentored undergraduate research. The total enrollment of the college is about 2000 [1]. There are 50 areas of study and 39 majors available at Wooster. Numerous courses are provided in each area, each one satisfies one or more general education requirements. There are seven general requirements defined by the College's curriculum [4]:

1. **W**: Writing Intensive
2. **C**: Studies in Cultural Difference
3. **R**: Religious Perspective
4. **Q**: Quantitative Reasoning
5. **AH**: Learning Across the Disciplines: Arts and Humanities
6. **HSS**: Learning Across the Disciplines: History and Social Sciences
7. **MNS**: Learning Across the Disciplines: Mathematical and Natural Sciences

To graduate from the College of Wooster, a student must have a minimum of 32 course credits and complete all the major courses and general requirements.

In the middle of a semester students have to register courses for the next semester. The registration is completed on ScotWeb, an internal platform provided by the college for viewing and managing student information. For students who have planned out their course schedules, the registration process is easy and quick. But in fact many students have difficulty choosing courses to take. Often times, it is the courses outside the major that students lack the competence to choose. They need those courses to fulfill the general requirements. At the same time, they want to make sure that the courses selected are interesting to them. But without further information other than the metadata of a course, it is hard for students to efficiently find the right courses for them.

Therefore, after understanding the science behind recommender systems, we decide to build a course recommender system in this project help students at the College of Wooster with the course selection process.

## 1.1 Recommender System

"*We are leaving the age of information and entering the age of recommendation.*" said Chris Anderson in his famous book "The Long Tail" [7]. Today, people are surrounded by an overwhelming amount of information. More information drives smarter decisions, yet sometimes having overloaded information reduces the quality of our decisions. This is particularly evident on the Internet. It is reported [3] in 2015 that in just one minute, there are 3.3 million posts posted on Facebook, 400 hours of videos uploaded to YouTube, etc. It is impossible for the users to find out the pieces he or she is truly interested in out of the data ocean in a short amount of time. A software called *Recommendation Systems* (RSs) has come up with a solution to handle this issue.

RSs are software tools providing personalized suggestions for items to users that they will likely be interested in. "Item" is a general term that denotes what content the system recommends to the users. It could be a movie, a book, or merchandise. In general, a RS focuses on one specific domain. The target users of RSs are all people that have potential need for help in evaluating the alternative items that an information system may offer. The problem of a RS can be summarized as the estimation of a utility function that automatically predicts how an user will like an item based on factors like past behavior, relations to other users, item similarity, etc. The system can provide personalized recommendations to individuals by learning their browsing history, looking for similar users in the community and recommend what they like, or finding similar items.

Today, RSs are implemented in many websites to filter information and suggest items immediately to users that fit their taste. For example, Amazon tells consumers what merchandise are often bought together, YouTube suggests relevant videos to an individual based on his/her previous watching history, and Facebook creates a tailored news feed for individuals. As the speed of data generation constantly increases, the need for recommendation systems rises.

The implementation of a RS can be divided into two steps:

1. **Learning process**, which learns the users' behavior or the relation between items to build a model that represents a user's taste or relationships between items.

2. **Decision process**, which takes in a user's preferences and constraints and uses the model built in the previous step to generate predictions of the most suitable items for the user.

The learning process is the core of a RS, it can be associated to a *data mining* problem. Data mining is a process of extracting interesting and useful information from existing large data set. Some of the most useful data mining methods, including specific machine learning algorithms, statistics, clustering, classification and association rule learning [6], are often used in building the recommendation model.

The most common approaches to recommendation are *content-based* and *collaborative filtering*. The first one solely looks at the items the target user has rated and finds highly similar items to recommend. The second method analyzes the behavior of an entire community and finds similar users to the current user, and recommends items those similar users have rated. The recommendation approaches are discussed in details in Chapter 2.

The objective of this research is to build a course recommendation system for The College of Wooster that serves current students. The system can generate personalized results based on students' courses taken in the past. In machine learning problems, the portion of data used to fit a model is called *training data*. The training data used to feed our system is a collection of students and their courses taken in the last seven years (from 2009-10 to 2015-16). For more information about the training data, please read Chapter 3. Several recommendation approaches are applied in the implementation of this system. Different approaches are tested and evaluated to see which one has the most effective performance.

## 1.2 Literature Review

Numerous studies have been done in the field of RS. Researchers have been proposing new algorithmic techniques and implementing RS in real life application domains. Several studies have been conducted in the E-learning field that focused on analyzing students' course selection behavior.

Lu et al. [11] provides an up-to-date snapshot of the different application domains of recommendation systems and approaches used. One of the domains discussed is E-learning in which it summarizes the techniques applied in a few studies. One technique mentioned is *association rule learning* which is a data mining technique commonly used to extract important transaction rules from data. Another one is collaborative filtering which is a common method used in all kinds of recommendation systems. These two appear to be the most common methods used to implement course recommendation systems as they are frequently seen in related works.

Aher et al. [5] presents several data mining techniques used to develop a course recommendation system. The authors use different combinations of clustering and association rule learning algorithms to learn about student's learning behavior from data to build models. Among all the combinations tested, they find *Simple K-means* and *Apriori* to be most useful. The results are compared with the results obtained using an open source data mining tool called Weka. The model was used to recommend courses to new students who have taken some courses. The problem described in this paper is similar to ours. The main difference is that the target users of their system are students in Computer Science & Engineering and Information Technology departments, whereas we want to serve all students in all departments.

Chang et al. [8] uses a user-based CF algorithm to build a course recommendation system. In this study, the authors use students' predicted course grade as the metric to rank potential courses. The data collected include students' courses taken, grades received, and the popularity of instructors in five years. They apply the idea of *Artificial Immune System* in clustering. This idea provides a way to find the best solution to an optimization problem, in which the objective function is the antigen and the possible solutions to the problem are the antibodies. The authors use the student data as antigens and using the weighted cosine similarity to calculate the affinity between antigens and antibodies. The results are then used to cluster the students. Finally, a user-based collaborative filtering method is applied on the data clusters to predict the rating of the courses and generate the recommendations. The model is evaluated by checking the mean average error and *confusion matrix*. A confusion matrix $C$ is a matrix representation of comparison between true and predicted results. By definition an element $C_{i,j}$ represents the number of observations known to be in group $i$ but predicted to be group $j$. A more thorough explanation of confusion matrix will be given in Chapter 4.

Volkovs and Yu [16] discuss building effective CF models in situations where there is only binary feedback available. Collaborative filtering works well in many application domains in which explicit feedback (e.g. number of stars, scores) are provided. However, the majority of existing models underperform for cases there is only implicit binary feedback (e.g. number of views of a movie, number of clicks on a news link). The authors use neighborhood similarity information to guide latent factorization and derive latent representations. Matrix factorization methods like *Singular Value Decomposition* (SVD) were used to enhance the performance of CF models. The observed binary matrix is transformed to a similarity score matrix. The score matrix is then factorized to produce accurate user or item representations. They test this approach on several large public datasets. The results show that their approach outperform the normal CF models on binary feedback data sets.

The remaining of this thesis is organized as follows. Chapter 2 provides an overview of the common approaches to implement recommender systems and describes three of them in details. These three approaches are *Content-based*, *Collaborative Filtering* and *Association Rule Learning*, they are applied in this project to build the course recommender system (CRS). Chapter 3 describes the data used to build the CRS and the implementations with different approaches. Chapter 4 investigates the metric for evaluating the performance of the CRS and discusses the evaluation results. Chapter 5 presents conclusion and future work.

# CHAPTER 2

# Approaches to Recommender Systems

Businesses see potential value in using massive amounts of existing information from their customers to build a recommender system that recommends customers items or services that they might be interested in, attempting to increase sales. The outcome has proven them right. Here are some examples [6]: on Netflix, 66.7% of the movies watched are recommended; on Google News, 38% more clickthrough have been generated by recommendations; on Amazon, 35% of sales are from recommendations. As a consequence, the role of recommender systems is becoming more important in business.

The problem of recommending items has been studied over the years and many approaches have been discovered. Based on the complexity and knowledge required, these approaches can be divided into two main categories: traditional approaches and advanced approaches. Traditional approaches have been proven successful in most of the scenarios and are commonly used in practice today. Advanced approaches incorporate the lastest research outcome from other fields to handle sophisticated recommender problems. Each approach has its own effectiveness in handling certain application domains and issues. Below is an overview of the common traditional and advanced approaches in recommender system development.

## Traditional Approaches

1. **Content-based RS**: Recommends items based on the features of items.
2. **Collaborative Filtering RS**: Recommends items based on the past behavior of users.
3. **Demographic RS**: Recommends items based on the demographic profile of the target user, assuming different recommendations should be generated for different demographic factors such as country or language.

4. **Hybrid RS**: Combines two or more approaches above to implement a RS.

## Advanced Approaches

1. **Learning to Rank**: Constructs ranking model from training data. It treats the recommender problem as a standard supervised classification problem. [6]

2. **Deep Learning**: Uses *Artificial Neural Network* (ANN) for collaborative filtering.

In this chapter, we discuss three recommendation approaches: *Content-based*, *Collaborative Filtering* and *Association Rule Learning*.

## 2.1 Content-based Approach

A content-based system learns to recommend items based on two ingredients: (1) the content of items and (2) the user's preferences. The content of an item can be described by its features. Features are developed from either explicit attributes (e.g., genre of movies) or text description (e.g., language of novels). Content-based approach is commonly used in domains in which the items are descriptive, such as web pages, articles and movies. The basic idea behind it is to calculate the scores between items using their features and compare the scores with user's preference to see the percentage of matching.

The most important step in content-based model is to develop features of items. The difficulty of doing so depends on how items are characterized. For example, for movies one can easily generate features directly from their genres. Each feature represents a genre. However, if one is interested in comparing movies by their scripts, then one might need to apply some *Natural Language Processing* techniques to extract keywords from the text and use those keywords as features of the movies. The first situation represents the *explicit feature* whereas the second reflects the *implicit feature*. In this section, the method of feature development and comparison of the two situations are discussed.

### 2.1.1 Explicit Feature

The first case occurs when the features of items are explicitly available. One such example is movies. Movies can be classified according to genres. There are action film, drama, comedy, etc. The genre information can be used as features to describe various movies. Suppose there are four movies across different genres that we have watched. Then these movies are mapped to a set of features (in this case genres) (Table 2.1). The binary values 1/0 represent whether the movie has a genre or not.

| **Movie** | **Action** | **Animation** | **Comedy** | **Musical** | **Sci-Fi** |
|---|---|---|---|---|---|
| The LEGO Batman Movie | 1 | 1 | 1 | 0 | 0 |
| La La Land | 0 | 0 | 1 | 1 | 0 |
| Ghost in the Shell | 1 | 1 | 0 | 0 | 1 |
| The Truman Show | 0 | 0 | 1 | 0 | 1 |

**Table 2.1:** Mapping movies to genres using binary representation.

To recommend movies to a user, the user has to provide his/her preferences about those genres. For example, given the preference can be quantified using a scale from 1-5, consider a person with strong preference for animation and comedy movies (Table 2.2).

| **User** | **Action** | **Animation** | **Comedy** | **Musical** | **Sci-Fi** |
|---|---|---|---|---|---|
| A | 3 | 5 | 5 | 1 | 3 |

**Table 2.2:** Mapping user to genres using a scale from 1-5.

To predict which film is more attractive to user A, a score is calculated for each movie by taking the **dot product** of the binary feature representation and the user preference. For instance, for movie *The LEGO Batman Movie*, its score for A is:

$$1 \times 3 + 1 \times 5 + 1 \times 5 + 0 \times 1 + 0 \times 3 = 13$$

The computation is repeated for every movie in the example and the scoring (Table 2.3) shows that *The LEGO Batman Movie* is most attractive to A while *La La Land* is least. Therefore, the system will recommend *The LEGO Batman Movie* and probably *Ghost in the Shell* to user A.

| **Movie** | **Score** |
|---|---|
| The LEGO Batman Movie | 13 |
| La La Land | 6 |

| **Movie** | **Score** |
|---|---|
| Ghost in the Shell | 11 |
| The Truman Show | 8 |

**Table 2.3:** The scoring of movies for user A.

It is nice to work with items that possess the quality of explicit features under content-based approach as the whole recommendation process can be done in a simple and straight forward manner, without using any complex algorithm.

### 2.1.2 Implicit Feature and TF-IDF Algorithm

For items whose features are not directly available, keywords extracted from the content are often regarded as features. During this process, some weighting methods are used to filter out the keywords that best represent the item from all words appeared. One weighting method called *Term Frequency and Inverse Document Frequency* (TF-IDF), is discussed below. We will use two sentences as example to show how TF-IDF works.

Consider the following two sentences:

1. "The dog is sitting on the floor."
2. "The cat is sitting on the sofa."

The first step in getting keywords is *tokenization*. Tokenization is a process of obtaining meaningful basic units called *tokens* from a large piece of text. A simple tokenization method to retrieve tokens from a sentence is to split the words of it on whitespaces. For example the split results of the first sentence are *the*, *dog*, *is*, *sitting*, *on*, *floor*. When analyzing multiple sentences, a set of tokens is obtained from each sentence. The union of all sets of tokens can be used as features of the items and the count of a token can viewed as the score of an item on the token. In this case, the features (tokens) of the two sentences are *the*, *dog*, *cat*, *is*, *sitting*, *on*, *floor*, *sofa*. As with the movie example from previous section, once features are obtained, items can be mapped to features (Table 2.4).

| **Sentence** | **the** | **dog** | **cat** | **is** | **sitting** | **on** | **floor** | **sofa** |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

| **Sentence** | **the** | **dog** | **cat** | **is** | **sitting** | **on** | **floor** | **sofa** |
|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

**Table 2.4:** Mapping sentences to word count.

One problem with this simple word count approach is that not every token has the same significance. The word *"the"* appears more frequent than other words but clearly it is not the main subject of either sentence. Thus words need to be weighted in order to select the significant keywords. That is why weighting methods like TF-IDF are used.

TF-IDF algorithm uses two scores to calculate the weight of a word: *term frequency* and the *inverse document frequency*. The term frequency measures how frequently a term appears in a document. It is calculated by dividing the number of appearance of term $t$ in a document by the total number of terms in the document. The inverse document frequency evaluates the importance of a term. It is calculated by the logarithm of dividing the total number of documents by the number of documents with term $t$ in it.

$$TF(t,d) = \frac{f_{t,d}}{\sum f_{t',d} : t' \in d} \tag{2.1}$$

$$IDF(t,D) = \log_e \frac{|D|}{|\{d \in D : t \in d\}|} \tag{2.2}$$

where $f_{t,d}$ is the frequency of $t$ in a document $d$, $D$ is the corpus that contains all target documents.

The TF-IDF score of a term $t$ is simply the product of $TF(t,d)$ and $IDF(t,D)$. For the word *"dog"*, its TF-IDF score calculated as follows:

$$TF(\text{"}dog\text{"}, sentence1) = \frac{1}{7}$$

$$IDF(\text{"}dog\text{"}, \{sentence1, sentence2\}) = \log_e \frac{2}{1} = 0.693$$

$$TD - IDF(\text{"}dog\text{"}) = \frac{1}{7} \times 0.693 \approx 0.099$$

Calculating the TF-IDF score for all features (words), we have:

| **Sentence** | **the** | **dog** | **cat** | **is** | **sitting** | **on** | **floor** | **sofa** |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.099 | 0 | 0 | 0 | 0 | 0.099 | 0 |
| 2 | 0 | 0 | 0.099 | 0 | 0 | 0 | 0 | 0.099 |

| Sentence | the | dog | cat | is | sitting | on | floor | sofa |
|---|---|---|---|---|---|---|---|---|

Table 2.5: Mapping sentences to TF-IDF scores.

The result shows that all the trivial terms obtain 0 for the TF-IDF score and the non-zero items are selected to be the features of the corpus. Dropping the zero items, the new feature matrix is:

| Sentence | dog | cat | floor | sofa |
|---|---|---|---|---|
| 1 | 0.099 | 0 | 0.099 | 0 |
| 2 | 0 | 0.099 | 0 | 0.099 |

Table 2.6: Mapping sentences to keywords.

At this point, TF-IDF is completed. It helps us to filter out the keywords of the two documents. Now these keywords can be considered as the features of the two sentences. Then the system can compare the two sentences and calculates the similarity between them using those keywords identified by TF-IDF. The similarity is calculated by taking the feature vector of each item and apply a similarity computation method (Section 2.2.1 dives into the topic of similarity computation in details.). Here one common method called *Cosine similarity* is used, the formula of which is given below:

$$\text{sim}_{Cosine}(A,B) = \frac{\overrightarrow{A} \cdot \overrightarrow{B}}{||\overrightarrow{A}|| \times ||\overrightarrow{B}||} \tag{2.3}$$

In this example, the feature vectors of the two sentences, $\overrightarrow{S_1}, \overrightarrow{S_2}$ are defined as:

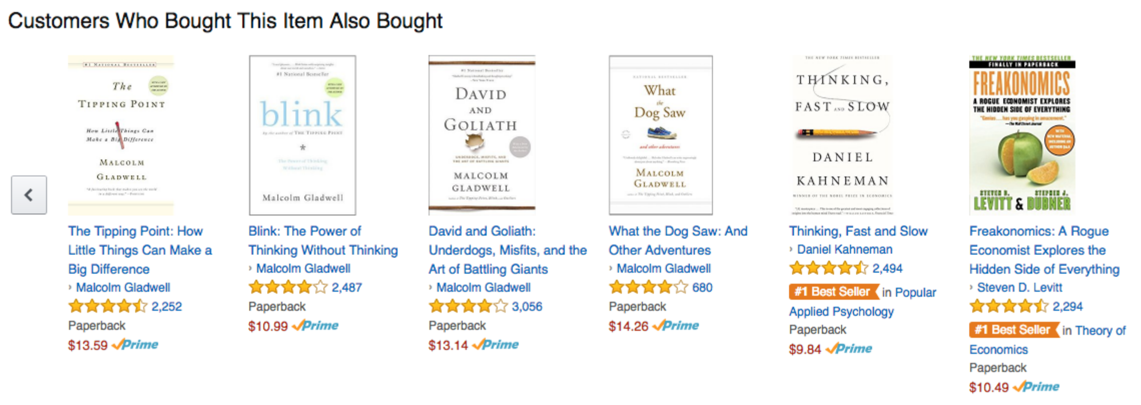
$$\overrightarrow{S_1} = (0.099, 0, 0.099, 0) \qquad \overrightarrow{S_2} = (0, 0.099, 0, 0.099)$$

the similarity between sentence 1 and 2 is 0 since $\overrightarrow{S_1} \cdot \overrightarrow{S_2} = 0$, indicating that they are uncorrelated.

The example shown is for demonstration purpose only. In reality, this technique is applied in more complex domains, for instance a news website. Each news article has its own keywords and scores on different keyword. After a user has read an article, a content-based system will calculate the similarity between the read article and all other news article based on the keywords from the read article, and recommend to the user articles that are more similar to the read article. So if a user has read a sports news with keywords "*basketball*", "*Cavs*", "*LeBron James*", the system is very likely to recommend another news article that has the same or similar keywords, but certainly not an irrelevant global trade news.

## 2.2 Collaborative Filtering

*Collaborative Filtering* (CF) is a method to predict a user's preference or rating of an item, based on his/her previous preferences and decisions made by other users. The two underlying assumptions of the CF approach are: (1) A person's preference does not change much over time; (2) If person A and person B share the same opinion on one issue, A is likely to have B's opinion on a different issue that B has encountered but A has not. Collaborative filtering is considered to be the most popular and widely implemented technique in RS [12]. Netflix [18] and Amazon [14] both have CF algorithms applied in their recommendation engine. Fig. 2.1 illustrates one instance of Amazon's recommender system.


Customers Who Bought This Item Also Bought

The Tipping Point: How Little Things Can Make a Big Difference
Malcolm Gladwell
2,252
Paperback
$13.59 Prime

Blink: The Power of Thinking Without Thinking
Malcolm Gladwell
2,487
Paperback
$10.99 Prime

David and Goliath: Underdogs, Misfits, and the Art of Battling Giants
Malcolm Gladwell
3,056
Paperback
$13.14 Prime

What the Dog Saw: And Other Adventures
Malcolm Gladwell
680
Paperback
$14.26 Prime

Thinking, Fast and Slow
Daniel Kahneman
2,494
#1 Best Seller in Popular Applied Psychology
Paperback
$9.84 Prime

Freakonomics: A Rogue Economist Explores the Hidden Side of Everything
Steven D. Levitt
2,294
#1 Best Seller in Theory of Economics
Paperback
$10.49 Prime

**Figure 2.1:** Amazon recommends related items to users.

The core of CF approach is computing similarity between subjects. Depending on the subject of focus, there are three major categories of CF approach:

1. **User-based**: Finds other users with similar tastes as the target user and recommend what they liked.

2. **Item-based**: Finds other items that are similar to the ones that the target user has liked and recommend them.

3. **Model-based**: Uses machine learning algorithms to develop a model to predict the target user's preference.

The first CF approach invented was the user-based algorithm. In fact, the term *collaborative filtering* comes from the process of user-based solution in which the opinions of other like-minded users are significantly valued. Years later, researchers at Amazon came up with a modified version that works better in the application scenario at that time, which is the item-based approach. Meanwhile, researchers in the RS field found that some machine learning techniques such as *Bayesian network*, *clustering* and *Association Rule Learning* produce reasonably good result when applied on RSs. These techniques view CF process as calculating the expected value of a user prediction, they are categorized as model-based approach.

In this section we first explore some common similarity computation methods used in the user-based and item-based algorithms, and then explain the three approaches one by one.

### 2.2.1 Similarity Computation

Similarity computation is the most important step in user-based and item-based CF algorithms because it forms the basis for filtering items. Similarity computation primarily relies on the user's rating history for items. A rating reflects a user's feedback on the experience with an item. There are two types of feedback: *explicit feedback* (illustrated in Fig. 2.3) and *implicit feedback* (illustrated in Fig. 2.3). Explicit feedback often comes in the form of rating (Amazon) or thumbs-up/down selection (YouTube). Users understand that the feedback provided is interpreted as a judgment of the quality of item. Conversely, implicit feedback comes in many forms. For instance the number of mouse clicks of web links or plays of songs, can all be counted as implicit feedback. Users providing implicit feedback are not trying to do assessment, rather, they are just satisfying their need. Yet the number of performing an action regarding items can be interpreted as an indirect indication of users' assessment on items. For example, if a person is browsing a web page full of short videos, and he watches a video of a playful panda over and over again, then it can be said that this person has a strong preference towards the panda video than any other video on the same page.

There can be a variety of forms of feedback, for the sake of uniformity, the term **rating** is used from now to represent any form of feedback. So, if a person gives 5 stars to a movie, we say this person's rating of the movie is 5 stars; if he reads three news out of eight news provided, we say his ratings of the three news are 1, and the rest five news are 0. In addition, the term **user profile** is used to describe the collection of ratings a user has in the past. So if the person in the news-reading example read the first, second and sixth news, then his user profile can be represented by a vector of ratings (1 1 0 0 0 1 0 0).

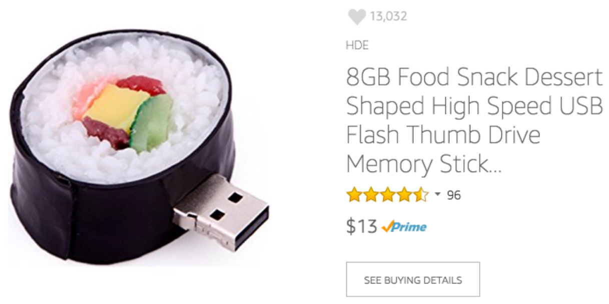
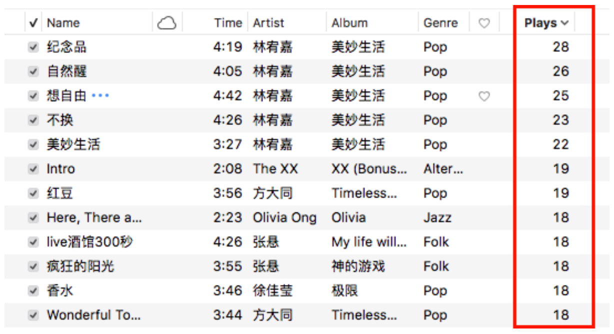

13,032
HDE
8GB Food Snack Dessert Shaped High Speed USB Flash Drive Memory Stick...
96
$13 Prime
SEE BUYING DETAILS

**Figure 2.2:** Explicit feedback: Products on Amazon.com are rated on a scale of 1 to 5 stars.



| ✓ | Name | ☁ | Time | Artist | Album | Genre | ♡ | Plays |
|---|---|---|---|---|---|---|---|---|
| ✓ | 纪念品 | | 4:19 | 林宥嘉 | 美妙生活 | Pop | | 28 |
| ✓ | 自然醒 | | 4:05 | 林宥嘉 | 美妙生活 | Pop | | 26 |
| ✓ | 想自由 ••• | | 4:42 | 林宥嘉 | 美妙生活 | Pop | ♡ | 25 |
| ✓ | 不换 | | 4:26 | 林宥嘉 | 美妙生活 | Pop | | 23 |
| ✓ | 美妙生活 | | 3:27 | 林宥嘉 | 美妙生活 | Pop | | 22 |
| ✓ | Intro | | 2:08 | The XX | XX (Bonus... | Alter... | | 19 |
| ✓ | 红豆 | | 3:56 | 方大同 | Timeless... | Pop | | 19 |
| ✓ | Here, There a... | | 2:23 | Olivia Ong | Olivia | Jazz | | 18 |
| ✓ | live酒馆300秒 | | 4:26 | 张悬 | My life will... | Folk | | 18 |
| ✓ | 疯狂的阳光 | | 3:55 | 张悬 | 神的游戏 | Folk | | 18 |
| ✓ | 香水 | | 3:46 | 徐佳莹 | 极限 | Pop | | 18 |
| ✓ | Wonderful To... | | 3:44 | 方大同 | Timeless... | Pop | | 18 |

**Figure 2.3:** Implicit feedback: iTunes records number of play of songs.

Although there are a number of ways to measure the similarity, three common ones are described in this section. These three are *Euclidean distance*, *Pearson correlation coefficient* and *Cosine similarity*.

To better illustrate each method, a movie example (Table 2.7) is used to show the difference between the three similarity measurements. There are three people and two movies in this example. The ratings of the two movies are given in Table 2.7. The ratings range from 1 being poor to 5 being great.

| | **Movie 1** | **Movie 2** |
|---|---|---|
| **Amy** | 3 | 4 |
| **Bob** | 2 | 5 |
| **Chris** | 4 | 2 |

**Table 2.7:** Three users rate two movies on a scale of 1 to 5.

### 2.2.1.1 Euclidean Distance

Recall the classic *Pythagorean Theorem* that computes the hypotenuse of a triangle, it can also be used to calculate the distance between two points. In an $n$-dimensional Euclidean space $\mathbb{R}^n$, if $p = (p_1, p_2, \dots, p_n)$ and $q = (q_1, q_2, \dots, q_n)$, then the Euclidean distance from $p$ to $q$ is given by the Pythagorean formula:

$$\begin{aligned}\text{Distance}(p,q) &= \sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\cdots+(p_n-q_n)^2} \\ &= \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}\end{aligned} \tag{2.4}$$

In the movie example above, two movies define a 2-dimensional space. Each person's rating can be viewed as a point in the space. The first coordinates are the ratings of movie 1, the second coordinates are the ratings of movie 2. Hence the distance between any two people can be calculated using Equation 2.4. The result indicates the similarity of three people's movie taste. The shorter the distance value, the more similar they are. For instance, to see whether Bob or Chris has a more similar taste to Amy's, the Euclidean distance between the two people and Amy can tell the answer:

$$Distance(Amy, Bob) = \sqrt{(3-2)^2 + (4-5)^2} = \sqrt{1^2 + 1^2} \approx 1.414$$

$$Distance(Amy, Chris) = \sqrt{(3-4)^2 + (4-2)^2} = \sqrt{1^2 + 2^2} \approx 2.236$$

The conclusion is that Bob is more similar to Amy than Chris is, since 1.414 < 2.236.

Now suppose there are more movies and users in the example, and this time not all people have watched all movies. Therefore, there are some missing ratings (Table 2.8).

| | **Movie 1** | **Movie 2** | **Movie 3** | **Movie 4** | **Movie 5** | **Movie 6** | **Movie 7** |
|---|---|---|---|---|---|---|---|
| **Amy** | 3 | 4 | - | 2 | 3 | 5 | 4 |
| **Bob** | 1 | 5 | 1 | - | 5 | - | 1 |
| **Chris** | 4 | 5 | 5 | 4 | - | 5 | 4 |
| **David** | - | - | - | 1 | 4 | - | - |

**Table 2.8:** Four users rate seven movies.

According to Equation 2.4, when calculating the Euclidean distance between two people, only ratings of items that they both have rated are used. As a result, in some cases, it skews the distance measurement. For example, the Amy-Chris distance computation uses five movie ratings, but the Amy-David distance uses only two. The former distance is in 5-dimension whereas the latter one is in 2-dimension. Such situation is referred as *shared zeros*. In summary, Euclidean distance works best when there are no missing values.

In a real application scenario, it is unlikely that there are only seven items in a system. The *Internet Movie Database (IMDb)*, an online database of films and television programs, has approximately 4.1 million titles [17]. Computing the similarity between two random users on IMDb would be way harder than computing an Amy-Who similarity in the small example above. Because chances are that a user has only rated a few dozens of movies out of 4.1 million. In other words, the data is **sparse** as the percentage of non-zero attributes (rating) is low. In this situation, when two users are compared together, it is likely that they will have shared zeros in common. However, a RS rule of thumb is that shared zeros should not be used to calculate similarity. Involving shared zeros in calculation is a major disadvantage of Euclidean distance. The other two methods that are described later avoid this shared zeros issue.

#### 2.2.1.2 Cosine Similarity

The cosine similarity between two users *A*, *B* is defined as:

$$\text{sim}_{Cosine}(A,B) = \frac{\overrightarrow{A} \cdot \overrightarrow{B}}{\|\overrightarrow{A}\| \times \|\overrightarrow{B}\|} = \frac{\sum_{i\in I} r_{A,i} r_{B,i}}{\sqrt{\sum_{i\in I}(r_{A,i})^2}\sqrt{\sum_{i\in I}(r_{B,i})^2}} \tag{2.5}$$

where $I$ is the set of items that both users have rated, $r_{A,i}$ means rating of user A for item $i$.

The cosine similarity ranges from 1 meaning perfectly similar, to -1 meaning perfectly dissimilar, with 0 meaning two users are not correlated at all.

Oftentimes, it is used to find the angle between two vectors. Indeed, every user profile can be viewed as a vector of ratings. The degree of angle between two user profiles can represent how close the two profiles are similar to each other. The angle ranges from 0 being identical to 180 being opposite, that is 1 to -1 after taking the cosine function.

This way of thinking helps us to visualize the process of comparing two users and understand the meaning of values. Imagine a user profile is compared with itself, i.e., a user is compared with himself/herself. The answer is obvious 1 meaning perfectly similar, since the two subjects are identical. The picture of this is two vectors overlapping each other completely, so the angle between is 0, therefore the result is $\cos(0) = 1$. A perfectly dissimilar case is just two vectors going exact opposite directions, thus the angle between is 180 and of course $\cos(180) = -1$. The reason why 0 is the indication of uncorrelated property is because it happens when two user profiles are orthogonal to each other, meaning they just have totally different opinions, but not opposing each other.

The same movie example can be used to demonstrate how cosine similarity ignores shared zeros. To compute the Amy-Chris similarity and the Amy-David similarity, the first step is to convert the user profiles to vectors and replace the missing values with 0.

$$\overrightarrow{U_{Amy}} = (3,4,0,2,3,5,4) \quad \overrightarrow{U_{Chris}} = (4,5,5,4,0,5,4) \quad \overrightarrow{U_{David}} = (0,0,0,1,4,0,0)$$

then

$$\text{sim}_{Cosine}(Amy, Chris) = \frac{U_{Amy} \cdot U_{Chris}}{\left\|U_{Amy}\right\| \times \|U_{Chris}\|}$$

$$= \frac{12+20+0+8+0+25+16}{\sqrt{3^2+4^2+0^2+2^2+3^2+5^2+4^2} \times \sqrt{4^2+5^2+5^2+4^2+0^2+5^2+4^2}} = 0.8217$$

and similarly

$$\begin{aligned}\text{sim}_{Cosine}(Amy, David) &= \frac{U_{Amy} \cdot U_{David}}{\left\|U_{Amy}\right\| \times \left\|U_{David}\right\|} \\ &= \frac{0+0+0+2+12+0+0}{\sqrt{3^2+4^2+0^2+2^2+3^2+5^2+4^2} \times \sqrt{0^2+0^2+0^2+1^2+4^2+0^2+0^2}} = 0.3820\end{aligned}$$

The results indicate that Amy and Chris are similar movie viewers. Because the ratings in this example are non-negative, the range of cosine similarity is $[0, 1]$.

### 2.2.1.3 Pearson Correlation Coefficient

Another potential issue that causes problem for RS is the variability of users' rating behavior. Everyone has their criteria in rating a item. For example, in the movie example Bob and Chris expose different rating behavior. Bob's opinion on movies is binary, giving either 1s or 5s. On the other hand, Chris seems to be very generous in giving ratings, all of his ratings are 4s and 5s. It is hard for us to say that Bob's '1' means the same as Chris's '1'. In this case, the similarity result may not reflect the "true" similar ratings. This situation is termed *grade-inflation* in data mining. A way to fix this problem is to use the *Pearson correlation coefficient*.

The Pearson correlation coefficient between two users $A$, $B$ is defined as:

$$\text{sim}_{Pearson}(A, B) = \frac{\sum_{i \in I}(r_{A,i} - \overline{r_A})(r_{B,i} - \overline{r_B})}{\sqrt{\sum_{i \in I}(r_{A,i} - \overline{r_A})^2}\sqrt{\sum_{i \in I}(r_{B,i} - \overline{r_B})^2}} \tag{2.6}$$

where $\overline{r_A}$ stands for user A's average rating. The value of Pearson correlation ranges from 1 meaning perfect agreement, to -1 meaning perfect disagreement.

A new example (Table 2.9) is given, along with chart explanation (Fig. 2.9) to help to understand the formula. In the new example the users are allowed to give half value rating (e.g. 1.5, 2.5, . . . ).

| | **Movie 1** | **Movie 2** | **Movie 3** | **Movie 4** | **Movie 5** |
|---|---|---|---|---|---|
| **Ben** | 3 | 4 | 1 | 5 | 2 |
| **Clark** | 4 | 4.5 | 3 | 5 | 3.5 |

**Table 2.9:** Two users rate five movies.

It can be seen that the two users have different rating behaviors. Ben's ratings are between 1 and 5, whereas Clark's ratings are between 3 and 5. However, plotting Ben's ratings against Clark's results in a linear line (Fig. 2.4). This indicates that Ben and Clark have perfect agreement. Both of them give Movie 4 the highest rate, and then Movie 2, Movie 1, and so on. In this case, Ben's '1' is equivalent to Clark's '3'.

A linear line means that the Pearson correlation coefficient between them is 1. To verify that computationally, the values are substituted into Equation 2.6 to check the answer.

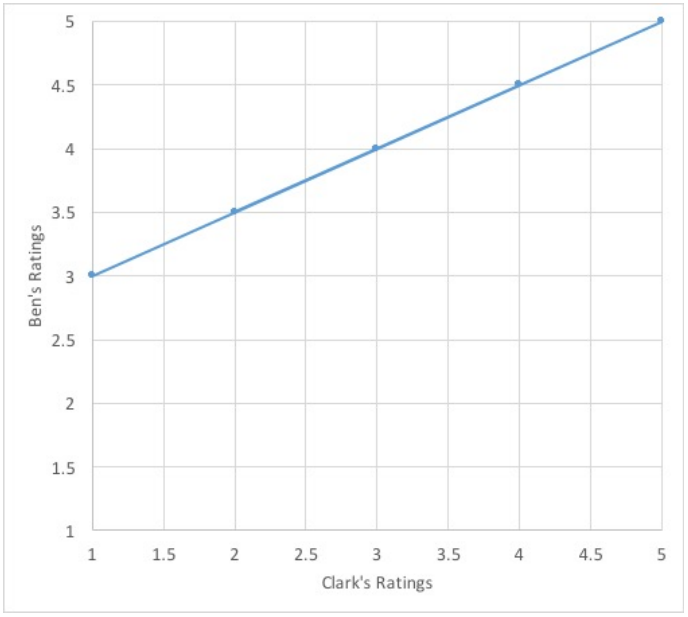


**Figure 2.4:** The linear line indicates that Ben and Clark have perfect agreement.

First compute $\overline{r_{Ben}}$ and $\overline{r_{Clark}}$:

$$\overline{r_{Ben}} = \frac{1}{5}(3 + 4 + 1 + 5 + 2) = 3$$

$$\overline{r_{Clark}} = \frac{1}{5}(4 + 4.5 + 3 + 5 + 3.5) = 4$$

then the numerator is the sum of the products of the differences of each rating and mean rating given by each user

$$\sum_{i \in I}(r_{A,i} - \overline{r_A})(r_{B,i} - \overline{r_B}) = (3-3)(4-4) + (4-3)(4.5-4) + (1-3)(3-4) + (5-3)(5-4) + (2-3)(3.5-4) = 5$$

and the denominator is:

$$\sqrt{\sum_{i\in I}(r_{A,i}-\overline{r_A})^2}\sqrt{\sum_{i\in I}(r_{B,i}-\overline{r_B})^2} = \sqrt{0^2+1^2+(-2)^2+2^2+(-1)^2}\times\sqrt{0^2+0.5^2+(-1)^2+1^2+(-0.5)^2}$$

$$= \sqrt{10}\times\sqrt{2.5} = 5$$

thus the Pearson correlation coefficient between Ben and Clark is

$$\mathrm{sim}_{Pearson}(Ben, Clark) = \frac{5}{5} = 1$$

The graph of a perfect disagreement is reversing the perfect agreement diagonal line horizontally.

In conclusion, the choice of similarity computation should depend on the data. If it is dense, i.e., only a few missing values, then use Euclidean distance; If it is sparse, use Cosine similarity; If it exhibits grade-inflation, then consider Pearson correlation coefficient. In reality, Cosine similarity and Pearson correlation coefficient are most widely used.

### 2.2.2 User-based Collaborative Filtering

User-based CF collects user profiles (ratings by users) as input and computes the similarity between users. Given an target user A, it can predict A's rating of items that A has not rated before based on the most similar users to A.

As mentioned in the previous section, the two most common similarity computation methods used in user-based CF are *Cosine similarity* and *Pearson correlation coefficient*. Both methods compare a user's profile with another one's and returns a similarity score. To serve a target user A, the user-based system will calculate the similarity score between A and all other users, record the similarity score from each comparison, and select $K$ users whose scores are within the top $K$. These $K$ users are called the *neighbors* of user A.

Usually neighbors have feedback for items that the target user has not rated. The neighbors' opinions can be used to predict A's rating of unrated items. For example, if the two most similar neighbors have high positive rating for an item $i$, the user-based system will trust them and recommend $i$ to user A. The formula that the system applies to predict A's rating on an item $i$ is given by:

$$\mathrm{Rating}(A,i) = \overline{r_A} + \frac{\sum_{B\in N} sim(A,B)\times(r_{B,i}-\overline{r_B})}{\sum_{B\in N} sim(A,B)} \tag{2.7}$$

where $i$ is an item unrated by user A, $N$ is the set containing the top $K$ neighbors from the previous step. For each neighbor $B$, this formula calculates $B$'s average rating and subtracts it from B's rating on $i$ to obtain $B$'s bias. The bias is then weighted by his/her similarity score. The total weighted bias is the sum of all neighbors' weighted bias divided by the sum of all neighbors' similarity score, it can be positive or negative. In the end, the result is added to the mean rating of A, the sum of which is the final predicted rating of user A on item $i$.

The prediction process is repeated for all items in the stock that the target user has no feedback for. In the end, these items are sorted by their associated predicted rating values. The top $K$ items are returned to the target user as the final recommendations. Notice that even though $K$ is used in both the number of neighbors and the number of recommendations, it really means an arbitrary number that the developer thinks that will produce the best result for the RS. The number of neighbors and recommendations do not have to be the same. It can be 10 neighbors and 20 recommendations.

Another movie example is given below (Table 2.10) to illustrates the process of a user-based RS using the cosine similarity method. There are four users and five movies in the example. All users have watched all movies and rated them. Now a new user A registers the system and gives ratings to all movies except Movie 5 because A has not watched it. The goal is to predict A's rating for Movie 5.

| | **Movie 1** | **Movie 2** | **Movie 3** | **Movie 4** | **Movie 5** |
|---|---|---|---|---|---|
| **1** | 3 | 4 | 5 | 1 | 4 |
| **2** | 2 | 5 | 3 | 3 | 5 |
| **3** | 4 | 2 | 3 | 2 | 2 |
| **4** | 5 | 4 | 4 | 5 | 3 |
| **A** | 4 | 3 | 5 | 3 | ? |

**Table 2.10:** Movie ratings for user-based model example.

As mentioned in Section 2.2.1.2, in cosine similarity computation missing values are replaced with 0. Applying Equation 2.5, the cosine similarity score between user A and other users are calculated:

| **User A** | **User 1** | **User 2** | **User 3** | **User 4** |
|---|---|---|---|---|
| 1.0 | 0.82706556 | 0.72111678 | 0.92032578 | 0.91438291 |

**Table 2.11:** User-based model example: cosine similarity computation.

The result (Table 2.11) shows that user 3 and 4 are more similar with A, so they are identified as neighbors of A. The next step is to compute Rating($A, Movie5$) using the prediction formula. The average rating given by *User A* is:

$$\overline{r_A} = \frac{4+3+5+3}{4} = 3$$

The average rating given by *User 3* and *User 4* are:

$$\overline{r_3} = \frac{4+2+3+2+2}{5} = 3 \quad \overline{r_4} = \frac{5+4+4+5+3}{5} = 4.2$$

Then *User A*'s predicted rating on Movie 5 is:

$$\begin{aligned} \text{Rating}(A, Movie5) &= \overline{r_A} + \frac{sim(A,3) \times (r_{3,Movie5} - \overline{r_3}) + sim(A,4) \times (r_{4,Movie5} - \overline{r_4})}{sim(A,3) + sim(A,4)} \\ &= 3 + \frac{0.92 \times (2-3) + 0.91 \times (3-4.2)}{0.92 + 0.91} = 2.1 \end{aligned}$$

Hence, the system thinks user A will rate 2.1 for Movie 5.

Imaging there are more movies unrated by A in the data set. In that case, the user-based system will repeat the process above for all unrated movies. Then each unrated movie will be given a predicted rating by A ranging from 1 to 5. At the end, movies with high predicted rating will be recommended to A. This is how user-based system works to recommend items to users.

### 2.2.3 Item-based Collaborative Filtering

Item-based CF was invented by people at Amazon.com, Inc. in 1998 to solve two major potential problems of the user-based system [14]:

1. **Sparsity**: If the item sets are so large that even target users may have interacted with less than 1% of the items. In this case, a user-based algorithm may fail to find nearest neighbors and therefore fail to recommend to the target user.

2. **Scalability**: The number of computation in user-based algorithm grows as the number of both users and items grow. With an enormous number of users and items, the system will suffer scalability problems.

Rather than finding similar users, the item-based algorithm detects items that are similar to items rated by the target user. At the end, the $K$ most similar items are recommended to the user. It is similar to content-based filtering as both methods compute the similarity between items. But unlike content-based filtering where the features of items are used to calculate the similarity, in item-based CF the past ratings from other users form the basis of similarity computation.

The similarity between items can be calculated using the same methods used to find similar neighbors, i.e., Cosine similarity and Pearson correlation coefficient, with slightly formula change. In used-based approach the similarity computation looks at the set of items rated by both users, in item-based method the similarity computation focuses on the set of users that have rated both items. For example the cosine similarity between items $i$ and $j$ is given by:

$$\text{sim}_{Cosine}(i,j) = \frac{\vec{i} \cdot \vec{j}}{\| \vec{i} \| \times \| \vec{j} \|} = \frac{\sum_{u \in U} r_{u,i} r_{u,j}}{\sqrt{\sum_{u \in U} (r_{u,i})^2} \sqrt{\sum_{u \in U} (r_{u,j})^2}} \tag{2.8}$$

where $U$ is the set of users that have rated both item $i$ and $j$, and $r_{u,i}$ denotes user $u$'s rating for item $i$. The difference can be seen easily compared with the cosine similarity in user-based approach (Equation 2.5).

There are two ways to rank the recommended items once similarity computation is finished. One is to use solely the similarity information of all input items, without weighting the users' rating. The other one is to apply the *weighted sum* formula which weights the users' rating.

The first method is simple. For each item $i$ in the total item set $I$, the similarity scores between $i$ and each item $j$ in the target user's rated item set $J$ are added together. At the end, $I$ is sorted by the accumulated similarity score. The top $K$ results are recommended to the user.

The second method predicts the rating of a user A on item $i$ using the *weighted sum* formula which is given by:

$$\text{Rating}(A,i) = \frac{\sum_{j \in S} sim(i,j) \times (r_{A,j})}{\sum_{j \in S} sim(i,j)} \tag{2.9}$$

where $S$ is the set of all similar items with item $i$.

The same movie example is used again to demonstrate the process of an item-based model. Recall that in the example there are four existing users and one new user and there are five movies in total. The goal was to predict user A's rating on Movie 5 (see Table 2.10). In this example, similarity computation is done using cosine similarity.

The first step is to find similar movies to Movie 5. To do so, every movie is compared with Movie 5 and only ratings by users who have rated both Movie 5 and the other movie are considered when computing similarity. For example, the similarity between Movie 1 and 5 is calculated as follows:

First, ratings of each movie is transformed into a vector, shown as $I_{Movie1}$ and $I_{Movie5}$.

$$\overrightarrow{I_{Movie1}} = (3,2,4,5,4) \quad \overrightarrow{I_{Movie5}} = (4,5,2,3,0)$$

The missing value $r_{A,Movie5}$ is replaced by 0. Next, the vectors are plugged into the cosine similarity formula:

$$\text{sim}_{Cosine}(Movie1, Movie5) = \frac{\overrightarrow{I_{Movie1}} \cdot \overrightarrow{I_{Movie5}}}{||\overrightarrow{I_{Movie1}}|| \times ||\overrightarrow{I_{Movie5}}||}$$
$$= \frac{3 \times 4 + 2 \times 5 + 4 \times 2 + 5 \times 3 + 4 \times 0}{\sqrt{3^2 + 2^2 + 4^2 + 5^2 + 4^2} \times \sqrt{4^2 + 5^2 + 2^2 + 3^2 + 0}} = 0.732$$

Doing this for Movies 1-5, we get:

| Movie 5 | Movie 1 | Movie 2 | Movie 3 | Movie 4 |
|---|---|---|---|---|
| 1.0 | 0.732 | 0.927 | 0.787 | 0.746 |

**Table 2.12:** Cosine similarity of all movies in comparison with Movie 5.

The next step is to select items to recommend to the user. As mentioned before, there are two ways to do so, direct method and weighted sum.

### 2.2.3.1 Direct Method

In the direct method, the system will compute a pairwise similarity for every item in the stock and use that to generate a list of recommended items. The pairwise similarity can be obtained by performing the calculation above for every movie in the example. The similarity scores are shown in Table 2.13.

| | Movie 1 | Movie 2 | Movie 3 | Movie 4 | Movie 5 |
|---|---|---|---|---|---|
| **Movie 1** | 1 | 0.886 | 0.952 | 0.932 | 0.732 |
| **Movie 2** | 0.886 | 1 | 0.939 | 0.897 | 0.927 |
| **Movie 3** | 0.952 | 0.939 | 1 | 0.866 | 0.787 |
| **Movie 4** | 0.932 | 0.897 | 0.866 | 1 | 0.746 |
| **Movie 5** | 0.731 | 0.927 | 0.787 | 0.746 | 1 |

**Table 2.13:** Pairwise cosine similarity results of all movies.

Next, for each movie that A has rated, the system will check its similarity score with other movies and accumulate the value for each movie in the stock. In this example, since A has rated Movies 1-4, the system will look at the first to forth rows of the pairwise table and aggregate the values for each movie. Suppose there is a list that keeps track of the final accumulated similarity score for each movie and it is initialized to all 0s before using the direct method.

$$[0, 0, 0, 0, 0]$$

Then for the first movie A has rated, Movie 1, the system scan the pairwise similarity score of Movie 1 and add each score to the final rating list. The updated list is now:

$$[1, 0.886, 0.952, 0.932, 0.732]$$

which is just the first row of the table. And for the second movie A has rated, Movie 2, the system repeats the same work, adding the second row of the pairwise table to the list. The list becomes:

$$[1.886, 1.886, 1.891, 1.829, 1.659]$$

The same process is repeated two more times, one for Movie 3 and one for Movie 4. The final list is:

$$[3.769, 3.722, 3.757, 3.695, 3.192]$$

Thus, the accumulated similarity score of Movie 5 is 3.192. If the system ignored the fact that A has watched the first four movies and recommend all movies to A, then the order of the movies it recommends would be Movie 1, 3, 2, 4 and 5.

#### 2.2.3.2 Weighted Sum

In user-based model, after user similarity is computed an arbitrary number of users are selected as neighbors of the target user. In item-based model with weighted sum method, usually there is a threshold value $\epsilon$ that signals that items whose similarity score greater than $\epsilon$ are said to be the *neighbors* of the target item. The set of similar items to item $i$, $S_i$, is defined as:

$$S_i = \{j \in I | \text{sim}_{Cosine}(i, j) > \epsilon\}$$

where $I$ is the set of all items.

If $\epsilon$=0.75, then the table shows that $S_{Movie5} = \{Movie2, Movie3\}$. and the predicted rating of Movie

5 by user A using the weighted sum formula (Equation 2.9) is:

$$\begin{aligned}\text{Rating}(A, Movie5) &= \frac{sim(Movie5, Movie2) \times r_{A,Movie2} + sim(Movie5, Movie3) \times r_{A,Movie3}}{sim(Movie5, Movie2) + sim(Movie5, Movie3)} \\ &= \frac{0.927 \times 3 + 0.787 \times 5}{0.927 + 0.787} = 3.918\end{aligned}$$

Hence, the system predicts that user A will rate 3.918 for Movie 5 when $\epsilon = 0.75$. This is different from the result obtained through user-based approach which was a 2.1 rating for Movie 5.

### 2.2.4 Association Rule Learning

Many model-based CF methods use data mining techniques to solve recommendation problems. One of them is *association rule learning* and it is useful in discovering rules hidden in large database of consumption transactions. It is used in many application domains. For example in E-learning systems, association rule learning is used to analyze students' course selection behavior [11]. Therefore, it is an option for implementing a course recommender system in our project.

In association rule learning, the uncovered relationships are represented in the form of **association rules** or sets of **frequent items**. A classic application of association rule learning is the tale of diapers and beers, in which the technique is used to analyze shopping items that are brought together by looking at market basket transactions. At the end it revealed a surprising rule which shows customers who bought diapers and milk together are also likely to buy beer. Below is a short example showing how this association is discovered.

First of all, we need to ensure the raw data is in a useful form, ready to be processed. In the case of merchandise, it has to be in the form of transactions (Table 2.14). Each transaction is called **itemset**, meaning a set of items brought together.

**Definition 2.2.1.** *Let $I = \{i_1, i_2, ..., i_m\}$ be the set of all items in a market basket data and $T = \{t_1, t_2, ..., t_n\}$ be the set of all transactions. Each transaction $t$ contains a subset of items from $I$.*

| **TID** | **Items** |
|---|---|
| **1** | Milk, Soda, Apple |
| **2** | Apple, Beer, Diaper |
| **3** | Apple, Beer, Milk, Diaper |
| **4** | Beer, Diaper, Milk, Soda |
| **5** | Diaper, Milk, Soda |

| **TID** | **Items** |
|---|---|
| **6** | Beer, Soda |

**Table 2.14:** Market basket transactions for association analysis.

**Definition 2.2.2.** *An association rule describes the associative relationship between two itemsets. For itemsets $i_a$ and $i_b$, the rule $\{i_a \rightarrow i_b\}$ indicates that if people bought $i_a$, then they are likely to buy $i_b$.*

There are multiple association rules that can be extracted from the market basket table above, the rule $\{Apple \rightarrow Milk\}$ is one example. The goal is to discover significant association rules from the data. Two measurements are used to determine the association between items.

#### 2.2.4.1 Support

For a given itemset $i$, its support is defined as the proportion of transactions in which $i$ appears.

$$\text{Support}(i) = \frac{\text{number of } i}{\text{total number of items}} \tag{2.10}$$

In the market transaction example, the support of *{Beer}* is 4 out of 6, or 66.7%. And the support of *{Diaper, Beer}* is 3 out of 6, or 50%. It shows how popular an item is among all. Sometimes there is a certain level of support that impacts the total sales significantly, such a level is used as **support threshold** to filter out less popular itemsets. Given a support threshold $s$, then the itemsets with support values greater than $s$ are called *frequent itemsets*. A frequent $N$-itemset is a frequent itemset of length $N$. For example, if the support threshold is 50%, the frequent itemsets are itemsets that appear in at least 50% of the basket transactions.

#### 2.2.4.2 Confidence

The confidence of an association rule $\{i_a \rightarrow i_b\}$ measures how likely the itemset $i_b$ is purchased when the itemset $i_a$ is purchased. It is measured by the proportion of transactions with itemset $i_a$, in which itemset $i_b$ also appears. In other words, it is calculated as:

$$\text{Confidence}(\{i_a \rightarrow i_b\}) = \frac{\text{Support}(\{i_a, i_b\})}{\text{Support}(\{i_a\})} \tag{2.11}$$

For example, the confidence of the association rule *Beer* $\rightarrow$ *Diaper* is Confidence($\{Diaper \rightarrow Beer\}$) = $\frac{\text{Support}\{Diaper,Beer\}}{\text{Support}\{Diaper\}} = \frac{3/6}{4/6} = 75\%$.

#### 2.2.4.3 Finding Association Rules

Generally association rule learning algorithm decomposes the problem into two subtasks:

1. **Frequent Itemset Generation**, which finds all itemsets with support $\geq s$, where $s$ is the support threshold.

2. **Rule Generation**, which extracts all rules with confidence $\geq c$, where $c$ is the confidence threshold. These itemsets are called *strong rules*.

Finding frequent itemsets is often more computationally expensive than generating strong rules. Take the market example mentioned earlier as example, if the market owner wants to run an association analysis on all the products in the market, he or she would need to calculate the support values for every possible itemset. In reality, a market would not only sell 5 products. But even for just 10 products, which is still far away from a practical value, there are 1,023 different possible itemsets! Apparently a huge amount of computation is required to exhaustively test for every itemset if the dataset is large.

The visualization of the enumeration of itemsets can be shown in a lattice structure. The item lattice for 5 products $I = \{a, b, c, d, e\}$ (Fig. 2.5) shows that there are five layers and 31 possible itemsets in total. For a dataset with $n$ items, there are $2^n - 1$ possible itemsets.

#### 2.2.4.4 The *Apriori* Algorithm

The exponential growth of candidate itemsets induces great difficulties to run an association analysis in terms of the amount of computation. Fortunately, the number of candidate can be reduced using the *Apriori* principle. *The Apriori principle* states that any subset of a frequent itemset must be frequent.

The idea of *Apriori* principle can be illustrated on the same example of items $I = \{a, b, c, d, e\}$. If itemset $\{c, d, e\}$ is frequent, then any transaction that contains subsets of $\{c, d, e\}$ must also be frequent (Fig. 2.6). Therefore, any itemset that contains $\{c, d\}$, $\{c, e\}$, $\{d, e\}$, $\{c\}$, $\{d\}$ and $\{e\}$, is frequent. All of these frequent itemsets are shaded in the figure. The inverse of this conditional statement is also true: If an itemset is infrequent, then all of its supersets must be infrequent as well. By this principle, once an itemset is found to be infrequent, any transaction that contains its supersets can be pruned

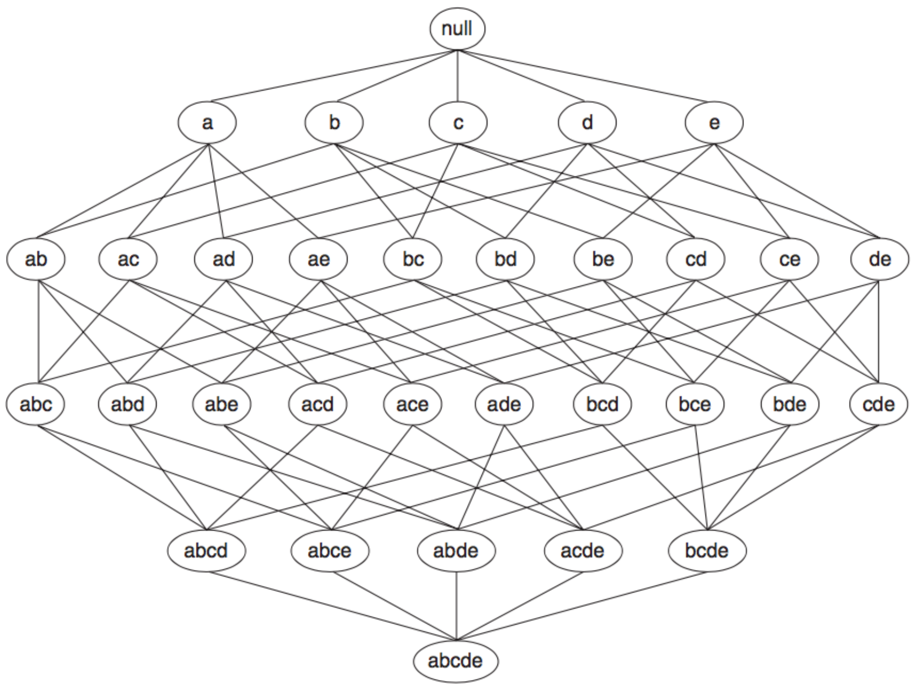


**Figure 2.5:** An item lattice illustrating all $2^5 - 1$ itemsets for $I = \{a, b, c, d, e\}$.

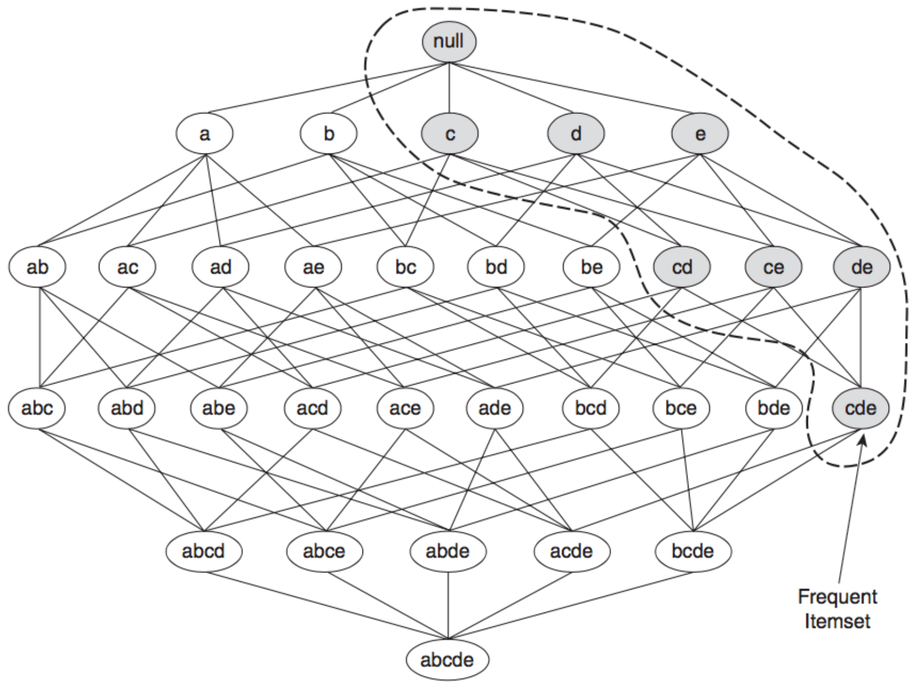


**Figure 2.6:** The *Apriori* principle: Subsets of the frequent itemset $\{c, d, e\}$ must be frequent.

immediately from the lattice. In practice, the itemsets are reduced based on the support measure, therefore this strategy is also known as *support-based pruning* [15]. For example, if itemset $\{a, b\}$ is found to be infrequent because its support is lower than a support threshold, then all the nodes containing the supersets of $\{a, b\}$, from $\{a, b, c\}$, $\{a, b, d\}$ and $\{a, b, e\}$ all the way up to $\{a, b, c, d, e\}$, are pruned (Fig. 2.7). The remaining itemsets are the frequent itemsets.

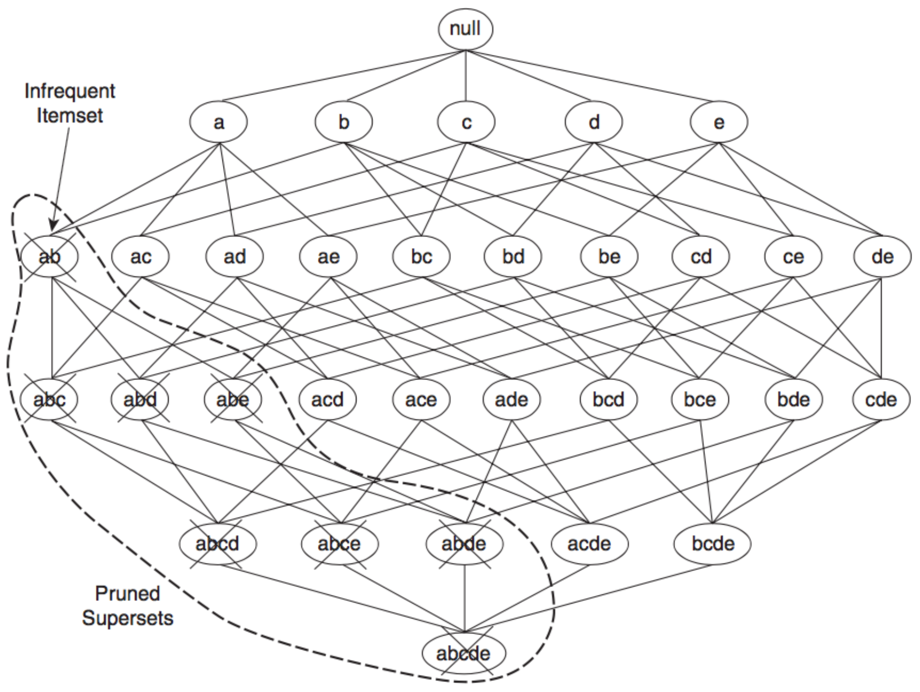


**Figure 2.7:** Support-based pruning: Nodes containing the supersets of an itemset with low support are pruned.

### 2.2.4.5 Rule Generation

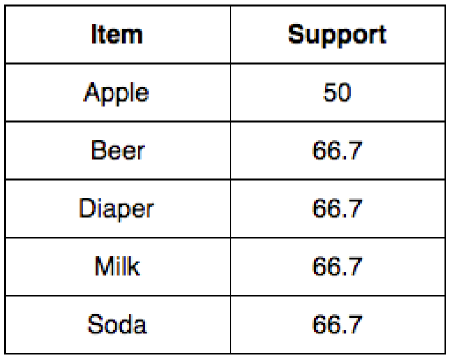
Given a frequent itemset $Y$, its association rules are all non-empty subsets $X \subset Y$ such that $X \rightarrow Y - X$ satisfies the minimum confidence requirement. In frequent itemset generation several items can exponentially generate itemsets, similarly association rule generation has the limitation of expensive computation due to massive enumeration. For example, based on the frequent 3-itemset $\{a, b, c\}$, there are six possible association rules:

1. $a \rightarrow b, c$
2. $b \rightarrow a, c$
3. $c \rightarrow a, b$
4. $a, b \rightarrow c$
5. $b, c \rightarrow a$
6. $a, c \rightarrow b$

In general, each frequent $k$-itemset can have $2^k - 2$ association rules [15]. To avoid enumerating unnecessary rules, the Apriori principle is applied again: Rules whose confidence is low than the threshold confidence will not be considered.

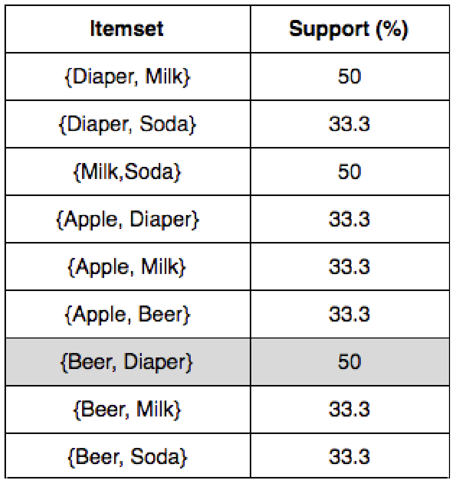
#### 2.2.4.6 Learning the Diaper and Beer Rule

Having introduced the concepts of frequent itemsets and rule generation, we put them together and apply them on the market transaction example (Table 2.14). Suppose the threshold values for support and confidence are set to be $s = 30$ and $c = 70$. The frequent itemsets satisfying both constraints from the market transactions will be:



| Item | Support |
|---|---|
| Apple | 50 |
| Beer | 66.7 |
| Diaper | 66.7 |
| Milk | 66.7 |
| Soda | 66.7 |

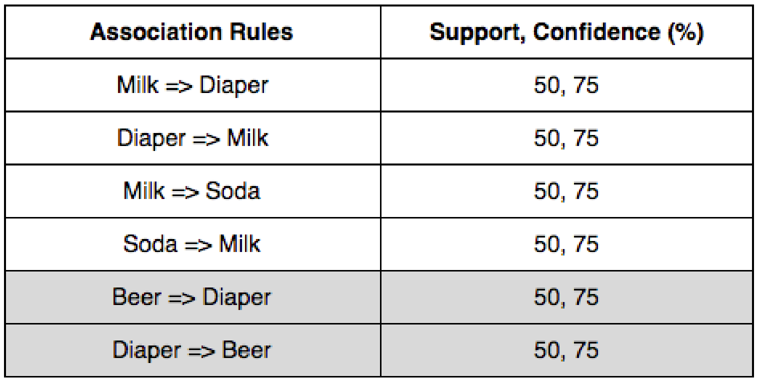
**Figure 2.8:** Frequent 1-Itemset mined from the market transaction data.



| Itemset | Support (%) |
|---|---|
| {Diaper, Milk} | 50 |
| {Diaper, Soda} | 33.3 |
| {Milk,Soda} | 50 |
| {Apple, Diaper} | 33.3 |
| {Apple, Milk} | 33.3 |
| {Apple, Beer} | 33.3 |
| {Beer, Diaper} | 50 |
| {Beer, Milk} | 33.3 |
| {Beer, Soda} | 33.3 |

**Figure 2.9:** Frequent 2-Itemset mined from the market transaction data.

Although frequent 3-itemset can be further generated, the frequent 2-itemset (Fig. 2.9) has already proven that Diaper and Beer are often bought together. From frequent-2-itemsets, the derived association rules are:



| Association Rules | Support, Confidence (%) |
|---|---|
| Milk => Diaper | 50, 75 |
| Diaper => Milk | 50, 75 |
| Milk => Soda | 50, 75 |
| Soda => Milk | 50, 75 |
| Beer => Diaper | 50, 75 |
| Diaper => Beer | 50, 75 |

**Figure 2.10:** Association rules derived from the frequent 2-itemset.

Two of the rules confirm the association between the consumption of diaper and beer.

# CHAPTER 3

# Implementation of the Course Recommender System

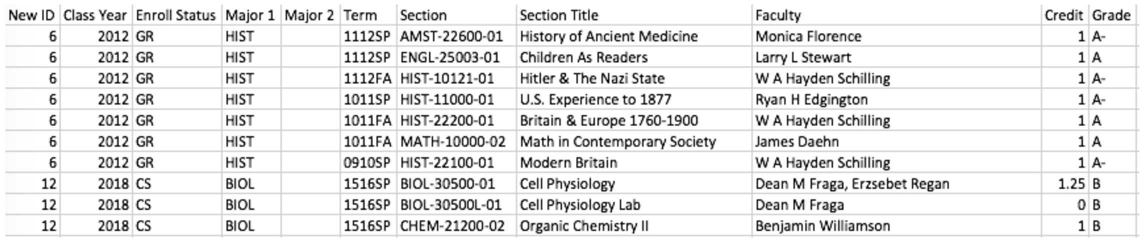
This chapter describes the data, the design and the implementation of the course recommender system (CRS) software using different recommendation approaches. The entire CRS is developed using Python 3.5. The main libraries used to process the data are: `Pandas`, `Numpy`, `Scipy` and `Sklearn`.

## 3.1 Data Collection

The data is obtained from the registrar office of the College of Wooster. It includes student information from class of 2010 to class of 2020 and their courses taken from 2009-10 spring semester to 2015-16 spring semester at the college. It is stored in a single CSV file which has the following data fields:

1. New ID (Student ID in database)
2. Enroll Status (Graduated, Withdrawn or Current student)
3. Major 1
4. Major 2
5. Term
6. Course Number
7. Course Title
8. Faculty
9. Credit
10. Grade

In the data received, the names of the students are not provided because of privacy protection. Instead, every student is assigned a unique ID which appears in the *New ID* field. It is worth noticing that the IDs are not all continuous and do not start with 1. Fig. 3.1 shows a screen shot of the head portion of the data. It shows that the first student record has ID 6 and the second student has ID 12. It also shows that each student's record spreads over several rows. Each row contains information of a course taken by the student.



| New ID | Class Year | Enroll Status | Major 1 | Major 2 | Term | Section | Section Title | Faculty | Credit | Grade |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2012 | GR | HIST | | 1112SP | AMST-22600-01 | History of Ancient Medicine | Monica Florence | 1 | A- |
| 6 | 2012 | GR | HIST | | 1112SP | ENGL-25003-01 | Children As Readers | Larry L Stewart | 1 | A |
| 6 | 2012 | GR | HIST | | 1112FA | HIST-10121-01 | Hitler & The Nazi State | W A Hayden Schilling | 1 | A- |
| 6 | 2012 | GR | HIST | | 1011SP | HIST-11000-01 | U.S. Experience to 1877 | Ryan H Edgington | 1 | A- |
| 6 | 2012 | GR | HIST | | 1011FA | HIST-22200-01 | Britain & Europe 1760-1900 | W A Hayden Schilling | 1 | A |
| 6 | 2012 | GR | HIST | | 1011FA | MATH-10000-02 | Math in Contemporary Society | James Daehn | 1 | A |
| 6 | 2012 | GR | HIST | | 0910SP | HIST-22100-01 | Modern Britain | W A Hayden Schilling | 1 | A- |
| 12 | 2018 | CS | BIOL | | 1516SP | BIOL-30500-01 | Cell Physiology | Dean M Fraga, Erzsebet Regan | 1.25 | B |
| 12 | 2018 | CS | BIOL | | 1516SP | BIOL-30500L-01 | Cell Physiology Lab | Dean M Fraga | 0 | B |
| 12 | 2018 | CS | BIOL | | 1516SP | CHEM-21200-02 | Organic Chemistry II | Benjamin Williamson | 1 | B |

**Figure 3.1:** A portion of the raw data.

## 3.2 Data Preprocessing

There are a few defects in the raw data that require fixing. First and foremost there are a lot of outdated course numbers. Before dive into this issue, it is essential to understand the course numbering system at the college. As shown in Fig. 3.2, the College of Wooster uses a five-digit course numbering system. The first four letters stand for the department or program abbreviation. The first three digits indicate the primary course number, followed by two digits representing the secondary course number, which shows whether there is a special focus for the course. In addition, Fig. 3.1 shows that in fact there are two extra digits at the end of a course number which represent the section number.

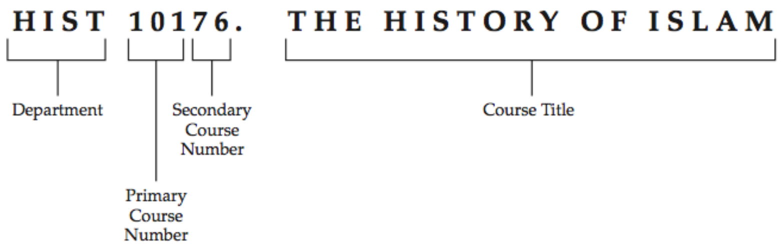


**Figure 3.2:** The structure of the course numbering system at the College of Wooster.

A RS keeps track of all the *unique* items in a system. In our CRS, items are the courses which are represented by course numbers. The raw data stores the complete course number, but in fact only the first four letter plus the first three digits are enough for differentiating courses. For instance, the two course numbers *AFST-10000-01* and *AFST-10000-02* both refer to the course *Intro to Africana Studies*, in CRS they are viewed as the same item. Thus, a solution to this is to replace all the complete course numbers with an abbreviated version. In this case, the two course numbers should be replaced by AFST-100.

Another issue with the course data is that lab sessions and lectures are separated but in CRS they should be combined. Some lecture courses are accompanied by a lab session, and the lab itself is listed on separate row in the data. For example in Fig. 3.1, Student 12 has three course records, the first one is *Cell Physiology* for which the course number is *BIOL-30500-01*, and the second one is *Cell Physiology Lab* for which the course number is *BIOL-30500L-01*. Notice the extra 'L' in the second course number indicate that that is the lab session of the first lecture course. The solution to this is to remove all the rows in the data that contain a lab session.

After fixing the issues regarding the course number format, the next step is dealing with outdated course number data. The curriculum of the college is reformed every few years. As a result, a set of course numbers may change and even some department abbreviations are updated. For example, in 2013 the Computer Science department removed some courses and added some new ones. Two of the removed courses are *CSCI-151 Computer Programming I* and *CSCI-152 Computer Programming II*. These two courses are replaced with *CSCI-100 Scientific Computing* and *CSCI-110 Imperative Programming*. But our data contains both the old course numbers and the updated course numbers. The problem of this is that the CRS will treat an old course number and the updated one as two different courses, which has a negative effect on the accuracy of the recommendation. To fix this issue, we went over the latest course catalogue (2016-2017) [4], recorded all the new course numbers that do not exist in the data and check for the alternatives for the outdated course numbers in the data. Finally, the outdated course numbers were all updated, either replaced with a new one if there is an alternative or removed if there is not. For details of the replacement and removal of courses, please see Table A.1 and A.2 in Appendix A.

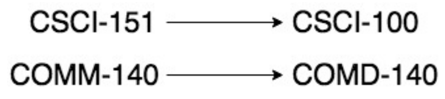


**Figure 3.3:** Update old course numbers (left) with new ones (right).

In the end, there are 634 courses and 5,332 students in our data. From now on, we use the term *Student-Course data* to refer to the cleaned data.

## 3.3 Design and Implementation of the CRS

In Chapter 1 we reviewed some RS work done by others, especially those applied in the education field. Inspired by those works, we decide to apply approaches described in their paper to build our CRS. These include *user-based CF* and *association rule learning* technique. In addition, two methods of the *item-based CF* described in Chapter 2 will also be applied to see the performance.

### 3.3.1 Use Case Analysis for the CRS

In software engineering a good practice is to analyze the *use case scenarios* before developing the software. A *scenario* is a sequence of actions that illustrates user behavior when using the software. Fig. 3.4 captures the use case scenarios of our CRS.

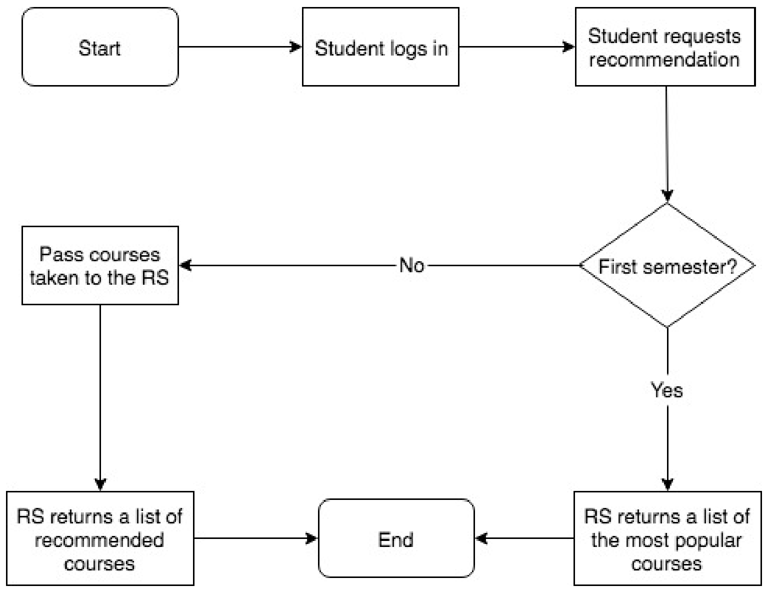

Start
Student logs in
Student requests recommendation
First semester?
No
Yes
Pass courses taken to the RS
RS returns a list of recommended courses
End
RS returns a list of the most popular courses

**Figure 3.4:** Flow chart for using the CRS.

We assume that the CRS has access to the course registration data on ScotWeb, which enables the Student-Course data to be updated as soon as students register new courses. In that way, the student does not need to type in his/her courses taken history, because the CRS can directly extract them from the up-to-date Student-Course data.

It takes only a few steps for a student to get a list of recommendations from the CRS. First the student enters his/her college ID and password to log in the system. Then the CRS checks the user profile to see his/her course history. Then the CRS will take the course history as input and and use a recommendation approach to return a list of 20 recommendations. The reason for the amount 20 is because it is neither too small nor too large.

### 3.3.2 Cold Start Issue and Solution

The situation where there is none or not enough user or item information for a recommendation approach to run is referred as *cold start* issue. In our case, the cold start problem occurs when the target student is a first-year student in his/her first semester. Because there is no course taken by the student, the CRS can not interpret the student's preference on courses and therefore fail to make useful recommendations.

We decide to adopt a naive way to handle this problem by recommending these freshmen the most popular courses in data. Going through the Student-Course data, the 21 most popular courses are counted as follows: (Fig. 3.5).

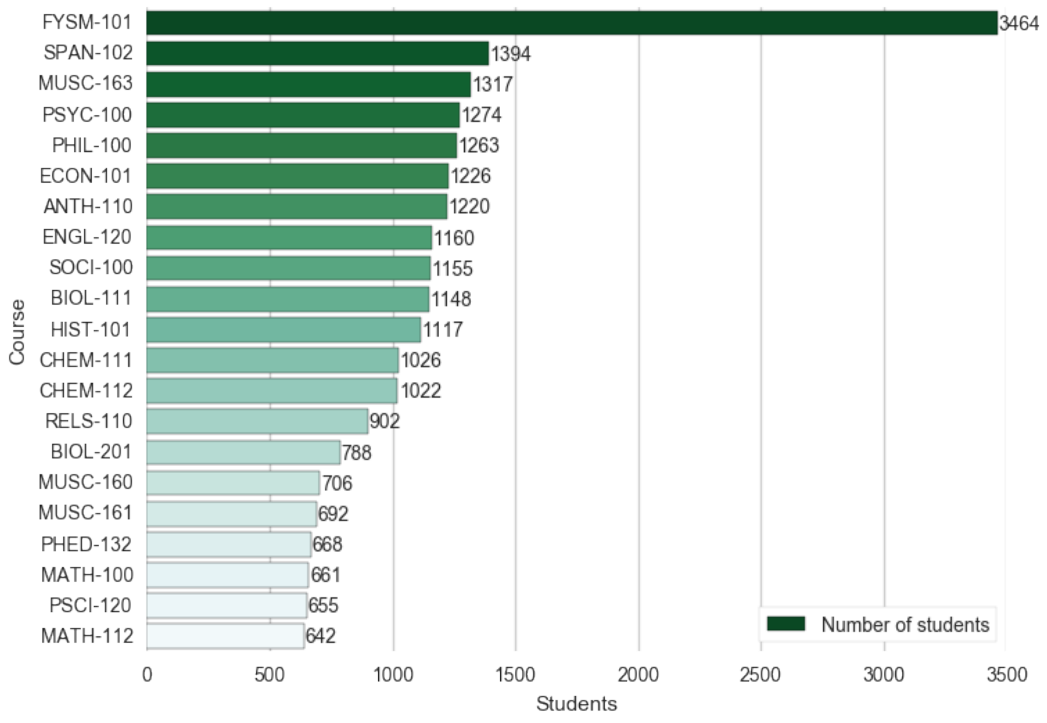


**Figure 3.5:** The 21 most popular courses in the last seven years.

FYSM-101 stands for the *First-year Seminar*, it is ranked the first because it is a required course for all freshmen. The courses other than FYSM-101 will be recommended to the first-semester students. The rest of this chapter discuss the implementations of different approaches that handle students who have taken at least one course.

### 3.3.3 Problems with Applying Content-based Approach for CRS

Chapter 2 introduces two cases of content-based approach. One is when data has explicit features (like movies) and the other one is when the features can be implicit evaluated (like articles). In our case, it is unfortunate that the course data has neither explicit nor implicit features available for implementing the content-based approach. Had the raw data contained general requirement information for each course, that information could have been used as features for the courses just like genres were used as features for movies in Section 2.1.1. In that way, the similarity between courses could be calculated based on the general requirement features. Or if the course description data was available, a feature-mining algorithm such as TF-IDF could be applied to extract keywords from the description. Then other courses can be checked to see how close their keywords are compared to the keywords of courses taken by the student.

Although the general requirement and course description data can be scraped from ScotWeb, only records of the courses in the recent two years are displayed on ScotWeb. Thus, content-based approach is not considered in building our CRS because our data does not support it.

### 3.3.4 Applying User-based Collaborative Filtering for CRS

The first recommendation approach to try on the data is the user-based CF and is illustrated in Fig. 3.6. There are six steps for the user-based CRS to recommend courses to a target student:

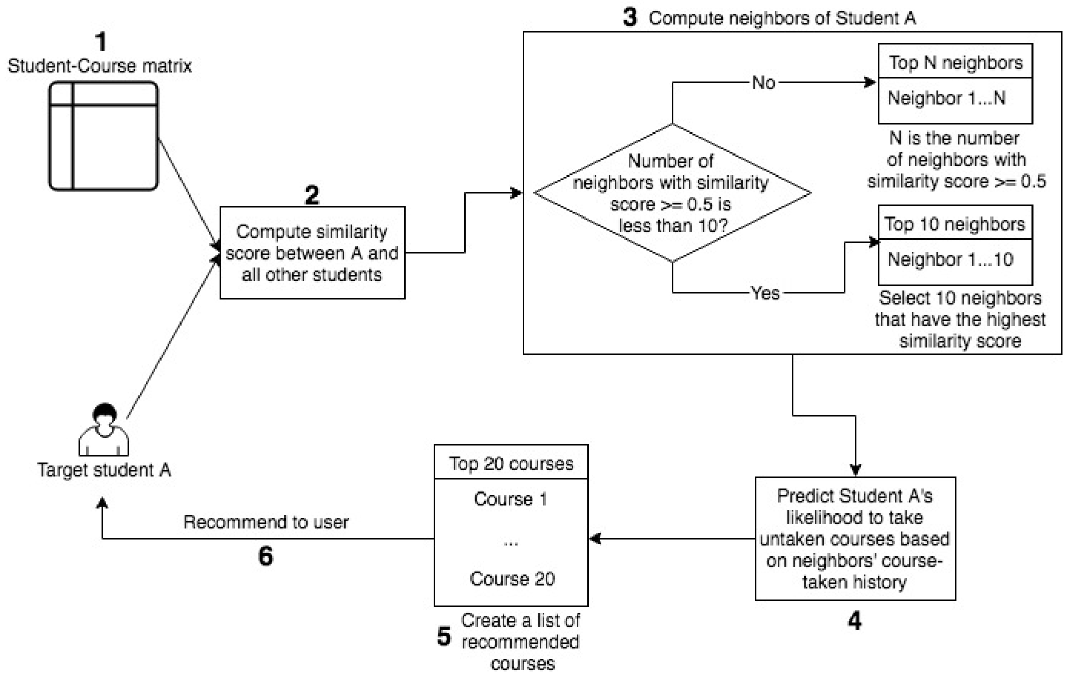

1
Student-Course matrix
2
Compute similarity score between A and all other students
3 Compute neighbors of Student A
Number of neighbors with similarity score >= 0.5 is less than 10?
No
Top N neighbors
Neighbor 1...N
N is the number of neighbors with similarity score >= 0.5
Yes
Top 10 neighbors
Neighbor 1...10
Select 10 neighbors that have the highest similarity score
Target student A
Predict Student A's likelihood to take untaken courses based on neighbors' course-taken history
4
Top 20 courses
Course 1
...
Course 20
5 Create a list of recommended courses
Recommend to user
6

**Figure 3.6:** User-based CRS flowchart.

1. Create student-course matrix, i.e., courses that each student has taken in the past.
2. Compute similarity score between the target student and all other students.
3. Compute the 10 or less most similar students with the target student and identify them as the neighbors.

4. Predict the target student's likelihood to take all courses not taken based on neighbors' course-taken history.

5. Select the top 20 courses based on likelihood value.

6. Recommend the courses from Step 5 to the target user.

Next we briefly describe each of the six steps.

### Step 1. Create Student-Course Matrix

Unlike the movie example given in Chapter 2 which has explicit feedback data (ratings) from users, there is no numerical value available for our course data. However, implicit data can be obtained by looking whether a student has taken a course or not. Using this information, we construct a matrix with 5,332 rows (all students) and 634 columns (all courses). Each entry of the matrix is assigned 1 meaning the student has taken the course or 0 otherwise. In this way, each row of the matrix (Table 3.1) represents a user profile.

| **Course Number** | AFST-100 | AFST-200 | AFST-213 | ... | WGSS-310 | WGSS-320 |
|---|---|---|---|---|---|---|
| **Student ID** | | | | | | |
| **6** | 0 | 0 | 0 | ... | 0 | 0 |
| **12** | 1 | 0 | 0 | ... | 0 | 0 |
| **20** | 0 | 0 | 0 | ... | 0 | 0 |
| **21** | 0 | 0 | 0 | ... | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |

**Table 3.1:** A portion of the Student-Course matrix which is a collection of user profiles.

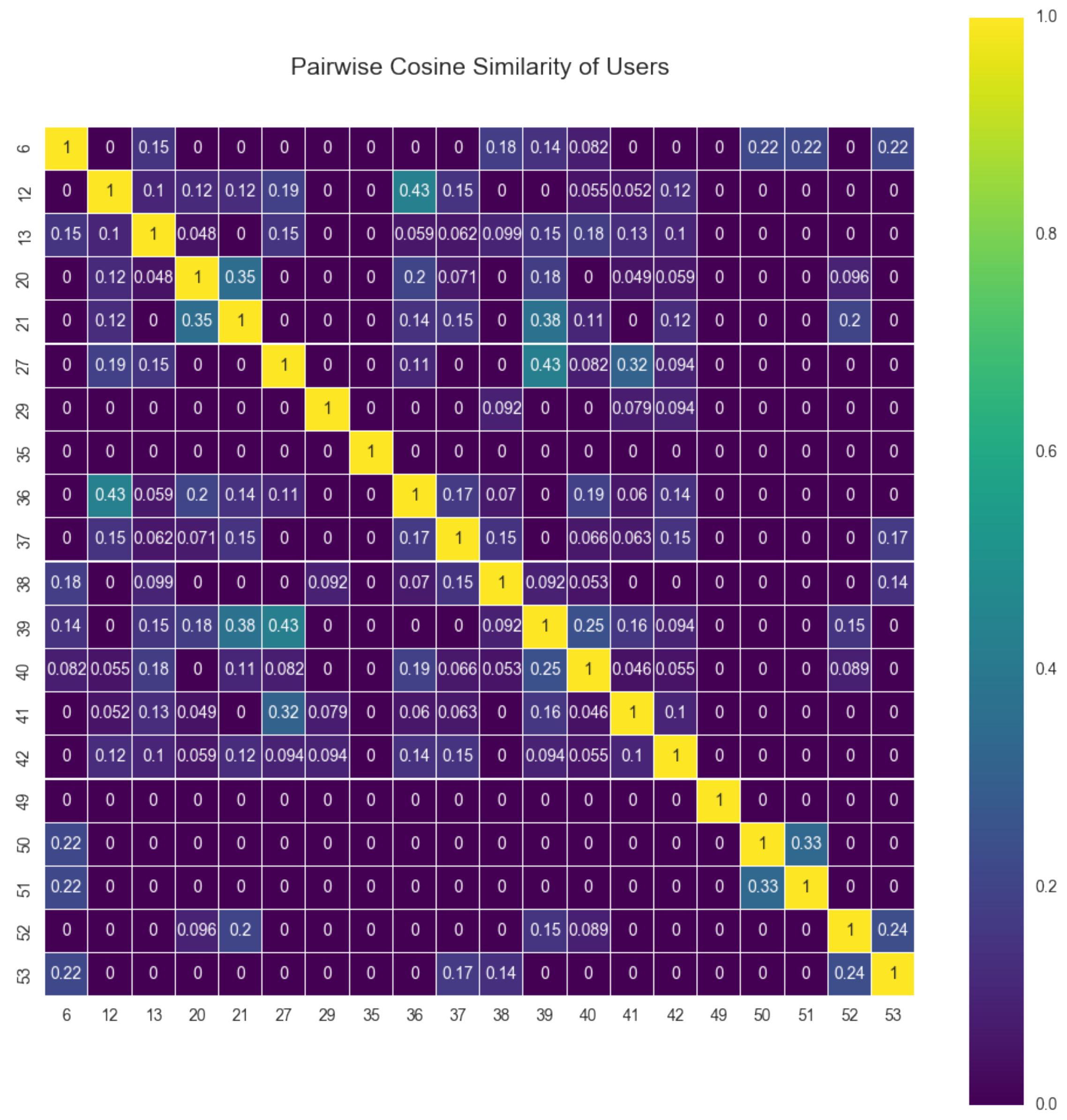
### Step 2. Compute User Similarity

Suppose the student whose ID is 12 (this student will be refer as Student 12) has asked for course recommendations, Student 12 is now the target user. In this step the similarity scores between Student 12 and all other students are computed to find nearest neighbors. There are two things to consider: (1) The method of similarity computation to use; (2) The way of getting the neighbors. As concluded in Section 2.2.1, the choice of method depends on the data. In this case, the number of courses is 634, but since a student takes only about 32 courses at most, it is imaginable that there are many shared zeros between students. Because the data is sparse, cosine similarity is used to calculate the similarity between students.

There are two ways to get all neighbors of a target user. The first way is to directly extract the target user profile from the Student-Course matrix, calculate the score with the rest of the user profiles and select the highest results. Another way is to compute the pairwise similarity score for all the students. The pairwise computation compares every student profile with every other student profile and records the similarity scores in a matrix.

There is a *space-time trade off* between the two ways. When the data size is large, e.g., hundreds of thousands of students and courses, the first way becomes computationally expensive. Suppose there are $M$ students and $N$ courses, then the run time of computing similarity score between one target student and all other students is $O(MN)$. Every time a new target user comes for recommendation, the system will cost that much run time to obtain the neighbors. If a user queries the CRS twice, the system will repeat the same process again. In contrast, the second way builds up a lookup table from which the similarity scores of the neighbors of a user can easily be checked. The cost of it is the need for an extra amount of memory, but it is more time-efficient than the first way when the data size is large. The data in this project is relatively small but is expected to grow in the future, so the second way is more preferable.

The pairwise similarity matrix has a size of $5,332 \times 5,332$ as there are 5,332 students, and it is symmetric. Fig. 3.7 visualizes part of this matrix. The rows and columns are labeled with the student IDs, with the first student ID being 6. The $ij^{th}$ entry of the matrix is the cosine similarity score between $i^{th}$ and $j^{th}$ student IDs. The color of each cell indicates the strength of similarity. The more yellowish the cell, the more similar it is. In contrast, the more purplish, the less similar. The entries on the diagonal are all ones and perfectly yellowish since every student is perfectly similar with themselves. The rest of the entries have various values and shades of color.



| | 6 | 12 | 13 | 20 | 21 | 27 | 29 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 49 | 50 | 51 | 52 | 53 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 1 | 0 | 0.15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.18 | 0.14 | 0.082 | 0 | 0 | 0 | 0.22 | 0.22 | 0 | 0.22 |
| 12 | 0 | 1 | 0.1 | 0.12 | 0.12 | 0.19 | 0 | 0 | 0.43 | 0.15 | 0 | 0 | 0.055 | 0.052 | 0.12 | 0 | 0 | 0 | 0 | 0 |
| 13 | 0.15 | 0.1 | 1 | 0.048 | 0 | 0.15 | 0 | 0 | 0.059 | 0.062 | 0.099 | 0.15 | 0.18 | 0.13 | 0.1 | 0 | 0 | 0 | 0 | 0 |
| 20 | 0 | 0.12 | 0.048 | 1 | 0.35 | 0 | 0 | 0 | 0.2 | 0.071 | 0 | 0.18 | 0 | 0.049 | 0.059 | 0 | 0 | 0 | 0.096 | 0 |
| 21 | 0 | 0.12 | 0 | 0.35 | 1 | 0 | 0 | 0 | 0.14 | 0.15 | 0 | 0.38 | 0.11 | 0 | 0.12 | 0 | 0 | 0 | 0.2 | 0 |
| 27 | 0 | 0.19 | 0.15 | 0 | 0 | 1 | 0 | 0 | 0.11 | 0 | 0 | 0.43 | 0.082 | 0.32 | 0.094 | 0 | 0 | 0 | 0 | 0 |
| 29 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0.092 | 0 | 0 | 0.079 | 0.094 | 0 | 0 | 0 | 0 | 0 |
| 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 36 | 0 | 0.43 | 0.059 | 0.2 | 0.14 | 0.11 | 0 | 0 | 1 | 0.17 | 0.07 | 0 | 0.19 | 0.06 | 0.14 | 0 | 0 | 0 | 0 | 0 |
| 37 | 0 | 0.15 | 0.062 | 0.071 | 0.15 | 0 | 0 | 0 | 0.17 | 1 | 0.15 | 0 | 0.066 | 0.063 | 0.15 | 0 | 0 | 0 | 0 | 0.17 |
| 38 | 0.18 | 0 | 0.099 | 0 | 0 | 0 | 0.092 | 0 | 0.07 | 0.15 | 1 | 0.092 | 0.053 | 0 | 0 | 0 | 0 | 0 | 0 | 0.14 |
| 39 | 0.14 | 0 | 0.15 | 0.18 | 0.38 | 0.43 | 0 | 0 | 0 | 0 | 0.092 | 1 | 0.25 | 0.16 | 0.094 | 0 | 0 | 0 | 0.15 | 0 |
| 40 | 0.082 | 0.055 | 0.18 | 0 | 0.11 | 0.082 | 0 | 0 | 0.19 | 0.066 | 0.053 | 0.25 | 1 | 0.046 | 0.055 | 0 | 0 | 0 | 0.089 | 0 |
| 41 | 0 | 0.052 | 0.13 | 0.049 | 0 | 0.32 | 0.079 | 0 | 0.06 | 0.063 | 0 | 0.16 | 0.046 | 1 | 0.1 | 0 | 0 | 0 | 0 | 0 |
| 42 | 0 | 0.12 | 0.1 | 0.059 | 0.12 | 0.094 | 0.094 | 0 | 0.14 | 0.15 | 0 | 0.094 | 0.055 | 0.1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 50 | 0.22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0.33 | 0 | 0 |
| 51 | 0.22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.33 | 1 | 0 | 0 |
| 52 | 0 | 0 | 0 | 0.096 | 0.2 | 0 | 0 | 0 | 0 | 0 | 0 | 0.15 | 0.089 | 0 | 0 | 0 | 0 | 0 | 1 | 0.24 |
| 53 | 0.22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.17 | 0.14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.24 | 1 |

**Figure 3.7:** Visualization of pairwise cosine similarity of the first 20 students.

## Step 3. Compute the Top 10 Neighbors

The information of Student 12 is extracted and shown in Table 3.2. It shows that this student is a biology major and has taken many biology and chemistry courses.

| ID | Class Year | Major 1 | Major 2 | Courses Taken |
|---|---|---|---|---|
| 12 | 2018 | BIOL | - | BIOL-305 CHEM-212 IDPT-405 RELS-219 BIOL-201 BIOL-203 CHEM-211 IDPT-405 BIOL-202 CHEM-112 IDPT-199 MATH-111 SOCI-100 AFST-100 BIOL-111 CHEM-111 FYSM-101 |

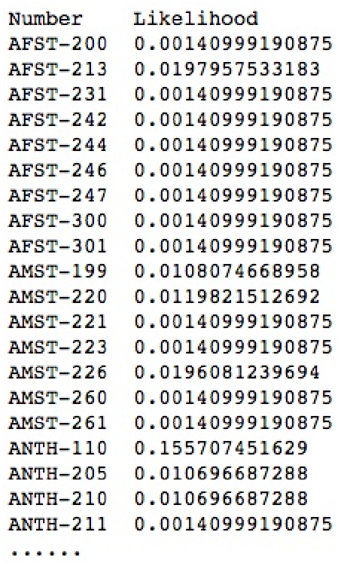
**Table 3.2:** Information of Student 12.

To find the nearest neighbors of Student 12, first his/her row vector is selected from the pairwise similarity matrix. Since there is no negative rating (course taken is a binary value) in the data, the range of cosine similarity is between 0 and 1. Therefore, the vector consists of 5,332 values between 0 and 1. Among these values, those greater than 0.5 are selected as nearest neighbors of Student 12. It turns out that there are 100 entries satisfy the condition, meaning there are 100 neighbors for Student 12. However, not every student is guaranteed to have a neighbor using this method. There are some students whose similarity vector has no value greater than 0.5. In order to address this issue, a mean test is run to check the average number of neighbors of all students. The result of the test says the average number of neighbors is 10.81. Now the program will check the number of neighbors whose similarity score is greater than 0.5 for each student. If there are more than 10 neighbors, all of them are selected as nearest neighbors. Otherwise if there are less than 10 neighbors, the top 10 highest results are used. In this way, everyone is guaranteed to have at least 10 neighbors.

Table A.3 in Appendix A lists the information of the top 10 closest neighbors to Student 12. It shows that among these 10 neighbors, 4 of them are biology majors, 4 are biochemistry and molecular biology majors and 1 is neuroscience major. All of them have taken an adequate amount of biology and chemistry courses. In addition, all of them are in the same year with the target student. All of these common information indicate that the user-based model is efficient in finding similar users.

### Step 4&5. Create a List of Recommended Courses

After the nearest neighbors are identified, the likelihood of Student 12 taking other courses he/she has not taken is calculated using Equation 2.7. Fig. 3.8 illustrates the likelihood values for a subset of the 634 courses sorted by alphabetical order.

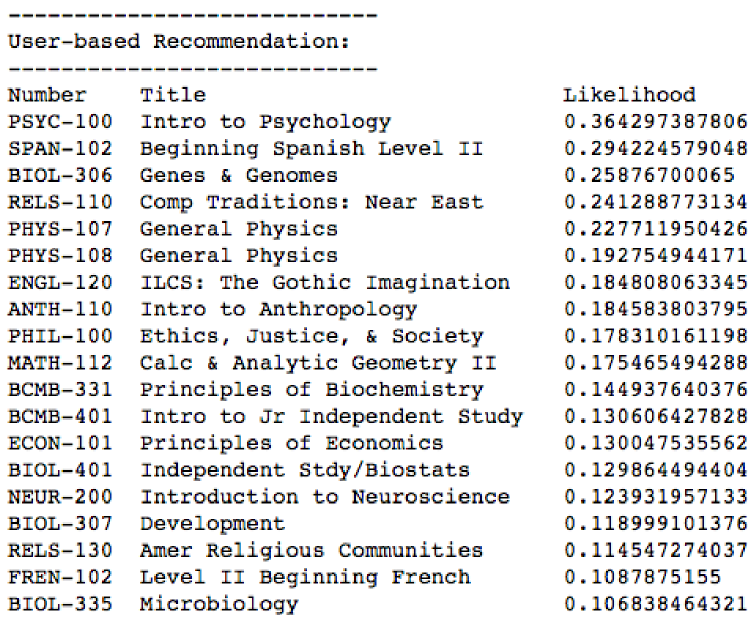


```
Number     Likelihood
AFST-200   0.00140999190875
AFST-213   0.0197957533183
AFST-231   0.00140999190875
AFST-242   0.00140999190875
AFST-244   0.00140999190875
AFST-246   0.00140999190875
AFST-247   0.00140999190875
AFST-300   0.00140999190875
AFST-301   0.00140999190875
AMST-199   0.0108074668958
AMST-220   0.0119821512692
AMST-221   0.00140999190875
AMST-223   0.00140999190875
AMST-226   0.0196081239694
AMST-260   0.00140999190875
AMST-261   0.00140999190875
ANTH-110   0.155707451629
ANTH-205   0.010696687288
ANTH-210   0.010696687288
ANTH-211   0.00140999190875
......
```

**Figure 3.8:** Likelihood of Student 12 to take other courses.

Step 6. Recommend the Top 20 Courses to Target User

In the end, as shown in Fig. 3.9, 20 courses having the highest likelihood values are selected and recommended to Student 12.



```
----------------------------
User-based Recommendation:
----------------------------
Number     Title                          Likelihood
PSYC-100   Intro to Psychology            0.364297387806
SPAN-102   Beginning Spanish Level II     0.294224579048
BIOL-306   Genes & Genomes                0.25876700065
RELS-110   Comp Traditions: Near East     0.241288773134
PHYS-107   General Physics                0.227711950426
PHYS-108   General Physics                0.192754944171
ENGL-120   ILCS: The Gothic Imagination   0.184808063345
ANTH-110   Intro to Anthropology          0.184583803795
PHIL-100   Ethics, Justice, & Society     0.178310161198
MATH-112   Calc & Analytic Geometry II    0.175465494288
BCMB-331   Principles of Biochemistry     0.144937640376
BCMB-401   Intro to Jr Independent Study  0.130606427828
ECON-101   Principles of Economics        0.130047535562
BIOL-401   Independent Stdy/Biostats      0.129864494404
NEUR-200   Introduction to Neuroscience   0.123931957133
BIOL-307   Development                    0.118999101376
RELS-130   Amer Religious Communities     0.114547274037
FREN-102   Level II Beginning French      0.1087875155
BIOL-335   Microbiology                   0.106838464321
```

**Figure 3.9:** Our user-based CRS recommends Student 12 twenty courses.

The result shows that the likelihoods are low in general. Part of the reason is because of the sparsity of the data. Because of the binary rating, every single component in Equation 2.7 is between

0 and 1, therefore the aggregated result is expected to be low. In fact, the reality that some students do not have neighbors with similarity score greater than 0.5 also reflects the issue of sparsity. As mentioned in the previous chapter, one possible solution to address the sparsity issue in user-based model is to use an item-based solution.

### 3.3.5 Applying Item-based Collaborative Filtering for CRS

Building a CRS with item-based model is similar to building a user-based model. The main difference is that all user-oriented operations are changed to item-oriented. As shown in Fig. 3.10, there are five steps in the item-based CF approach.

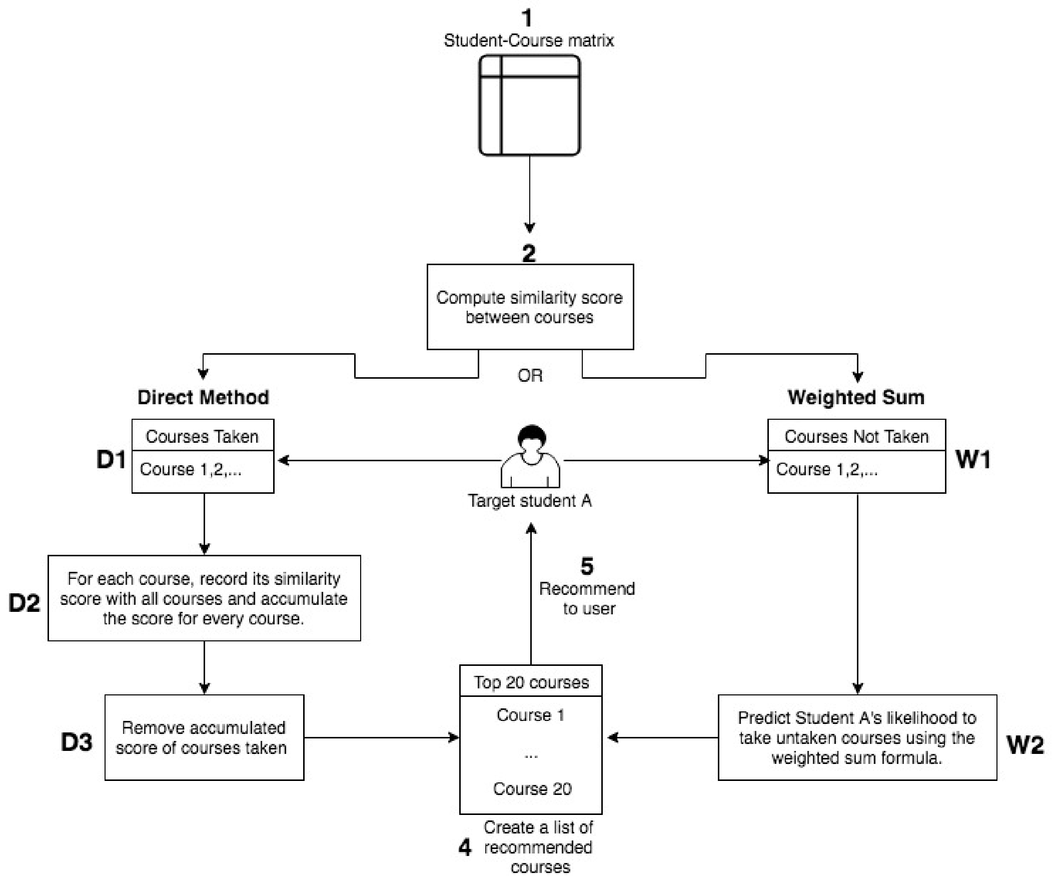

1
Student-Course matrix
2
Compute similarity score between courses
OR
Direct Method
Weighted Sum
D1
Courses Taken
Course 1,2,...
Target student A
Courses Not Taken
Course 1,2,...
W1
D2
For each course, record its similarity score with all courses and accumulate the score for every course.
5
Recommend to user
D3
Remove accumulated score of courses taken
Top 20 courses
Course 1
...
Course 20
4
Create a list of recommended courses
Predict Student A's likelihood to take untaken courses using the weighted sum formula.
W2

**Figure 3.10:** Item-based CRS flowchart.

1. Create Student-Course matrix, i.e., courses that each student has taken in the past.

2. Compute similarity score between all courses.

3. Choose direct method or weighted sum method to calculate the score or likelihood of each course.

4. Select the top 20 courses based on accumulated score / likelihood value.

5. Recommend the courses from Step 4 to the target user.

Next we briefly describe each of the five steps.

#### Step 1. Create Student-Course matrix

It is the same as the first step in the user-based CF approach. The implicit feedback that whether a student has taken a course or not is used to construct a matrix of all students and all courses with entries being either 1 meaning taken or 0 otherwise (Table 3.1).

#### Step 2. Compute Similarity Between All Courses

The most important step in item-based approach is computing the similarity between all items in the system. In this case, the items are the courses. Because the similarity values are used frequently in the later steps, it is better to build a look up table of the values, i.e., compute the pairwise cosine similarity between the courses, so that duplicated computations are avoided. As shown in Fig. 3.11, the pairwise item similarity matrix is $634 \times 634$ and has the same properties as the pairwise user similarity matrix (Fig. 3.7).

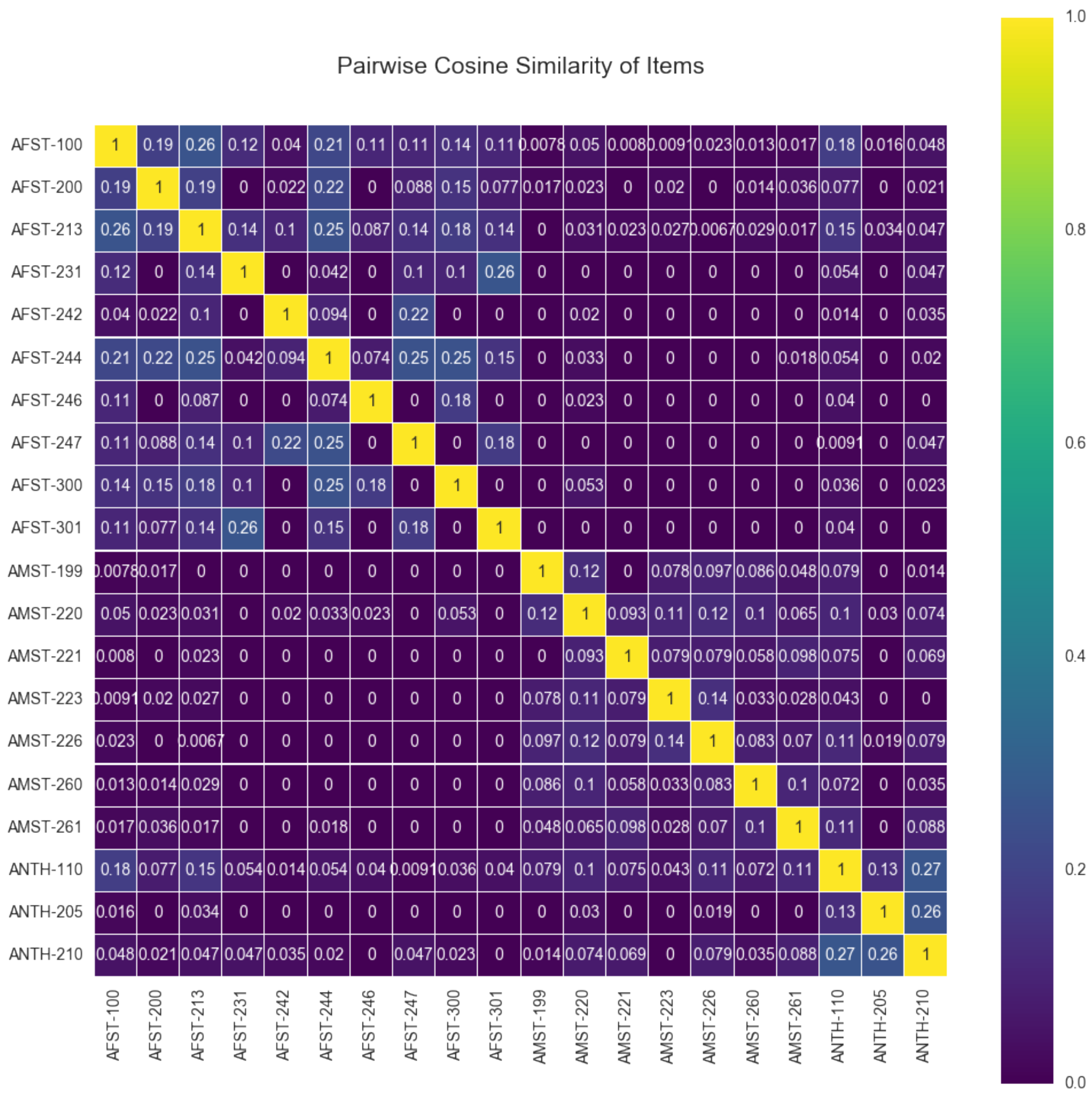
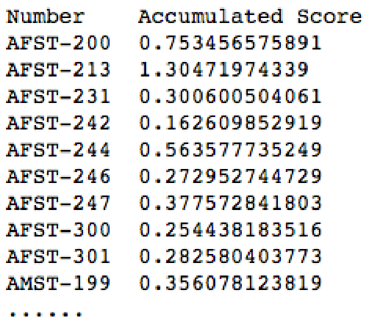


**Figure 3.11:** Visualization of pairwise cosine similarity of the first 20 courses.

After the similarity matrix is completed, a target user's profile is put in to run through the process. We use Student 12 again as an example to show the process of direct method and weighted sum method. From previous section we know that Student 12 has taken 13 courses, among which most are biology and chemistry (Table 3.2). Both direct and weighted sum methods are described below, first is the direct method, then the weighted sum method.

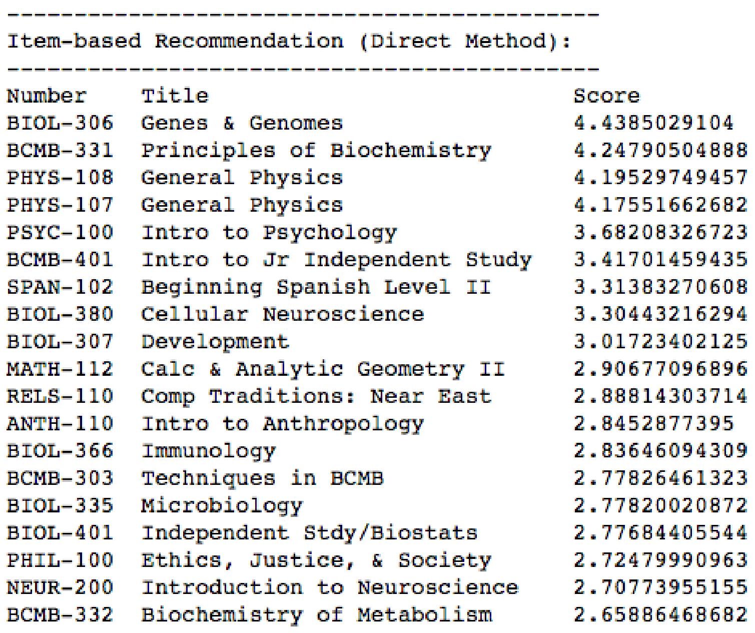
## Direct Method

The direct method, as discussed in Chapter 2, initializes a list of zeros that serve as the initial condition of accumulating similarity score for each item in the system. In this example, the initial list has 634 zeros, corresponding to 634 courses. Then for each of the 13 courses taken by Student 12, the system extracts its row vector from the similarity matrix and adds it to the accumulation list. At the end, values at the positions corresponding to those 13 courses taken are removed from the list. The resulted list is the final candidate list. Fig. 3.12 displays this final candidate list in alphabetical order. After that, 20 courses that have the highest accumulated similarity scores are recommended to Student 12, shown in Fig. 3.13.



```
Number     Accumulated Score
AFST-200   0.753456575891
AFST-213   1.30471974339
AFST-231   0.300600504061
AFST-242   0.162609852919
AFST-244   0.563577735249
AFST-246   0.272952744729
AFST-247   0.377572841803
AFST-300   0.254438183516
AFST-301   0.282580403773
AMST-199   0.356078123819
......
```

**Figure 3.12:** The accumulated scores for the first 10 courses using the direct method.



```
--------------------------------------------
Item-based Recommendation (Direct Method):
--------------------------------------------
Number    Title                            Score
BIOL-306  Genes & Genomes                  4.4385029104
BCMB-331  Principles of Biochemistry       4.24790504888
PHYS-108  General Physics                  4.19529749457
PHYS-107  General Physics                  4.17551662682
PSYC-100  Intro to Psychology              3.68208326723
BCMB-401  Intro to Jr Independent Study    3.41701459435
SPAN-102  Beginning Spanish Level II       3.31383270608
BIOL-380  Cellular Neuroscience            3.30443216294
BIOL-307  Development                      3.01723402125
MATH-112  Calc & Analytic Geometry II      2.90677096896
RELS-110  Comp Traditions: Near East       2.88814303714
ANTH-110  Intro to Anthropology            2.8452877395
BIOL-366  Immunology                       2.83646094309
BCMB-303  Techniques in BCMB               2.77826461323
BIOL-335  Microbiology                     2.77820020872
BIOL-401  Independent Stdy/Biostats        2.77684405544
PHIL-100  Ethics, Justice, & Society       2.72479990963
NEUR-200  Introduction to Neuroscience     2.70773955155
BCMB-332  Biochemistry of Metabolism       2.65886468682
```

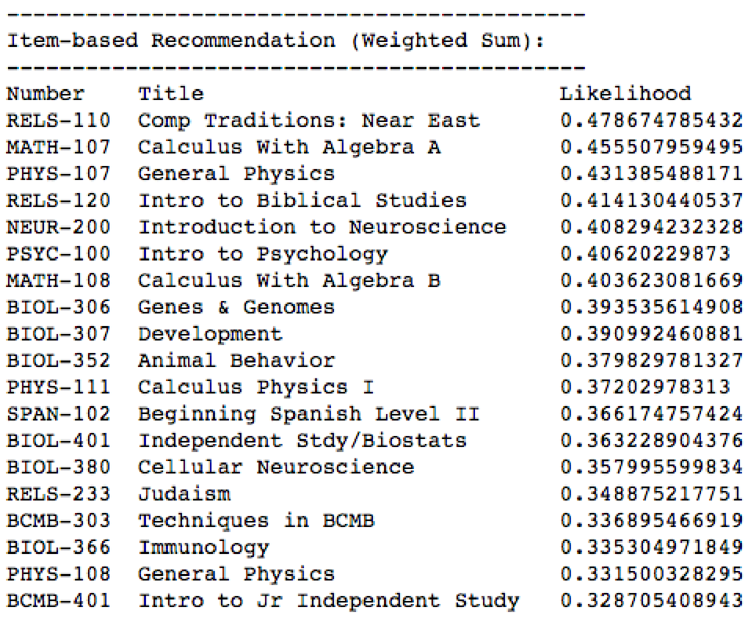
**Figure 3.13:** Our item-based CRS using direct method recommends twenty courses to Student 12.

### Weighted Sum

The weighted sum method involves more calculations than the direct method. Its search space is the courses not taken by Student 12, i.e., the other $634 - 13 = 621$ courses. For each one of these courses, denoted by $c$, the system extracts its row vector from the similarity matrix. The row vector contains the neighbors, denoted by $S$, of the untaken course $c$. Then, the system applies Equation 2.9 to get the predicted rating for $c$ using the information of each neighbor $j$ in $S$.

$$\text{Rating}(A, c) = \frac{\sum_{j \in S} sim(c, j) \times (r_{A,j})}{\sum_{j \in S} sim(c, j)}$$

At the end, 20 courses with highest likelihood are recommended to Student 12 (Fig. 3.14).



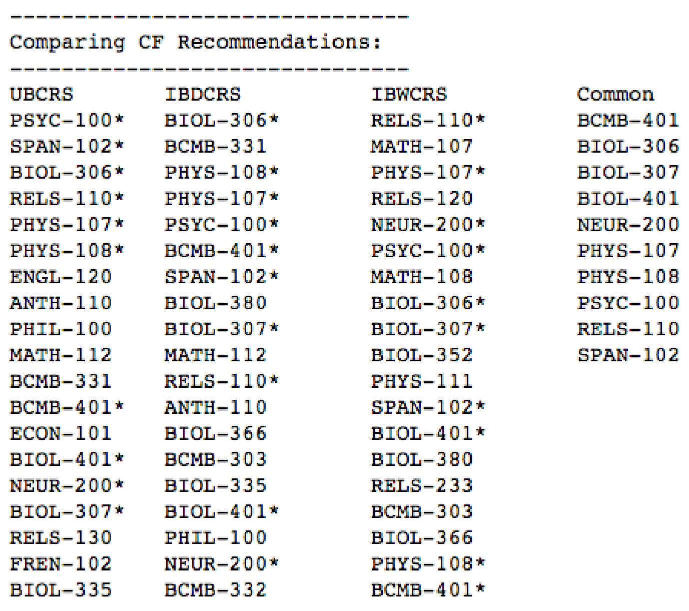
```
-------------------------------------------
Item-based Recommendation (Weighted Sum):
-------------------------------------------
Number    Title                          Likelihood
RELS-110  Comp Traditions: Near East     0.478674785432
MATH-107  Calculus With Algebra A        0.455507959495
PHYS-107  General Physics                0.431385488171
RELS-120  Intro to Biblical Studies      0.414130440537
NEUR-200  Introduction to Neuroscience   0.408294232328
PSYC-100  Intro to Psychology            0.40620229873
MATH-108  Calculus With Algebra B        0.403623081669
BIOL-306  Genes & Genomes                0.393535614908
BIOL-307  Development                    0.390992460881
BIOL-352  Animal Behavior                0.379829781327
PHYS-111  Calculus Physics I             0.37202978313
SPAN-102  Beginning Spanish Level II     0.366174757424
BIOL-401  Independent Stdy/Biostats      0.363228904376
BIOL-380  Cellular Neuroscience          0.357995599834
RELS-233  Judaism                        0.348875217751
BCMB-303  Techniques in BCMB             0.336895466919
BIOL-366  Immunology                     0.335304971849
PHYS-108  General Physics                0.331500328295
BCMB-401  Intro to Jr Independent Study  0.328705408943
```

**Figure 3.14:** Our item-based CRS using weighted sum method recommends twenty courses to Student 12.

So far three CF approaches and methods have been applied to the Student-Course data to implement a CRS. From now on we will be using the following notations for these three distinct CRSs:

- **UBCRS**: user-based course recommender system.
- **IBDCRS**: item-based course recommender system using the direct method.
- **IBWCRS**: item-based course recommender system using the weighted sum method.

Each one produces a list of 20 courses to the same student Student 12. Comparing the results produced by the three recommendations (see Fig. 3.15), we find that there are 10 courses that appear in all results. Those shared recommendations are followed by a star.



```
-------------------------------
Comparing CF Recommendations:
-------------------------------
UBCRS       IBDCRS         IBWCRS         Common
PSYC-100*   BIOL-306*      RELS-110*      BCMB-401
SPAN-102*   BCMB-331       MATH-107       BIOL-306
BIOL-306*   PHYS-108*      PHYS-107*      BIOL-307
RELS-110*   PHYS-107*      RELS-120       BIOL-401
PHYS-107*   PSYC-100*      NEUR-200*      NEUR-200
PHYS-108*   BCMB-401*      PSYC-100*      PHYS-107
ENGL-120    SPAN-102*      MATH-108       PHYS-108
ANTH-110    BIOL-380       BIOL-306*      PSYC-100
PHIL-100    BIOL-307*      BIOL-307*      RELS-110
MATH-112    MATH-112       BIOL-352       SPAN-102
BCMB-331    RELS-110*      PHYS-111
BCMB-401*   ANTH-110       SPAN-102*
ECON-101    BIOL-366       BIOL-401*
BIOL-401*   BCMB-303       BIOL-380
NEUR-200*   BIOL-335       RELS-233
BIOL-307*   BIOL-401*      BCMB-303
RELS-130    PHIL-100       BIOL-366
FREN-102    NEUR-200*      PHYS-108*
BIOL-335    BCMB-332       BCMB-401*
```

**Figure 3.15:** Comparing the sample recommendation result of three CF approaches.

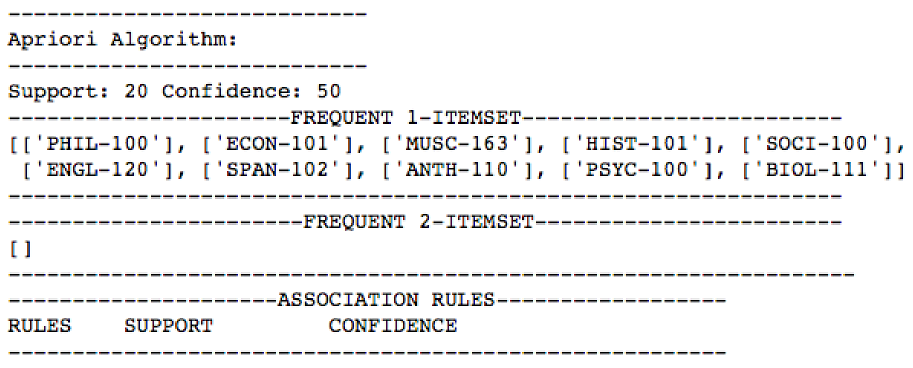
The last approach we apply is association rule learning.

### 3.3.6 Applying Association Rule Learning for CRS

Imagine the college is a market and the course is merchandise, then each student's course history can be viewed as a market transaction. The goal of applying association rule learning on the Student-Course data is to discover courses that are frequently taken together and rules of form like *Course 1*, *Course 2* => *Course 3* that tells us about the hidden course-taking pattern. There are two plans to divide the data for applying association rules mining with Apriori algorithm to build the CRS. We will go over them one by one.

#### 01 All Transactions

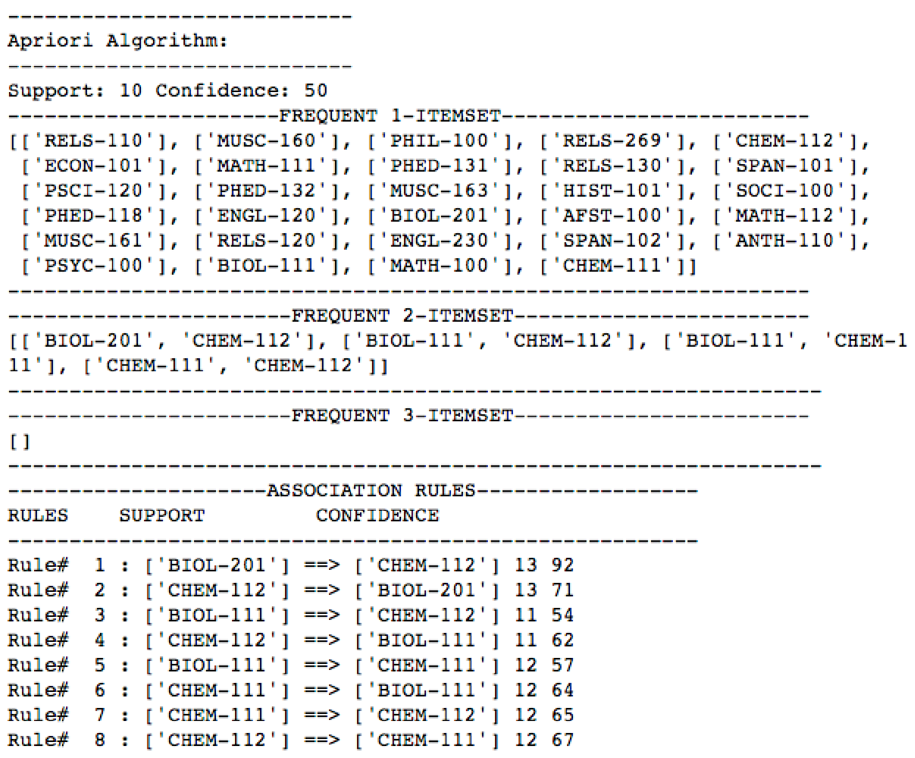
The first one is using directly all the course transactions. All students' course history is collected together into a giant list to run the association analysis. The next thing to do is tunning the values of support threshold $s$ and confidence threshold $c$ in order to obtain meaningful result. Starting with *s*=30% and *c*=70%, no result is obtained. Then by lowering the values to *s*=20% and *c*=50%, frequent-1-itemset starts to show (see Fig. 3.16).



```
----------------------------
Apriori Algorithm:
----------------------------
Support: 20 Confidence: 50
----------------------FREQUENT 1-ITEMSET-------------------------
[['PHIL-100'], ['ECON-101'], ['MUSC-163'], ['HIST-101'], ['SOCI-100'],
 ['ENGL-120'], ['SPAN-102'], ['ANTH-110'], ['PSYC-100'], ['BIOL-111']]
-----------------------------------------------------------------
-----------------------FREQUENT 2-ITEMSET------------------------
[]
------------------------------------------------------------------
---------------------ASSOCIATION RULES------------------
RULES    SUPPORT         CONFIDENCE
--------------------------------------------------------
```

**Figure 3.16:** Association rules result for all course transactions at *s*=20% and *c*=50%.

The figure shows the most popular courses, most of which are shown in Fig. 3.5, appearing in 20% of the course transactions. However, without further frequent *N*-itemset, there is no association rule. Continuing to reduce *s* to 10% and frequent 2-itemset and association rules are revealed. (see Fig. 3.17). As shown in the figure, frequent 2-itemset is dominated by chemistry and biology courses.



```
----------------------------
Apriori Algorithm:
----------------------------
Support: 10 Confidence: 50
----------------------FREQUENT 1-ITEMSET-------------------------
[['RELS-110'], ['MUSC-160'], ['PHIL-100'], ['RELS-269'], ['CHEM-112'],
 ['ECON-101'], ['MATH-111'], ['PHED-131'], ['RELS-130'], ['SPAN-101'],
 ['PSCI-120'], ['PHED-132'], ['MUSC-163'], ['HIST-101'], ['SOCI-100'],
 ['PHED-118'], ['ENGL-120'], ['BIOL-201'], ['AFST-100'], ['MATH-112'],
 ['MUSC-161'], ['RELS-120'], ['ENGL-230'], ['SPAN-102'], ['ANTH-110'],
 ['PSYC-100'], ['BIOL-111'], ['MATH-100'], ['CHEM-111']]
-----------------------------------------------------------------
-----------------------FREQUENT 2-ITEMSET------------------------
[['BIOL-201', 'CHEM-112'], ['BIOL-111', 'CHEM-112'], ['BIOL-111', 'CHEM-1
11'], ['CHEM-111', 'CHEM-112']]
------------------------------------------------------------------
-----------------------FREQUENT 3-ITEMSET------------------------
[]
------------------------------------------------------------------
---------------------ASSOCIATION RULES------------------
RULES    SUPPORT         CONFIDENCE
--------------------------------------------------------
Rule#  1 : ['BIOL-201'] ==> ['CHEM-112'] 13 92
Rule#  2 : ['CHEM-112'] ==> ['BIOL-201'] 13 71
Rule#  3 : ['BIOL-111'] ==> ['CHEM-112'] 11 54
Rule#  4 : ['CHEM-112'] ==> ['BIOL-111'] 11 62
Rule#  5 : ['BIOL-111'] ==> ['CHEM-111'] 12 57
Rule#  6 : ['CHEM-111'] ==> ['BIOL-111'] 12 64
Rule#  7 : ['CHEM-111'] ==> ['CHEM-112'] 12 65
Rule#  8 : ['CHEM-112'] ==> ['CHEM-111'] 12 67
```

**Figure 3.17:** Association rules result for all course transactions at *s*=10% and *c*=50%.

On one hand, this reflects that the majority of students take chemistry and biology courses and these two types of courses are often taken together, but it also shows this approach is not able to provide personalized recommendations as it fails to uncover less popular courses and their association rules. Indeed, endlessly decreasing $s$ may improve the situation, but it violates the original intention of this algorithm.

### 02 Major By Major

In the works of Lu et al. [11] and Aher et al. [5], association rule learning are applied on courses from one or two departments and descent results are shown. Inspired by their works, we split the course data by department and apply association rule learning on each department's courses. The first thing we do is to count the popularity of majors in the past seven years, shown in Fig. 3.18.

As with the first method, we expect it is easier to get the results for the more popular majors than the less popular ones since there are more data. To verify this hypothesis, we select one popular and one less popular majors (History and Art History) to run the association analysis. It turns out at $s$=20% and $c$=50%, there are 2 rules for History major and 7 for Art History major. At $s$=10% and $c$=50%, there are 27 rules for History major and 197 for Art History major. This shows the assumption does not hold.

Further checking the number of courses in each department, we want to find if there is an inverse relationship between the number of rules and the number of courses. Because there are 41 courses labeled "HIST" and 18 labeled "ARTH". It is suspected that the more courses there are, the more sparse the data becomes, thus less frequent itemsets and fewer association rules are found. However, using the data of Music major at at $s$=20% and $c$=50%, we obtain 1,997 rules. Therefore, the number of rules for a major is uncorrelated with the number of courses in that department.

After multiple times of changing parameters, we decide to omit this major-by-major approach because of several key reasons. First, there is no clear metric for determining the support and confidence threshold that ensure the quality and quantity of frequent itemsets and association rules for each major. Tuning the parameters for each major is time-consuming and there is no way to validate whether the current parameter values are optimal. Plus, the frequent itemsets and association rules change as the data increase, and in that situation a new set of parameters need to be determined. Secondly, for most majors, the majority of frequent itemsets are courses from the their own department. And since most of them are required for the major, students have to take them no matter what. Such recommendations are not useful because there is no degree of freedom to reject

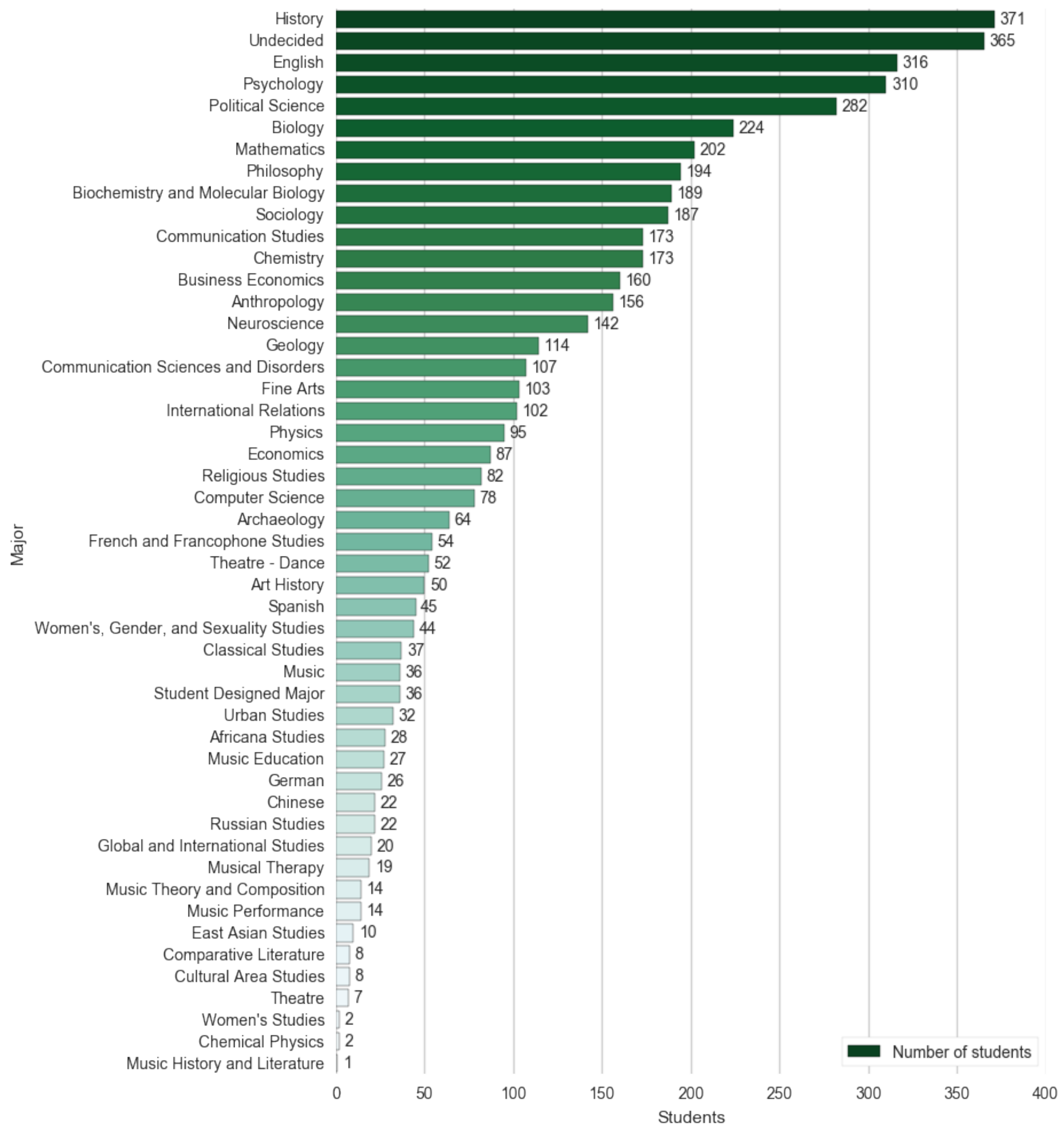


**Figure 3.18:** Majors ranked by popularity in the last seven years.

them. Thirdly, this approach produces more general result if the number of people in a major is large. For major like Chemical Physics where there are only two people, the result is highly limited to those two people's course history.

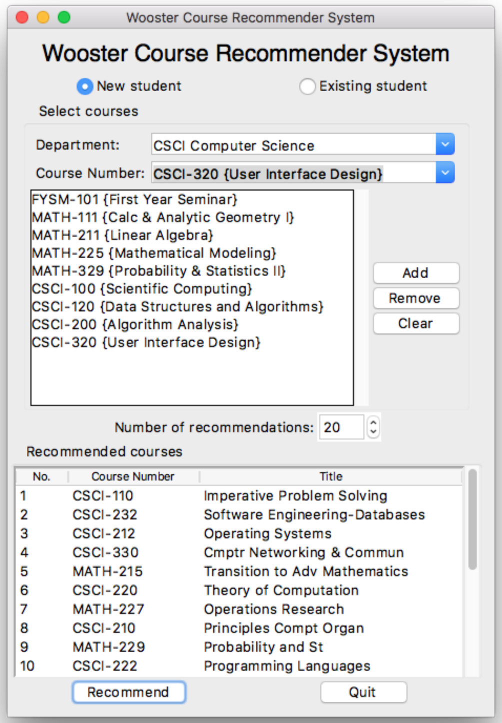
In conclusion, the biggest problem of implementing association rule learning is setting the values for support and confidence threshold. As mentioned before, since there is no metric to measure the quality of the parameters, and in long term if the data change, which is certain to grow, the parameters need to be reset to another pair of optimal values, association rule learning is more tricky and less practical compared to the collaborative filtering technique. Hence, it is left for future work.

Finally we decide to implement the CRS using the collaborative filtering approaches.

## 3.4 User Interface

Another part of the CRS implementation is creating a user interface (UI) for the script files so people can try the system without typing commands. As mentioned before, the implementation code is written in Python language. There is a built-in package in Python called `tkinter` that builds graphical user interface for Python programs. It is used in this project to create the UI of the CRS.

The goal of the UI is to provide two ways for people to try the CRS. One is choosing an arbitrary combination of courses as input, the other is testing the CRS with an existing student data. For this purpose, two separate pages are created, one is named "New student" and the other is called "Existing student". The final appearance of the two pages of our UI are shown in Fig. 3.19 and 3.20.



Wooster Course Recommender System

[✓ New student] Existing student

Select courses

Department: CSCI Computer Science

Course Number: CSCI-320 {User Interface Design}

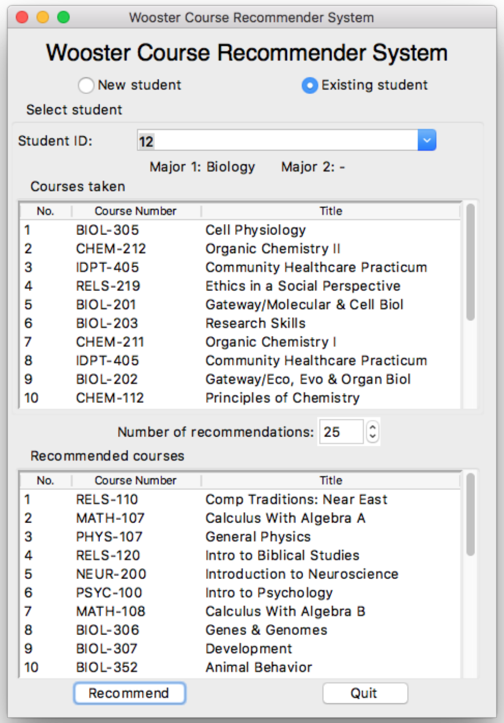
FYSM-101 {First Year Seminar}
MATH-111 {Calc & Analytic Geometry I}
MATH-211 {Linear Algebra}
MATH-225 {Mathematical Modeling}
MATH-329 {Probability & Statistics II}
CSCI-100 {Scientific Computing}
CSCI-120 {Data Structures and Algorithms}
CSCI-200 {Algorithm Analysis}
CSCI-320 {User Interface Design}

Number of recommendations: 20

Recommended courses

| No. | Course Number | Title |
|---|---|---|
| 1 | CSCI-110 | Imperative Problem Solving |
| 2 | CSCI-232 | Software Engineering-Databases |
| 3 | CSCI-212 | Operating Systems |
| 4 | CSCI-330 | Cmptr Networking & Commun |
| 5 | MATH-215 | Transition to Adv Mathematics |
| 6 | CSCI-220 | Theory of Computation |
| 7 | MATH-227 | Operations Research |
| 8 | CSCI-210 | Principles Compt Organ |
| 9 | MATH-229 | Probability and St |
| 10 | CSCI-222 | Programming Languages |

**Figure 3.19:** New student page of the CRS UI: user can create a random input.



Wooster Course Recommender System

New student [✓ Existing student]

Select student

Student ID: 12

Major 1: Biology Major 2: -

Courses taken

| No. | Course Number | Title |
|---|---|---|
| 1 | BIOL-305 | Cell Physiology |
| 2 | CHEM-212 | Organic Chemistry II |
| 3 | IDPT-405 | Community Healthcare Practicum |
| 4 | RELS-219 | Ethics in a Social Perspective |
| 5 | BIOL-201 | Gateway/Molecular & Cell Biol |
| 6 | BIOL-203 | Research Skills |
| 7 | CHEM-211 | Organic Chemistry I |
| 8 | IDPT-405 | Community Healthcare Practicum |
| 9 | BIOL-202 | Gateway/Eco, Evo & Organ Biol |
| 10 | CHEM-112 | Principles of Chemistry |

Number of recommendations: 25

Recommended courses

| No. | Course Number | Title |
|---|---|---|
| 1 | RELS-110 | Comp Traditions: Near East |
| 2 | MATH-107 | Calculus With Algebra A |
| 3 | PHYS-107 | General Physics |
| 4 | RELS-120 | Intro to Biblical Studies |
| 5 | NEUR-200 | Introduction to Neuroscience |
| 6 | PSYC-100 | Intro to Psychology |
| 7 | MATH-108 | Calculus With Algebra B |
| 8 | BIOL-306 | Genes & Genomes |
| 9 | BIOL-307 | Development |
| 10 | BIOL-352 | Animal Behavior |

**Figure 3.20:** Existing student page of the CRS UI: user can test the CRS using existing student data.

On the *New student* page, users can use the two drop down menus (Fig. 3.21) to select a course taken, and then hit the *Add* button below the menus (Fig. 3.22) to add the selected course inside the course input box. Users can also remove any item from the input box or clear the content of the box by clicking the *Remove* or *Clear* buttons.

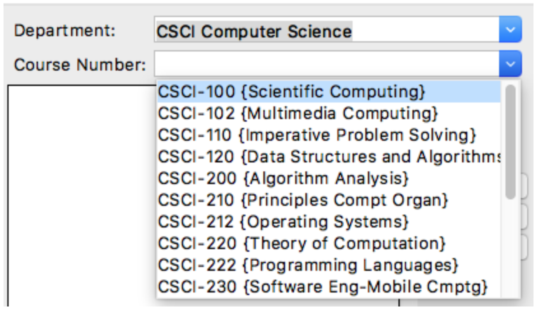

Department: CSCI Computer Science
Course Number:
CSCI-100 {Scientific Computing}
CSCI-102 {Multimedia Computing}
CSCI-110 {Imperative Problem Solving}
CSCI-120 {Data Structures and Algorithms
CSCI-200 {Algorithm Analysis}
CSCI-210 {Principles Compt Organ}
CSCI-212 {Operating Systems}
CSCI-220 {Theory of Computation}
CSCI-222 {Programming Languages}
CSCI-230 {Software Eng-Mobile Cmptg}

**Figure 3.21:** Select a course using two drop down menus.

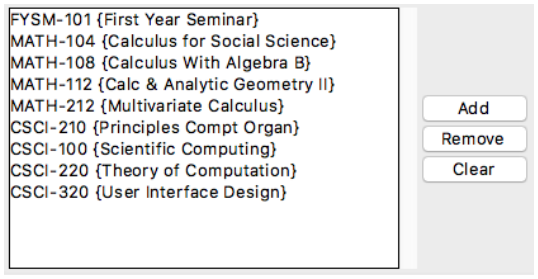
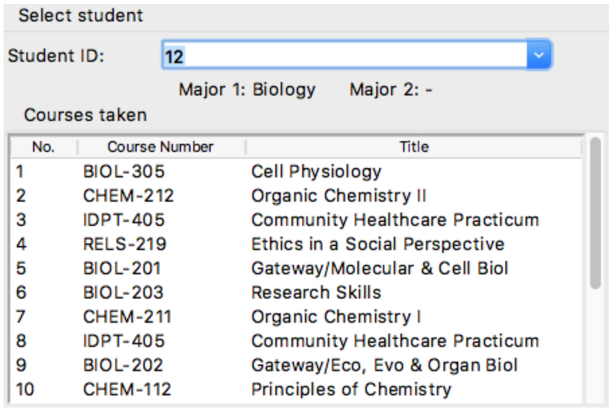

FYSM-101 {First Year Seminar}
MATH-104 {Calculus for Social Science}
MATH-108 {Calculus With Algebra B}
MATH-112 {Calc & Analytic Geometry II}
MATH-212 {Multivariate Calculus}
CSCI-210 {Principles Compt Organ}
CSCI-100 {Scientific Computing}
CSCI-220 {Theory of Computation}
CSCI-320 {User Interface Design}

**Figure 3.22:** Add, remove and clear the course input box.

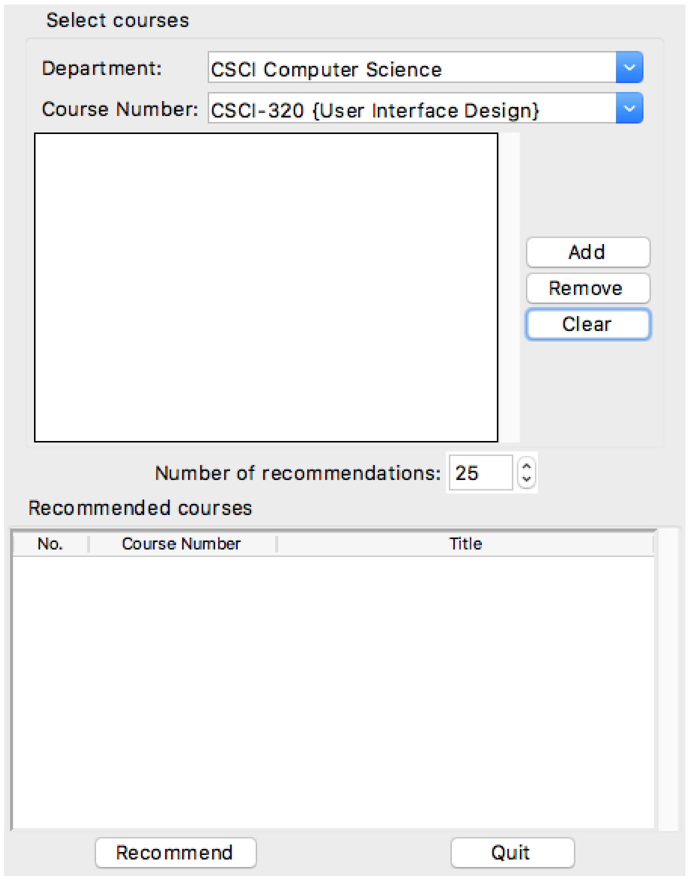
On the *Existing student* page, users can use the drop down menu to select a student ID in the data. Once a student ID is chosen, the text label below will show the major(s) information of the chosen student, and the input box below will display all the courses that this student has taken (Fig. 3.23).


Select student
Student ID: 12
Major 1: Biology Major 2: -
Courses taken

| No. | Course Number | Title |
|---|---|---|
| 1 | BIOL-305 | Cell Physiology |
| 2 | CHEM-212 | Organic Chemistry II |
| 3 | IDPT-405 | Community Healthcare Practicum |
| 4 | RELS-219 | Ethics in a Social Perspective |
| 5 | BIOL-201 | Gateway/Molecular & Cell Biol |
| 6 | BIOL-203 | Research Skills |
| 7 | CHEM-211 | Organic Chemistry I |
| 8 | IDPT-405 | Community Healthcare Practicum |
| 9 | BIOL-202 | Gateway/Eco, Evo & Organ Biol |
| 10 | CHEM-112 | Principles of Chemistry |

**Figure 3.23:** Select a existing student data as input to the CRS.

At the bottom of both pages is the recommendation box. The spin box above allows users to modify the number of recommendations from 20 to 100. The default value is 20. Once users click the *Recommend* button, the courses in the input box will be sent to the CRS and a list of courses with the chosen size will be returned and displayed in the recommendation box. If the input box is empty, by default the most popular courses will be recommended (Fig. 3.24).


Select courses

Department: CSCI Computer Science

Course Number: CSCI-320 {User Interface Design}

Number of recommendations: 25

Recommended courses

| No. | Course Number | Title |
|---|---|---|

**Figure 3.24:** Recommend the most popular courses when there is no input.

At this point, it is not sure yet which CF model will be used as the final model. In the next chapter, the three models will be compared to see which one is the best. The winner will be the final model.

CHAPTER 4

# Evaluation of the Course Recommender System Models

An important issue associated with RS deployment is the necessity of evaluation. In fact, a well-developed RS has to have evaluation in different stages of the life cycle. For example in the design phase, an evaluation can be conducted by running different algorithms on the same training data and compare their performance. There are known experiment design practices that can be followed, but in some cases, a self-designed metric better captures the property of RS and thus provides more convincing feedback.

The quality of experiment itself is critical as different metrics have different effectiveness in measuring the accuracy of RS. Designing the settings of experiment must be done carefully in order to minimize the systematic error. This requires a well understanding of the context of RS, including the input and output and the process of recommendation.

## 4.1 Accuracy Measurement of RS

There are two types of output of RS: *Ratings* and *Usage*. The way of evaluation for each type of output is different.

### 4.1.1 Ratings Prediction Measurement

An example of ratings prediction is when we try to predict the rating a user would give to a movie, say 1 to 5. We would use the users' ratings history as the training data for building such a system. The two most known and popular metrics for evaluating the results of ratings prediction are **Root Mean Squared Error** and **Mean Absolute Error**.

**Root Mean Squared Error (RMSE)** is the standard deviation of the *residuals*. Residuals are the difference between the actual values and predicted values. It measures how spread the residuals are.

Suppose there exists a set $S$ of user-item pairs $(u, i)$ for which the predicted ratings produced by the RS $\hat{r}_{ui}$ and the actual ratings $r_{ui}$ are known. Then RMSE of this RS is given by:

$$RMSE = \sqrt{\frac{1}{|S|} \sum_{(u,i)\in S} (\hat{r}_{ui} - r_{ui})^2} \tag{4.1}$$

**Mean Absolute Error (MAE)** is another popular metric for measuring the error of result. As its name suggests, MAE is the average of absolute error of all recommendations. It is given by:

$$MAE = \frac{1}{|S|} \sum_{(u,i)\in S} |\hat{r}_{ui} - r_{ui}| \tag{4.2}$$

### 4.1.2 Usage Prediction Measurement

Unlike ratings-predicting applications, usage-predicting RSs recommend items that users are likely to use. For example, on Amazon, when shoppers put one merchandise into the cart they are given a row of items that Amazon thinks they may also be interested in. In such applications, *Recall* and *Precision* are often used to evaluate the accuracy of the results.

In order to define recall and precision precisely, we begin by introducing the *confusion matrix*. It is a table that is often used to describe the performance of a classification model on test data for which the actual results are known. A confusion matrix is composed of four measurements:

1. **True Positives** (*TP*): number of instances predicted to be class $A$ and they do belong to class $A$.
2. **True Negatives** (*TN*): number of instances predicted not to be class $A$ and they do not belong to class $A$.
3. **False Positives** (*FP*): number of instances predicted to be class $A$ but they do not belong to class $A$.
4. **False Negatives** (*FN*): number of instances predicted not to be class $A$ but they do belong to class $A$.

For a usage-predicting RS, the confusion matrix is given by:

| | **Recommended** | **Not recommended** |
|---|---|---|
| **Used** | True Positive (TP) | False Negative (FN) |
| **Not used** | False Positive (FP) | True Negative (TN) |

**Table 4.1:** Confusion matrix of a usage-predicting RS.

These four measures can be combined in different ways to form advanced performance measures. For example *Recall*, *Precision* and *F1*:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.3}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.4}$$

$$\text{F1} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.5}$$

Recall and precision are concepts in the field of Information Retrieval. Precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned [2]. To better understand these two concepts, consider a teacher asks a student to write down the birthdays of 5 classmates. If the student recall 5 of them correctly, then his/her recall ratio is 1.0. If the student only got 3 of them correctly, then his/her recall is $\frac{3}{5} = 0.6$. However, having high recall does not show that the student has great memory. If the student answers 10 times to get all of them correct, i.e., half correct and half incorrect, then this student has perfect recall but he is not **precise**. In this case, the precision of this student is $\frac{5}{10} = 0.5$, which is not ideal. Sometimes a classifier is good at one and not so good at the other. Therefore, the harmonic mean of recall and precision, *F1*, is considered.

In our case, the CRS returns a list of courses to the target student, which can be viewed as it trying to predict what courses the student is taking in the future. Hence, it is an usage-predicting RS and therefore usage prediction measurements are applied to evaluate the performance of the CRS.

## 4.2 Evaluating the Course Recommender System Models

There are three CRS models implemented: UBCRS, IBDCRS and IBWCRS. All of them use the collaborative filtering approach. To evaluate the performance of these CF-based CRS, an experiment is designed and shown as follows (Fig. 4.1)

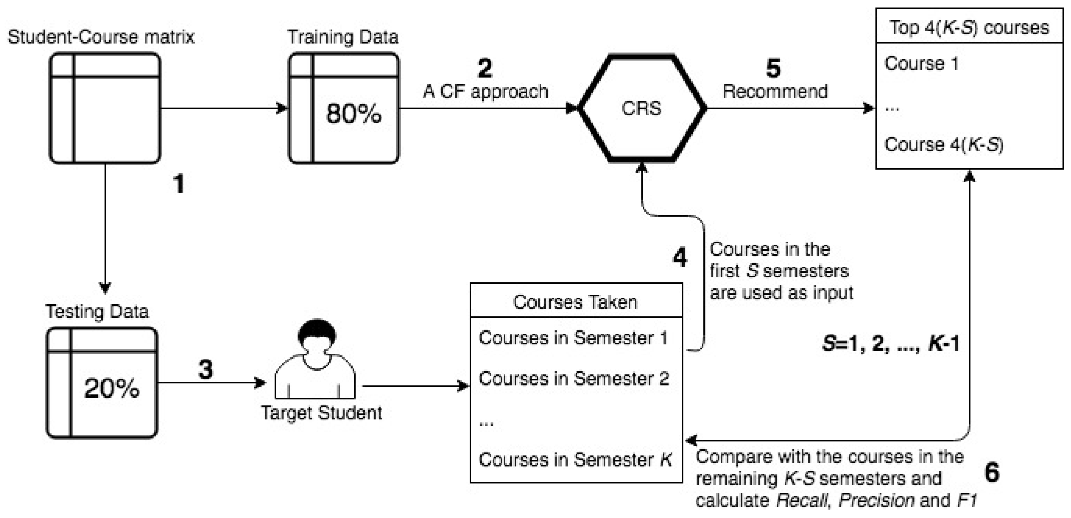


**Figure 4.1:** CRS evaluation flowchart.

1. Randomly split Student-Course data into two partitions: 80% are used as training data and 20% are used as testing data.

2. Create a CRS with one CF approach using the training data.

3. Randomly pick a student from the testing data and load his/her course history across $K$ semesters, where $K$ is max number of semesters the student spends in the college.

4. Select the courses taken in the first $S$ semesters as input to the CRS, where $S$ = 1 to $K$-1.

5. CRS recommends $4(K - S)$ courses to the student.

6. Compare the remaining courses of the student with the recommended course list and calculate *Recall*, *Precision* and *F1*.

The diagram illustrates the work flow of for evaluating the CRS performance with one randomly selected student. One complete round of evaluation is done by repeating the entire process for every student in the testing data. Further, to ensure the statistical accuracy of the evaluation, the evaluation is run for 10 rounds and each time the Student-Course data is partitioned randomly. In each round, the values of recall, precision and f1 are recorded. At the end we take the average of the measurements for each model and compare the models. Next the steps 4-6 are explained in details.

Chapter 1 explains that a student's graduation requires 32 course credits. Distributing the total course credits over the four years, that is 4 courses per semester, assuming each course worths one credit. Using this as the benchmark, we can simulate the usage by students after each semester by splitting their course history by semesters and inputing each semester's data to the CRS separately and comparing the recommended courses and the actual results. This means a student's data is used at most seven times in evaluation. In the first time, only the first semester courses are used as input and others are hidden. It is like simulating the student using the CRS as if he/she was at the middle of the first semester. Then the recommended courses are compared with the actual courses taken in the rest semesters and the measurements are recorded. In the second time, the second semester courses are used, so on and so forth. To illustrate this idea with example, consider a student whose course history is shown in Fig. 4.2.

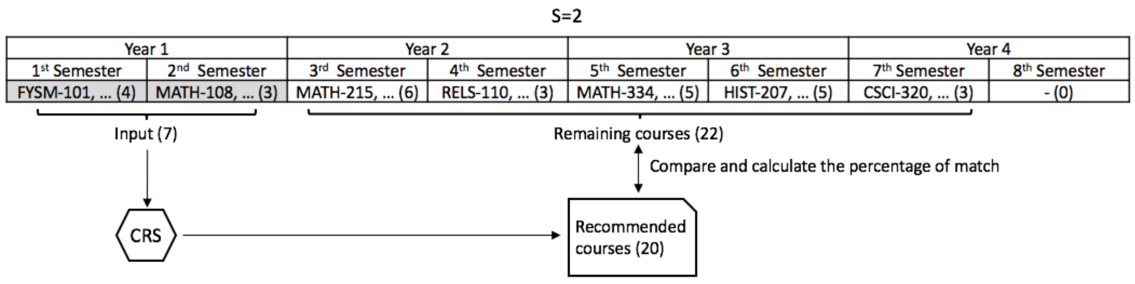


**Figure 4.2:** Evaluating the CRS using the first two-semester data.

This student has courses taken in seven semesters. Numbers in the parentheses indicate the number of courses in a semester. Although it shows that this student does not take four courses every semester, which is what most students do in reality, this does not affect the evaluation. Because this student has courses in seven semesters, his/her data is used six times in evaluation. The figure shows the second time of evaluation ($S = 2$). The seven courses taken in the first two semesters are passed to the CRS. And then, the CRS recommends $4(7 - 2) = 20$ courses to the student. These courses are what CRS predict that the student will take in the next five semesters. Later, the results are compared with the 22 courses that were actually taken in the last five semesters to compute *Recall*, *Precision* and *F1*. But before that, the values of TP, FP and FN are recored. The way they are recorded is described as follows:

**Algorithm 1** Algorithm for recording TP, FP and FN.

```
*recommendations* ← a list of courses the CRS predict the student will take.
*actuals* ← a list of courses the student actually took.
for course in recommendations do
    if course in actuals then
        $TP = TP + 1$
    else
        $FP = FP + 1$
    end if
end for
for course in actuals do
    if course not in recommendations then
        $FN = FN + 1$
    end if
end for
```

Using TP, FP and FN, we compute recall, precision and f1. Among these three measurements, recall is the easiest to understand and the most useful. If eleven of the recommendations are actually taken by the student, then the recall is simply dividing it by the number of actual courses: $\frac{11}{22} = 50\%$.

However, consider the fact that the goal of our CRS is not to predict what exact courses that a student is to take but to suggest a set of areas to consider about, we define a **looser** version of the experiment in addition to the current one (**strict** version). In the strict experiment, the complete course number is used when comparing recommended courses and actual courses. But in the loose experiment, only the department code, i.e., the first four letters of the course number, is used. Fig. 4.3 and 4.4 show the difference between the *recall* obtained from the two experiments on the same data.

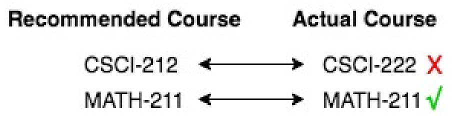


Figure 4.3: In strict experiment, the recall is 50%.

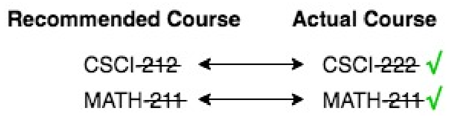


Figure 4.4: In loose experiment, the recall is 100%.

At the end, the performance of the three models will be evaluated based on the values of the three measurements obtained from both the strict and loose experiments.

## 4.3 Results

The final results are summarized in three figures below. Each figure illustrates one measurement for seven groups of data.

### 4.3.1 Recall

The figure of recall (Fig. 4.5) shows that IBWCRS has higher percentage than the other two models in every test group. In the loose experiment, the maximum recall of IBWCRS is about 60%, which is a good indication of its effectiveness. As for the other two models, there is an interesting pattern over the test cases. For the first four test groups ($S = 1, 2, 3, 4$), IBDCRS has advantage over UBCRS. But the amount of advantage decreases as $S$ increases. Then at $S = 5$, UBCRS is almost tied with IBDCRS and at $S = 6$ UBCRS overtakes IBDCRS and the difference gets larger as $S$ gets toward the end.

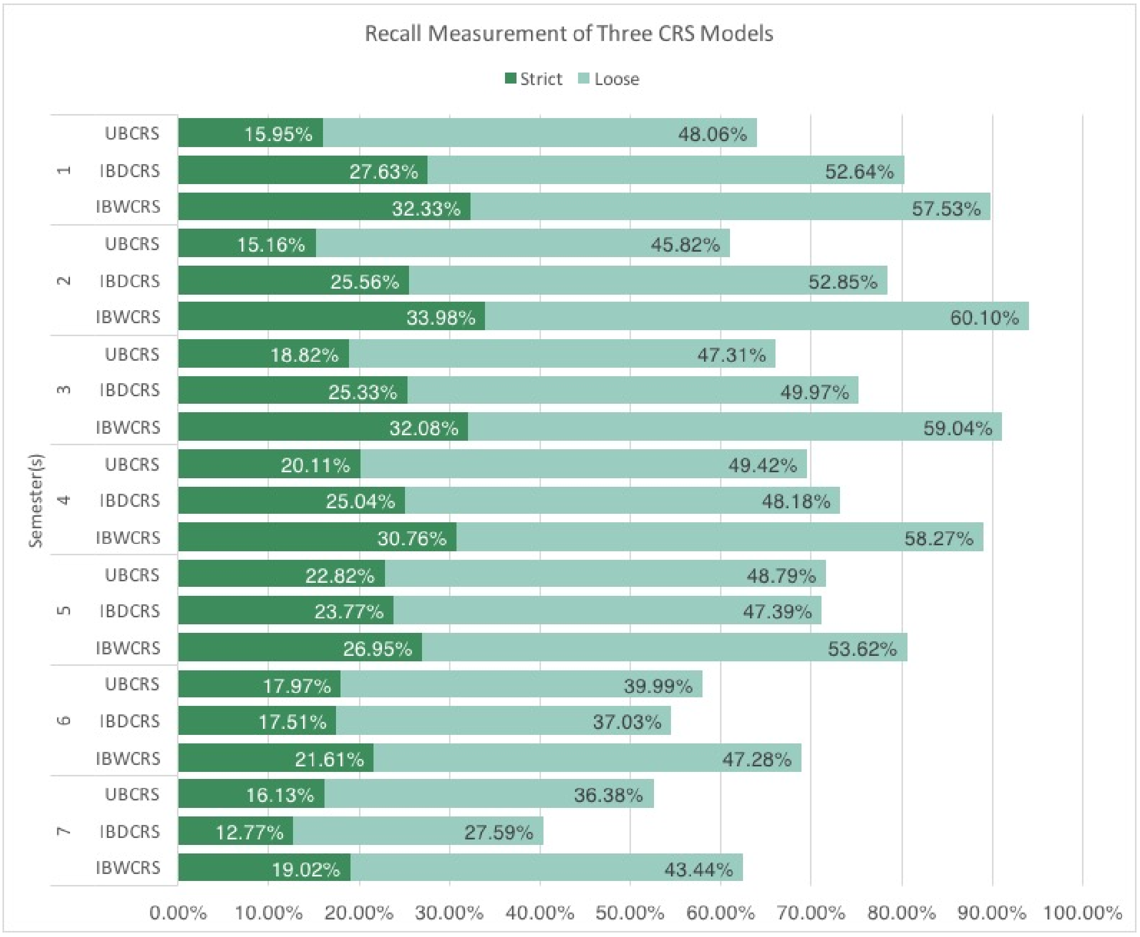


**Figure 4.5:** Recall comparison of UBCRS, IBDCRS and IBWCRS.

### 4.3.2 Precision

The figure of precision Fig. 4.6 displays the same trend as the figure of recall. IBWCRS continues to dominate the results in every test group.

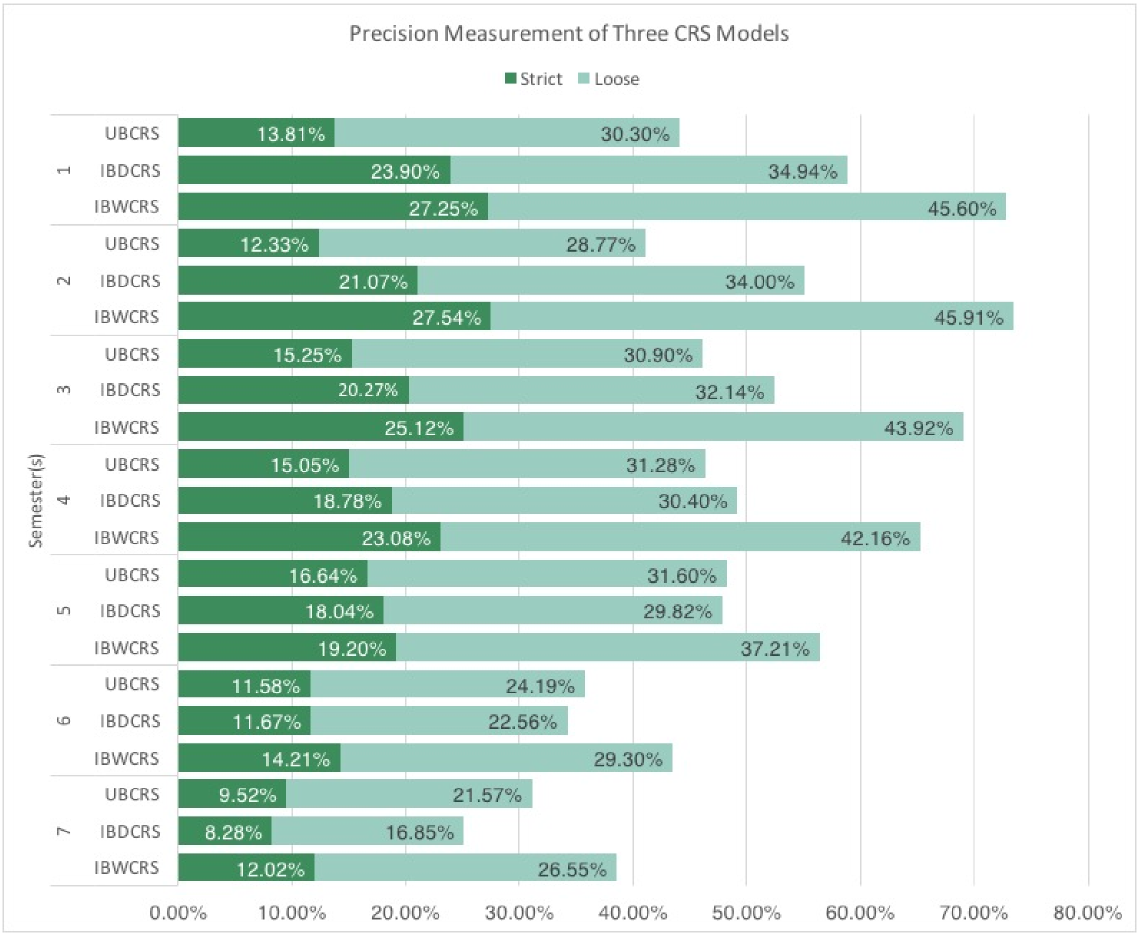


**Figure 4.6:** Precision comparison of UBCRS, IBDCRS and IBWCRS.

### 4.3.3 F1

Since f1 is the harmonic mean of precision and recall, the figure of f1 (Fig. 4.7) balances the trend of the precision figure (Fig. 4.6) and recall figure (Fig. 4.5). However as mentioned above, both figures reflect the same trend. This means the f1 figure demonstrates that IBWCRS has the highest f1-score overall.

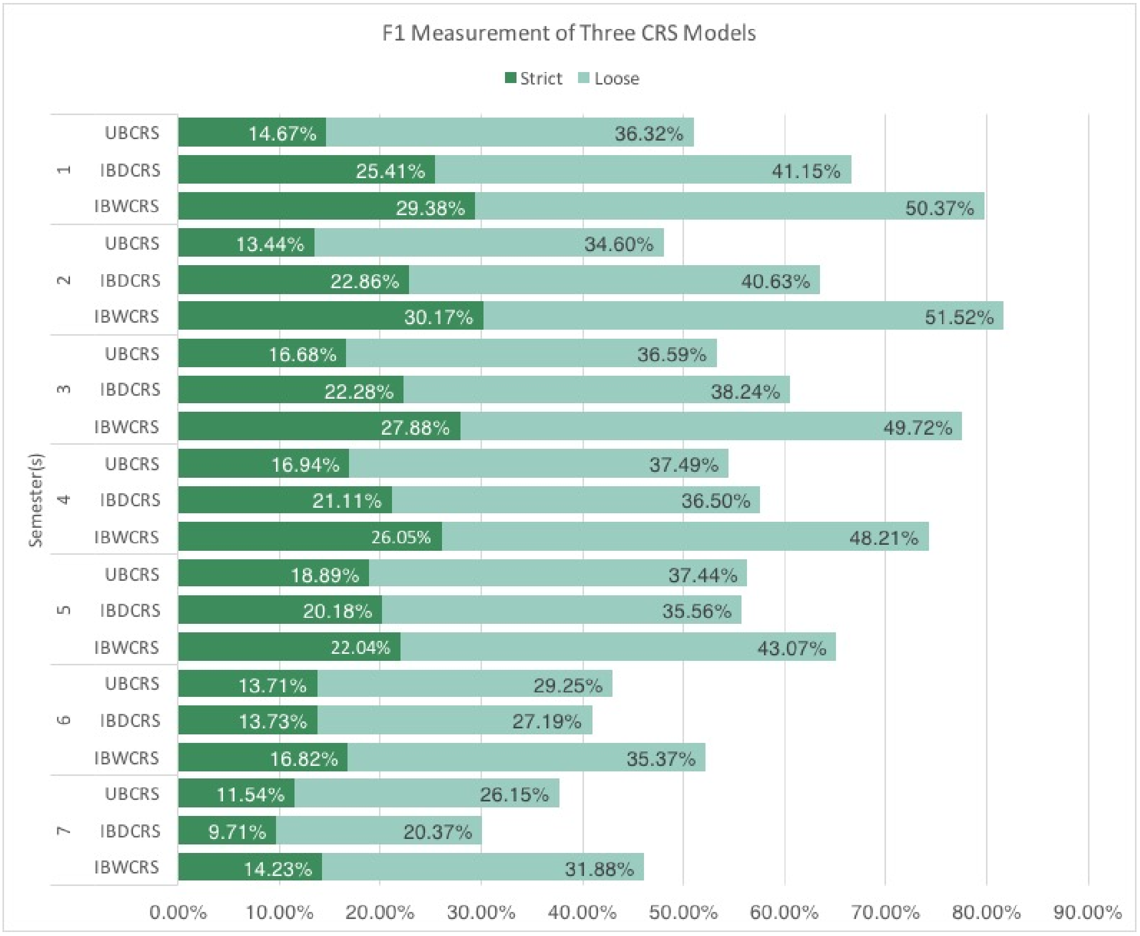


**Figure 4.7:** F1 comparison of UBCRS, IBDCRS and IBWCRS.

### 4.3.4 Run time

Recall that in each round of experiment a random 20% of the total 5,332 students are used as testing targets, which is approximately 1,067 students. Figure 4.8 records the average time taken for running the entire experiment for 1,067 testing students. As shown in the graph, IBDCRS is incredibly faster than the other two models, using only 45 seconds in both experiments. UBCRS and IBWCRS cost about the same time to finish an experiment, using more than 2 hours. The main reason why UBCRS and IBWCRS are much slower than IBDCRS is that these two approaches involve computing the neighbors of users ir items which can take up a significant portion of time, whereas IBDCRS does not need any additional computation other than the pairwise similarity.

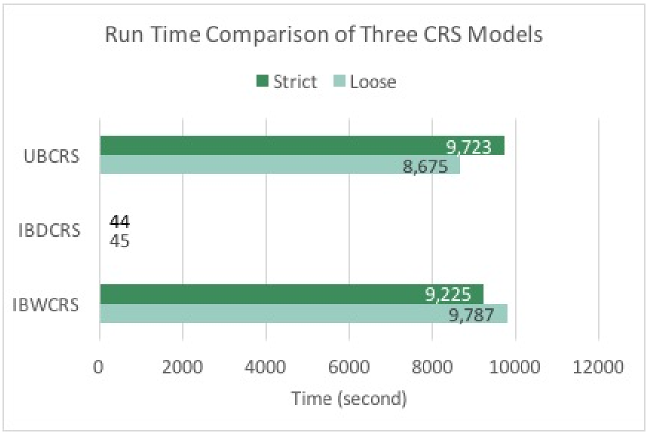


**Figure 4.8:** Run time comparison of UBCRS, IBDCRS and IBWCRS.

In conclusion, IBWCRS outperforms the other two models in every measurement, making it undoubtedly the most effective model among the three models according to our experiments. It is chosen to be the final model of our CRS.

CHAPTER 5

# Conclusion and Future Work

The primary goal of this research project is to create a course recommender system that provides personalized recommendations to students in the College of Wooster. In this thesis, we have investigated three common approaches (*Content-based*, *Collaborative Filtering* and *Association Rule Learning*) of building a recommender system and applied them to implement the course recommender system. The training data for this system contains information about students from class of 2009 to 2020 and courses in the last seven years (2009-2016). Among the three approaches mentioned before, two of the them (*Content-based* and *Association Rule Learning*) are abandoned because they do not work well with our existing data. Finally, we implement the course recommender system with three *Collaborative Filtering* models (UBCRS, IBDCRS and IBWCRS). For all three models, the similarity between subjects is calculated using Cosine similarity because the data is sparse.

The performances of the three models are compared using self-designed experiments and three measurements: *Recall*, *Precision* and *F1*. The results show that IBWCRS is more effective than UBCRS and IBDCRS in all the categories for all the test cases. Hence, IBWCRS is chosen to be the final model for our course recommender system. The average recall of our best model is about 28.10% in strict condition and 54.18% in loose condition. The values seem fairly low, but the intention of this course recommender system is to offer a guide to students, not a standard. After all, the course recommender system does not select the courses for the students, all it does is providing personalized suggestions. As long as the students feel that the recommendations are helpful and related to themselves, we think this CRS is successful.

In the future, there are several things that could be done to improve the CRS:

1. We may collect more data fields and test more recommendation approaches as well as more ways of similarity computation.

2. The current solution of the cold start issue is simple and straightforward. Some students may not prefer any of the popular courses. In the future, we may create a psychological test based on which the CRS can recommend personalized results.

3. Even though we spent a lot of time going through the course catalog, but because we did not consult each department for the correctness of the course replacement/removal, there might still be some incorrect and missing course information in the data. Plus, there might be new courses coming up in the future which will replace some existing courses. Hence, we need to consistently update the course data.

4. We may store the data in a relational database management system (RDBMS) instead of CSV files which is used currently. As data increases, a RDBMS is much more efficient than CSV in extracting and organizing data.

5. The program is written in Python using various libraries that are not really time-efficient. The main reason for using them is they provide functions that can conveniently manage and visualize data from CSV file. If we changed the way of storing data to database, we might as well use other libraries that are faster in processing data.

# APPENDIX A

# Tables

This appendix contains tables that are too large to fit in the regular pages. There are three tables in this appendix. The first two provide the details of our data preprocessing (see Section 3.2), including the replacement and removal of the outdated courses. The third table lists the 10 most similar users to Student 12, which is computed using user-based CF (see Section 3.3.4).

| **Original Course Number** | **Title** | **Replaced With** |
|---|---|---|
| ARTD-120 | Introduction to Art History | ARTH-101 |
| ARTD-151 | Introduction to Drawing | ARTS-151 |
| ARTD-153 | Introduction to Painting | ARTS-153 |
| ARTD-155 | Introduction to Printmaking | ARTS-155 |
| ARTD-163 | Introduction to Sculpture | ARTS-163 |
| ARTD-165 | Introduction to Ceramics | ARTS-165 |
| ARTD-171 | Intro to Digital Imaging | ARTS-171 |
| ARTD-204 | American Art & Nat'l Identity | ARTH-204 |
| ARTD-206 | Early Medieval Art | ARTH-206 |
| ARTD-207 | Late Medieval Art | ARTH-207 |
| ARTD-208 | Italian Renaissance Art | ARTH-208 |
| ARTD-212 | Baroque Art 1600-1700 | ARTH-212 |
| ARTD-214 | Nineteenth-Century Art | ARTH-214 |
| ARTD-216 | Gender in 20th Century Art | ARTH-216 |
| ARTD-220 | African Art | ARTH-220 |
| ARTD-221 | Islamic Art | ARTH-221 |
| ARTD-251 | Intermediate Drawing | ARTS-251 |

| **Original Course Number** | **Title** | **Replaced With** |
|---|---|---|
| ARTD-253 | Intermediate Painting | ARTS-253 |
| ARTD-259 | Intermediate Photography | ARTS-259 |
| ARTD-263 | Intermediate Sculpture | ARTS-263 |
| ARTD-270 | Concept Strategies Photography | ARTS-151 |
| ARTD-360 | Contemporary Art | ARTH-360 |
| BIOL-200 | Foundations of Biology | BIOL-111 |
| BUEC-255 | Organization of the Firm | BUEC-355 |
| BUEC-271 | Portfolio Theory & Analysis | BUEC-365 |
| CHEM-110 | Introductory Chemistry | CHEM-111 |
| CHEM-120 | Principles of Chemistry Lab | CHEM-112 |
| CHEM-399 | Biophysical Chemistry | CHEM-334 |
| COMM-140 | Clinic Practicum | COMD-140 |
| COMM-141 | Intro to Comm Sci & Disorders | COMD-141 |
| COMM-143 | Phonetic Transcrption & Phnlgy | COMD-143 |
| COMM-144 | Audiology Clinic Practicum | COMD-144 |
| COMM-145 | Lang Development in Children | COMD-145 |
| COMM-344 | Speech and Hearing Sciences | COMD-344 |
| COMM-345 | Speech and Hearing Science | COMD-344 |
| COMM-370 | Audiologic Rehabilitation | COMD-370 |
| CSCI-151 | Computer Programming I | CSCI-100 |
| CSCI-152 | Computer Programming II | CSCI-110 |
| CSCI-199 | Animation, Gmg & 3-D Virt Wrld | CSCI-102 |
| CSCI-251 | Prin of Computer Organization | CSCI-210 |
| CSCI-252 | Algorithms | CSCI-200 |
| CSCI-354 | File & Database Systems | CSCI-232 |
| CSCI-357 | Machine Intelligence | CSCI-310 |
| ECON-216 | Public Finance | ECON-315 |
| EDUC-240 | Interdisciplinary Fine Arts | EDUC-140 |
| GRK-101 | Beginning Greek Level I | GREK-101 |
| GRK-102 | Beginning Greek Level II | GREK-102 |

| **Original Course Number** | **Title** | **Replaced With** |
|---|---|---|
| HIST-203 | Roman Civilization | HIST-205 |
| IDPT-101 | First-Year Seminar | FYSM-101 |
| LAT-1010 | Beginning Latin Level I | LATN-101 |
| LAT-1020 | Beginning Latin Level II | LATN-102 |
| MATH-235 | Numerical Analysis | MATH-327 |
| MATH-241 | Probability & Statistics I | MATH-229 |
| MATH-242 | Probability & Statistics II | MATH-329 |
| MATH-300 | Introduction to Topology | MATH-330 |
| MATH-302 | Real Analysis I | MATH-332 |
| MATH-304 | Abstract Algebra | MATH-334 |
| MATH-306 | Functions of Complex Variable | MATH-336 |
| MATH-319 | Special Topics: Number Theory | MATH-299 |
| MUSC-211 | Music History I | MUSC-212 |
| NEUR-323 | Behavioral Neuroscience Lab | PSYC-323 |
| NEUR-380 | Cellular Neuroscience | BIOL-380 |
| PHED-130 | Varsity Sports, 2nd Half | PHED-131 |
| PHYS-101 | ALGEBRA PHYSICS I | PHYS-107 |
| PHYS-102 | ALGEBRA PHYSICS II | PHYS-108 |
| PHYS-121 | Astronomy of Stars & Galaxies | PHYS-105 |
| PHYS-122 | Astronomy of the Solar System | PHYS-104 |
| PHYS-205 | Modern Physics | PHYS-201 |
| PHYS-208 | Math Methods for Phys Sci | PHYS-202 |
| PHYS-303 | Modern Optics | PHYS-330 |
| PHYS-377 | Condensed Matter | PHYS-325 |
| PSYC-240 | Educational Psychology | PSYC-326 |
| PSYC-340 | Clinical Psychology | PYSC-331 |
| RUSS-199 | Artist & Tyrant | RUSS-260 |
| SOCI-215 | American Masculinities | SOCI-211 |
| SOCI-342 | Social Statistics | SOAN-341 |

| Original Course Number | Title | Replaced With |
|---|---|---|

Table A.1: Replacement of the outdated course numbers.

| Course Number | Title |
|---|---|
| ARAB-101 | Beginning Arabic Level I |
| ARAB-102 | Beginning Arabic Level II |
| ARAB-110 | Introduction to Arabic |
| ARTD-310 | Istanbul, Rome, Mexico City |
| ARTD-325 | Museum Studies |
| ARTD-401 | Independent Study |
| BIOL-199 | Conservation Biol in Tropics |
| BIOL-395 | Restoration Ecology |
| CCIV-260 | Rels in Ancient Mediterranean |
| CHIN-199 | War & Culture in China |
| CMLT-220 | Theory/Practice Transl(in Eng) |
| COMM-130 | Radio Workshop |
| COMM-220 | Intrapersonal Communication |
| COMM-221 | Interpersonal Communication |
| COMM-225 | Group & Organizational Comm |
| COMM-227 | Intercultural Communication |
| COMM-229 | Mass Communication Processes |
| COMM-231 | Radio, TV, & Film in America |
| COMM-233 | Mediated Gndr, Race, Sexuality |
| COMM-235 | Media, Culture and Society |
| COMM-244 | Audiology |
| COMM-250 | Principles of Rhetoric |
| COMM-252 | Argumentation & Persuasion |
| COMM-254 | Political Rhetoric |
| COMM-316 | Anatomy & Phys of Speech |
| COMM-332 | Visual Communication |

| **Course Number** | **Title** |
|---|---|
| EAST-199 | Introduction to East Asia |
| EDUC-199 | Lit for Children & YA Readers |
| EDUC-242 | Curriclum Stds Upper Elem Year |
| ENVS-205 | Entrepreneurship & the Environ |
| ENVS-235 | Gardening Practicum |
| HIST-298 | Making History: Theories/Meth |
| IDPT-406 | Global Social Entrepreneur Sem |
| IDPT-407 | Social Entrep Internship |
| MATH-219 | Number Theory |
| MUSC-210 | Basic Repertoire |
| PHIL-301 | Ontological Commitments |
| PHYS-199 | Physics of Sustainable Energy |
| PHYS-203 | Foundations of Physics Lab |
| PHYS-204 | Foundations of Physics |
| SOAN-240 | Research Methods |
| SOCI-111 | Social Problems |
| SPAN-399 | Don Quixote (in English) |
| THTD-100 | Arts & Entrepreneurship |
| THTD-104 | The Impulse to Create |
| THTD-243 | Exploring India Home & Abroad |
| THTD-300 | Theatre As Social Change |
| WGSS-310 | Sem in Feminist Learn&Teach |
| WGSS-320 | Queer Theory |

**Table A.2:** Removal of the outdated course numbers.

| | ID | Similarity Score | Class Year | Major 1 | Major 2 | Courses Taken |
|---|---|---|---|---|---|---|
| 1 | 4383 | 0.75 | 2018 | BIOL | - | BIOL-305 CHEM-212 MATH-111 SOCI-100 BIOL-202 BIOL-203 CHEM-211 ENGL-161 BIOL-201 CHEM-112 RELS-110 RELS-130 ANTH-110 BIOL-111 CHEM-111 FYSM-101 |
| 2 | 4420 | 0.727606875109 | 2018 | B&MB | - | CHEM-212 ENGL-120 IDPT-405 PHYS-108 RELS-219 BIOL-305 CHEM-211 IDPT-405 PHYS-107 BIOL-201 CSCI-100 IDPT-199 MATH-111 SOCI-100 BIOL-111 CHEM-112 ECON-101 FYSM-101 |
| 3 | 4687 | 0.6875 | 2018 | BIOL | - | BIOL-202 BIOL-305 IDPT-405 RELS-110 BIOL-100 BIOL-201 BIOL-203 IDPT-199 SPAN-202 BIOL-111 CHEM-112 RELS-110 SPAN-201 CHEM-111 FYSM-101 MATH-111 SPAN-102 |
| 4 | 4532 | 0.66697296885 | 2018 | B&MB | - | CHEM-212 ECON-101 IDPT-405 PHYS-108 SOAN-202 BIOL-305 CHEM-211 IDPT-199 PHYS-107 ARTH-208 BIOL-201 CHEM-112 MATH-111 BIOL-111 CHEM-111 ENGL-120 FYSM-101 IDPT-199 |
| 5 | 4463 | 0.66697296885 | 2018 | BIOL | - | CHEM-212 FREN-102 IDPT-405 MATH-102 BIOL-202 BIOL-203 CHEM-211 FREN-101 IDPT-199 ARTH-208 BIOL-201 CHEM-112 MUSC-216 BIOL-111 CHEM-111 FYSM-101 PSYC-100 |

| | ID | Similarity Score | Class Year | Major 1 | Major 2 | Courses Taken |
|---|---|---|---|---|---|---|
| 6 | 4863 | 0.666666666667 | 2018 | - | - | BIOL-201 CHEM-112 IDPT-199 MUSC-132 SOCI-100 BIOL-111 CHEM-111 FYSM-101 MATH-111 |
| 7 | 4656 | 0.645497224368 | 2018 | BIOL | - | BIOL-202 CHEM-212 COMD-145 RELS-241 BIOL-201 BIOL-203 CHEM-211 PHED-126 BIOL-111 CHEM-112 FREN-102 CHEM-111 FREN-101 FYSM-101 MATH-111 |
| 8 | 4605 | 0.645497224368 | 2018 | B&MB | - | BIOL-305 CHEM-212 RELS-130 THTD-303 BIOL-201 CHEM-211 SOCI-209 THTD-303 BIOL-111 CHEM-112 SOCI-100 SPAN-102 CHEM-111 FYSM-101 MATH-111 SPAN-101 |
| 9 | 4870 | 0.625 | 2018 | NEUR | - | BIOL-202 CHEM-212 COMM-152 IDPT-405 BIOL-306 CHEM-211 IDPT-199 PSYC-250 BIOL-201 FREN-102 NEUR-200 PSYC-100 BIOL-111 CHEM-112 FYSM-101 MATH-111 |
| 10 | 4561 | 0.625 | 2018 | B&MB | - | BIOL-305 CHEM-212 MATH-112 BIOL-306 CHEM-211 HIST-231 MATH-111 ANTH-110 BIOL-201 CHEM-112 HIST-207 BIOL-111 CHEM-111 FYSM-101 HIST-101 IDPT-199 |

**Table A.3:** Similar neighbors of Student 12.

# References

1. About wooster. URL http://www.wooster.edu/about/. 1

2. Precision-recall. URL http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html. 56

3. What happens online in 60 seconds? - smart insights digital marketing advice, Aug 2016. URL http://www.smartinsights.com/internet-marketing-statistics/happens-online-60-seconds/. 2

4. The College of Wooster, 1 edition, 2016. URL https://www.wooster.edu/_media/files/academics/catalogue/full/16-17.pdf. 1, 33

5. Sunita B Aher and LMRJ Lobo. Combination of machine learning algorithms for recommendation of courses in e-learning system based on historical data. *Knowledge-Based Systems*, 51:1–14, 2013. 4, 49

6. Xavier Amatriain. Recommender systems: Collaborative filtering and other approaches. URL http://www.slideshare.net/xamat/recommender-systems-machine-learning-summer-school-2014-cmu. 3, 6, 7

7. Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006. ISBN 1401302378. 2

8. Pei-Chann Chang, Cheng-Hui Lin, and Meng-Hui Chen. A hybrid course recommendation system by integrating collaborative filtering and artificial immune systems. *Algorithms*, 9(3):47, 2016. doi: 10.3390/a9030047. URL http://dx.doi.org/10.3390/a9030047. 4

9. Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

10. G.D. Linden, J.A. Jacobi, and E.A. Benson. Collaborative recommendations using item-to-item similarity mappings, July 24 2001. URL https://www.google.com/patents/US6266649. US Patent 6,266,649.

11. Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. Recommender system application developments: a survey. *Decision Support Systems*, 74:12–32, 2015. 3, 25, 49

12. Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 0387858199, 9780387858197. 12

13. M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, July 2010. ISSN 0740-7459. doi: 10.1109/MS.2009.161.

14. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001. 12, 21

15. Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321321367. 28, 29

16. Maksims Volkovs and Guang Wei Yu. Effective latent models for binary feedback in recommender systems. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 313–322. ACM, 2015. 4

17. Wikipedia. IMDb — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=IMDb&oldid=763982262`, 2017. [Online; accessed 06-February-2017]. 15

18. Wikipedia. Netflix Prize — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Netflix%20Prize&oldid=761558843`, 2017. [Online; accessed 07-February-2017]. 12