



BOOK RECOMMENDATION SYSTEM

Machine learning approaches and prediction algorithms for recommendation systems



12. SEPTEMBER 2019
SAMY SAMMOUR

Contents page

1. Introduction	2
2. Objectives	2
3. Recommendation Systems (RS)	2
3.2 Terms of use	3
4. BookRec Demo Project	4
5. Content-Based Filtering (CBF)	8
5.1 Factors	9
5.2 Score	11
5.3 Input Iteration Threshold	13
5.4 The algorithm	14
5.5 Advantages and disadvantages	17
6. Collaborative Filtering (CF)	19
6.1 User Temperature UTM	20
6.2 Output Weight OPW	22
6.3 Score	23
6.4 The algorithm	25
6.5 Advantages and disadvantages	28
7. Content-Based Filtering VS. Collaborative Filtering	30
8. Hybrid Filtering (HF)	32
8.1 Item Strength	32
8.2 Filter weight	33
8.3 Score	33
8.4 The algorithm	35
8.5 Advantages and disadvantages	36
9. Points of failure	38
10. Future perspective	40
11. Conclusion and future work	42
12. References	43

1. Introduction:

Modern Web applications are nowadays focusing more on increasing the interaction between users, and recommendation systems have become a common scenario. Users are not just expecting platforms to manage their work but also to help them find what to do next. People have to make many decisions in their life whether it is a product to buy, a film to watch or a book to read. The need for finding recommendations that help them to make decisions has been increased in the last 10 years. [1]

2. Objectives:

The aim of this study is to explain recommendation systems and the three major approaches: Content-Based, Collaborative and Hybrid filtering, and to develop a project to predict books based on them. The study explains the main differences between the three approaches and points out some of the issues that each one of them is facing. It also analyses some training sets to provide a reasonable solution for each issue.

The project is a platform (Website) developed to predict books in order to help its users to find interesting books to read next.

3. Recommendation Systems (RS):

Recommendations Systems are algorithms that provide predictions to users to helping them find out items they may be interested in. These suggested predictions are a result of a decision-tree process based on pre-collected information.

RSs are focusing on helping users who lack personal experience in evaluating a huge number of items in a system, reducing the unwanted options by excluding the non-interesting items from the dataset.

The items could vary from system to another whether they are products to buy, movies to watch or books to read. The RSs are normally meant to deal with a certain type of items only. Different types have different properties and attributes and of course different requirements, and that identifies every system with unique keys. This specification will require a unique implementation for each system.

RSs are personalized to users, which makes the results vary between users and groups. Two users who share the same interest and the same purchased items may receive different predictions based on their location.

Recommendation Systems processes and analyses the training data to predict a suitable output among the huge number of items which will require the system to collect information about the users, their preferences and previous activities. Those collected data will help to define the criteria that the decision-tree will use to find user's interest. [2]

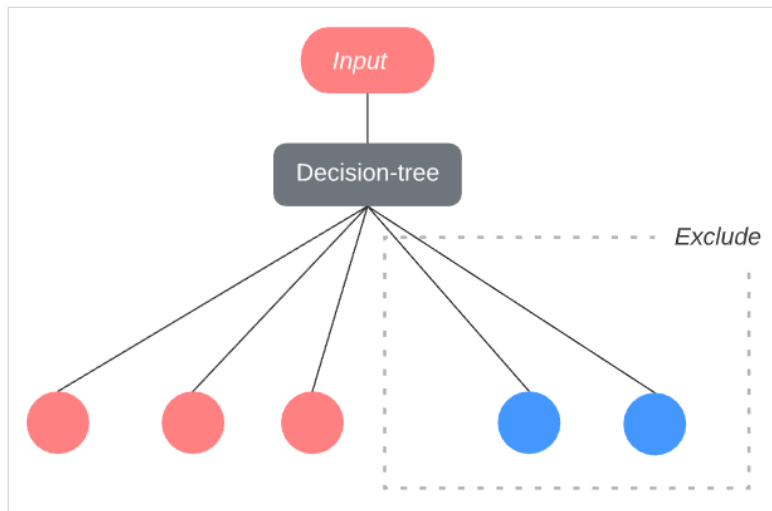


Figure 1

There are many approaches used for recommendation systems. The three most dominating approaches are:

- Content-Based Filtering (CBF)
- Collaborative Filtering (CF)
- Hybrid Filtering (HF)

These approaches use different behaviour to provide a suitable prediction and they will be mentioned in detail later in this study.

Before start explaining these approaches let's just introduce some terms that will be used in this study.

3-2 Terms of use:

Training set is a set of pre-collected data used as an example to help to spot the issues and to find suitable solutions.

Inputs are a set of items or data models that used as an input of the recommender to which the prediction will use to find items. Those inputs change among requests.

Prediction is a list of items selected from the recommender as a suggested list corresponded to the inputs

Item represents a single data model used by the recommender as an input or output

Match is an item that corresponds fully or partially with the user's interests

Perfect match is a match that perfectly matches user's interests

4. BookRec Demo Project:

BookRec project is a platform (Website) that helps users to find books that they may be interested in reading based on their previously read books. It also implements the three filtering algorithms: Content-Based, Collaborative, Hybrid filtering.

The following flowchart gives a better understanding of the project and how it works, *figure 2*

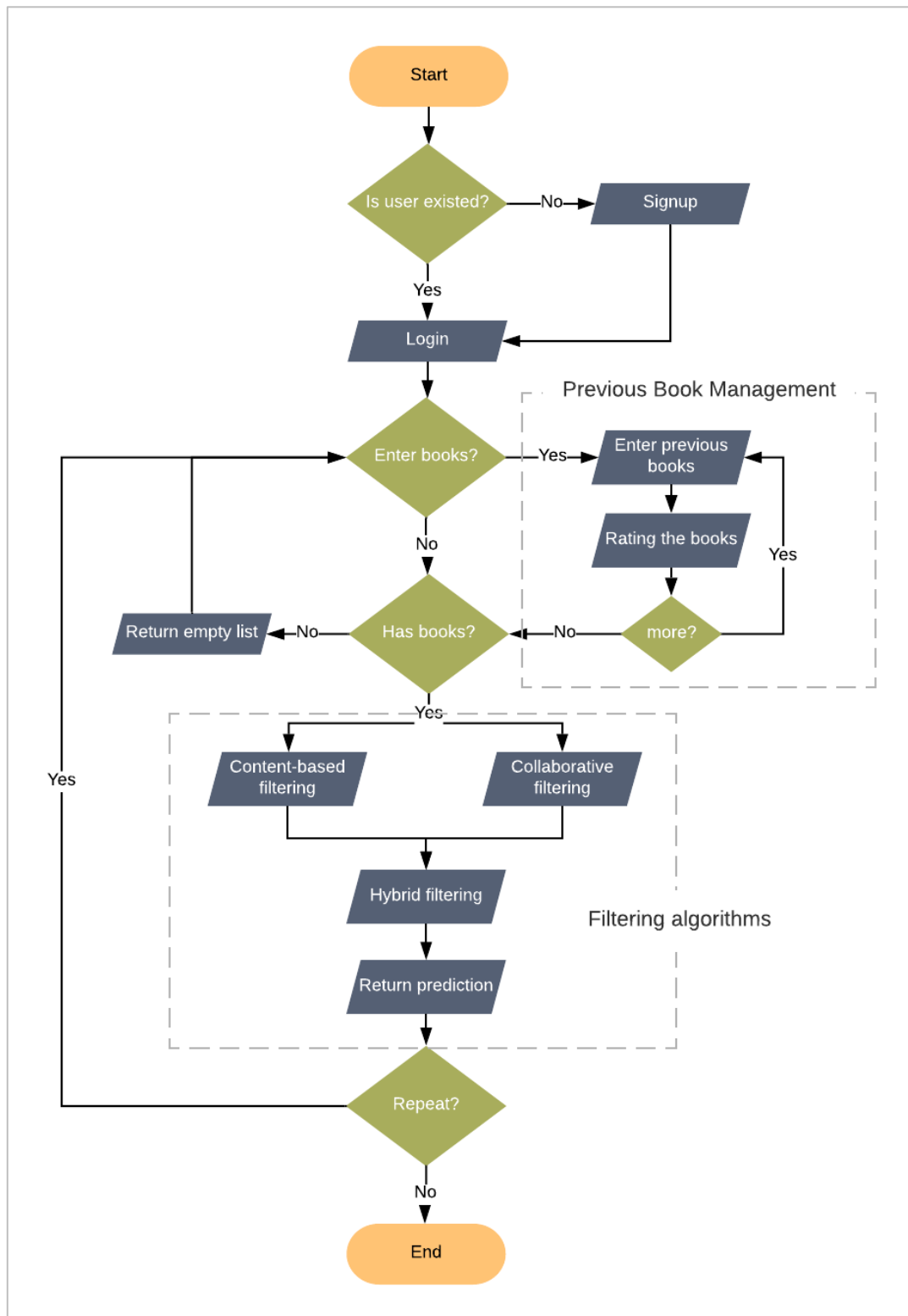


Figure 2

The prediction in BookRec is individual and depends on pre-entered books by each user. Therefore, the platform provides a simple user management portal that allows users to register, edit their personal data and add the books they have read.

Register and login will be the first step in the project to get a prediction as it is the first stage shown in the flowchart in, *figure 2*

The second stage is managing users' books which will be used as input items for the recommenders.

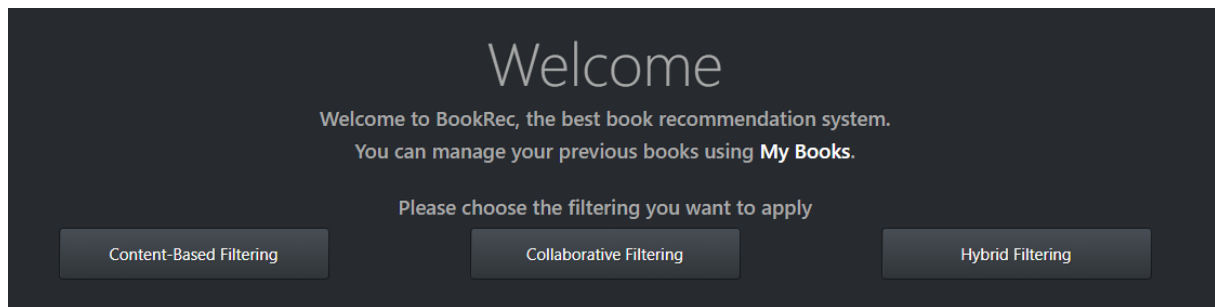
BookRec gives the users the ability to add books to their history from more 60 thousand books saved in the database and to rate them. Those books have been imported from the public non-commercial website "books.google.com" and will not be used beyond the purpose of this project [Online: See google terms: <https://developers.google.com/books/terms>] [Online: See google terms: <https://developers.google.com/terms/>]

What Is To Be Done?	
Category	Fiction
Language	English
Country	Germany
Mature Rating	NOT MATURE
Author	Nikolai Chernyshevsky
Publisher	Cornell University Press
Publish date	29.05.2014
★★★★★	

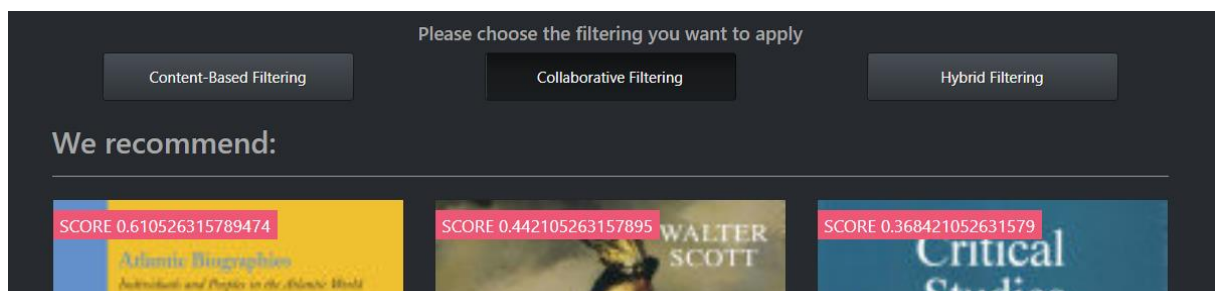
Murder By Ice	
Category	Fiction
Language	English
Country	Germany
Mature Rating	NOT MATURE
Author	Lorraine Burrell Hughes
Publisher	Xlibris Corporation
Publish date	01.02.2012
★★★★★	

Beautiful Disaster	
Category	Fiction
Language	German
Country	Germany
Mature Rating	MATURE
Author	Jamie McGuire
Publisher	Piper Verlag
Publish date	16.04.2013
★★★★★	

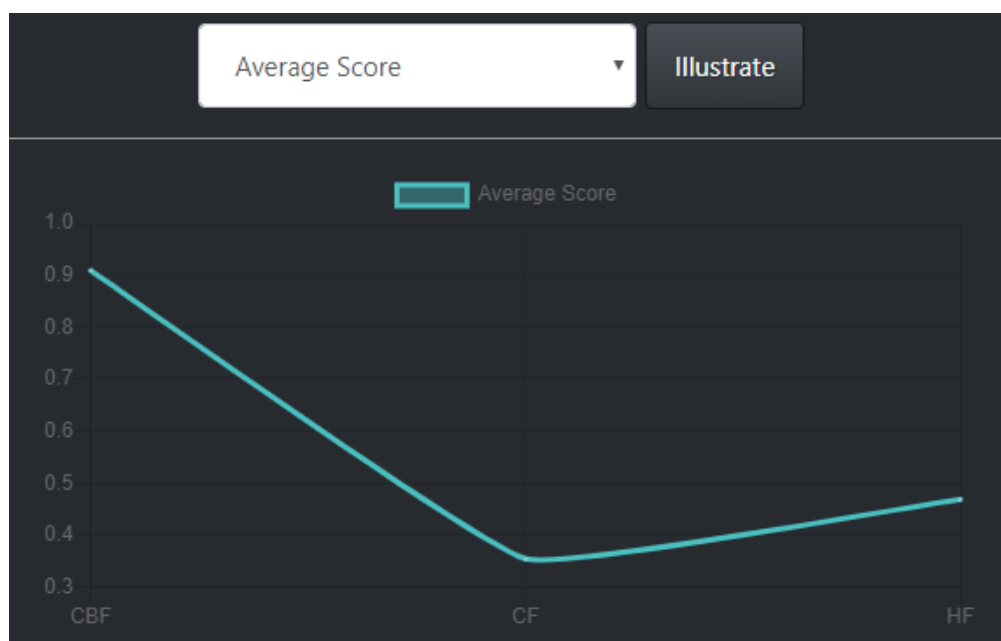
After adding the users' books, the three types of filtering will be available to find a prediction and can be started separately.

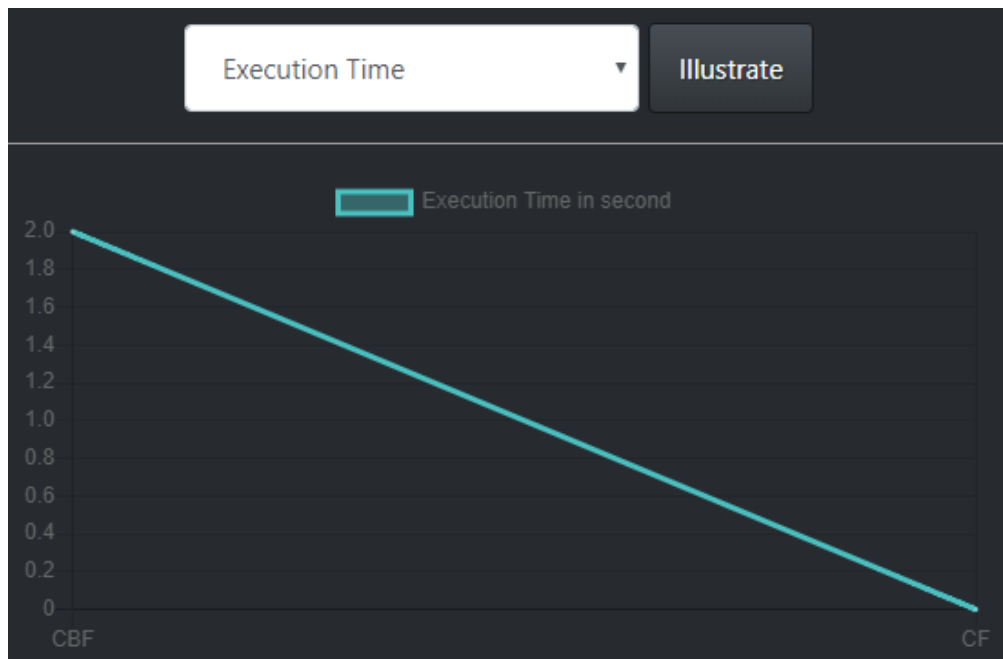


All filtering algorithms provide a prediction based on the input books and they calculate a score of each suggested item in the prediction that is displayed on each item for the ease of testing:



The platform provides also a simple chart system that compares the different types of filtering based on two different criteria: the average score of the items in the prediction and the execution time for each filtering. Those charts are specified to each user individually which will help to evaluate each type of filtering with different sets of inputs.





The project has been developed using Microsoft .NET Core Razor Pages framework. The business layer, which has been used to represent the data model, is Entity Framework Core.

Both the website domain and database are hosted in Microsoft Windows Azure.

The project is accessible online using the link:

<https://bookrec.azurewebsites.net/>

The source code can be found in Github with MIT Licence:

<https://github.com/samysammour/BookRec>

The functionality of each algorithm is tested using unit testing with mock data that can be found under the link: <https://github.com/samysammour/BookRec/tree/master/tests/UnitTests/Recommender>

5. Content-Based Filtering (CBF):

Content-based filtering is the most dominating recommendation system used in the market today. Huge applications like Amazon, Netflix and Facebook are using it to recommend items to their users. A real-life example would be user suggestion whom the user may be interested in adding to his social network based on their location, work, university studies or interests.

CBF is based on the similarities between objects (books, products, ... etc), referred to as items. It compares user profile with other items in the training set and finds the most similar item to it. For example, CBF can compare a book (input) within a category Fiction to find other books with the same category and suggest them as the output subset.

User profile is a term that represents all the data specified by the user that the CBF requires. It consists of static information that can be manually or automatically collected such as location (IP address), age, interesting tags, ... etc., or the user's previous activities like purchased products, visited items or watched movies.

CBF focuses on user profile and the content of the items and ignores users' experiences. It sets factors (the properties will be filtered on) to weigh the score of the outcome (prediction). Content can be very broad, and it becomes important to narrow down the factors based on relevancy. Otherwise, the number of predictions could be endless. The first issue with CBF becomes which factors to prioritize.

This study focuses only on one side of user's profile which is previous activities.

To understand the concept of CBF, a small demonstration could be helpful: the table below represents a small training set of books, and those books will be referred to with their IDs.

Id	Title	Category	Language	Author	Year published
1	The Arts of Korea	Art	EN	Jamie	1670
2	Art in museums lectures	Art	EN	Grace	1830
3	Arts of Power	Art	EN	Randolph	2019
4	What Is to Be Done?	Fiction	EN	Nicolai	1950
5	Murder by Ice	Fiction	EN	Lorraine	2012
6	Beautiful Disaster	Fiction	DE	Jamie	2013

Table 1

At first glance, we notice that the books can be categorized with different factors (e.g. category, language, author, etc.). Isolating either factor of category or language will leave us with two groups (Art and Fiction, and English and German, respectively), while the author factor categorizes the books into 5 different groups (note that Jamie has written two books). There are, on the other hand, no similarities between years. We will learn in the next section how some books may share the same era.

5.1 Factors:

Factors are the properties of the items by which they will be filtered. Those factors are not static and can be changed between algorithms. Therefore, the factors which are being used to predict products are different from those that will be used to predict books.

Factors don't just decide which properties will be checked, they also set the *weight*, or relevance, of each property due to the fact that some factors are more important than others.

However, not all of them have the same effect on the output. In fact, some of them can be completely ignored.

Table 1 defines 6 factors while in BookRec project there are 16 factors.

Factors can be split into three categories based on their impact on relevancy:

- **Hot** factors are those which provide the most relevant results. Filtering hot factors has the biggest impact on output and has the highest priority.

The process for designating a factor as hot is to analyse all the properties logically and consider the impact that each factor will have. For example, if a user is reading only English books, then why should the prediction contain non-English books?

Counting on that, both Category and Language are hot factors in the previous training set, *table1*. In the BookRec project, the 4 hot factors that have the most relevance are:

- Category: Fiction, historical, art, ... etc.
- Language
- Country in which the book is available.
- Maturity rating: determines the appropriate age level of the reader
- **Warm** factors are those which have less of an effect on the result than hot factors, but they will still be filtered. Those factors add value to the output score, but they may or may not impact the result. The author is a good example of a warm factor. Users who read books by a certain author would probably like to read other books by the same author. However, they may also be interested in reading books by other authors if these books are similar to what they read.

If hot factors can be marked as required, then warm factors will be 'nice to have.'

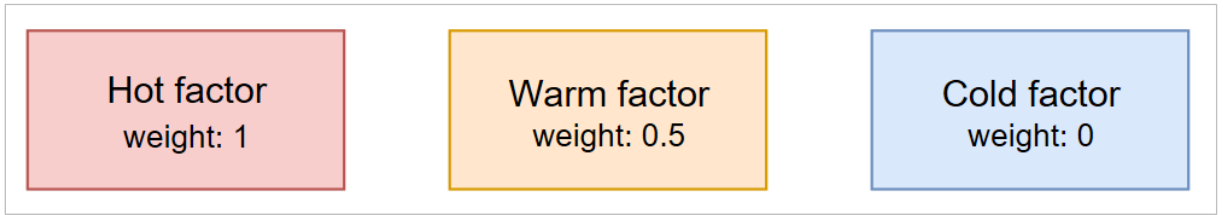
BookRec project defines 3 warm factors are:

- Author of the book.
- Publisher
- Year published (Publishing date range): Users are often interested in reading books from a specific era. In other words, the exact year the book was written in is irrelevant to the output. Rather, the era in which the book was written will impact the output. This publishing year will be grouped and scaled up to the nearest century. For instance, the publishing year of 1601 will be rounded to the 1700 and year 1999 will be rounded to 2000 while the year 1900 will not be scaled. So, a year group will be 100 years range.
- **Cold** factors are those which have no effect on the prediction. They will not be considered as filter criteria and can be ignored. Those properties are normally unique values that identify each item but are not relevant. Both Id and Title of the book are

good examples of cold factors. Their value is unique for each book and will normally not match. Even if two books share the same title completely or partially, it will not bring any value to the output. Examples of the 9 cold factors in BookRec project are all other properties not mentioned above including book cover, ISDN, page count, print type, etc.

Factors, as mentioned above, don't share the same level of impact. Each factor type has a different weight, which can be defined as the value that each will add to the final item score when this factor matches the input. Because the cold factors don't have any impact on the result, they will have a weight of 0. In fact, they will not be compared at all and will be completely ignored. Nonetheless, a hot factor has double the impact of a warm factor.

A single weight, in this case, will be represented by '1' for a hot factor and '0.5' for a warm factor.



In BookRec project, there are 4 hot factors and 3 warm factors. To calculate the total weight of a book, we multiply the amount of each factor by its respective weight and add them together, as represented in the following equation:

$$Total_w = (4 * 1) + (3 * 0.5) = 5.5$$

The resulting value represents the maximum weight when all the factors match the input or as we call it, a perfect match. Item weight will then be the summation of all matching hot factors' weight added to the summation of all matching warm factors' weight:

$$Weight_i = \sum_{1}^n weight_h + \sum_{1}^m weight_w = \sum_{1}^n 1 + \sum_{1}^m 0.5$$

n is the number of matching hot factors and m is the number of matching warm factors

Weight(i): the weight of the item

Weight(h): hot factor weight

Weight(w): warm factor weight

A book does not have to match all of the factors to be selected, but rather should reach a satisfying level which is pre-defined based on the requirement of the system. BookRec project sets this level to the summation of all hot factors which are required in the result.

$$Minimum_w = \sum_{1}^n 1 = 4$$

n is the number of hot factors

To calculate item weights from *table 1*, let's say a user, Bob, has read a new book 7.

Id	Title	Category	Language	Author	Year published
7	The art in Germany	Art	EN	Jamie	2005

Book 7 is an art book and has been written in English by Jamie in 2005 (follow the ear of 2100). When we compare the similarities from book 7 to the training set, we see that all the books in the training set share at least one factor with book 7. But do all of them reach the minimum satisfying weight?

To answer this question, we need to calculate the weight of each training item (book) based on the similarity to the input (book 7) and check if it exceeds the minimum weight. Id and title, both cold factors, are therefore disregarded. The hot factors are category and language in this example, while the author and the year are the warm factors. The years can be grouped into four eras: 1700, 1900, 2000, 2100.

Book 1 matches the category, language and the author but does not match the year-range. Its weight will be then:

$$Book1 : Weight_i = (1 * 2) + (0.5 * 1) = 2.5$$

Following the same equivalent, book 2 shares only hot factors, and the weight is 2. The weight of book 3 is 2.5, and of book 4 and 5 is 1 while of book 6 is 1 which shares only warm factors.

To know which items (books) are going to be selected in the prediction, we need to calculate the minimum weight, which is, in this case, 2 (considering $n = 2$). The total weight is 3, which would be the weight of a perfect match.

When comparing the minimum weight to each item weight, we notice that Book1, 2 and 3 have reached a satisfying level and will be selected in the prediction. Book 4, 5 and 6 have failed and will be eliminated.

BookRec sets a fixed value for all hot factors and another value for all warm factors for the ease of the implementation. However, in other systems, these values can vary between factors to define the impact of each individual factor on the prediction.

The next step will be calculating the *score* of the output.

5.2 Score:

The weight of the item is relative and related to the number of factors, which means it can be increased when the number of hot and warm factors increases. That will lead, at one point, to an unclear meaning of the weight. what is considered a good value in different systems?

A user or a programmer who does not understand how this algorithm works, will probably not understand what this value actually means. What is considered a good weight in one system may be considered a bad weight in another with more factors, or even in the same system if the factor definition has changed. In the previous example, a book with 2.5 weight is highly recommended. But in BookRec project, this value does not even reach the satisfaction level which is 4. For those reasons, a score for each item is needed.

The score is a value that represents the confidence of the suggestion. The score value is a scale from 0 to 1 (1 is the highest score) that will be calculated for each item based on its

weight. An item is considered as a perfect match when its score hits 1, whereas an item with a score of 0 will be immediately eliminated.

The score is the percentage of the element weight that shows how confident the match is due to the total weight:

$$Score = \frac{Item_w}{Total_w}$$

After calculating the score of the predicted items (books) in the previous example, the rounded value for both book1 and 3 are 0.8334 and for book 2: 0.6667.

The example in *table 1* is a simple example where the final result is only three books, but what if there were thousands of items that match the input? In BookRec project, the training set is more than 60 thousand books and that may lead to thousands of matches for some popular inputs.

The user is expecting to see not just the matches but also the best matches. Book 1 and 3 are more likely to interest Bob than Book 2 although Book 2 is still a point of interest.

This prediction needs to be sorted by score in descending order. Book 1 and 3 will appear first followed by book 2.

This sorted prediction should also be limited to a maximum number of items that will be displayed to the user. This limitation may also be extended by the user if desired. Some applications, like Amazon, provide the user with an extendable list of items with the ability to scroll or paginate to get more items.

BookRec CBF prediction is limited to 10 items with no ability to extend, as it is not the purpose of the project.

The previous algorithm produces items that match the input, but there is one issue remaining open in the previous filtering and appears when we have multiple books as inputs. Issue: When should an input value have the effect and be considered as a factor value?

To understand this problem, let's return to the previous example. Bob has read only one book; the values of the properties in this input will be considered as a factor value and will be matched with other items. But if Bob read 9 out of 10 books under the category Fiction and only 1 historical book, will both categories affect the output?

Just because Bob read one historical book, doesn't mean he is interested in this category.

An example story: A software engineering student is interested in technology and all the books he has read are related to this subject. He had in his final year one course about the moral usage of information and he had to read a philosophy book in order to write his final report. The input list contains now both categories: technology and philosophy, and they will both be considered as a value of the hot factor (category). The algorithm will iterate through the items in the training set and pick all that have one of the two categories.

But, what if his professor, the author of the philosophy book, has written many books on this subject, and the algorithm has found 10 perfect matches of the philosophy category? That will cause the prediction to contain only philosophy books which are not at all an interest to him. In this case, the algorithm has failed to predict.

These input values should be filtered first before being matched.

5.3 Input Iteration threshold:

The solution for this issue would be to define a threshold to which an input value will be picked as a factor value. The Input Iteration Threshold is the minimum iteration of an input value.

Value Iteration (VI) is the percentage of how often the value iterates to the total number of inputs.

$$VI = \frac{Iteration_v}{Iteration_t}$$

Bob read 9 fiction books, which makes the VI 0.9 and only 0.1 for historical.

Now, we need to set a logical threshold. The easy way to do it is to set a static threshold (e.g. 30%). So, an input value will have an effect only when its VI hits more than 30% of the overall inputs.

When comparing the VIs of both fiction and historical books Bob has read with the static threshold, historical category (0.1) has not exceeded the threshold (30%) and will then be disqualified as a factor value.

This solution partially solves the problem. But if all input values did not exceed the threshold, the CBF will not receive input and will predict nothing in return. If Bob, for example, read each 2 books out of 10 in a different category. The iteration will be equally divided to each category and VI will have the value of 0.2 for each. This does not exceed 30%, and therefore, all values will be disqualified.

Besides that, having a static value is not the correct solution. For some factors (e.g. author) the iteration will not likely happen very often compared to the category, which will normally be grouped into 2 to 4 categories. Thus, the threshold should be calculated for each factor separately.

The threshold represents the average of the iterations which is the summation of the VIs divided by the number of iterations:

$$Threshold = \frac{\sum_1^m VI_i}{n}$$

n is the number of inputs

m is the number of iterated value

In the example where Bob read only two categories, the threshold of the category will be: $(0.9 + 0.1 / 2) = 0.5$ For that, the historical category will still be disqualified, but in the second case where Bob read 10 books in 5 different categories, the threshold will be: 0.2 Now all the books will be selected and that solves the problem.

5.4 The algorithm:

All the issues of CBF have been covered and now it is the time to put everything together.

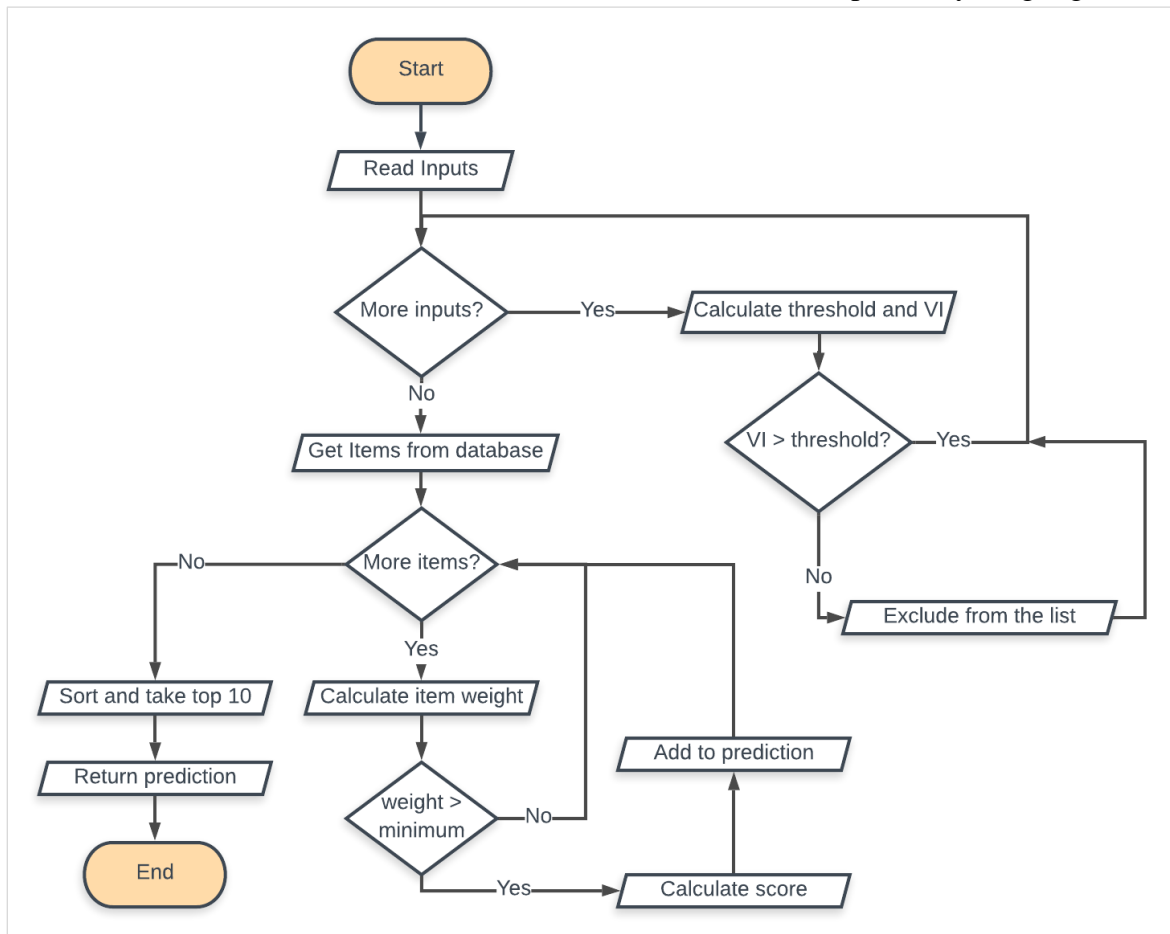


Figure 3

The flowchart above illustrates the flow of this algorithm, *figure 3*.

The algorithm receives the input (books have been read by the user) as a list and iterates through all the elements to find which value will be selected after calculating the threshold of each factor and the VI of the iteration of each value.

The following pseudo-code demonstrates the calculation of VI and threshold to decide whether to pick the input value as a factor value.

```
START
  INPUT books
  DEFINE inputValues

  FOR every book in books
    FOR every factor in book.Properties
      COMPUTE threshold, VI
      IF VI > threshold
        ADD value :=> inputValues
      ENDIF
    ENDFOR
  END
```

BookRec defines CBF options as a helper class that prepares all the necessary values needed for the algorithm.

```
var options = new ContentBasedRecommenderOptions(inputs);
```

The initialization of the options class will automatically filter the inputs for each hot and warm factor.

```
public ContentBasedRecommenderOptions(List<Book> inputs)
{
    EnsureArg.IsNotNull(inputs);

    this.FilterCategories(inputs);
    this.FilterMaturityRating(inputs);
    this.FilterLanguageCode(inputs);
    this.FilterCountry(inputs);
    this.FilterAuthors(inputs);
    this.FilterPublisher(inputs);
    this.FilterPublishedDate(inputs);
}
```

Each factor has its own filtering method because some values do not represent the actual filter criteria like publishing date. Its value is not the value that should be filtered, rather it should be rounded to the nearest century. Filter categories is an example of this function:

```
private void FilterCategories(List<Book> inputs)
{
    var groups = inputs.GroupBy(x => x.Categories);
    var threshold = (double)groups.Sum(group => group.Count()) / groups.Count();
    this.Categories = groups.Where(group => group.Count() >= threshold)
        .SelectMany(group => group.Select(x => x.Categories))
        .Where(x => x != null).Distinct().ToList();
}
```

Option class also contains other values like the minimum weight and helper functions to calculate the score and the satisfaction value for both warm and hot factors.

The options object holds the filtered values, and these values can now be checked against the items in the database to find similar items (matches). This is the second part of the algorithm as shown in the following pseudo-code.


```

START
  INPUT books
  // Calculate scores
  DEFINE score,
    scores,
    items: database books,
    minimum: minimum weight
    total: total weight
  FOR every item in items
    CALCULATE weight =:>
      FOR every factors in books
        CASE item = HOT FACTOR
          RETURN 1
        CASE item = WARM FACTOR
          RETURN 0.5
      IF weight > minimum
        SET score = weight / total
        ADD score to scores
      ENDIF
    ENDFOR

  SORT scores descending
  RETURN top 10 scores
END

```

The algorithm calculates the weight for each item in the database and checks it against the minimum weight. Only the items that pass the minimum weight will be added to the score list to be considered as a match. This score list will be then ordered by the score in descending order to prepare it for delivery.

Finding the weight of each item will be the next step. Item values should be checked against all selected values from the inputs that the options class has already calculated. This calculated value is the summation of all the matched hot and warm factors. Notice that hot and warm factor satisfaction is not equal, but actually differs due to the differences of the impact (the added value to the score). Below is a function that calculates the weight from the matched hot factors only:

```

public double HotFactorsSatisfaction(Book book)
=> (this.Categories.Any(x => x == book.Categories) ? 1 : 0) +
    (this.LanguageCode.Any(x => x == book.LanguageCode) ? 1 : 0) +
    (this.Country.Any(x => x == book.Country) ? 1 : 0) +
    (this.MaturityRating.Any(x => x == book.MaturityRating) ? 1 : 0);

```

The following code is an implementation for the complete algorithm after putting all the pieces together using Microsoft Entity Framework and C# code.

```
(from book in this.repository.DbContext.Books
let weight = options.HotFactorsSatisfaction(book) +
               options.WarmFactorsSatisfaction(book)
where weight >= options.MinWeight()
where inputs.All(x => x.Id != book.Id)
select new PredictionModel
{
    Book = book,
    Score = options.CalculateScore(weight)
}).OrderByDescending(x => x.Score).Take(10).ToListAsync();
```

The returned list of this algorithm will be then the prediction of this algorithm.

The implementation of Content-Based filtering can be found using the link:
<https://github.com/samysammour/BookRec/tree/master/src/Recommender/ContentBasedRecommender>

5.5 Advantages and disadvantages:

This algorithm provides a realistic prediction with an easy pattern “We recommend only what a user is interested in.”

CBF has the following advantages:

1. User independence: CBF analyses only the user’s profile that contains the items he already liked, bought or read, and the prediction is based on his interest only. This prediction is concrete and specified to a certain user regardless of what others may or may not like.
2. Transparency: CBF does not recommend items based on other unknown users that share similar interests just because they were interested in items the user is interested in. When a user buys a new phone, he will probably need a charger, a headphone and a screen protector. Those products are in the same category and are picked up from what the user already bought. Just because another user who bought a phone, has also bought a bike, does not make the bike as an interest to the user.
3. No cold start: A prediction will always be provided even if the user was the first visitor of this application or if he has not bought anything yet. Each user always has some standard data (static information in his profile) that CBF can use to predict like location, IP, age ... etc. This data guarantees that the user will see at least some items based on his location even if he has not been interested in items yet.

As with every algorithm, CBF does not fit every requirement. Only analysing the user’s items may not always be enough to provide the right prediction. The disadvantages of CBF are:

1. Ignoring other users: although this point can be seen as an advantage in some systems, it can still be considered as a disadvantage in others. The CBF ignores other users’ interest completely, including the rating of the items. People who a user may know is a

good example of such a situation. A user may not match another user based on their location, but they have met each other at a mutual friend's party and will likely add each other to their social network.

2. Lack of information: CBF finds similarities between items based on their contents. The more information this algorithm gathers, the more accurate the result will be. The wrong prediction may occur if the information is inadequate, or in case of non-clear (empty values) or dirty data (unvalidated data) like Germany as a country value, which can be written as Germany or Deutschland.
3. Over-specialization: CBF matches the items with the input after calculating the weight of the items in the training set. But, what if all the weights of all items did not pass the minimum weight? A total perfect filtering algorithm can sometimes suggest nothing. The specification of an item may sometimes be unique and not match any other item.

These disadvantages lead to another type of filtering (collaborative filtering).

6. Collaborative Filtering (CF):

Collaborative Filtering produces recommendations based on patterns of usage and rating. Unlike CBF, Collaborative Filtering does not focus on item similarities but rather takes into account what other users like and their opinion about items.

CF pattern is a combination of users' items who share the same input and their experiences with this item (rating)

CF is considered one of the most important filtering approaches in recommendation systems. In fact, it plays a central role in the Netflix system (one of the best recommendation systems in the world). It has also been used in social media to find friends of friends.

Items' content cannot always be enough to suggest what the user may need. A user who has bought a hammer, for example, may also need gloves to protect his hands. Gloves and hammers do not share the same category and may also be from different manufacturers and even different countries. However, many users who bought the hammer have also bought gloves. Therefore, based on users' purchase history, the user may be interested also in gloves, which could be a valuable item added to the prediction. [9]

CF has been massively improved since 2006 when the Netflix prize had begun. This has encouraged thousands of developers, scientists and industries to build new techniques to improve the prediction accuracy. [9]

CF observes users' behaviour and collects information about the items in the system, taking their experiences into consideration.

This algorithm faces many challenges that need to be solved. So, let's start with a little demonstration to understand these challenges and find a suitable solution for it.

Id	Title	User	Rating
1	The Arts of Korea	Bob	4
2	Art in museums lectures	Bob	5
3	Arts of Power	Bob	3
1	The Arts of Korea	Alice	3
3	Arts of Power	Alice	4
4	Murder by Ice	Alice	5
5	Beautiful Disaster	Alice	4
2	Art in museums lectures	Laura	5
6	Time travel	Laura	4
4	Murder by Ice	Laura	5
7	The art in Germany	Laura	3
8	Usage of information	Jack	4
5	Beautiful Disaster	Jack	3

Table 2

This table, *table 2*, demonstrates a training set of books that have been read by different users and have been rated in a 1 to 5 using a star-based rating system. Bob has asked for a prediction for what to read next. We have here three users (other than Bob) who read and rated books, which will be the source of the prediction that may interest Bob. Although the table shows 5 Books that Bob did not read, is he really interested in all of them?

The user Jack does not share any books with Bob and has completely different interests, while both Laura and Alice do share books. The first challenge becomes whether all books based on the reading history of the users should be in the prediction? Do they have the same level of importance?

6.1 User Temperature (UTM):

As mentioned above, Jack does not share any books with Bob. Therefore, all the books read by Jack are not of interest to Bob and should be ignored. That leaves us with 5 items. (2 read by Alice and 3 read by Laura). Book 4 is iterated twice and read by both of them.

We notice that Alice has shared 2 books with Bob, while Laura has shared only 1. That makes her closer to Bob, which gives the books she read a higher priority.

The graph below illustrates the relationship between users and their books.

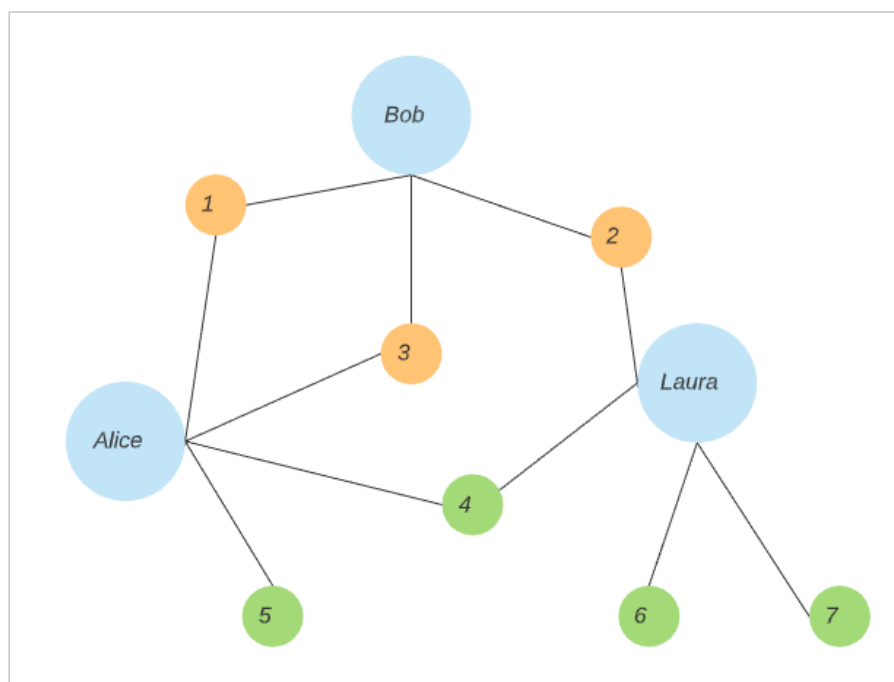


Figure 4

But let's not rush to conclusions. The rating of the books is also important to determine which user is a better match because a lower-rated book is still not a big interest. Both Alice and Laura read Book 1, but they have rated it differently. Laura shares the same rating of book 2 with Bob which makes the books suggested by Laura's reading relevant based on the rating.

But is the number of shared books or the rating more important in the prediction?

UTM represents the match value in the scale of 0 to 1 between the input user (Bob) and the user who shares interest (Alice and Laura); In other words, how close the input user is to those who have shared books and provided a rating.

User temperature will take both match criteria, the rating and the number of the shared books, into consideration. But first, we need to find the impact of each star in the rating.

To understand the rating system, let's isolate each match and compare it separately with Bob's. We can see that Bob and Alice did not rate the same two books equally.

Id	Title	User	Rating
1	The Arts of Korea	Bob	4
2	Art in museums lectures	Bob	5
3	Arts of Power	Bob	3
1	The Arts of Korea	Alice	3
3	Arts of Power	Alice	4




Table 3

Bob has rated Book 1 with one star more than Alice and Book 2 with one star fewer. In order for Alice to match Bob's rating, she needs to add one more star for Book 1 (+1) and to lose one star for Book 2 (-1).

Counting on finding the summation of the stars of the shared books for each user is not enough, because in this case, both Bob and Alice hit 7 stars collectively, but they are still not a perfect match.

Each star is actually connected to its book and has an impact on the temperature. Alice has missed Bob by two stars regardless if she has a star more or less. But after a closer look, Alice matches only two of Bob's books. This will make her miss 2 stars from the shared books and 5 stars from the book that she completely misses, increases the number of missing stars to 7. The sum of the absolute value of the differences will be then:

$$\sum_{i=1}^m |r_i - ur_i|$$

m is the number of input items

r is the rating of the input item

ur is the rating of the matched item

We do not just need to find the difference, but we also need to calculate the impact of this difference in the overall rating. We can achieve that by dividing it by the summation of the input stars of all input books.

$$\frac{\sum_{i=1}^m |r_i - ur_i|}{\sum_{i=1}^m r_i} = \frac{7}{14} = 0.5$$

m is the number of input books

r is the rating of input book

The last equation calculates the impact of the differences in the rating between the input user and the matched user related to the number of the shared books. This means that Alice missed matching Bob by 0.5 and therefore, Alice's temperature will be: $1 - 0.5 = 0.5$

To put everything together, the total equivalent will be:

$$UTM = 1 - \frac{\sum_{i=1}^m |r_i - ur_i|}{\sum_{i=1}^m r_i}$$

A perfect match, in this case, will only occur when a user shares all input books and matches the rating of them. The difference will then be 0, and the UTM will be: $1 - 0 = 1$

Let's calculate the UTM for Laura using the same equivalent.

Id	Title	User	Rating
1	The Arts of Korea	Bob	4
2	Art in museums lectures	Bob	5
3	Arts of Power	Bob	3
2	Art in museums lectures	Laura	5

Table 4

Laura matches the rating of book 2, but she still lacks 2 books in common with Bob. The difference for Laura will also be 7, which means she will have the same UTM as Alice.

In the training set, there is still a user, Jack, who does not share any book with Bob and misses all the stars. This makes his UTM equal to 0 ($1 - 1 = 0$), and his result will be automatically excluded from the result.

We measured in the previous section the temperature of each user compared to the input user. These temperatures help the CF pick up the books read by the matches. And that will lead us to the second challenge: Are all books from all matches equally interesting?

6.2 Output Weight (OPW):

All the books from users who have a temperature value are a point of interest. These books, however, are not equally interesting to the input user. Books from a user with higher temperature are more interesting than those from a lower temperature user.

An example story: a group of students have watched superhero movies together at weekends during study time and were very happy because they shared the same interest. During summer vacation, they travelled to their origin country, and some of them broke the agreement and watched movies alone. When they got together the next year, they decided to re-watch the missed movies together, but they were not sure which ones to watch. Therefore, they tried to see which movies were the best among all of them by collecting opinions about the movies to decide which one would be most interesting to watch next.

Examining this story from the previous information in *table 2* will lead to the choosing criteria which consist of the rating of the items read by the matching users.

By putting only Alice on the spot, we notice that she read 2 other books that Bob has not yet read. Alice liked Book 4 more than Book 5 and rated it higher which makes it more interesting to Bob than Book 5.

The criteria would then be to pick up the higher books first, and the result based on Alice only and sorted by relevance would be: Book 4, Book 5.

Following the same pattern, the result based on Laura will be then: Book 4, Book 6, Book 7.

Laura and Alice currently share the same temperature. But recently, Laura read another book, which adjusts the training set.

Id	Title	User	Rating
3	Arts of Power	Laura	3

Laura has a new temperature because of the change in her reading. Her temperature now is 0.71428 which is higher than Alice's temperature (0.5). This change made Laura's books more interesting than Alice's books regardless of the rating.

Ignoring the rating between users is not completely correct. A book with a rating of 1 from a higher temperature user is still not as good of a prediction than a book which has a rate of 5 from a lower temperature user.

Dividing the rating of the book to the maximum rating (5 stars) will give the value of the interest of this book, and 1 will be the perfect match.

This value of interest should also relate to the user's temperature to find the impact of each book among all users.

OPW is the weight of each book related to its rating and user temperature.

$$OWP_i = \frac{temperature_u * r_i}{5}$$

r(i) is the rating of the selected item
u is the selected user

Sorting the prediction by score based on the OPW of all books suggested from Alice and Laura will give us the following list:

L: Book4, L: Book6, A: Book4, L: Book7, A: Book5 (L: Laura, A: Alice)

Alice	
Book4:	0.5, Book5: 0.4
	₃ ₅
Laura	
Book6:	0.571424, Book4: 0.71428, Book7: 0.428568
	₂ ₁ ₄

Before delivering the prediction to Bob, there is one more point. The prediction is now sorted by relevance but Book 4 appears twice (has been read by both Alice and Laura). The first index where the book appears is 1 and the second is 3 in the prediction. It is not logical for a prediction to repeat the same item more than once. Instead, we need to choose one of them, or maybe calculate a new value based on both.

6.3 Score:

The score is the value of each item in the prediction that determines how highly this item is recommended. There are two cases that appear in the prediction:

1. The item appears only one time in the prediction. Its score will then be the value of the OPW that has been calculated in the previous step.
2. The second case, as with Book 4, is when the item is repeated, which will be explained next.

The iteration of an item does not, however, duplicate the effect of its impact. Rather, it increases its worth. Nonetheless, if an item is suggested based on only one user, it does not mean it is not interesting.

Let's say a user has read 10 books, and we have 20 users who share reading. One user shares all the books with the input user and the other 19 users share only one book with him. Even if a book has been read by all 19 users who have the lowest temperature, it would not make it the best choice. Rather, a book read by the only user who is a perfect match and has a temperature of 1 is still more interesting.

The first logical solution that comes is to calculate the average of the OPWs. But, based on the definition of the average, the number between the highest value and the lowest value, if one of the users has a low temperature, it will influence all other OPWs and decrease the overall value and negatively impact the iteration. Therefore, the average is not the correct solution.

The following graph will help in understanding the issue:

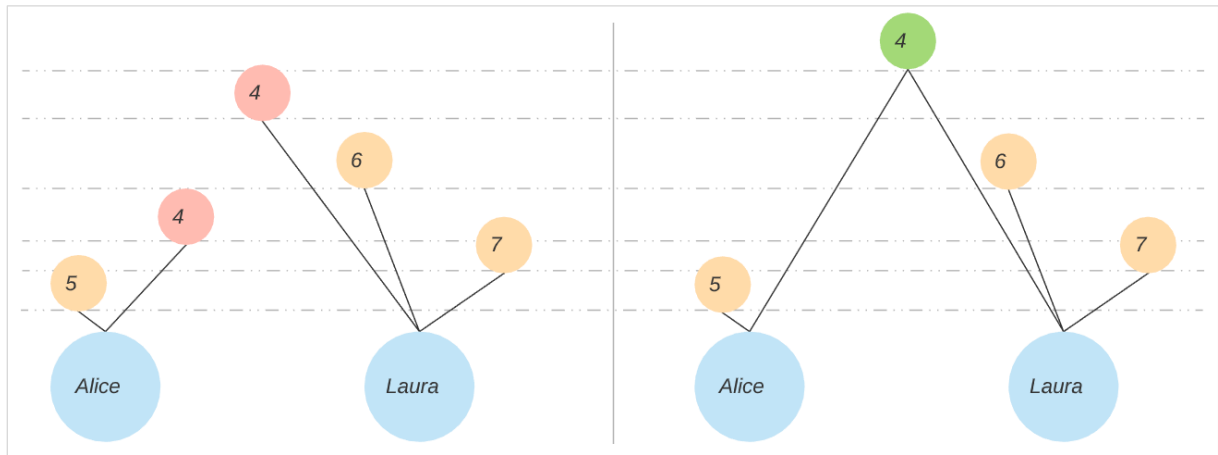


Figure 5

The goal would be to combine the repeated node number 4 and increase its OPW as shown on the right side but without exceeding the maximum value (1).

The value, which the iteration will add should increase the maximum value of the iterated item, which will, in this case, be book 4 from Laura (0.71428).

The first step is to calculate the value that each iteration will add to the maximum value. By looking at the value 0.71428, we notice that the window left for the win is $1 - 0.71428 = 0.28572$.

The previous value is the maximum value that can be added to the total score, but let's find out how much each iteration will add. This value divided by the number of iterations (users) will be the amount that each iteration can add, or the Iteration Max Value (IMV).

$$IMV = \frac{1 - \max(OPW)}{m} = \frac{1 - 0.71428}{2} = 0.14286$$

m is the number of iterations

The fact that the temperature between Laura and Alice is different, means, the value added by Laura is higher than the value added by Alice and this value can be up to the value of IMV.

Multiplying the IMV by the temperature of each user will give the total value that an iteration will influence. The summation of all the added value will then represent the total increase of the OPW.

$$\sum_1^m \left(\frac{1 - \max(OPW)}{m} * UTM_i \right)$$

The final step is to add the calculated value from the equivalent above to the highest value between iterations which will provide the final score in case of iteration:

$$Score = \max(OPW) + \sum_1^m \left(\frac{1 - \max(OPW)}{m} * UTM_i \right)$$

That will be for Book 4: $0.71428 + 0.17348 = 0.88776$

This new score increases the interest in book 4 which, in our example, will not change the order. However, the iteration in other cases may change the order of the items in the prediction.

We have calculated the score of all books in the prediction by either taking the OPW value when there is no iteration or by calculating the new score as mentioned before when there is an iteration.

The final step will then be to order these books by score relevantly. The prediction in this training set only yields 4 books, but the results could potentially be massive. The input user would probably dislike having to sift through thousands of relevant books. Limiting the prediction item count saves time and effort, where, in some platforms, the user can still extend this limitation (paging, infinity-scrolling, ... etc.). The limitation for results has been set to a fixed value of 10 for ease of use in the BookRec Project.

6.4 The algorithm:

This algorithm is divided into three parts:

1. Calculate the temperature of the matching users
2. Calculate the OPW of their books
3. Find the final Score

The flowchart for CB is as shown in *figure 6*

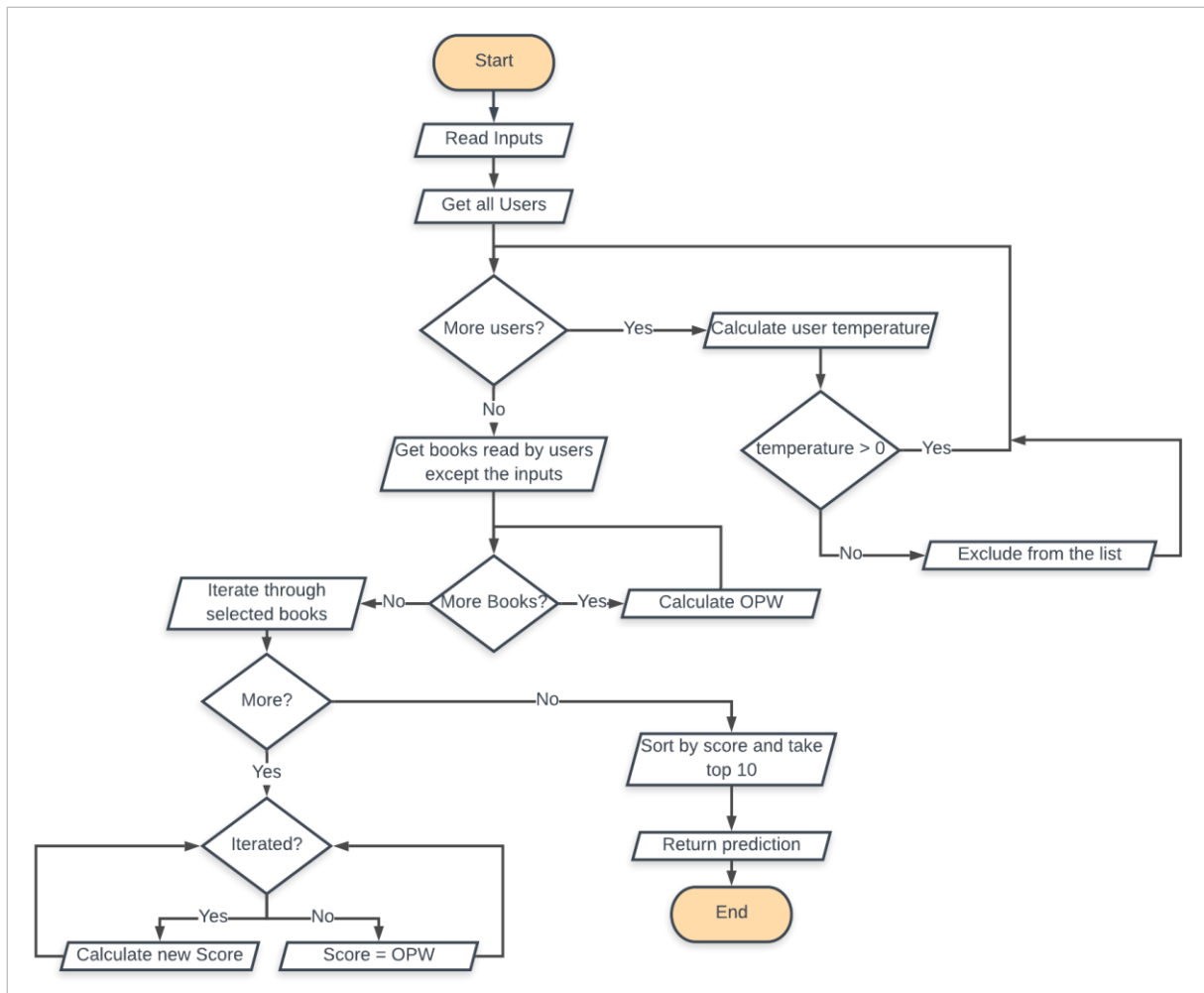


Figure 6

The algorithm receives the inputs and iterates through all users to find those who share inputs by calculating the temperature of the users and excluding those who have a temperature of 0

The following pseudo-code demonstrates the calculation of user temperature.

```

START
  INPUT books

  DEFINE users: database users, matchingUsers
  FOR every user in users
    DEFINE difference, Temperature
    FOR every book in user.Books
      CALCULATE difference:=> rating - user rating
    ENDFOR

    ASSIGN Temperature:=> 1- (difference / sum rating)
    IF Temperature > 0
      ADD User :=> matchingUsers
    ENDIF
  ENDFOR
END
  
```

BookRec defines a Collaborative Filtering helper which contains the necessary functions that this algorithm needs

```
var helper = new CollaborativeRecommenderHelper(inputs);
```

The initialization of this helper will calculate the total rating of the inputs which will be needed later on to calculate the temperature of each user

```
public CollaborativeRecommenderHelper(List<UserBook> inputs)
{
    EnsureArg.IsNotNull(inputs);

    this.CalculateTotalRating(inputs);
    this.SetInputs(inputs);
}
```

The user temperature UTM should be calculated for each user separately by finding the star differences between their books and the input books and that will be done using the following method in the helper class:

```
public double CalculateUTM(IGrouping<string, UserBook> userGroup)
=> (double)1 - this.Inputs.Select(input =>
{
    var book = userGroup.FirstOrDefault(x => x.BookId == input.Key);
    return book == null ? input.Value : input.Value - book.Rating;
}).Sum(value => (double)value / this.TotalRating);
```

The second part of this algorithm will be to calculate the OPW for each suggested book by the users who have temperature > 0, and which is different than the input books.

```
START
  INPUT books

  DEFINE users: users with temperature > 0
  FOR every user in users
    FOR every book in user.Books
      IF books do not include book
        CALCULATE OPW
      ENDIF
    ENDFOR
  ENDFOR
END
```

The helper contains the function to calculate the OPW for each book

```
public double CalculateOPW(double temperature, int rating)
=> temperature * rating / 5;
```

The final step of this algorithm is to calculate the score when there is an iteration.

The weighted output list will be grouped by books and the score will be calculated for each group:

```
outputWeights.GroupBy(x => x.Book.Id)
    .Select(group =>
    {
        var prediction = new PredictionModel()
        {
            Book = group.FirstOrDefault().Book
        };

        if (group.Count() == 1)
        {
            prediction.Score = group.FirstOrDefault().Score;
        }
        else
        {
            var maxWeight = group.Max(x => x.Score);
            var imv = (1 - maxWeight) / group.Count();
            prediction.Score = (double)maxWeight + group.Sum(x => imv * x.Temperature);
        }

        return prediction;
    }).OrderByDescending(x => x.Score).Take(10).ToList();
```

In the end, the list will be ordered by score relatively and limited to the top 10 books and delivered to the user as a prediction.

The implementation of Collaborative filtering can be found using the link:

<https://github.com/samysammour/BookRec/tree/master/src/Recommender/CollaborativeRecommender>

6.5 Advantages and disadvantages:

CF collaborate users with each other and build the prediction based on their collaboration.

CF advantages are:

1. Serendipity: this filtering is isolated from the domain knowledge. It does not know the interest of the user but still predicts based on the cooperation between users. It helps the user to discover new interests that he may like. The book prediction may contain books from new categories even if the user did not mark them as his interest. These new categories may introduce the user to a new interest that he may like to read.
2. User-to-user: This filtering increases the collaboration between the users and predicts based on their experiences. A widely watched movie could be more likely included in the prediction, but the same movie may be excluded from the prediction when it has a low rating. Users have control over the prediction and they together cooperate to help each other to find interesting items and gaining better experiences.

The disadvantages of CF are:

1. Complexity: It may consider one of the most relevant disadvantages. The complexity of CF is related to the number of the users and their previous items and will not just be increased due to the increase of the users but also due to the increase of there activities within time and could affect the performance. It may also predict a massive result that leads to uninteresting items. BookRec has more than 60 thousand books and it may contain thousands of users that read tens of books and each new entry will increase the complexity of this algorithm, although the number of items (books) still the same.
2. Data sparsity: CF consists basically on rating. Users do not always rate what they are using, and they maybe rate only the items that they do not like. This may lead to ambiguous data that may be ignored just because of the unclear rating system. A user will never be a match if he has never rated a book. Even, if he rated the shared books, his other books will be ignored when they are not rated. Books from a perfect match can be omitted due to lake of stars. That will force the filtering to search for the second-best which decreases the accuracy.
3. Eager requests: CF prediction is based on other user's experiences. This filtering will fail to predict when the user is the first one who is using the application or the first one within a certain interest. If the system does not have users, it will fail to predict although it may contain thousands of items. That will leave a bad experience and maybe cause the user to leave the system which is exactly the opposite purpose of all recommendation systems.

Both CBF and CF have their advantages and disadvantages and it may be a wonder, which to use. That will be discussed next.

7. Content-Based Filtering VS. Collaborative Filtering:

Content-based filtering and collaborative filtering use a different source of information, whereas CBF predicts based on user preferences of items, while CF predicts using user-to-user recommendation.

CBF needs to collect information about user preferences to be able to predict items, the more the collected information is the more accurate the prediction will be. On the other hand, CF suggests items from other users based on the shared items between users and their rating and it needs a large dataset of rated items to provide an accurate prediction.

Although of their disadvantages, both of them are still used widely in the market. System requirements help to decide which filtering to choose due to the difference in nature and the usage between both filterings.

Amazon, for example, is counting mainly on CBF and suggests items to buy next based on user's previous visited items or certain interests. While Spotify used CF to suggest music that the user may listen to next based on top-rated songs.

1. Usage:

CBF ensures that the predicted items are following the same preferences pattern of the user, and it can be very useful for systems which they need to predict items related to the user himself like stores, articles, books ... etc., Amazon store is a good example. CBF ignores (partially ignores) rating systems and feedback which may not be that important in some systems, and this point can be improved by including visiting or purchasing counters which Amazon did.

CF consists of rating and feedback, but unlike CBF, it does not ensure that the prediction follows the same preferences pattern. The recommended items may or may not be in the same category or even in the same language or country. CF is very useful for systems that are not necessary bounded to a geographical location or systems that related to time periods and need to keep the user up-to-date, and Spotify is a nice real example who is mainly using CF to suggest music and help to explore new genres and to keep up-to-date, and suggests the most heard music today when a song hits a trend and go viral.

2. Complexity and performance:

CBF is related to the training set size. Changes on items will change the complexity of the algorithm. This complexity is almost equal between requests and may sometimes slightly differ. When a system has 100 items, each prediction needs to check the whole training set to find matches and the time for each request will be related to 100. If there are no changes to the items or user preferences, the prediction will be the same for any given time.

CF works differently. It is related to the usage of the training set. The complexity will change when users are using the training set. The complexity of CF is not equal between requests even for the same items. When a request needs one second to find a prediction another may need 10 seconds as its related to the shared items between users. In young systems, CF may respond rapidly, but within time, the complexity can be obviously increased. The same input may have a different prediction at a different time according to the usage of the items.

Producing new items like writing a book or composing a song is normally slower than using it, which weigh the complexity of the CF above the other filtering (CBF). The number of music listened to by users, for example, is way beyond the number of songs on Spotify.

CF can still be shrink and grouped to certain factors to reduce the complexity like shrinking the matching users to only who live in Germany when suggesting items to a user located in Germany.

3. Limitation:

CBF is limited to the user's profile. User's profile contains all the information about the user like location, age and tags, and his previous items. The prediction will fail in case of inadequate information in the profile. In a real-world scenario, the user's profile will always have a minimum level of information, like age, gender, location ... etc. This information may be enough to be used to suggest items to the user. If a new user, for example, signed up into Amazon, he will get at least prediction related to the country he is logged in from and that can be calculated using his IP address or his internet provider. The prediction can always be improved when the user enriches his profile within time.

The limitation of CF is to users' interactions. The prediction may fail in eager requests, when the user is the first one to sign up into the website or the first with a certain interest, in other words, when there are inadequate interactions to provide a prediction. A user may not get a prediction at the beginning when he registers himself to the website, but he may within time receive some prediction when he shows his interest in some items. The accuracy of the prediction is related to these interactions.

The three previous comparison criteria show the differences between both filtering and which to choose in which system. But why not use both filterings together to gain more accuracy? And here where Hybrid Filtering, the third type of filtering, comes into action.

8. Hybrid Filtering (HF):

Hybrid algorithm as a definition is an algorithm that combines more than one algorithm which solve the same problems (two or more). It either switches between them depending on the data or combines the result to improve the performance and tuning the result.

Hybrid Filtering term is used usually when combining both CBF and CF together to win both advantages and reduce the disadvantages.

To understand HF more, let's have the following story:

A music company wanted to develop a website to help the people listening to the music online and they decided to create a recommendation system using collaborative filtering based on users' experiences and rating. They discovered after a while that most users who were facing an empty page with "no prediction can be found" leave the website immediately and never come back because the recommender failed to recommend music to them (Eager requests). So, they thought of a solution to predict items based on the country where users sign in using (CBF) as a backup when CF recommender fails to suggest items.

This scenario describes one of the advantages of HF. Collaborative filtering can sometimes recommend items that do not match the user's profile. Combining the results with CBF can eliminate or at least reduce them and shrink the result into only interesting items with a high rating.

HF can be implemented in many ways.

1. The first implementation is to switch between the algorithms. The same approach that the company in the previous story will use. When there is inadequate information for one filtering approach to predict, HF will automatically switch to the second one. (When there are no shared items between users, CBF will be used to provide a suggestion)
2. The second implementation is to combine the result from both filtering approaches which is used in BookRec and will be discussed next.

8.1 Item strength:

Both Content-Based and Collaborative filtering use different approaches to predict. The predicted items from both filterings may or may not match. But what if they match?

To answer this question, let's try to understand the challenge first. The following table, *table 5*, represents a prediction from both filterings for the same input user, Bob.

CBF		CF	
Id	Score	Id	Score
1	0.7	6	0.9
2	0.6	1	0.5
3	0.6	7	0.5
4	0.1	4	0.3

Table 5

The CBF in the previous training set predicted different Books from CF except for Book 1 and 4 which are suggested from both filtering, and that makes them more interesting for the user than the other books.

Item strength is a term used to define if an item predicted from all algorithm (strong) or only from one (weak). The strength of the item decides which priority does the item have.

Apparently, Book 1 and 4 are strong items because they are suggested from both CBF and CF while all others are weak items. But is that enough to decide the order of the items?

Counting on item strength alone does not necessarily decide the correct order. Book 4, for example, scored a low value in both filterings, but Book 6 scored a very high value only in CF, the question will be, does Book 4 have a higher priority than Book 6?

In some hybrid systems, recommenders are not handled equally. They may count on one approach more than the other, although they are using both filterings. That leads to the next step which is to calculate the weight of each filter type.

8.2 Filter Weight:

Simply, taking the average value of the scores in all of the predictions does not give a realistic solution. The average value of Book 1 is 0.6 which is less than the value suggested from CBF only, and the same for Book 4 which is 0.2. But, all other books (weak items) hold the old score. The average will ignore the item strength completely.

Therefore, we need first to find the impact of each filtering on the final prediction. Filter Weight is a value between 0 to 1 that represents the percentage that each type of filtering (two or more) will add to the final score. These weights may be equal or may differ among them, but the summation of all weights should equal to 1 to avoid the overflow or the gaps that may lead to loss of values.

BookRec deals with both filtering equally, so each filtering has a value of 0.5

8.3 Score:

HF calculates a new score of each item in the prediction list from both filtering approaches. This new score should be related to each filter weight that will shrink the impact of each of the final score.

Backing to the example, Book 1 appears in both filtering prediction and has the value of 0.7 and 0.5

The filter is considered in the example as mentioned (0.5), which means, each filtering has the exact half effect on the final score. Multiplying the old score with filter weight of each filtering will calculate the new score:

$$Score = (Score_m * Weight_m) + (Score_n * Weight_n)$$

m is the item in CBF

n is the item in CF

The new score of Book 1 will be then for Book 1: $(0.7*0.5) + (0.5*0.5) = 0.6$

The books which have been only predicted from one algorithm will lose at least the complete filter weight of the other filter type.

Repeating the same equivalent for all books will calculate the new score as shown in this table:

Book	CBF (0.5)		CF (0.5)		HF
	Old score	New score	Old score	New score	Score
1	0.7	0.35	0.5	0.25	0.6
2	0.6	0.3		x	0.3
3	0.6	0.3		x	0.3
4	0.1	0.05	0.3	0.15	0.2
6		x	0.9	0.45	0.45
7		x	0.5	0.25	0.25

Table 6

We notice from the values in *table 6* that book 1 is taking the lead. It scores acceptable values in both filtering and the final score is the highest value among the other. But Book 6 is still interesting based on its high score (almost a perfect match), although it is suggested from one filtering (CF). It makes it even more interesting than book 4 which is recommended by both filterings. The final order of the suggested book will be then: 1, 6, 2, 3, 7, 4

The difference between Book 1 and 4 is the original score in each filtering. Book 4, however, was not a very interesting suggestion in both of them.

By choosing filter weight as 0.5, the HF score will be similar to the average of the strong items, but it will at least cut the value of the weak items into half. When the filter weights are not equal, the percentage each filter will add will change and will in return change the final score, but it will still ensure that the impact of each filtering is included.

The score may also differ and may give more accurate values if we applied each filtering two times by splitting the inputs into two lists. Each half will be processed by the two filterings and the final list will return 4 different scores for each book.

The final prediction follows the same pattern as CBF and CF. It will be ordered by relevant and limited to a specific number of items, which also could be controlled by the user. BookRec sets the limitation to 10 items.

8.4 The algorithm:

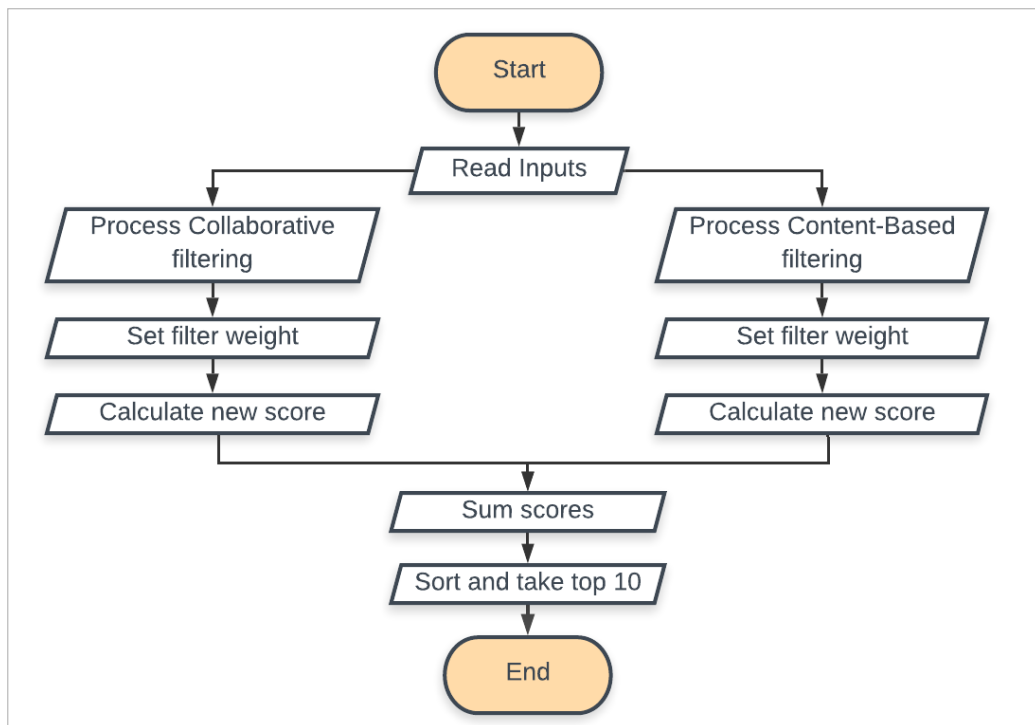


Figure 7

Hybrid algorithm is simple. It processes the input list using both approaches and waits of the result to combine it and prepare it for delivery.

BookRec introduces an option class that contains the filter weight of both algorithms and the necessary functions to calculate the new score

```
var options = new HybridRecommenderOptions();
```

This options class sets the filter weight for both algorithms to 0.5:

```
public HybridRecommenderOptions()
{
    this.CBFWeight = 0.5;
    this.CFWeight = 1 - this.CBFWeight;
}
```

HF algorithm starts processing the inputs using both filtering and waits for the prediction to be finished.

```
var cbfTask = this.contentBasedRecommender.GetPredicationsByBooksAsync(inputs);
var cfTask = this.collaborativeRecommender.GetPredicationsByBooksAsync(inputs);

await Task.WhenAll(cbfTask, cfTask).ConfigureAwait(false);

var cbfPrediction = cbfTask.Result;
var cfPrediction = cfTask.Result;
```

The next step is to calculate the new score of each book related to its filter weight and then combine the result of both filtering together:

```
var output = cbfPrediction.Select(prediction =>
{
    prediction.Score = options.CBFScore(prediction.Score);
    return prediction;
}).SafeConcat(cfPrediction.Select(prediction =>
{
    prediction.Score = options.CFScore(prediction.Score);
    return prediction;
}));
```

The new score is now prepared for each book and the algorithm will group similar items (books) together as a final step and sum the grouped scores. HF will then order them by score relevantly and take the top 10 items to return them as a prediction.

```
return output.GroupBy(x => x.Book.Id).Select(group => new PredictionModel()
{
    Book = group.FirstOrDefault().Book,
    Score = group.Sum(x => x.Score)
}).OrderByDescending(x => x.Score).Take(10).ToList();
```

This algorithm brings value to the output and improves the prediction, but it does not actually add any extra logic.

The implementation of Hybrid filtering can be found using the link:

<https://github.com/samysammour/BookRec/tree/master/src/Recommender/HybridRecommender>

8.5 Advantages and disadvantages:

Hybrid Filtering combines both filtering approaches, which will eliminate some predicted items that do not match the user's profile or at least tune the result for a better experience.

The main advantages are:

1. Including user profile: Collaborative filtering is very powerful and useful filtering, but it ignores user profile which leads, sometimes, to unwanted prediction. However, HF helps to solve the problem. It is not based on user profile like CBF, but it at least eliminates the items which differ completely from the user's profile and that will improve, in return, the output. Songs which suggested from both CBF and CF are highly recommended, and they interest the user because they are both suggested from other users who share the same interest and match the user's profile. With a wider list of prediction, the result can be quite trimmed to those who exactly fit the user.
2. Solving eager requests: As mentioned in CF, the filtering is suffering from eager requests. Simply, If there is no match, there is no prediction. HF guarantees that if one of the filterings failed to predict, the result will, at least, contain the prediction from the other approach (all others, if there is more than one approach) and the user will not be left empty-handed without a prediction.

This approach solves the music company problem mentioned in the introduction of HF.

3. Solve over-specialization: CBF fails, sometimes, to predict items when there are no similarities. Similar to point 2, HF will predict all the items from CF when CBF fails.

This filtering is also suffering from some disadvantages, the following are the main three:

1. Increase complexity: HF is combining multiple algorithms together and wait until the finishing of all of them to start calculating the new score. Which means, the minimum complexity of HF is the maximum complexity of all filtering. If CF needs 1 second to predict, for example, and CBF needs only 0.5 seconds, HF will wait at least 1.5 seconds (or 1 second in case of parallel programming) and then start processing the prediction. It also increases the complexity of unit testing. With good implementation using parallel programming, HF performance can be improved, and its complexity can be reduced.
2. Inadequate prediction: The value of HF appears when there are enough items predicted by the filtering algorithms. The more items in the predictions the better the tuning. HF counts mostly on the number of strong items, but when both filterings fail to predict or do not predict enough items, HF will not add any value to the output and will itself fail. Instead, HF will just increase the complexity in this case. If each one of CBF and CF predicted only 3 different Books, HF will do nothing rather than resorting the items. But what if the items are already sorted
3. Unbalanced Filter Weight: HF calculates the new score based on each filter weight. The weak items with low score predicted by the high weighted filtering in unbalanced filter weight system will have an advantage over perfect matches predicted from the low weighted one. If a system mainly counts on CF (filter weight 0.9) more than CBF (filter weight 0.1), a perfect match from CBF will have a maximum value of 0.1 while a low scored item (e.g. 0.12) predicted from CF will have a higher HF score (0.108) and will appear in the list before the perfect match in CBF, which almost means ignoring all items from the low filter weighted algorithm in case of no strong items even if they are perfect matches.

HF combines more than one filter together and requires for all of them to be implemented which consume times and effort, but it still widely used in real-life scenarios. Netflix is using HF to predict items where it is still convenient for them to use and its value is still worthy of the implementation.

These are the three main algorithms used in recommendation systems in the market. There are still other types like Risk-aware recommendation system or mobile recommendation system which can be used separately or mixed with other types.

9. Points of failure:

There are many ways to implement Recommendation Systems and developers need sometimes to develop their own method based on system specifications. This study shows a way to implement a recommendation system. Yet, there are still points of failure where the study has failed. The following points are maybe the main failures:

1. Performance:

The performance of the algorithms is the main issue in this study which is a requirement nowadays. The math, for instance, can be improved to minimize the complexity of the filtering which, in return, will improve the performance itself.

In this small application with a limited database and few transactions, the performance was acceptable. But in huge systems, performance could be critical and the whole system may collapse. Imagine visiting Netflix and waiting for 2 minutes to get movies recommendations. This could be a reason to leave the application.

The performance can also be improved with machine learning approaches, where the system observes and learns the prediction for later retrieval when the same input is being requested (predict the prediction).

2. Uninteresting prediction in CF:

CF is not restricted to certain groups, it rather tries to match all the users in the database (training set) ignoring the similarities between their profiles. Why should CF try to find matching users who do not share the same user's profile?

If a father watched an animation movie with his children, it does not necessarily mean that all the movies he watched could be a suggestion for other kids.

Grouping the users by matching their profiles first may improve the performance because it minimizes the number of items in the training set.

3. Ignoring static information in CBF:

The limitation of Content-Based filtering to only the activities in the user profile in this study is still not optimal and ignores all other static information in the profile which may lead to inaccurate prediction. It fits, however, the case study of suggesting books based on previous reading, but it may not (probably will not) fit other systems. It is important to predict interesting products to the user to buy next, but it will still not be helpful if the products are not available in his country. This issue could be critical sometimes when predicting a horror movie or a movie with adult content to a minor without paying attention to the age restriction.

4. Ignoring items rating in CBF:

Most modern applications have a rating system. This rating can be used to determine the differences between items and to define what a good rating or a bad rating item may be. If the user is interested in buying construction tools, he would like to see not just similar but the best items that have good price and good quality (good review), which the math in this study does not take it into consideration.

This problem can be solved by trying to sort the items first by rating or activities before comparing them.

5. Score of an iteration in CF:

CF calculates a new score for the items when there is an iteration. The score of each item in the prediction will be increased when there is an iteration. This new value is calculated for each iteration separately which is not convenient when there is more than one iteration. For example, an item x iterates between 2 users and the highest score is 0.5, that means, each user can add up to 0.25 to the final score. But if an item y iterates between 10 users and has the same highest score as item x, each user will add only 0.05 to the new score of item y, which is less than the value of item x that iterates only 2 times. In one prediction, the maximum value that can be added to each item in case of an iteration should be limited to the maximum number of iteration among all items, which is, in this case, 0.05, regardless of the iteration of each item itself. That will make the maximum value that each user can add to item x is only 0.05

Those are the main issues that this study is suffering from. The performance may be seen as the most important point and can be seen as the work to do next.

10. Future perspective:

Users receive a superabundance of information from social media, internet and tv every day which they cannot handle. They are expecting help to process this information in almost every aspect of their life. [15]

[image reference: <https://medium.com/futuristone/the-future-directions-of-recommender-systems-72beae7c2dd2>]



Figure 7

The type of information is being collected nowadays has been widened to include more than just solid information about the user's activities. Rather, the user's profile has been expanded to collect information about user's life, emotions, work situation, personal likes and dislikes ... etc. to predict, what I like to call it, a *warm prediction*.

The interest of users is connected to their emotions and life situation. Users will not have probably the same interest when they come back home from a tiring day at work as when they are on a vacation.

The time factor to which an action occurs plays also a big role in the prediction. Rating the same item by the same user may differ due to the day of the week or the sequence of previously purchased items.

Personal Assistant Agents (PAAs) like SIRI or Cortana and wearable devices like smartwatches and smart glasses can be used to help to retrieve information about users and their daily activities, emotions and energy.

Emotion-Based Recommendation Systems are using this information involving user's interests from a wide range of domains like sports, politics, work ... etc. which are being collected, stored and filtered on the time factor [15]

The challenge with these recommendation systems is choosing the correct mood at the correct time from this huge amount of information.

Emotional information may not be connected directly to the data, but it adds an important value to the prediction. Those warm predictions can be seen as the future perspective of recommendation systems.

Warm predictions should not always be limited to a specific subject like products or movies. They can also expand the domain to predict not only what a user is searching for, but what are the searching terms that the user may be interested in searching at this time.

We may see in the future small wearable devices like HoloLens that suggest not just the movie that I would like to watch, rather if I am ever interested in watching a movie right now. I may be very tired to concentrate for two hours watching a movie, instead, I may like to hear some relaxing music. [16]

User privacy will pop up as an opposite opinion of warm prediction. In the end, the user can choose what to share.

11. Conclusion:

This study explained recommendation systems in general and focused on the three popular approaches used: Content-Based filtering, Collaborative filtering and Hybrid filtering. It mentioned the differences between the three approaches and pointed out a way to implement these algorithms to develop a demo book recommendation system.

This project has failed in some points as mentioned above. As recommendation systems are being widely used and developed these days, these issues make the prediction not acceptable and open a new window of improvement.

This study pointed out the future perspectives and defined the direction in which the development of the recommendation systems should follow.

12. References:

1. Peter Brusilovsky, Alfred Kobsa, Wolfgang Nejdl (2007) *The Adaptive Web: Methods and Strategies of Web Personalization*, 2007th Edition edn., Germany: Springer Berlin Heidelberg.
2. Francesco Ricci, Lior Rokach, Bracha Shapira, Paul Kantor (2011) *Recommender Systems Handbook*, Springer US.
3. *Using Content-Based Filtering for Recommendation* (2011) Robin van Meteren, Maarten van Someren, : NetlinQ Group, Gerard Brandtstraat 26-28, 1054 JK, Amsterdam, The Netherlands, robin@netlinq.nl 2 University of Amsterdam, Roeterstraat 18, The Netherlands, marten@swi.psy.uva.nl
4. Martin P. Robillard, Walid Maalej, Robert J Walker, Thomas Zimmermann (2014) *Recommendation Systems in Software Engineering*, 1 edn., Germany: Springer-Verlag Berlin Heidelberg.
5. Parul Aggarwal, Vishal Tomar, Aditya Kathuria (2017) *Comparing Content Based and Collaborative Filtering in Recommender Systems*, Volume-3 edn. *International Journal of New Technology and Research (IJNTR)*.
6. Bee-Chung Chen, Deepak K. Agarwal (2016) *Statistical Methods for Recommender Systems*, UK: Cambridge University Press.
7. Luis M. de Campos, Juan M. Fernández-Luna *, Juan F. Huete, Miguel A. Rueda-Morales (2010) 'Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks', *International Journal of Approximate Reasoning*, 51(), pp. 785.
8. Guy Shani, Asela Gunawardana (2010) 'Evaluating Recommendation Systems', in Guy Shani, Asela Gunawardana (ed.) *Recommender System Handbook*. USA: Microsoft Research Redmond USA, pp. 257.
9. Richard Vézina, Dorin Militaru (2003) *COLLABORATIVE FILTERING: THEORETICAL POSITIONS AND A RESEARCH AGENDA IN MARKETING*, Québec (Québec) Canada G1K 7P4: Faculté des sciences de l'administration Université Laval.
10. Pazzani, M.J. *Artificial Intelligence Review* (1999) 13: 393.
<https://doi.org/10.1023/A:1006544522159>
11. Geetha G., Safa M., Fancy C., Saranya D. (2018) 'A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System', *Journal of Physics*, 1000(), pp.
12. Melville P., Sindhwani V. (2011) *Recommender Systems*. In: Sammut C., Webb G.I. (eds) *Encyclopedia of Machine Learning*. Springer, Boston, MA
13. Klamka R., Cuong P.M., Cao Y. (2009) *You Never Walk Alone: Recommending Academic Events Based on Social Network Analysis*. In: Zhou J. (eds) *Complex Sciences. Complex 2009. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol 4. Springer, Berlin, Heidelberg
14. Nitin Pradeep Kumar, Zhenzhen Fan (2015) *Hybrid User-Item Based Collaborative Filtering*, University of Singapore: Institute of Systems Science, National University of Singapore 25 Heng Mui Keng Terrace, Singapore- 119615.
15. Costa H., Macedo L. (2013) *Emotion-Based Recommender System for Overcoming the Problem of Information Overload*. In: Corchado J.M. et al. (eds) *Highlights on Practical Applications of Agents and Multi-Agent Systems. PAAMS 2013. Communications in Computer and Information Science*, vol 365. Springer, Berlin, Heidelberg

16. Ferwerda, Bruce and Markus Schedl. "Enhancing Music Recommender Systems with Personality Information and Emotional States: A Proposal." UMAP Workshops (2014).
17. Xiaoyuan Su and Taghi M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, vol. 2009, Article ID 421425, 19 pages, 2009. <https://doi.org/10.1155/2009/421425>.
18. Marcus Stolzenberger (2009) *Empfehlungssysteme: Transparente Visualisierung im mobilen Umfeld*, Germany: Diplomica Verlag GmbH.
19. Charu C. Aggarwal (2016) *Recommender Systems: The Textbook*, 2016 edition edn.: Springer.
20. Amancio Bouza (2012) *Hypothesis-Based Collaborative Filtering*: Lulu.com.
21. Mohammad Amin Rashidifar (2015) *HYBRID MOVIE RECOMMENDERS BASED ON NEURAL NETWORKS AND DECISION TREES*: Anchor Academic Publishing (aap_verlag).
22. Dhanda, Mahak & Verma, Vijay. (2016). Recommender System for Academic Literature with Incremental Dataset. *Procedia Computer Science*. 89. 483-491. 10.1016/j.procs.2016.06.109.
23. Prof. Dr. Andreas Nürnberger, Prof. Dr. Klaus Turowski, Prof. Dr. Alesia Zuccala (2015) *Towards Effective Research-Paper Recommender Systems and User Modeling based on Mind Maps*, Angenommen durch die Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg.
24. Joonseok Lee, Kisung Lee, Jennifer G. Kim, Sookyung Kim (n.d.) *Personalized Academic Paper Recommendation System*
25. Kokate, Shrikant. (2018). Hybrid Content-Based Filtering Recommendation Algorithm on Hadoop.