

project report

COMPARING THE COLLABORATIVE FILTERING ALGORITHM WITH naïve BAYES ON THE FILM recommendation SYSTEM

liem, daniel aditya santoso

18.k1.0036

Faculty of Computer Science

Soegijapranata Catholic University

2020

APPROVAL AND RATIFICATION PAGE

COMPARING THE COLLABORATIVE FILTERING ALGORITHM WITH NAÏVE BAYES ON THE FILM RECOMMENDATION SYSTEM

by

Liem, Daniel Aditya Santoso – 18.K1.0036

This project report has been approved and ratified

By the Faculty of Computer Science on January, DD. YYYY

With approval,

|  |  |
| --- | --- |
|  | Supervisor  Hironimus Leong, S.Kom, M.Kom  NPP: 05812007273 |
| Examiners,  1.)  Yonathan Purbo Santosa S.Kom., M.Sc  NPP :  2.)  Yulianto Tejo Putranto S.T., M.T.  NPP :  3.)  Y.b. Dwi Setianto S.T., M.Cs.  NPP :  4.)  Rosita Herawati S.T., M.I.T.  NPP : |  |
|  | Dean of Faculty of Computer Science  Robertus Setiawan AJi Nugroho, Ph.D  NPP: 058.1.2004.264 |

DECLARATION OF AUTHORSHIP

I, the undersigned:

Name : LIEM, DANIEL ADITYA SANTOSO

ID : 18.K1.0036

declare that this work, titled “COMPARING THE COLLABORATIVE FILTERING ALGORITHM WITH NAÏVE BAYES ON THE FILM RECOMMENDATION SYSTEM”, and the work presented in it is my own. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at Soegijapranata Catholic University
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
3. Where I have consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the work of others, the source is always given.
5. Except for such quotations, this work is entirely my own work.
6. I have acknowledged all main sources of help.
7. Where the work is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Semarang, MONTH, DD, YYYY



LIEM, DANIEL ADITYA SANTOSO

18.K1.036

ACKNOWLEDGMENT

First, I would like to say a big thank you to all the lecturers in the computer science faculty at Soegijapranata Unika Semarang who have accompanied the entire learning process so that I can be at this point. Especially to Mr. Hironimus Leong who has been a supervisor for the past few months who continues to motivate and provide.

I also thank my family who always support me in completing this thesis. And also my friends who always accompany me and always give me motivation.

And lastly, I would like to thank myself for always struggling all this time in doing this thesis and always finding the motivation to always do this thesis on time.

ABSTRACT

The many movies that are circulating and the many platforms that provide movie streaming platforms raise a question, namely what algorithm is the most suitable for use in providing movie recommendations. Of course, each of these streaming platforms uses different algorithms and factors.

In this study the author tries to compare two algorithms in providing movie recommendations based on the rating factor. The algorithm used is Collaborative Filtering with Cosine Similarity and also nave Bayes. Both authors tested using a dataset from movieLens.org as much as 10,000 data.

And in the results, Collaborative Filtering got better results through MSE and RMSE testing than nave Bayes. But the prediction score of each movie in each algorithm has a similar and the same score because it only uses the rating factor.

Keyword: naïve bayes, collaborative filtering, cosine similarity

TABLE OF CONTENTS

COVER i

[APPROVAL AND RATIFICATION PAGE (Heading plain) ii](#_Toc92713390)

[DECLARATION OF AUTHORSHIP iii](#_Toc92713391)

[ACKNOWLEDGMENT iv](#_Toc92713392)

[ABSTRACT (Abstract Title) v](#_Toc92713393)

[TABLE OF CONTENTS vi](#_Toc92713394)

[LIST OF FIGURE viii](#_Toc92713395)

[LIST OF TABLE 1](#_Toc92713396)

[CHAPTER 1 INTRODUCTION 2](#_Toc92713397)

[1.1 Background 2](#_Toc92713398)

[1.2 Problem Formulation 3](#_Toc92713399)

[1.3 Scope 3](#_Toc92713400)

[1.4 Objective 3](#_Toc92713401)

[CHAPTER 2 LITERATURE STUDY 4](#_Toc92713402)

[CHAPTER 3 RESEARCH METHODOLOGY 8](#_Toc92713403)

[3. 1 Data Collection 8](#_Toc92713404)

[CHAPTER 4 ANALYSIS AND DESIGN 10](#_Toc92713405)

[4.1 Pre Processing Dataset 10](#_Toc92713406)

[4.2 Collaborative Filtering 14](#_Toc92713407)

[4.3 Naïve Bayes 18](#_Toc92713408)

[4.4 Perbandingan Collaborative Filtering dengan Naïve Bayes 21](#_Toc92713409)

[CHAPTER 5 IMPLEMENTATION AND RESULTS 22](#_Toc92713410)

[5.1 Collaborative Filtering 22](#_Toc92713411)

[5.2 Naïve Bayes 26](#_Toc92713412)

[5.3 Result 30](#_Toc92713413)

[5.4 Analysis 31](#_Toc92713414)

[CHAPTER 6 CONCLUSION a](#_Toc92713415)

[REFERENCES b](#_Toc92713416)

[APPENDIX c](#_Toc92713417)

LIST OF FIGURE

[**Gambar 5.1.1 Cosine Similarity Code** 16](#_Toc85461505)

APPROVAL AND RATIFICATION PAGE (Heading plain) ii

DECLARATION OF AUTHORSHIP iii

ACKNOWLEDGMENT iv

TABLE OF CONTENTS vi

LIST OF FIGURE viii

LIST OF TABLE 1

CHAPTER 1 INTRODUCTION 2

1.1 Background 2

1.2 Problem Formulation 3

1.3 Scope 3

1.4 Objective 3

CHAPTER 2 LITERATURE STUDY 4

CHAPTER 3 RESEARCH METHODOLOGY 8

3. 1 Data Collection 8

CHAPTER 4 ANALYSIS AND DESIGN 10

4.1 Pre Processing Dataset 10

4.2 Collaborative Filtering 14

4.3 Naïve Bayes 18

4.4 Perbandingan Collaborative Filtering dengan Naïve Bayes 21

CHAPTER 5 IMPLEMENTATION AND RESULTS 22

5.1 Collaborative Filtering 22

5.2 Naïve Bayes 26

5.3 Result 30

5.4 Analysis 31

CHAPTER 6 CONCLUSION a

REFERENCES b

APPENDIX c

LIST OF TABLE

[**Tabel 4.3.1 Rating Dataset** 11](#_Toc85461980)

[**Tabel 4.3.2 Movie Dataset** 12](#_Toc85462066)

[**Tabel 4.3.3 RatingMovie Dataset** 12](#_Toc85462127)

[**Tabel 4.3.4 Preprocessing Table Collaborative Filtering** 13](#_Toc85462235)

# CHAPTER 1 INTRODUCTION

## Background

Movies have become our daily needs in terms of entertainment. It's no secret that several well-known platforms such as Netflix, Youtube, Disney+ Hotstar, and others have more or fewer millions and even tens of millions of films on each of these platforms. Therefore, the recommendation engine is at the heart of various movie provider platforms.

The number of these films is what causes a new problem, namely what film to watch next. Almost all platforms must collect data on movies watched by their users, the problem is if a platform still doesn't have data from that user or a new user. Then what are the movie recommendations that should be presented to the new user if a platform still doesn't have viewing data from that user or still has very little data from that user. Of course, users from each platform want to get movie recommendations that they think are most suitable for them, if not, then it's likely that these users will not use the platform anymore.

However, this problem has been solved with the recommendation engine. There are many algorithms used to create recommendation engines. Each algorithm has its advantages and disadvantages. Some algorithms recommend a film based on data between users, but there is also an algorithm that recommends also based on the relationship from one film to another. Then there is also a combined algorithm of the two, namely user-item, the user-item relation produces more maximum recommendations than relying on only one of the components above.

But of course, there are always shortcomings from the algorithms listed above, such as users who are new and don't have any viewing records. There is also an accuracy that is not optimal due to the low rating of each film. And in this study, the algorithm that I will use is item-based which uses data from ratings per film.

In this study, I tried to make 2 recommendation engines using 2 different algorithms, and then I will measure them concerning MSE and RMSE. And I will try this research with the cosine similarity approach to find the proximity between 1 object to another. From this research, it is expected to see which of the two algorithms has the least error.

## Problem Formulation

1. Which of the Collaborative Filtering and Nave Bayes has the lowest MSE and RMSE values ?
2. What factors can be used as a reference to compare the two algorithms ?

## Scope

The data used is the data that I downloaded from movielends.org. All the data that I use already have all the data records so there are no users who don't have data records at all.

## Objective

Comparing the algorithm between Collaborative Filtering and Naïve Bayes then calculating the RMSE value, and predicting the films that will be recommended to users while maintaining the genre.

Comparing two different algorithms against the film recommendation engine and finding out which of the two algorithms is the most effective, based on the smallest MSE and RMSE values. In collaborative filtering, we use the approach through Cosine Similarity.

# CHAPTER 2 LITERATURE STUDY

Khusna et al. [1] ⁠ perform an analysis on the product recommendation system on the Gadget Shield by using the collaborative filtering method based on user-to-user references. The data was obtained by the author by conducting several direct interviews with 17 respondents and managed to get 40 mobile phone product data and also 15 user data. Where the data in the training then make predictions to each user to test it. Then from the average data that has been obtained, it was tested using the RMSE test method and obtained an accuracy result of 0.496 or 90.08%. So that this research can be said to be quite good, but there are still some shortcomings that can be improved, such as the use of an algorithm that is more accurate than the Euclidean distance.

Wahyudi, Indah Survyana [2]⁠ trying to solve problems in the film industry, namely the film recommendation engine that has different search results or recommendation results based on the data or queries used. In this study, the author tries to overcome a common problem in the use of collaborative filtering algorithms for film recommendations, namely sparsity by using the best model. And get a validation result of 0.96 and a test result of 0.94 on a data amounting to 100 thousand, and a validation result of 0.86 and a test result of 0.96 on a data of 1 million. filtering by 62% and with the genre similarity method with 75% results.

N.Safir, S.Ziad [3] This research is intended to find out how the performance of an algorithm with data continues to grow. The algorithm used in this research is item-based collaborative filtering and also FunkSVD, Collaborative filtering itself here is still divided into 3 more, namely cosine-based similarity, correlation-based similarity, and also computing prediction. Different neighborhoods are then tested using MSE and RMSE. The results in testing datasets of 100 thousand and 1 million there is no significant difference, but when tested with continuously growing data FunkSVD gets a larger percentage compared to item-based collaborative filtering. From this research, we can see that in data that continues to grow, FunkSVD can show better performance than collaborative filtering.

N.Raval dan V.Khedar [4]⁠ In this study, researchers tested various recommendation algorithms with the aim of getting the algorithm that had the best performance. Various kinds of algorithms were tested such as collaborative filtering, ALS (Alternating Least Squares), rapid miner, and there is also a method of grouping data with a neural network. In this test, the researcher writes that the results of various tests still have their respective strengths and weaknesses. To be able to develop a better recommendation algorithm, the researcher gives suggestions to use the libraries that already exist in python to look for hidden factors that can be the key to the recommendation.

K.Goldberg et al. [5]⁠ researching to test a new algorithm, namely, eigenstate which is accurate and more efficient to users in 1 time online. Of course, the algorithm used is the Eigentaste algorithm, Eigentaste itself is collaborative filtering that uses universal queries to get data ratings based on rating data. Then this test is tested using the normalized mean absolute error with several other factors such as user data that is offline and also online by 20%. Then also in testing the Uniform and noise distribution model, the NMAE result is better, which is 33%.

A.Varma [6] In this study the researcher tries to understand various recommendation algorithms and also to compare their performance with the dataset from MovieLens. The algorithm used is collaborative filtering and also content-based recommendation. After passing several tests using a number of different datasets, several problems were obtained, such as the larger the dataset, the larger the RMSE test results, the smaller the level of accuracy. Therefore, the researchers hope to be able to add content-based collaborative filtering to add several criteria to be able to categorize several films, it can also be based on recommendations based on actors, directors, and writers.

L.Shuxian, F .Sen [7] In this study, the researcher tried to personalize the users to find similarities between one user to another. By using the nave Bayes method on the MovieLens dataset, researchers got better performance than item-based collaborative filtering or user-based collaborative filtering. For nave Bayes itself, the precision value is 0.642, and for collaborative filtering item-based and user-based, it is 0.1788 and 0.1984. Therefore, the use of nave Bayes in the recommendation engine can have a significant impact.

Sahu, Satya Prakash et al. [8] researching comparing several algorithms aimed at finding the algorithm that has the best performance by conducting a comprehensive comparative analysis. And the results of the analysis are measured by one factor, namely the cold start problem where a user's data does not have any watch records or references. The methods used are content-based filtering, collaborative-based filtering, hybrid content collaborative-based filtering, k-mean clustering, and also nave Bayes. This test is tested with several dataset sizes such as 10 thousand, 50 thousand, and also 100 thousand datasets. In all these tests, the one who got the highest score was nave Bayes while the one who got the lowest score was content-based collaborative filtering. So that in handling the cold start problem with several sizes of the nave Bayes dataset, it is the one who has the highest predictive value among the other algorithms tested in this study.

Anchal Dubey dan Raju Ranjan [9] conduct research in solving problems such as cold start problems, sparsity, and also scalability which is usually found in content-based collaborative filtering and hybrid collaborative filtering. In solving this problem, the researcher tries to combine two algorithms, namely nave Bayes with collaborative filtering by predicting the rating and preference of each user. With several technical stacks such as XML, JAVA, Python, Android Studio, and also firebase. With all APIs built using nodeJs and also with database i.e. MongoDB. Android studio here plays a role in social media to bridge some users in sending some of the films they like and see the reaction of the user who receives the message whether he gives a negative or positive response. Can also be developed with a group between users to share. By conducting this research, the researcher got several reviews from several users for a film. With this review, it is hoped that in the future it can help other researchers in developing their recommendation engine.

Poonam Sharma dan Lokesh Yadav [10] conducted research using content-based collaborative filtering and also collaborative filtering in a film recommendation engine with the aim of accelerating the accuracy of recommendations based on ratings from users in each film. The results of this study are authentic data collection to be able to provide recommendations that are more accurate based on several factors such as likes, ratings, comments, password modification. With methods such as clustering, similarity, and classification that can reduce the MAE value which will increase the accuracy and precision of the recommendation engine. In the future, researchers can hope to use a hybrid recommender using clustering and similarity to get better performance.

The difference between my project and the projects from the journal above is the use of methods and also the use of analytical research. I will try to compare item-based collaborative filtering algorithms with nave Bayes in a movie recommendation engine. The dataset that I took myself was taken from MovieLen, which amounted to approximately 10,000 user history data who have rated a film. Then to compare the performance between the two algorithms I will calculate the MSE and also the RMSE which I will calculate with the two algorithms. In the journals above, they usually try to combine collaborative filtering with nave Bayes in the recommendation engine in order to get maximum results. In some journals there are also those who try to compare collaborative filtering with nave Bayes by using genre as its classification.

# CHAPTER 3 RESEARCH METHODOLOGY

## 3. 1 Data Collection

The dataset used in this project is a downloadable dataset from MovieLens.org itself but I got this dataset from Kaggle. This dataset itself has 2 sheets, namely master movies and master ratings. In the master movies, there are several variables such as movieId, title, and genre. Then the master rating itself has userId, movieId, rating, and timestamp variables. With a rating value range from 0 to 5.

From the data above we can take several factors that can support this project. We can do collaborative filtering as well as nave Bayes by using ratings from item by item, then we can also use user's watch data to find similarities between one user and another.

**3.2 Algorithms**

The algorithm that will be used in this research is item-based collaborative filtering and also naive Bayes. Both are used because both algorithms are quite widely used in making recommendation engines. Then the research at this time will be the main factor is the rating. Here we can predict a film that we will recommend to users by calculating the similarity of one item to another based on its rating using the naive Bayes method, and can also recommend films using the item-based collaborative filtering method.

**3.3 Design**

The data that will be processed is data that comes from the master rating, with the movieId, we simply need to manage from the movieId first, then when we display the recommendation results we will take data from the master movie CSV which contains information on the title of the film and also the genre. The amount of the data will be calculated by comparing the amount of all data, then from the accumulated data, we will calculate the percentage as a benchmark for similarity. For collaborative filtering, we will use an algorithmic approach with the help of Cosine Similarity to find the closest distance. Then for the nave bayes itself, we will try to calculate the distance based on the percentage of each film.

**3.4** **Coding**

Here the programming language that I will use is python 3.8 because python can process large amounts of data. In python, some libraries are quite adequate in the process of working on this research, but in the algorithm that I will be working on I will not use any libraries. The library that I will use is like pandas to read CSV files, then there is NumPy to perform the calculation process and also to change some forms of matrices, then I will use matplotlib to display some graphs as a form of visualization, and other libraries.

**3.5 Analysis**

This project aims to measure the accuracy of the two algorithms, namely item-based collaborative filtering with nave Bayes in making a film recommendation engine. With the dataset obtained from MovieLens, there are 10,000 data ratings from users and about 149,000 movie title data. We will test the level of accuracy and efficiency. The results of this analysis can be seen using the MSE and RMSE methods as a benchmark for the comparison of these two algorithms. The one who gets the smallest value from the test results will have the best performance.

# CHAPTER 4 ANALYSIS AND DESIGN

## 4.1 Pre Processing Dataset

In this study, the author uses two algorithms, namely collaborative filtering item-based using cosine similarity and also nave Bayes in comparing the two algorithms in the film recommendation engine. Where the dataset used is the same for both algorithms, namely the dataset that the author took from MovieLens with references from Kaggle. In this dataset, there are approximately 105,000 data provided but what the author will use is 10,000 datasets in this study. In the dataset, there are several data sheets that have been provided by MovieLens, but the author only took 2 sheets from several sheets that have been provided, namely the movie sheet and also the rating sheet.

Table 4.1.1 Rating Dataset

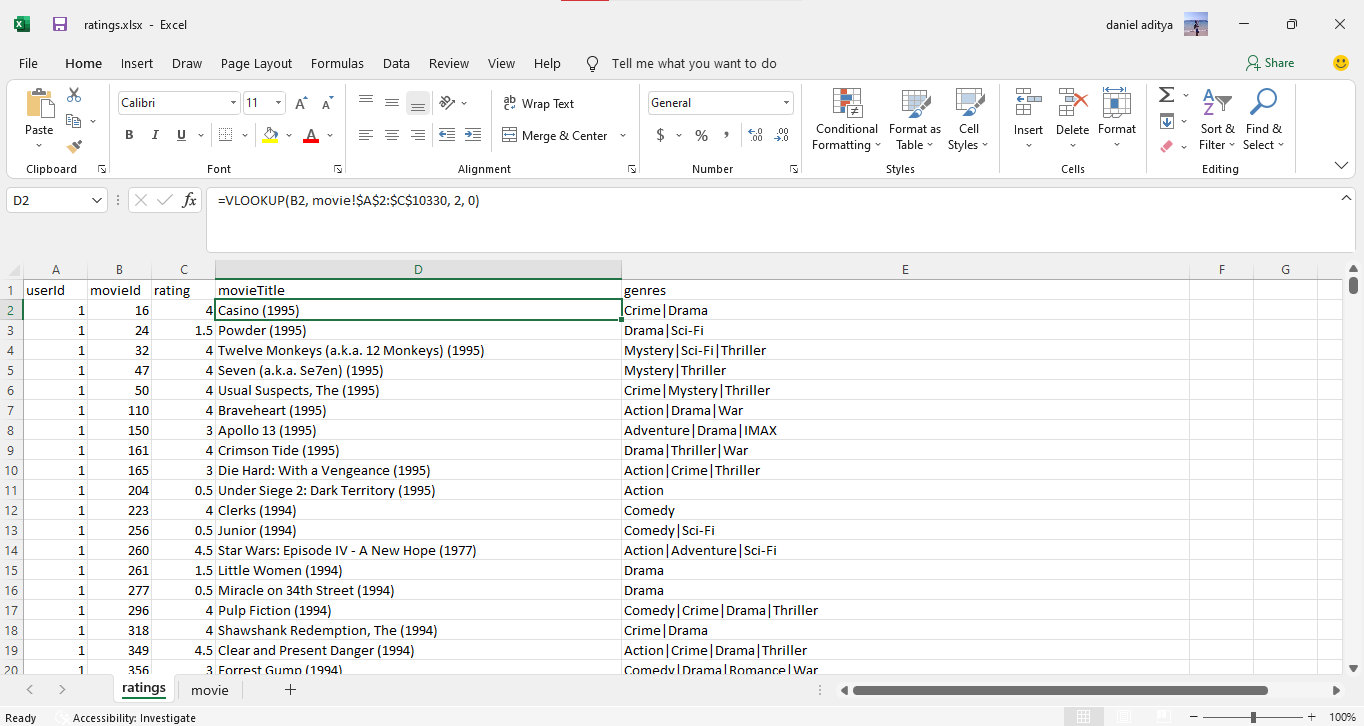
|  |  |  |  |
| --- | --- | --- | --- |
| userId | movieId | Rating | timestamp |
| 1 | 3 | 1.4 | 1.22E+09 |
| 1 | 9 | 3.5 | 1.22E+09 |
| 2 | 3 | 4.6 | 1.22E+09 |
| 3 | 7 | 4.3 | 1.22E+09 |
| 3 | 3 | 2.7 | 1.22E+09 |
| 4 | 10 | 4.8 | 1.22E+09 |
| 4 | 7 | 2.5 | 1.22E+09 |
| 4 | 10 | 3.6 | 1.22E+09 |
| 5 | 3 | 3.4 | 1.22E+09 |
| 5 | 6 | 2.7 | 1.22E+09 |

Table 4.1.2 Movie Dataset

|  |  |  |
| --- | --- | --- |
| movieId | title | Genres |
| 1 | Toy Story (1995) | Adventure|Animation|Children|Comedy|Fantasy |
| 2 | Jumanji (1995) | Adventure|Children|Fantasy |
| 3 | Grumpier Old Men (1995) | Comedy|Romance |
| 4 | Waiting to Exhale (1995) | Comedy|Drama|Romance |
| 5 | Father of the Bride Part II (1995) | Comedy |
| 6 | Heat (1995) | Action|Crime|Thriller |
| 7 | Sabrina (1995) | Comedy|Romance |
| 8 | Tom and Huck (1995) | Adventure|Children |
| 9 | Sudden Death (1995) | Action |
| 10 | GoldenEye (1995) | Action|Adventure|Thriller |

It can be seen in the rating dataset table (4.1.1) that there are various columns such as userId, movieId which is a foreign key of movieId in the movie dataset table (4.1.2), rating, and also timestamp. Then in the movie dataset table (4.1.2), there is a movieId which is the primary key in the movie dataset table, a title that contains the title of the movie, and also genres which contains the genre category of the movie in the movieId. In preprocessing this data, the author combines the two sheets into one with Microsoft Excel tools using the vLookUp feature by combining the two sheets into one first. Then the author omitted the timestamp column in the rating dataset because it was deemed not to be used. Then the author uses the vlookup function so

4.1.3 Function vLookUp Microsoft Excel



where B2 is column B in the second row which contains movieId 16. Then the author takes the reference data that we will vlookup from the movie which can be seen by the author using the AC column in the vlookup in the movie sheet where column A is movieId, B is movieTitle and C is genres his. To get the movieTitle, the author takes the second value as can be seen at the end of the formula, which is 2 and the number 0 at the end of the formula is the comparison we use to find the same data between sheets where the data this time is movieId. So that the author gets the final data results like

4.1.4 Table ratingMovie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| userId | movieId | rating | movieTitle | genres |
| 1 | 3 | 1.4 | Grumpier Old Men (1995) | Comedy|Romance |
| 1 | 9 | 3.5 | Sudden Death (1995) | Action |
| 2 | 3 | 4.6 | Grumpier Old Men (1995) | Comedy|Romance |
| 3 | 7 | 4.3 | Sabrina (1995) | Comedy|Romance |
| 3 | 3 | 2.7 | Grumpier Old Men (1995) | Comedy|Romance |
| 4 | 10 | 4.8 | GoldenEye (1995) | Action|Adventure|Thriller |
| 4 | 7 | 2.5 | Sabrina (1995) | Comedy|Romance |
| 4 | 10 | 3.6 | GoldenEye (1995) | Action|Adventure|Thriller |
| 5 | 3 | 3.4 | Grumpier Old Men (1995) | Comedy|Romance |
| 5 | 6 | 2.7 | Heat (1995) | Action|Crime|Thriller |

Where the author only uses a few columns such as userId, movieId, rating, movieTitle, and also genres.

Then for later use in calculating the MSE and RMSE formulas, the author has prepared data for the actual data, namely by processing the data table (4.1.1 Rating Dataset) by calculating the average rating and also sorting the data based on the highest average rating and also based on the number most ratings. First, the writer enters the two data from (tables 4.1.1 and 4.1.2) into the database which becomes 2 tables then makes a query from the two tables which becomes the movie ranking data from this dataset.

4.1.5 Query untuk mendapatkan ranking movie data

1. SELECT DISTINCT(movie.movieId), sum(rating) as total\_rating, count(rating) jumlah\_rating, round(sum(rating)/count(rating), 3) as average\_rating , movie\_title.title as title, movie\_title.genre as genres
2. FROM `movie`
3. left join movie\_title on movie.movieId = movie\_title.movieId
4. GROUP BY(movieId)
5. order by average\_rating desc, sum(rating) desc

4.1.6 Ranking Dataset

|  |  |  |  |
| --- | --- | --- | --- |
| movieId | averageRating | movieTitle | genres |
| 10 | 4.2 | GoldenEye (1995) | Action|Adventure|Thriller |
| 9 | 3.5 | Sudden Death (1995) | Action |
| 7 | 3.4 | Sabrina (1995) | Comedy|Romance |
| 4 | 3.4 | Waiting to Exhale (1995) | Comedy|Drama|Romance |
| 8 | 3.1 | Tom and Huck (1995) | Adventure|Children |
| 3 | 3.025 | Grumpier Old Men (1995) | Comedy|Romance |
| 6 | 2.7 | Heat (1995) | Action|Crime|Thriller |
| 1 | 2.4 | Toy Story (1995) | Adventure|Animation|Children|Comedy|Fantasy |
| 2 | 1.3 | Jumanji (1995) | Adventure|Children|Fantasy |
| 5 | 1.1 | Father of the Bride Part II (1995) | Comedy |

## 4.2 Collaborative Filtering

In this research, collaborative filtering used by the author is item-based collaborative filtering using the cosine similarity formula. There are several steps contained in collaborative filtering that will be used by the author.

4.2.1 Flowchart collaborative Filtering

First, the author will preprocess the data from the ratingMovie table (table 4.1.4) again so that we can use the formula for this cosine similarity later. Where in this preprocessing we will group the rating ratings in 1 column based on the same movieId and eliminate the userId column, and also we will combine these ratings with a comma separator (,) which aims to be able to separate them later using the split function in python. So from the ratingMovie dataset above, the writer gets the preprocessing data as follows.

4.2.2 Table Preprocessing Data Collaborative Filtering

|  |  |  |  |
| --- | --- | --- | --- |
| movieId | rating | movieTitle | genres |
| 3 | 1.4, 4.6, 2.7, 3.4 | Grumpier Old Men (1995) | Comedy|Romance |
| 9 | 3.5 | Sudden Death (1995) | Action |
| 7 | 4.3, 2.5 | Sabrina (1995) | Comedy|Romance |
| 10 | 4.8, 3.6 | GoldenEye (1995) | Action|Adventure|Thriller |
| 6 | 2.7 | Heat (1995) | Action|Crime|Thriller |

After preprocessing the data, the author makes input for the user or the author himself to determine the movie reference that will be the reference item by selecting the movieId. In collaborative filtering with cosine similarity, this time the cosine similarity formula used by the author is

4.2.3 Function Cosine Similarity

where A is obtained from the movieId input by the user or author, and B is all data except the data A itself. To calculate the formula there are several steps. The first is to calculate A.B , in this example, the author will assume that A is movieId 3 and B is 9.

For ||A|| and also ||B|| will look like this.

So when combined, the cosine similarity value of movieId 9 to movieId 3 is

After the author conducted testing on all movieIds against movieId 3, the following results were obtained (Cosine similarity score results were taken from all movieIds against movieId 3).

4.2.4 Table Cosine Similarity Score

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| movieId | rating | movieTitle | score | genres |
| 3 | 1.4, 4.6, 2.7, 3.4 | Grumpier Old Men (1995) | - | Comedy|Romance |
| 9 | 3.5 | Sudden Death (1995) | 2.76 | Action |
| 7 | 4.3, 2.5 | Sabrina (1995) | 2.168 | Comedy|Romance |
| 10 | 4.8, 3.6 | GoldenEye (1995) | 1.965 | Action|Adventure|Thriller |
| 6 | 2.7 | Heat (1995) | 3.533 | Action|Crime|Thriller |

4.2.5 Table Cosine Similarity Sorted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| movieId | rating | movieTitle | score | genres |
| 6 | 2.7 | Heat (1995) | 3.533 | Action|Crime|Thriller |
| 9 | 3.5 | Sudden Death (1995) | 2.76 | Action |
| 7 | 4.3, 2.5 | Sabrina (1995) | 2.168 | Comedy|Romance |
| 10 | 4.8, 3.6 | GoldenEye (1995) | 1.965 | Action|Adventure|Thriller |
| 3 | 1.4, 4.6, 2.7, 3.4 | Grumpier Old Men (1995) | - | Comedy|Romance |

After getting the value from the cosine similarity, the writer displays some of the results of the sequence of movie titles and also their genres based on the top order of the score.

Then as a comparison value, the author calculates the MSE and RMSE from the above data. By using the following formula:

4.2.6 Function MSE

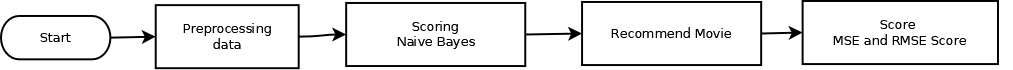
where n is the number of data entries, which is 5 in this example based on (Table 4.2.5), then Fi is the actual output which here we assume as the average rating of each movie from the movie ranking in (Table 4.1.6) , and Yi is the predicted output, which is the rating based on the score from the calculation of cosine similarity.

Then for the calculation of RMSE, the calculation used with MSE is actually almost the same, only the difference is that RMSE uses the root for the final result.

4.2.7 Function RMSE

And the MSE result is 2.89005625 while the RMSE is 0.289005625.

## 4.3 Naïve Bayes

 In making a movie recommender using the nave Bayes formula, this time the author uses the following plot.

In this study, the author started by preprocessing the data that was already owned from the results of preprocessing the previous data (Table 4.1.4). In preprocessing data, here we will try to regroup the rating data for each movie that has been given by the user based on its userId so that we have preprocessing data like the following. And also the author groups the rating data given by the user to a movie to be able to add up how many ratings the user has given to the existing movies.

4.3.2 Movie Data

|  |  |  |
| --- | --- | --- |
| movieId | Rating [userId, rating] | averageRating |
| 3 | [1, 1.4],[2, 4.6], [3, 2.7], [5, 3.4] | 3.025 |
| 9 | [1, 3.5] | 3.5 |
| 7 | [3, 4.3], [4, 2.5] | 3.4 |
| 10 | [4, 4.8], [4, 3.6] | 4.2 |
| 6 | [5, 2.7] | 2.7 |

4.3.3 Table User Data

|  |  |
| --- | --- |
| userId | Rating [movieId, rating] |
| 1 | [3, 1.4],[9, 3.5] |
| 2 | [3, 4.6] |
| 3 | [7, 4.3], [3, 2.7] |
| 4 | [10, 4.8], [7, 2.5], [10, 3.6] |
| 5 | [3, 3.4], [6, 2.7] |

The author groups all rating values on the same movieId as the separator by providing userId information on the rating given to the movieId, then there are also authors calculating the movieId collection that has been rated by the user by making it a separate array. After getting the values in the table above, we begin to perform calculations using the nave Bayes formula, namely the calculation of probabilities with the following formula:

4.3.3 Function Naïve Bayes

To be able to use the nave Bayes formula, we will calculate the probability of movie A against all the probabilities of the rating from the user that has been given to the movie A. So P(B|A) is the rating value divided by all the ratings for that 1 movie, then P(A) is the number of ratings on 1 movie divided by the number of all ratings on all the movies, then P(B) is the number of rating values given by the user divided by the number of all ratings that the user has given to all existing movies divided by the number of all existing ratings. For example, we will calculate the probability of movieId 3 then:

|  |  |  |
| --- | --- | --- |
| movieId | score | averageRating |
| 3 | 0.0130 | 3.025 |
| 9 | 0.7143 | 3.5 |
| 7 | 0.6074 | 3.4 |
| 10 | 0.5801 | 4.2 |
| 6 | 0.4426 | 2.7 |

After calculating all the nave Bayes scores from each movie to the user, the writer then sorts the data based on the nave Bayes scores.

|  |  |  |
| --- | --- | --- |
| movieId | score | averageRating |
| 9 | 0.7143 | 3.5 |
| 7 | 0.6074 | 3.4 |
| 10 | 0.5801 | 4.2 |
| 6 | 0.4426 | 2.7 |
| 3 | 0.0130 | 3.025 |

Then after getting the order based on the highest nave Bayes score, the author provides movie recommendations according to the data that has been sorted. After that the author calculates the MSE value and RMSE value using the formula:

Where Fi is the sequential rating value of the top ranking based on the nave Bayes score, then Yi is the sequential rating value of the top ranking of the actual data or data that we have sorted (Table 4.1.6). Then the author calculates the total rating on the nave Bayes score with the rating in (table 4.1.6) which we then rank and multiply by 1/ the amount of data on the rating on the Nave Bayes score.

Then to calculate the RMSE the author uses the following formula:

# 4.4 **Perbandingan Collaborative Filtering dengan Naïve Bayes**

In this test, the author compares the two algorithms between collaborative filtering using cosine similarity with nave Bayes in giving recommendations to movies. The comparison is done by the author using MSE (Mean Square Error) and also RMSE (Root Mean Square Error), where MSE is the average squared error value between the original image and the predicted image, while RMSE itself is a measurement method by measuring the difference in the value of the prediction. a model as an estimate of the observed value.

The smaller the MSE and RMSE values, the better the algorithm. From the example above, we get:

|  |  |  |
| --- | --- | --- |
|  | MSE | RMSE |
| Collaborative Filtering | 1.55 | 2.89005625 |
| Naïve Bayes | 0.1635625 | 0.4044286093 |

From the data above, we can see that the best algorithm with the same dataset between collaborative filtering and nave Bayes is nave Bayes because nave Bayes has a smaller MSE value and RMSE value compared to collaborative filtering.

# CHAPTER 5 IMPLEMENTATION AND RESULTS

## 5.1 **Collaborative Filtering**

In collaborative filtering, the author makes 6 functions in collaborative filtering coding.

1. import pandas as pd
2. import numpy as np
3. import math
4. #nrows untuk membatasi jumlah data
5. nRows = 10000
6. RatingSet = pd.read\_csv("ratings\_movies.csv",delimiter=";", nrows=nRows)
7. RatingSet = RatingSet.values

Lines 1-3 are libraries that I use, such as pandas, NumPy, math. Pandas here are used to read datasets in CSV. Then NumPy is used to process arrays and also for basic calculations. Then math here is used to square a number. In lines 6-8 the author reads a dataset in CSV named ratings\_movies, field delimiter ';' (semicolon), and also reads data as much as nRows, which is 10,000.

1. def ExplodeRatingSet(data):
2. return data.split(',')
3. def ExplodeGenre(data):
4. return data.split('|')

Lines 9-10 are a function to split a string with a separator ',' (comma) and with a data parameter that is a sentence with a string data type. Likewise in lines 12-13 only the separator is '|'.

1. def CountAverageRating(data):
2. DataRate = []
3. for i in range(len(data)):
4. DataTemp = [ExplodeRatingSet(data[i][0])[1], str(ExplodeRatingSet(data[i][0])[2]), ExplodeRatingSet(data[i][0])[3], ExplodeRatingSet(data[i][0])[4]]
6. count = 0
8. for a in range(len(DataRate)):
9. if DataRate[a][0] == DataTemp[0] and DataRate != []:
11. Ratings = str(DataRate[a][1]) + ',' + str(DataTemp[1])
12. DataTemp = [DataTemp[0], Ratings, DataTemp[2], DataTemp[3]]
13. DataRate[a] = DataTemp
14. count = 1
16. if count == 0:
17. DataRate.append(DataTemp)
18. #print('Data Appendedd')
20. return DataRate

In line 14 the author defines a function with the name CountaverageRating with a parameter named data. The parameter will be filled with the data that was just read from the CSV data. Then in line 17, the writer loops the data that will be managed by the writer, and in line 18, the writer creates a variable called DataTemp which will hold ExplodeRatingSet(data[i][0])[1] , as seen in the author calls the ExplodeRatingSet function by sending parameter data[i][0]. data[i][0] itself is data per row in the dataset why [i] then [0] because the form of the data itself is like [ 1, 16, 4, Casino] then it is necessary to open 1 block array first and then after the author splits the data based on ',' ( comma). The 1st array ( ExplodeRatingSet(data[i][0])[1] ) is the movieId, then the 2nd array is the rating, the 3rd array is the movieTitle, and the 4th array is the genres. In line 20 the author creates a variable called count the function is if after checking whether the movieId in the DataTemp is not yet in the DataRate then he will add new data to the DataRate array (lines 30-32), but if the movieId already exists then on line 25 we combine rating of the same movieId so that from [1] then added with a delimiter ',' so that it becomes [1, 3] which is where the DataRate array that has the same movieId will be updated with the rating increasing continuously (Line 26-27) and also the count variable will be 1 so it will not add new data.

1. def CosineSimilarity(id, data):
2. Ranking = []
4. #Get Movie Average
5. for a in range(len(data)):
6. if data[a][0] == id:
7. Movie = data[a]
8. break
10. MovieAverage = ExplodeRatingSet(Movie[1])
12. #Scoring Ranking With Cosine Similarity
13. for a in data:
14. OtherMovie = ExplodeRatingSet(a[1])
15. B = 1
16. Pb = 0
17. C = 1
18. Pc = 0
20. #Hitung B
21. for b in MovieAverage:
22. B = float(B) \* float(b)
23. Pb = float(Pb) + float(b)\*\*2
25. for c in OtherMovie:
26. C = float(C) \* float(c)
27. Pc = float(Pc) + float(c)\*\*2
29. Pb = round(math.sqrt(Pb), 3)
30. Pc = round(math.sqrt(Pc), 3)
32. BxC = round(B + C, 3)
33. BC = round(Pb \* Pc, 3)
35. Score = round(BxC / BC, 3)
37. Score = round(Score / 1000, 2)
39. Ranking.append([a[0], a[1], a[2], Score, a[3]])

42. return Ranking

Line 35 here the author creates a function with the name CosineSimilarity which has 2 parameters, namely id, and data, where id is the movieId of the movie to be compared and also data is the data from the CountAverageRating function. The first thing the author does, namely on lines 39-42, is looking for data that matches the movieId received from the parameter, namely id, and stored in a variable named movie. Then on line 44 the writer saves all the rating data on the previously searched movieId in the MovieAverage variable. Then to use the cosine similarity formula according to (Function 4.2.3) the author loops all movie data. In line 48 the author saves all rating data on the movieId which is looped first, then creates empty variables, namely B, Pb, C, Pc which are given a default of 1, which is to calculate the total multiplication that will be calculated during the next loop and also the default value of 0 given to become a variable that holds the total sum. Then in lines 55-57 is where the author calculates the sum of the addition and multiplication of the searched movie (with the movieId corresponding to the parameter thrown), and lines 59-61 calculate the looped movieId. Then the author calculates everything starting from BxC and then divided by BC, after the score is known, the writer adds all the data into the variables that the author has prepared on line 36 which aims to collect all the scores from cosine similarity movie A against all movieId.

1. def CountValidation(ScoreValue, DataActual):
3. MSE = 0
4. for count in range(len(ScoreValue)):
5. rates = ExplodeRatingSet(ScoreValue[count][1])
6. AverageRating = 0
7. for value in rates:
8. AverageRating += float(value)
9. AverageRating = AverageRating/len(rates)
10. ActualRating = ExplodeRatingSet(DataActual[count][0])
11. MSE += pow((AverageRating-float(ActualRating[1])) , 2)
13. MSE = 1/len(ScoreValue) \* float(MSE)
14. print('\n1 / n : ', 1/len(ScoreValue))
15. print('\nMSE / len Value : ', MSE)
17. print('\n MSE Score : ', round(MSE, 3))
18. print('\n RMSE Score : ', round(math.sqrt(MSE), 3))

In line 77 the author creates a function with the name CountValidation by accepting 2 parameters with the name ScoreValue and also DataActual where ScoreValue is the result of a calculation on the CosineSimilarity function which has been sorted based on the score from the calculation of its cosine similarity. Then in line 80 we loop the ScoreValue data which aims to compare the two ratings from the data based on their ranking order. Lines 81-84 the author first saves the rating of the ScoreValue variable which will then be looped to calculate the average rating of the movie that we predicted earlier. After getting the average rating on the movie, the writer calculates by means of the average prediction rating which is subtracted from the rating on the actual data which will then be raised to the power of 2 and then the number of MSE variables that I have prepared previously on line 79. After looping an on the ScoreValue the author calculates the MSE according to the existing formula (line 89) , and to get the RMSE value the writer takes the value on the MSE (line 94).

1. def printScore(data, limit):
2. print('\n-----Movie Ranking-----')
3. for i in range(limit):
4. print('\nWith Score : ', data[i][3], ' Title : ', data[i][2], ' Genres : ', data[i][4])

Baris 95-98 penulis membuat function dengan nama printScore yang berfungsi untuk mem print data yang diterima pada parameter data dengan jumlah sebanyak limit.

1. Average = CountAverageRating(RatingSet)
2. Ranking = CosineSimilarity('16', Average)
3. Ranking = np.array(Ranking)
4. Ranking = Ranking[np.argsort(Ranking[:, 3])]
5. printScore(Ranking, 10)
6. DataActual = pd.read\_csv("ranking.csv",delimiter=";", nrows=nRows)
7. DataActual = np.array(DataActual)
8. print('/nData Actual :', DataActual[1][0])
9. CountValidation(Ranking, DataActual)

In lines 99-113 the author only calls functions that have been created previously to get the results of this collaborative filtering. In line 99 there the author calls the CountAverageRating function, then the results are saved to the Average variable after getting the results, the author calls the CosineSimilarity function by sending 2 parameters, namely 16 and also Average where 16 is the movieId which will be compared in the cosine similarity formula and Average is the data which has been managed from the previous function, namely the CountAverageRating function ( line 101 ). Then on line 103 the Ranking that the author has calculated with the cosine similarity function, the author changes it to a NumPy array, and also the author sorts the array data in the 3rd column, namely the score of the cosine similarity. Then on line 107 the author prints the order of data by calling the printScore function with 2 ranking parameters and also the author gives a limit for printing as much as 10 data. Then to get the MSE and RMSE values, the writer takes the actual data ranking dataset from the dataset (Line 109-110). Then the author takes Ranking and Actual Data where Rank is data with a score from the calculation of cosine similarity and Actual Data is ranking data from the actual movie to calculate the MSE and also its RMSE.

# 5.2 **Naïve Bayes**

In the nave Bayes algorithm itself, the author uses a method that is almost the same as collaborative filtering, namely by making all functions first which will be called at the end. For nave Bayes, the author has 6 functions.

1. import pandas as pd
2. import numpy as np
3. import math

Lines 1-3 are some of the libraries used by the author to create this nave Bayes algorithm.

1. nRows = 5000
2. RatingSet = pd.read\_csv("ratings\_movies.csv",delimiter=";", nrows=nRows)
3. RatingSet = RatingSet.values

In lines 4-6 here the author reads the dataset from CSV form using the pandas library and then saves it into the RatingSet variable.

1. def ExplodeRatingSet(data):
2. return data.split(',')
3. def SplitingData(data, seperator):
4. return data.split(seperator)

In lines 7-8 is a function to split the string into several words based on the separator and the separator in this function is ','. Then on lines 10-11 the author makes the same function as before, only this time the separator must be sent via parameters so that the use of this split function is more flexible.

1. def PreprocessingData(data):
2. MovieData = []
3. UserData = []
5. CountCoba = 0
7. for value in data:
8. value = SplitingData(value[0], ',')
9. #print('\nValue : ', value)
11. conditionMovie = 0
12. for countMovie in range(len(MovieData)):
14. if MovieData[countMovie][0] == value[1]:
15. MovieData[countMovie][1].append([value[0], value[2]])
16. MovieData[countMovie][2] = round((MovieData[countMovie][2] + float(value[2])) / 2, 3)
17. conditionMovie = 1
18. CountCoba += 1
20. if conditionMovie == 0:
21. MovieData.append( [ value[1], [[value[0], value[2] ]], float(value[2]) ])
22. #print('\nMovie Data : ', MovieData)
24. conditionUser = 0
25. for countUser in range(len(UserData)):
27. if UserData[countUser][0] == value[0]:
28. UserData[countUser][1].append([value[1], value[2]])
29. conditionUser = 1
31. if conditionUser == 0:
32. UserData.append([value[0], [ [value[1], value[2] ] ] ])
34. print('\n Count : ', CountCoba)
36. return MovieData, UserData

In line 12 here the author creates a preprocessing function by receiving 1 parameter named data which will later be filled with the dataset that has been stored in the previous RatingSet variable. Then on line 18, the writer loops the data. Then on line 22-33 is where the author creates a variable called conditionmovie which later if this variable remains 0 then there is no data in the MovieData array and will be added to the array with the format [movieId, [[userId, rating], [ userId, rating]] ] as soon as the conditionmovie variable is 1, there will be additional data [UserId, rating] on the same movieId during the data loop. Likewise, for lines 35-43, it's just that the shape of the array is different, in lines 35-43 the authors form a data array to store user data so that the array is in the form of [UserId, [ [movieId, rating], [movieId, rating] ] ].

1. def NaiveBayes(MovieData, UserData):
2. result = []
3. tempAtas = 1.0
4. tempBawah = 1.0
5. print('\nNaive Bayes')
7. SumValueRating = 0
8. for movie in MovieData:
9. #print('\nRating Luar : ', movie[1])
10. for rating in movie[1]:
12. SumValueRating += float(rating[1])
14. for MovieValue in MovieData:
16. MovieId = MovieValue[0]
18. SumMovieRating = 0
19. for Rating in MovieValue[1]:
20. SumMovieRating += float(Rating[1])
22. tempAtas = 1.0
23. tempBawah = 1.0
25. for Rating in MovieValue[1]:
27. UserId = Rating[0]
29. TotalUserRating = 0
30. for UserValue in UserData:
32. if UserValue[0] == UserId:
34. for UserRating in UserValue[1]:
36. TotalUserRating += float(UserRating[1])
37. tempAtas = tempAtas \* (float(Rating[1])/SumMovieRating)
38. tempBawah = tempBawah \* float(TotalUserRating/SumValueRating)
40. tempAtas = tempAtas \* (SumMovieRating/SumValueRating)
42. total = round(tempAtas/tempBawah, 3)
44. result.append([MovieId, total, MovieValue[2]])
46. return result

In line 48 the author creates a NaiveBayes function by accepting 2 parameters, namely movieData and also userData. Lines 54-59 the author performs the calculation of all the rating amounts first and is saved to the SumValueRating variable. After getting the number of ratings, the writer starts to loop the data on movieData , on lines 65-67 the writer does the summation to find out the number of ratings from 1 movie according to the current looping. In lines 72-86, the writer adds the rating according to the userId currently looping the movieData data and adds up the rating from the userId. Then to be able to enter into the nave Bayes formula, the author calculates the Upper temp and Lower temp were later to get the value from this naive Bayes algorithm, the Upper temp will be divided by the Bottom temp. After calculating the top temp and the bottom temp for the top temp, the writer multiplies by the total number of ratings in 1 movie divided by the total of all ratings on line 88. After that, the writer calculates the score in the total variable on line 90 and adds the data into the result array as a result finally along with MovieValue[2] which is the average rating of the movieId.

1. def CountValidation(NaiveBayes, DataActual):
2. MSE = 0
4. for count in range(len(NaiveBayes)):
6. MSE += pow( (float(NaiveBayes[count][2]) - float( ExplodeRatingSet(DataActual[count][0])[1] ) ) , 2)
8. MSE = 1/len(NaiveBayes) \* MSE
9. RMSE = math.sqrt(MSE)
11. print('\nMSE Score : ', MSE)
12. print('\nRMSE Score : ', RMSE)
14. Return

In line 95, the author creates a function called CountValidation by sending 2 parameters, namely nave Bayes which is a prediction movie ranking based on a nave Bayes score, and there are also actual data where the actual data is the original ranking of the dataset. Then in line 98, we loop the NaiveBayes data which aims to compare the two ratings from the data based on their ranking order. Row 100 the author calculates according to the mse formula, namely by means of the average prediction rating which is deducted by the rating on the actual data which will then be raised to the power of 2 and then the sum of the MSE variables that the author has prepared in line 96. After looping on NaiveBayes The author calculates the MSE according to the existing formula (line 102), and to get the RMSE value, the writer takes the value of the MSE (line 103).

1. MovieData, UserData = PreprocessingData(RatingSet)
2. NaiveBayes = NaiveBayes(MovieData, UserData)
3. NaiveBayes = np.array(NaiveBayes)
4. NaiveBayes = NaiveBayes[np.argsort(NaiveBayes[:, 1])]
5. DataActual = pd.read\_csv("ranking.csv",delimiter=";", nrows=nRows)
6. DataActual = np.array(DataActual)
7. CountValidation(NaiveBayes, DataActual)

In line 109 we call the PreprocessingData function by sending data namely RatingSet and from that function, we receive 2 outputs namely MovieData and UserData. After getting the data, the writer calculates the nave Bayes score in the NaiveBayes function by sending 2 parameters, namely MovieData and UserData which produce an output, namely NaveBayes which contains the Nave Bayes score and its movieId. Then in lines 112-113 the author makes the NaiveBayes array into a NumPy array and sorts the data based on the nave Bayes score column with the NumPy function. After that, on lines 115-116 the author takes the actual data on the CSV data with the pandas library. And on line 117 the author calls the CountValidation function by sending 2 parameters, namely the NaïveBayes array and the DataActual array.

# 5.3 **Result**

The results of the MSE and RMSE calculations implemented in the two algorithms are as follows:

|  |  |  |
| --- | --- | --- |
| **Algoritma** | **MSE** | **RMSE** |
| Collaborative Filtering Cosine Similarity | 2.361 | 1.536 |
| Naïve Bayes | 1.728 | 1.31456 |

The results of this study, for Collaborative Filtering, the MSE value is 2.361 and the RMSE value is 1.536. Meanwhile, the Naïve Bayes algorithm gets an MSE value of 1.728 and an RMSE value of 1.31456. It can be concluded that Naïve Bayes has a better performance than Collaborative Filtering Cosine Similarity in the case of movie recommendations, because it has MSE and RMSE values that are close to 0.

# 5.4 Analysis

In this study, the authors compared movie recommendations based on item-based only. Therefore, there are many movies that have the same score, all of that because the recommendation for this research is item-based, which is only based on the title of the movie. It would be much different if the research had many factors such as the director of the film, the actors and actresses who played in the film, the year of publication of the film, the genre of the film, duration, rating, review, and many others. If the above factors are listed in the research for the next movie recommendation, the score that will be obtained for one movie with another movie will have a much different score and the possibility of getting a similar score will be small.

# CHAPTER 6 CONCLUSION

In this study, the authors compare two algorithms, namely Collaborative Filtering with Cosine Similarity approach and Naïve Bayes on movie recommendation. As a comparison value, validation calculations are carried out, namely MSE (Mean Square Error) and also RMSE (Root Mean Square Error) where the algorithm that gets MSE and RMSE values ​​close to 0 is the most suitable algorithm. From the results of the study, the MSE value was 2.361 for Collaborative Filtering and 1.728 for Naïve Bayes. The RMSE value is 1.536 for Collaborative Filtering and for Naïve Bayes it is 1.31456. The results of MSE and RMSE are shown in the data above, so it can be concluded that Naïve Bayes has better recommendations than the Collaborative Filtering with the Cosine Similarity approach algorithm.

The limitation of this project is that the author uses a factor in the form of a movie rating that produces a predictive score. The predicted scores have similar values ​​due to the factor used in the form of movie ratings. So that in this project the movie recommendations from Naive Bayes and collaborative filtering using cosine similarity are less than optimal due to the limitations of these factors.

Suggestions for further research, adding more factors so that the value obtained reduces the similarity of scores to one another and produces better recommendations.

REFERENCES

[1] A. N. Khusna, K. P. Delasano, D. Chaerul, and E. Saputra, “Penerapan User-Based Collaborative Filtering Algorithm Studi Kasus Sistem Rekomendasi untuk Menentukan Gadget Shield Application of User-Based Collaborative Filtering Algorithm : Case Study of Recommendation System for Determining Gadget Shield,” vol. 20, no. 2, pp. 293–304, 2021, doi: 10.30812/matrik.v20i2.1124.

[2] I. S. Wahyudi, *Mesin Rekomendasi Film Menggunakan Metode Kemiripan Genre Berbasis Collaborative Filtering*. 2017.

[3] N. Safir and S. Ziad, “Evaluating Prediction Accuracy for Collaborative Filtering Algorithms in Recommender Systems,” *KTH R. Inst. Technol.*, 2016.

[4] N. Raval and V. Khedkar, “A review paper on collaborative filtering based moive recommedation system,” *Int. J. Sci. Technol. Res.*, vol. 8, no. 12, pp. 2507–2512, 2019.

[5] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, “Eigentaste: A Constant Time Collaborative Filtering Algorithm,” *Inf. Retr. Boston.*, vol. 4, no. 2, pp. 133–151, 2001, doi: 10.1023/A:1011419012209.

[6] A. N. Varma, “Movie Recommender System,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 9, no. 1, pp. 833–837, 2021, doi: 10.22214/ijraset.2021.32936.

[7] L. Shuxian and F. Sen, “Design and Implementation of Movie Recommendation System Based on Naive Bayes,” *J. Phys. Conf. Ser.*, vol. 1345, no. 4, 2019, doi: 10.1088/1742-6596/1345/4/042042.

[8] S. P. Sahu, A. Nautiyal, and M. Prasad, “Machine Learning Algorithms for Recommender System - a comparative analysis,” *Int. J. Comput. Appl. Technol. Res.*, vol. 6, no. 2, pp. 97–100, 2017, doi: 10.7753/ijcatr0602.1005.

[9] A. Dubey and R. Ranjan, “Movie Recommendation System using Naive Bayes Algorithm with Collaborative Filtering,” vol. 9, no. 7, pp. 2018–2020, 2020, doi: 10.21275/SR20603193402.

[10] P. Sharma and L. Yadav, “Movie Recommendation System Using Item Based Collaborative Filtering,” *Int. J. Innov. Res. Comput. Sci. Technol.*, vol. 8, no. 4, pp. 8–12, 2020, doi: 10.21276/ijircst.2020.8.4.2.

APPENDIX

**CODE**

1. import pandas as pd
2. import numpy as np
3. import math
4. class CF:
6. def ExplodeRatingSet(data):
7. return data.split(',')
9. def ExplodeGenre(data):
10. return data.split('|')
12. def CountAverageRating(data):
13. DataRate = []
15. for i in range(len(data)):
16. DataTemp =[CF.ExplodeRatingSet(data[i][0])[1], str(CF.ExplodeRatingSet(data[i][0])[2]), CF.ExplodeRatingSet(data[i][0])[3], CF.ExplodeRatingSet(data[i][0])[4]]
18. count = 0
20. for a in range(len(DataRate)):
21. if DataRate[a][0] == DataTemp[0] and DataRate != []:
23. Ratings = str(DataRate[a][1]) + ',' + str(DataTemp[1])
24. DataTemp = [DataTemp[0], Ratings, DataTemp[2], DataTemp[3]]
25. DataRate[a] = DataTemp
26. count = 1
28. if count == 0:
29. DataRate.append(DataTemp)
30. #print('Data Appendedd')
32. return DataRate
34. def CosineSimilarity(id, data):
35. Ranking = []
37. #Get Movie Average
38. for a in range(len(data)):
39. if data[a][0] == id:
40. Movie = data[a]
41. break
43. MovieAverage = CF.ExplodeRatingSet(Movie[1])
45. #Scoring Ranking With Cosine Similarity
46. for a in data:
47. OtherMovie = CF.ExplodeRatingSet(a[1])
48. B = 1
49. Pb = 0
50. C = 1
51. Pc = 0
53. #Hitung B
54. for b in MovieAverage:
55. B = float(B) \* float(b)
56. Pb = float(Pb) + float(b)\*\*2
58. for c in OtherMovie:
59. C = float(C) \* float(c)
60. Pc = float(Pc) + float(c)\*\*2
62. Pb = round(math.sqrt(Pb), 3)
63. Pc = round(math.sqrt(Pc), 3)
65. BxC = round(B + C, 3)
66. BC = round(Pb \* Pc, 3)
68. Score = round(BxC / BC, 3)
70. Score = round(Score / 1000, 2)
72. Ranking.append([a[0], a[1], a[2], Score, a[3]])

75. return Ranking
77. def sortingRanking(val):
78. return val[3]
80. def printScore(data, limit):
81. print('\n-----Movie Ranking-----')
82. for i in range(limit):
83. print('\nTitle : ', data[i][2], ' Genres : ', data[i][4])
85. def CountValidation(ScoreValue, DataActual):
87. MSE = 0
88. for count in range(len(ScoreValue)):
89. rates = CF.ExplodeRatingSet(ScoreValue[count][1])
90. AverageRating = 0
91. for value in rates:
92. AverageRating += float(value)
93. AverageRating = AverageRating/len(rates)
94. ActualRating = CF.ExplodeRatingSet(DataActual[count][0])
95. MSE += pow((AverageRating-float(ActualRating[1])) , 2)
97. MSE = 1/len(ScoreValue) \* float(MSE)
98. print('\n MSE Score : ', round(MSE, 3))
99. print('\n RMSE Score : ', round(math.sqrt(MSE), 3))
101. class NB:
102. def ExplodeRatingSet(data):
103. return data.split(',')
105. def SplitingData(data, seperator):
106. return data.split(seperator)
108. def PreprocessingData(data):
109. MovieData = []
110. UserData = []
112. CountCoba = 0
114. for value in data:
115. value = NB.SplitingData(value[0], ',')
116. #print('\nValue : ', value)
118. conditionMovie = 0
119. for countMovie in range(len(MovieData)):
121. if MovieData[countMovie][0] == value[1]:
122. MovieData[countMovie][1].append([value[0], value[2]])
123. MovieData[countMovie][2] = round((MovieData[countMovie][2] + float(value[2])) / 2, 3)
124. conditionMovie = 1
125. CountCoba += 1
127. if conditionMovie == 0:
128. MovieData.append( [ value[1], [[value[0], value[2] ]], float(value[2]), value[3] ])
130. conditionUser = 0
131. for countUser in range(len(UserData)):
133. if UserData[countUser][0] == value[0]:
134. UserData[countUser][1].append([value[1], value[2]])
135. conditionUser = 1
137. if conditionUser == 0:
138. UserData.append([value[0], [ [value[1], value[2] ] ] ])
140. print('\n Count : ', CountCoba)
142. return MovieData, UserData
144. def NaiveBayes(MovieData, UserData):
145. result = []
146. tempAtas = 1.0
147. tempBawah = 1.0
148. #print('\nNaive Bayes')
150. SumValueRating = 0
151. for movie in MovieData:
152. #print('\nRating Luar : ', movie[1])
153. for rating in movie[1]:
155. SumValueRating += float(rating[1])
157. for MovieValue in MovieData:
159. MovieId = MovieValue[0]
161. SumMovieRating = 0
162. for Rating in MovieValue[1]:
163. SumMovieRating += float(Rating[1])
165. tempAtas = 1.0
166. tempBawah = 1.0
168. for Rating in MovieValue[1]:
170. UserId = Rating[0]
172. TotalUserRating = 0
173. for UserValue in UserData:
175. if UserValue[0] == UserId:
177. for UserRating in UserValue[1]:
179. TotalUserRating += float(UserRating[1])
181. tempAtas = tempAtas \* (SumMovieRating/SumValueRating)

184. total = round(tempAtas/tempBawah, 3)
186. result.append([MovieId, total, MovieValue[2], MovieValue[3]])
188. return result
190. def CountValidation(NaiveBayes, DataActual):
191. MSE = 0
193. for count in range(len(NaiveBayes)):
195. MSE += pow( (float(NaiveBayes[count][2]) - float( NB.ExplodeRatingSet(DataActual[count][0])[1] ) ) , 2)
197. MSE = 1/len(NaiveBayes) \* MSE
198. RMSE = math.sqrt(MSE)
200. print('\nMSE Score : ', MSE)
201. print('\nRMSE Score : ', RMSE)
203. return
204. # Raw Data
206. nRows = 10000
207. RatingSet = pd.read\_csv("ratings\_movies.csv",delimiter=";", nrows=nRows)
208. RatingSet = RatingSet.values
209. # Actual Data
210. DataActual = pd.read\_csv("ranking.csv",delimiter=";", nrows=nRows)
211. DataActual = np.array(DataActual)
212. print('\nData Actual :', DataActual[1][0])
213. # Collaborative Filtering
214. Average = CF.CountAverageRating(RatingSet)
215. pd.set\_option('display.max\_columns', None)
216. print('\nAverage : ', pd.DataFrame(Average).head())
217. Ranking = CF.CosineSimilarity('16', Average)
218. Ranking = np.array(Ranking)
219. print('\n\nRanking :\n',Ranking)
220. Ranking = Ranking[np.argsort(Ranking[:, 3])]
221. CF.printScore(Ranking, 10)
222. CF.CountValidation(Ranking, DataActual)
223. # Naive Bayes
224. MovieData, UserData = NB.PreprocessingData(RatingSet)
225. pd.set\_option('display.max\_columns', None)
226. print('\nMovie Data : \n ', pd.DataFrame(MovieData).head(10))
227. print('\nUser Data : \n ', pd.DataFrame(UserData).head(10))
228. print('\n Movie Data : ', MovieData[1])
229. NaiveBayes = NB.NaiveBayes(MovieData, UserData)
230. NaiveBayes = np.array(NaiveBayes)
231. NaiveBayes = NaiveBayes[np.argsort(NaiveBayes[:, 1])]
232. print('\nOutPut Naive Bayes :\n', pd.DataFrame(NaiveBayes).tail(100))
233. NB.CountValidation(NaiveBayes, DataActual)
234. # Comparing Ranking Movies
235. for count in range(10):
237. print('\nCF : ', Ranking[count][2], ' NB : ', NaiveBayes[count][3], ' Actual : ', NB.ExplodeRatingSet(DataActual[count][0])[2] )