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.
5.  #nrows untuk membatasi jumlah data
6.  nRows = 10000
7.  RatingSet = pd.read_csv("ratings_movies.csv",delimiter=";", nrows=nRows)
8.  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.

```
9.  def ExplodeRatingSet(data):
10.     return data.split(',')
11.
12. def ExplodeGenre(data):
13.     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 '|'.

```
14. def CountAverageRating(data):
15.     DataRate = []
16.
17.     for i in range(len(data)):
18.         DataTemp = [ExplodeRatingSet(data[i][0])[1], str(ExplodeRatingSet(data[i][0])[2]),
    ExplodeRatingSet(data[i][0])[3], ExplodeRatingSet(data[i][0])[4]]
19.
20.         count = 0
21.
22.         for a in range(len(DataRate)):
23.             if DataRate[a][0] == DataTemp[0] and DataRate != []:
24.
```

```
25.                 Ratings = str(DataRate[a][1]) + ',' + str(DataTemp[1])
26.                 DataTemp = [DataTemp[0], Ratings, DataTemp[2], DataTemp[3]]
27.                 DataRate[a] = DataTemp
28.                 count = 1
29.
30.         if count == 0:
31.             DataRate.append(DataTemp)
32.             #print('Data Appendedd')
33.
34.     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.

```
35. def CosineSimilarity(id, data):
36.     Ranking = []
37.
38.     #Get Movie Average
39.     for a in range(len(data)):
40.         if data[a][0] == id:
41.             Movie = data[a]
42.             break
43.
44.     MovieAverage = ExplodeRatingSet(Movie[1])
45.
46.     #Scoring Ranking With Cosine Similarity
47.     for a in data:
48.         OtherMovie = ExplodeRatingSet(a[1])
49.         B = 1
```

```
50.        Pb = 0
51.        C = 1
52.        Pc = 0
53.
54.        #Hitung B
55.        for b in MovieAverage:
56.            B = float(B) * float(b)
57.            Pb = float(Pb) + float(b)**2
58.
59.        for c in OtherMovie:
60.            C = float(C) * float(c)
61.            Pc = float(Pc) + float(c)**2
62.
63.        Pb = round(math.sqrt(Pb), 3)
64.        Pc = round(math.sqrt(Pc), 3)
65.
66.        BxC = round(B + C, 3)
67.        BC = round(Pb * Pc, 3)
68.
69.        Score = round(BxC / BC, 3)
70.
71.        Score = round(Score / 1000, 2)
72.
73.        Ranking.append([a[0], a[1], a[2], Score, a[3]])
74.
75.
76.    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.

```
77.  def CountValidation(ScoreValue, DataActual):
78.
79.      MSE = 0
80.      for count in range(len(ScoreValue)):
81.          rates = ExplodeRatingSet(ScoreValue[count][1])
82.          AverageRating = 0
83.          for value in rates:
84.              AverageRating += float(value)
85.          AverageRating = AverageRating/len(rates)
86.          ActualRating = ExplodeRatingSet(DataActual[count][0])
87.          MSE += pow((AverageRating-float(ActualRating[1])) , 2)
88.
89.      MSE = 1/len(ScoreValue) * float(MSE)
90.      print('\n1 / n : ', 1/len(ScoreValue))
91.      print('\nMSE / len Value : ', MSE)
92.
93.      print('\n MSE Score : ', round(MSE, 3))
94.      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).

```
95.  def printScore(data, limit):
96.      print('\n-----Movie Ranking-----')
97.      for i in range(limit):
98.          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.

```
99.  Average = CountAverageRating(RatingSet)
```

```
100.
101.Ranking = CosineSimilarity('16', Average)
102.
103.Ranking = np.array(Ranking)
104.
105.Ranking = Ranking[np.argsort(Ranking[:, 3])]
106.
107.printScore(Ranking, 10)
108.
109.DataActual = pd.read_csv("ranking.csv",delimiter=";", nrows=nRows)
110.DataActual = np.array(DataActual)
111.print('/nData Actual :', DataActual[1][0])
112.
113.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.

```
import pandas as pd
import numpy as np
import math
```

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

```
4   nRows = 5000
5   RatingSet = pd.read_csv("ratings_movies.csv",delimiter=";", nrows=nRows)
6   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.

```
7   def ExplodeRatingSet(data):
8       return data.split(',')
9
10  def SplitingData(data, seperator):
11      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.

```
12  def PreprocessingData(data):
13      MovieData = []
14      UserData = []
15
16      CountCoba = 0
17
18      for value in data:
19          value = SplitingData(value[0], ',')
20          #print('\nValue : ', value)
21
22          conditionMovie = 0
23          for countMovie in range(len(MovieData)):
24
25              if MovieData[countMovie][0] == value[1]:
26                  MovieData[countMovie][1].append([value[0], value[2]])
27                  MovieData[countMovie][2] = round((MovieData[countMovie][2] + float(value[2])) / 2, 3)
28                  conditionMovie = 1
29                  CountCoba += 1
30
31          if conditionMovie == 0:
32              MovieData.append( [ value[1], [[value[0], value[2] ]], float(value[2]) ])
33              #print('\nMovie Data : ', MovieData)
34
35          conditionUser = 0
36          for countUser in range(len(UserData)):
37
38              if UserData[countUser][0] == value[0]:
39                  UserData[countUser][1].append([value[1], value[2]])
40                  conditionUser = 1
41
```

```
42          if conditionUser == 0:
43              UserData.append([value[0], [ [value[1], value[2] ] ] ])
44
45      print('\n Count : ', CountCoba)
46
47      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]]].

```
48  def NaiveBayes(MovieData, UserData):
49      result = []
50      tempAtas = 1.0
51      tempBawah = 1.0
52      print('\nNaive Bayes')
53
54      SumValueRating = 0
55      for movie in MovieData:
56          #print('\nRating Luar : ', movie[1])
57          for rating in movie[1]:
58
59              SumValueRating += float(rating[1])
60
61      for MovieValue in MovieData:
62
63          MovieId = MovieValue[0]
64
65          SumMovieRating = 0
66          for Rating in MovieValue[1]:
67              SumMovieRating += float(Rating[1])
68
69          tempAtas = 1.0
70          tempBawah = 1.0
71
72          for Rating in MovieValue[1]:
73
74              UserId = Rating[0]
75
76              TotalUserRating = 0
77              for UserValue in UserData:
78
```

```
79                  if UserValue[0] == UserId:
80
81                      for UserRating in UserValue[1]:
82
83                          TotalUserRating += float(UserRating[1])
84
85              tempAtas = tempAtas * (float(Rating[1])/SumMovieRating)
86              tempBawah = tempBawah * float(TotalUserRating/SumValueRating)
87
88          tempAtas = tempAtas * (SumMovieRating/SumValueRating)
89
90          total = round(tempAtas/tempBawah, 3)
91
92          result.append([MovieId, total, MovieValue[2]])
93
94      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.

```
95  def CountValidation(NaiveBayes, DataActual):
96      MSE = 0
97
98      for count in range(len(NaiveBayes)):
99
100         MSE += pow( (float(NaiveBayes[count][2]) - float( ExplodeRatingSet(DataActual[count][0])[1] )
    ) , 2)
101
102     MSE = 1/len(NaiveBayes) * MSE
103     RMSE = math.sqrt(MSE)
104
105     print('\nMSE Score : ', MSE)
106     print('\nRMSE Score : ', RMSE)
107
```

```
108     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).

```
109 MovieData, UserData = PreprocessingData(RatingSet)
110
111 NaiveBayes = NaiveBayes(MovieData, UserData)
112 NaiveBayes = np.array(NaiveBayes)
113 NaiveBayes = NaiveBayes[np.argsort(NaiveBayes[:, 1])]
114
115 DataActual = pd.read_csv("ranking.csv",delimiter=";", nrows=nRows)
116 DataActual = np.array(DataActual)
117 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.