REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

TEMPORAL DRIFT OF USER RATING ON MOVIE RECOMMENDER SYSTEMS

16011084 — Mustafa Katipoğlu

COMPUTER PROJECT

Advisor
RA. Sercan AYGÜN

Ocak, 2020

TABLE OF CONTENTS

LIST OF SYMBOLS

r_{ui}	Rating of user u on item i
$\hat{r}_{ui}$	Prediction rating of user u on item i
μ_u	Average rating of user u
ρ_{uv}	Pearson correlation in between user u and v
I_{uv}	Movies rated by user u and user v
k	K nearest neighbours
u,v	user of interest
i,j	item of interest

LIST OF ABBREVIATIONS

Movielens	Movie Lens Dataset
RMSE	Root Mean Square Error
TRS	Temporal Recommendation System
RS	Recommendation System
CF	Collaborative Filtering
GB	Giga Byte

LIST OF FIGURES

LIST OF TABLES

1
Introduction

As multitude of new applications arise, consumers are drowned with choices. In the wake of solving this problem, the recommendation systems (RS) have been used with different algorithms with a view to provide decent amount of choice by filtering only the related ones to the user's interest. One of the most widely used and researched recommendation technology is the Collaborative Filtering (CF) [1] which provides personalized recommendations with high accuracy. Common approaches to CF are latent factor models, which directly profile both users and products and neighborhood models, that analyze similarities between products or users. With the of Netflix Prize competition, it has been made tremendous progress on the accuracy CF [2] . A lesson that has been learnt through this competition is that the neighborhood and latent factor approaches address quite different levels of structure in the data, so none of them is optimal on its own [3]. This observation has led to discovery of new kinds of CF approaches which is the combination of the both neighborhood based methods and latent factor models. These integrated models have improved the prediction accuracy of latent factor models while maintaining the merits of the neighborhood based methods such as explainability of predictions and ability to handle new users without re-training the model [4] .

While integrated models have high accuracy rates, they still lack understanding interest of users in detail. In the search of these hidden underlying effects, the interest drifting has been observed which is the rating scores of users on items tend to drift over time [5] . As the research on the temporal dynamics [6] has emphasized that they have faced with different kinds of time-related effects along with complicated forms of concept drifts where interconnected preferences of many users are drifting in different ways at different time points. Also, The temporal diversity of recommendations is an important factor [7] that affects the quality of RS and provides the feedback of the diverse tastes of users over time. By addressing various types of temporal effects, one can get not only get predictions with high accuracy but also reveal the true interest of the users'.

While incorporating the temporal effects that we have just mentined into CF approaches, it has been seen that the new enriched approaches which has integrated both neighborhood based and factor based methods into one enriched approach like in [4] , [7] , [8] has shown to be more compatible [6] with these temporal effects.

With an extensive review of TRS, it has been found that most of the current temporal approaches have focused on short-term and long-term temporal effects, but few of the approaches have focused on temporal drift and decay of CF [9] . As it has been pointed out by Al-Hadi et al in the review, the achievements of current temporal recommendation systems which are mostly based on Matrix Factorization are not satisfactory.

We believe that the similarity measure used for evaluation of neighborhood relationships can have significant effect on improving TRS. As it has been proved by addressing data in a problem specific way in the similarity measure, the prediction accuracy can be increased [10] .

In this research, we are in the search of uncovering the temporal drift effects of users and also if possible try to introduce some baselines for integrating these hidden effects into other advanced methods.

2
Preliminaries

Since the project is about researching and evaluating the temporal drift effect of the user ratings, almost every step we take is related to the ratings data which is part of the MovieLens dataset that we have chosed as the main dataset of our project. That is why before moving forward, we need to analize and get to know the dataset. In this section, we see the detailed analysis of the MovieLens dataset, and move on to contributions of the project and our roadmap.

2.1 MovieLens Dataset

The dataset we use in this project is the Movie Lens dataset. The biggest advantage of the Movie Lens dataset is that it provides two version of the same dataset, one of which is small and the other one is big. Since the evaluation phase of the project requires us to analyze the dataset from different perspectives, we have got to work on the small version rather than the big version of the dataset. Otherwise, If we take the big dataset when we are evaluating, it would take unreasonable amount of time with the hardware that we have currently allocated for this project. For this reason, in the evaluation phase of the project we will be using the small version of the dataset. Later if we able to conclude with an algorithm that is proved to be working on the small version of the dataset, we will start working on the big version of the dataset as the last step to proof our conclusions.

The small version of the Movie Lens dataset includes 100,000 ratings and 9,000 movies that has been rated by 610 users as of February 2020.

The big version of the Movie Lens dataset includes 27 million ratings, 58,000 movies that has been rated by 280,000 users.

userId	movieId	rating	timestamp

Table 2.1 Format of Ratings

userId : Id of the user, ranges from 0 to 610.
movieId : Id of the movie that the rating is related.
rating : Given rating by the user that ranges from 0.5 to 5 with 0.5 increments.
timestamp : Unix based timestamp of the rating.

movieId	title	genres

Table 2.2 Format of Movies

movieId : The id of the movie.
title : The title of the movie which also includes the release year inside parentheses at the end.
genres : List of genres that the movie is related.

2.1.1 Analysis of The MovieLens Small Dataset

Since we will be working on the small version of the dataset in the evaluation phase, which is the biggest part of the project, we will be examining the small version of the dataset rather than the big version.

2.1.2 Analysis on Dataset Users

The **first insight** of the movielens dataset about users that is extracted from Figure 2.1 is that huge number of ratings are given in some intervals of the dataset, while in others these numbers are quite low. The high variance of the ratings count in different years could possibly result in bias in some analysis.

The **second insight** comes from Figure 2.2 and Figure 2.3 that is only few users tend to give high number of ratings.

The **last insight** about users in the dataset comes from Figure 2.4 that is different users have different average movie ratings, which proves the fact that some users are optimistic and give high ratings on average, while others are kind of pessimistic and tend to give low ratings on average.

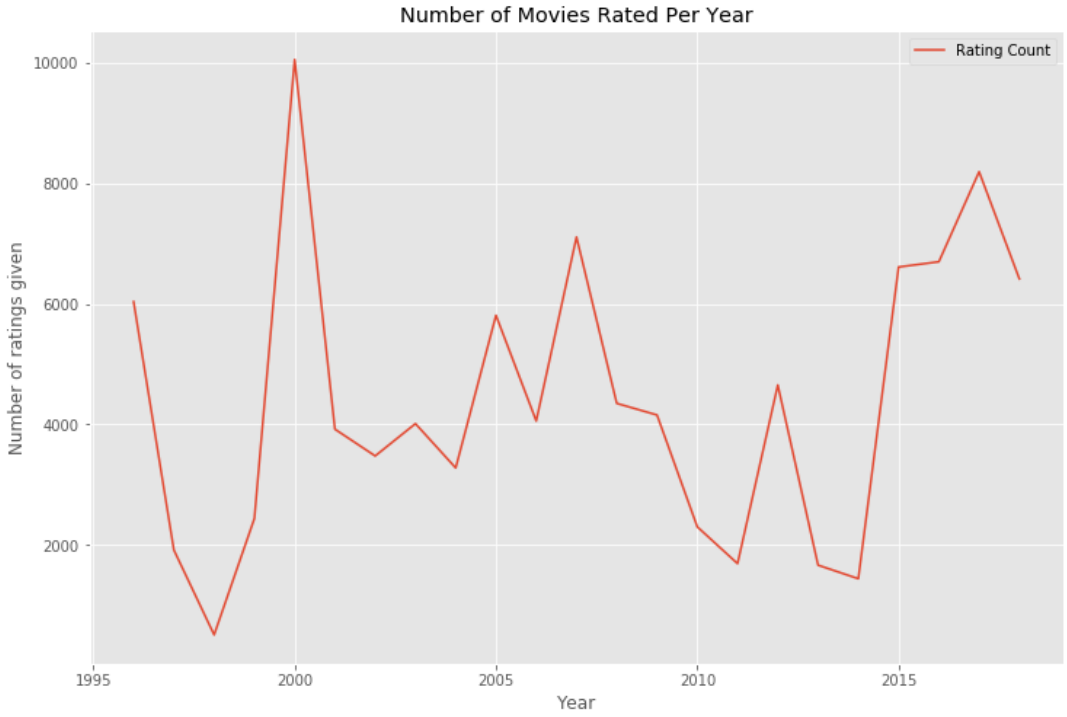

Figure 2.1 Number of Movies Rated Per Year

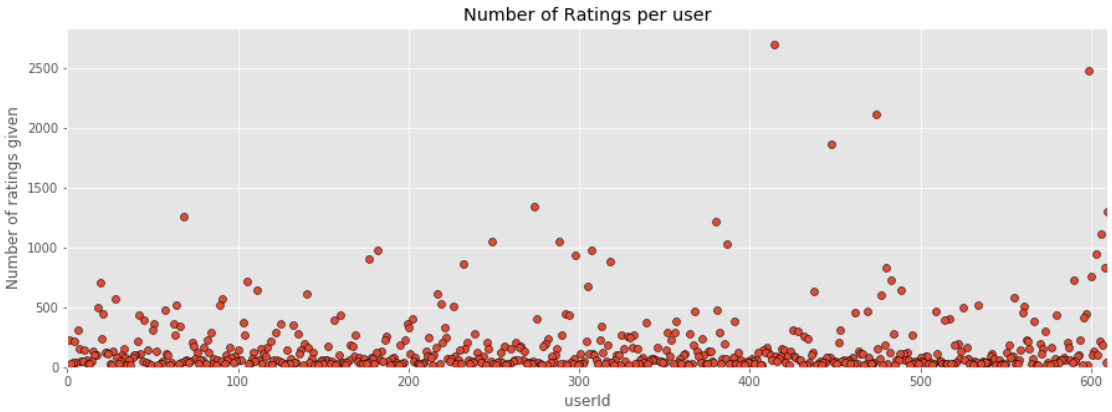

Figure 2.2 Number of Ratings Per User

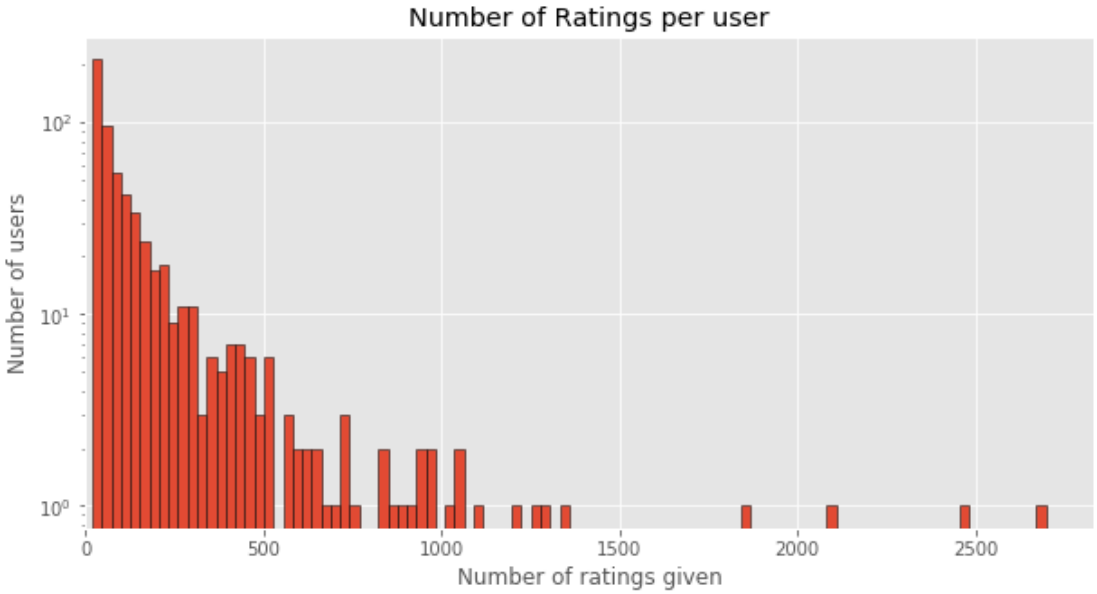

Figure 2.3 Number of Ratings Given By Users

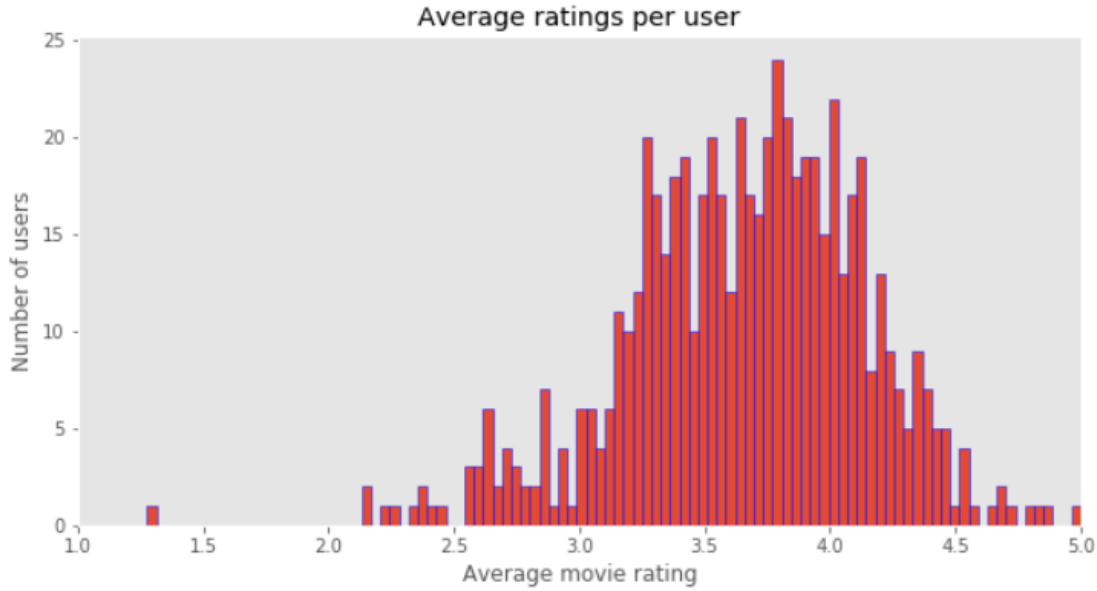

Figure 2.4 Number of Ratings Per User

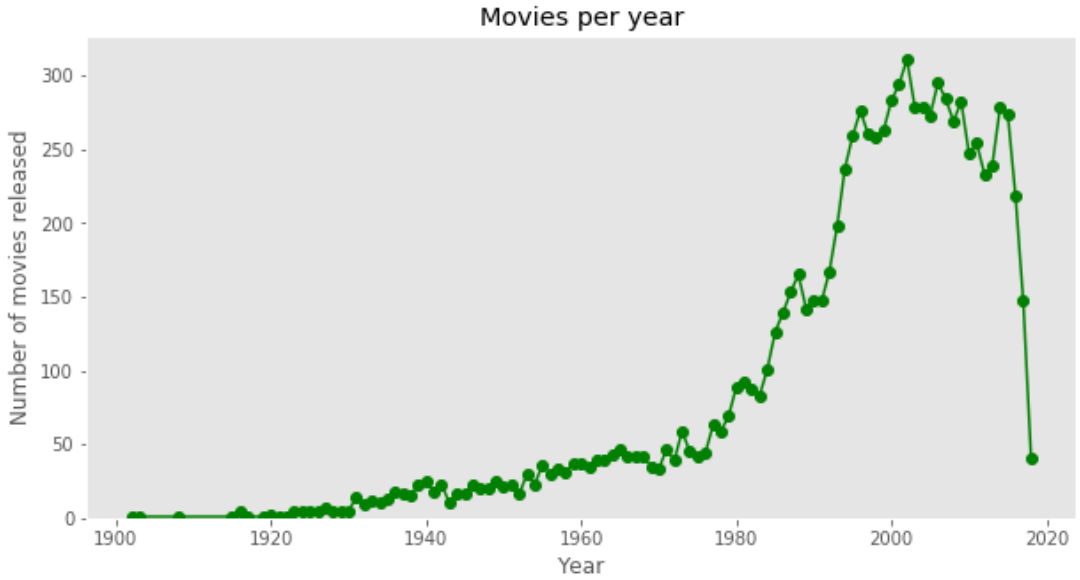

Figure 2.5 Movies Released Per Year

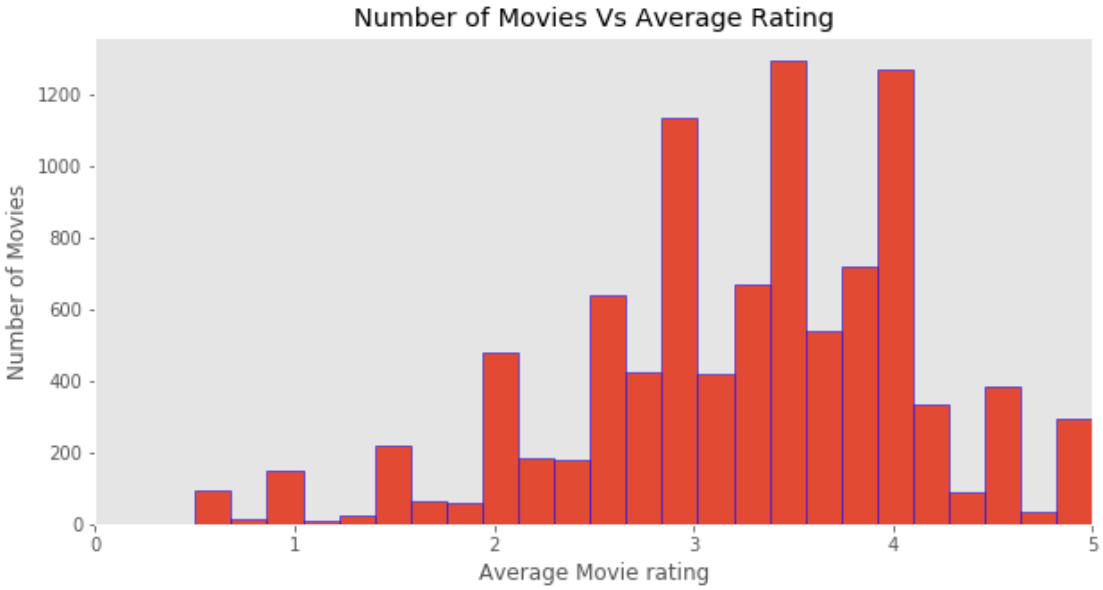

Figure 2.6 Number of Movies Vs Average Rating of Movies

2.1.3 Analysis on Dataset Movies

The **first insight** of the movielens dataset about movies that is extracted from Figure 2.5 is that the dataset provides movies released from a long range of period, starting from early 1900s up until 2020. This observation intuitively gives us a hint about the fact that the dataset almost represent the real cases.

The **second insight** comes from Figure 2.6 that is most movies have been rated in between 3 and 4 as expected in real life scenarios.

The **third insight** comes from Figure 2.7 that is interestingly movies that have been released before 1980 seem to have fluctuating average movie ratings while the movies that have been released after 1980 have more stable average ratings.

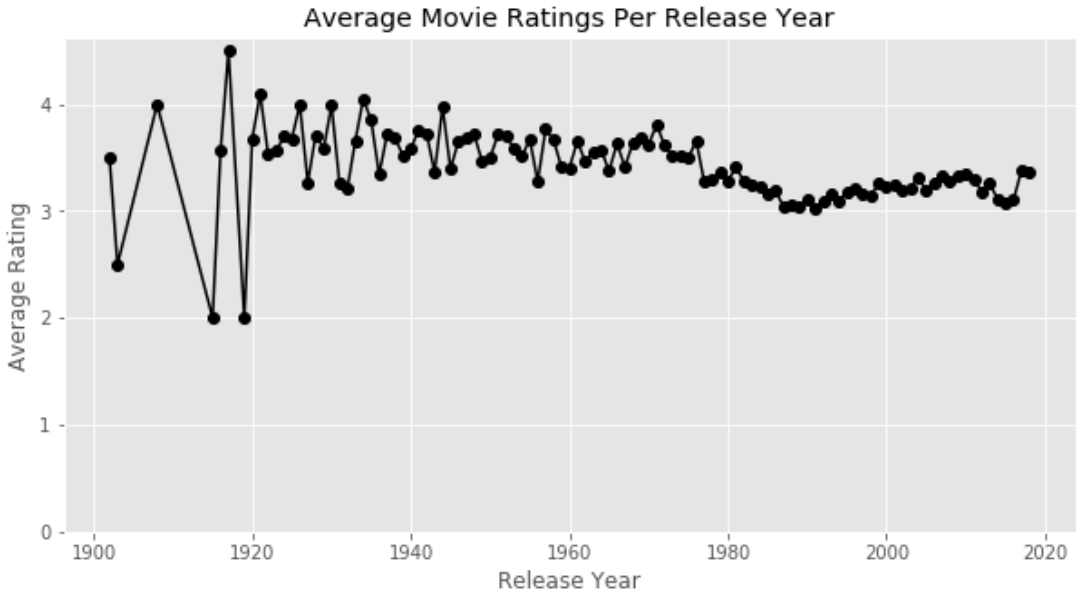

Figure 2.7 Average Movie Ratings Per Release Year

The **last insight** about movies comes from Figure 2.8 that is when we take a closer look at the average ratings of the movies that have been released after the first rating given, average ratings seem to change a lot as years pass by. Also, average movie ratings seem to have a tendency to go down. This observation raises different questions. Like, does the movies released in adjacent years changes in quality, or in some way the users are having a different trend as a result of the trend that they have just had by watching movies released in the year before the movie release. In other words, do the movies that have been released in the adjacent years tend to be similar which ultimately results in lower average movie ratings since the users are already over that kind of trend.

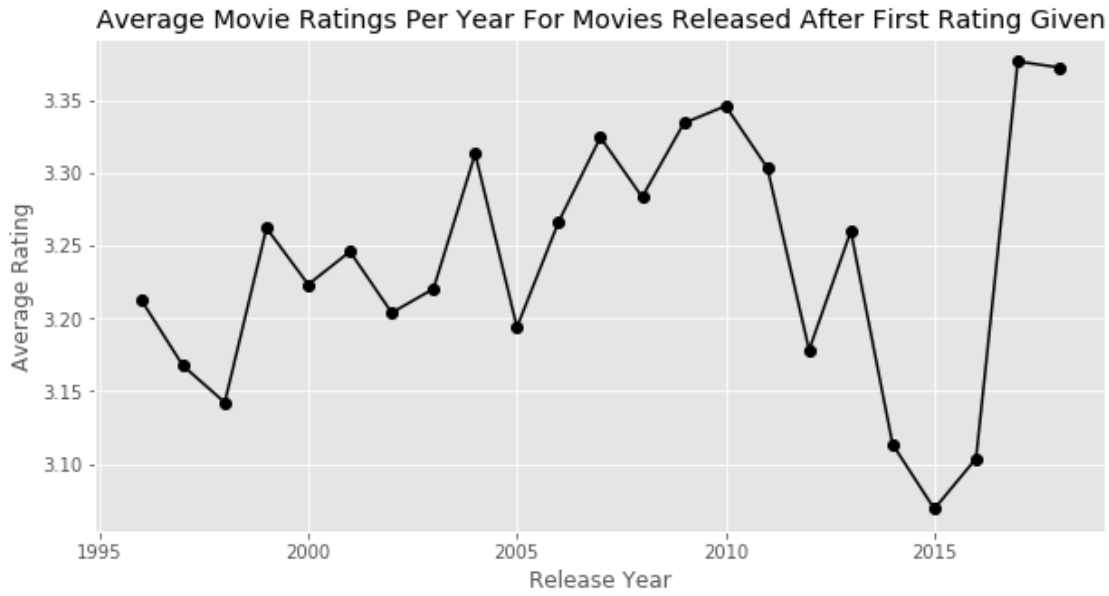

Figure 2.8 Average Movie Ratings Per Release Year For Movies Released After First Rating Given

2.2 The Project's Contributions and Road Map

Even though there has been plenty of research about the recommendation system, the temporal analysis part of the recommendation systems research is not mature enough as the other parts of the research. In this project, by focusing on the temporal drift effect of the user ratings, we aim to contribute to temporal analysis field of the recommendation system. The main focus of the project is about detecting the intervals of the users' activity where the users' ratings drift. If we able to detect the temporal drift effects of users successfully, then we can surely find the most similar temporal drifts that the other users already had before. By doing so, we can possibly increase the accuracy of the recommendation system with the help of the data that we have collected from the most similar users in this context.

The way we approach to this problem can be listed as follows:

1. Create a framework where we can evaluate the temporal parts of the data set.
2. Analyze and understand the data set.
3. Try to uncover the intervals where users could possible having temporal drift.
4. Try to develop baselines by making use of temporal drifts.

3
Feasibility

In this part of the project, the feasibility of the project has been investigated in order to determine the approach that we will be taking along with this project.

We first start by analyzing the technical feasibility. The technical feasibility has been analyzed in two parts. The first one is the software feasibility, we mention the qualities that have shaped the choice of programming language and the libraries. The second one is the hardware feasibility, we review the hardware that we will be using during development and deployment phase .

We move forward by continuing with the time feasibility section where we make the analysis of the time intervals. Later, we continue by taking a look at the legal feasibility. In legal feasibility, we review the permissions that we have required to take and the license restrictions. Lastly, we will be considering the economic feasibility in which we analyze the required budget required to develop and deploy the system.

3.1 Technical Feasibility

As we have examined, we analyze the technical feasibility as Software Feasibility and Hardware Feasibility.

3.1.1 Software Feasibility

As the programming language to be used in this project, we have chosen python programming language for a number of reasons:

- Ease of use
- Support for widely used data science libraries
- High community support

We have decided not to use any library other than basic data science libraries like pandas, numpy and matplotlib.

3.1.2 Hardware Feasibility

When developing the project, we have not felt any need to extra special hardware.That is why we have decided to move along with personal computer rather than any special hardware.

The hardware specifications of the personal computer used is shown in the Table 3.1.

Table 3.1 Hardware Specifications

RAM	16GB
Storage	256 GB SSD
Processor	Intel Core i7-8750 2.20 GHz
Graphics Card	GeForce GTX 1060
Operating System	Windows 10 Pro 64 bit

3.2 Time Feasibility

The project is planned to be finished in 78 days in spring term. Since the project is only done by one member, we have not made any task distribution. The tasks to be completed in the allocated period shown in the Figure 3.1 by using Gant diagram as a timeline.

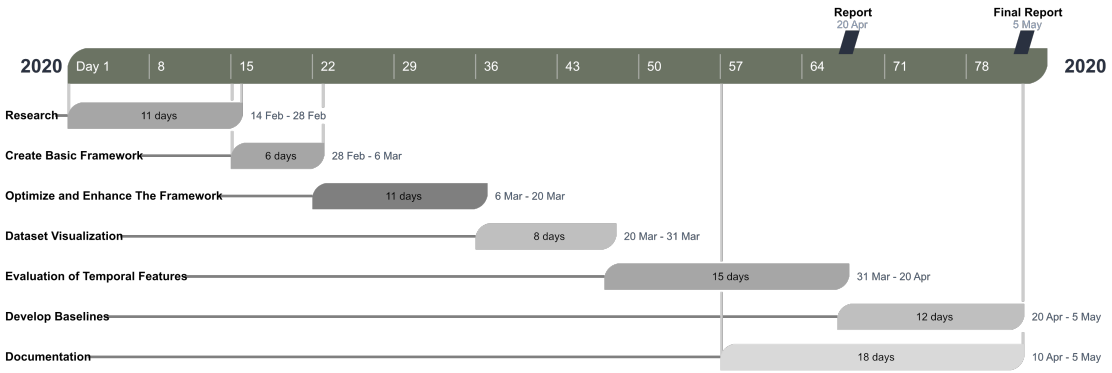

Figure 3.1 Gant diagram: Timeline

3.3 Legal Feasibility

Since we will be using only open source libraries in this non-profit project, there is no need to extra permission. There are some libraries that we will be using that they have licenses. But none of these libraries restrict us in any way in the context of this project.

numpy library has [11] BSD 3-Clause License.
pandas library has [12] BSD 3-Clause License.

3.4 Economic Feasibility

In this research focused project, we found no economic feasibility since the project is only focused on evaluating and providing better algorithm.

4
System Analysis

In this part of, we have analized the core topics of our temporal drift research. We first begin by analizing the dataset that we have used thoroughly since almost all analysis that we need to make is require us to know the dataset precisely. Then, we try to understand how the prediction is made by looking at the similarity metrics that we have used. Next, we talk about how we find k nearest neighbours and then move onto how we make predictions. Last but not least, we terminate the system analysis by looking at how we plan to analyze the temporal aspects of the dataset in the temporal analysis section.

We leave the UML design to the system design section.

4.1 Similarity Metrics

When we are calculating similarity in between users, we will be using the Pearson correlation.

4.1.1 Pearson Correlation

What the Pearson correlation gives us is that it gives the rate of similarity of 2 users. Values ranges in between (-1, 1) where 1 stands for both users act exactly same and -1 stands for both users act exactly opposite. Pearson correlation between two users is calculated as shown in equation (4.1)

$$Pearson\,Correlation \quad \rho_{uv} = \frac{\sum_{i \in I_{uv}} ((r_{ui} - \mu_u) * (r_{vi} - \mu_v))}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2} * \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \mu_v)^2}} \tag{4.1}$$

4.2 K Nearest Neighbors

When making movie prediction with neighborhood-based methods, we use the relationship in between the user and its neighbors. We define the neighbors of a user as the users that have the same taste. This way, when we make predictions, we can utilize nearest neighbors of a user in order to predict movie rating of the user, since all other movies they have watched in common up until the point where we calculate the correlations is quite similar.

4.2.1 Finding K Nearest Neighbors

The k nearest neighbors of a user is found by taking the Pearson correlation of a user with anybody else in the dataset, one by one. In this project, in order to simulate using better methods than Pearson correlation in terms of rmse, we have put a constraint on Pearson correlation that is we only take pearson correlation of two users if they have both rated 5 movies in common. This way, the predictions that we make will be taken from those pairs that already have some kind of relationship because of the inherent constraint in between them.

Since we use the correlations in between users repeatedly, we store the correlations of all users to all other users in a matrix where both rows and columns represents the users, and the values are the correlations between them. For example, first row, second column represents the correlation in between the first user and the second user in the dataset.

Then, whenever we need to find k nearest neighbor of a user, all we need to do is take the row of the user we are interested in, and search for the k biggest correlation values that it has. The users that corresponds to these k biggest correlations are the k nearest neighbors of the user.

4.3 Making Prediction

If we have already found out the k nearest neighbor of the user, making predictions is a quite easy. Whenever we want to predict a movie on the chosen user, we take the predictions of the user's k nearest neighbors on the movie. And by utilizing correlations in between the user and its neighbors, we predict the movie rating of the user. To do this, we take the weighted average of the neighbors' ratings by taking weights as the correlations in between the user and its neighbor. Even though Pearson correlation is good enough itself with datasets, as different users have different rating scale we face with bias. In order to filter bias, we will be making our predictions with a

zero centered approach by scaling down all users ratings by their own average rating.

$$Prediction \quad \hat{r}_{ui} = \mu_u + \frac{\sum_{v=1}^{k} \rho_{uv} * (r_{vi} - \mu_v)}{\sum_{v=1}^{k} \rho_{uv}} \tag{4.2}$$

4.4 Temporal Analysis

When we are making temporal analysis on our dataset, we are in the search for the time intervals that almost matches with the latest trends of the user. This is accomplished by using two different constraints. The aim of these two constraints is to detect the drift that the user is having and later the results of these two to be merged in order to make predictions with high accuracy.

4.4.1 Max Year Constraint

The first way of how we look at the data is by putting a time constraint which represents a virtual system time in which we do not know anything about the future but just the past of the point of the time constraint. If we ever find a time of the system that the user's interest drifts as the latest trend of the user drifts, then we can surely bypass all the ratings given after this date and just consider the ratings up until that point. The reason, we put a maximum time limit is that the user's actions appears to be drifting as time pass by and the drift that has occurred has led the user to act similarly to his-her old actions. We then use this repetitive nature of the actions and make personalized predictions.

4.4.2 Time Bin Constraint

The other way we look at the data is by looking parts of the data instead of long periods of the data. Since the user's latest actions may be related to actions of the user where it does not apparent when we look at the big picture but rather the small sections of the data. If we able to detect time bins which are quite like the user's latest actions, then we can use this data that we have collected from time bins to make predictions on the user. Here the main concern is to detect the right time bin size according to the latest drift so that we can detect similar drifts easily.

4.4.3 Merging Max Year and Time Bin Constraints

The question is even if the max year constraint and time bin constraint has detected the same drift as the latest one the user is having; will the data provide high accuracy

rates? Here we have a backup plan, which combines the results of the both constraints and tries to make predictions with high accuracy. The way we do is we first find max year constraint and time bin constraint independent from each other. Then, we take the max year, lets say 2010 and take the time bin with size of 4, and then we make the predictions on a virtual time limit of the max year and put additional rule to give higher priority to the data that has been found on the last 4 years. This way, we still take the max year limit but also increase the effect of the sections of data where the current trend of the user is so similar.

4.5 Accuracy Metrics

In the evaluation phase of the project we will be using only RMSE as the other researches on the topic. Later, we will be introducing new metrics, after we conclude a result by using RMSE.

RMSE is calculated using equation (4.3).

$$Root\ Mean\ Squared\ Error \quad RMSE = \sqrt{\frac{1}{|TestSet|} \sum_{(u,i) \in TestSet} (\hat{r}_{ui} - r_{ui})^2} \qquad (4.3)$$

5
System Design

In section, we start by designing a basic framework in order us to evaluate the different temporal aspects of the Movie Lens dataset and different datasets if also required. In section 5.1, we will be demonstrating the overall architecture the framework briefly.

As we have discussed in system analysis section, we have 2 different temporal constraints that we assume they show different temporal trends which reflects the mood and the change of the user's life. In section 5.2 and 5.3 we will be examining these temporal constraints and evaluate what are the best hyper parameters in order us to prove that with different temporal trends, we need to provide different recommendations to the user. In section 5.2, the Max Year constraint is aimed to be evaluated and then moving onto the Time Bin constraints in section 5.3.

While we can say its reasonably fast to calculate similarity, find k nearest neighbors and even make prediction, since we have used small version of the Movie Lens dataset, when it comes to evaluate temporal aspects of the dataset, things get tough. In the search of hyper parameters, we have faced with making predictions in the order of hundreds of thousands seconds which would ultimately lead to a runtime with an order of days to weeks by using the allocated personal computer. This problem has led us to provide caching solutions and not repeat almost none of the calculations. In section 5.4, we will be discussing the level of caching, the optimization of the functions that has run in bulk and details of the caching used.

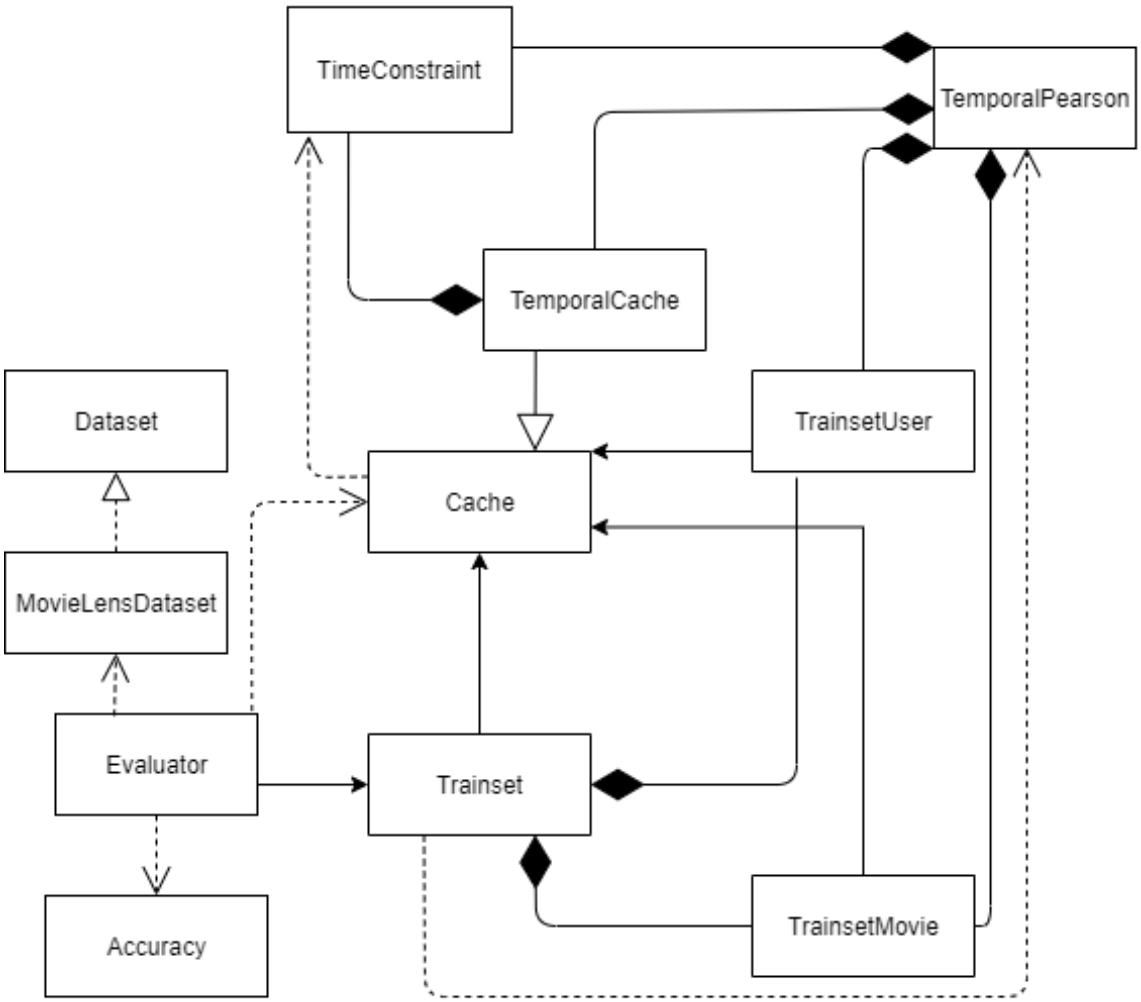

Figure 5.1 UML Class Diagram

5.1 Framework Architecture

We present the UML class diagram (Figure 5.1) of the overall architecture. We have used 8 different classes and 1 interface. In this section, we will be discussing the details of the class diagram in order to understand the architecture better.

Dataset Inteface Interface where it provides basic structure for any dataset to be compatible with other classes (Figure 5.2).

MovieLensDataset Class Implementation of the Dataset class. Loads the Movie Lens dataset (Figure 5.3).

Trainset Class Main class where we take everything required and make predictions (Figure 5.4).

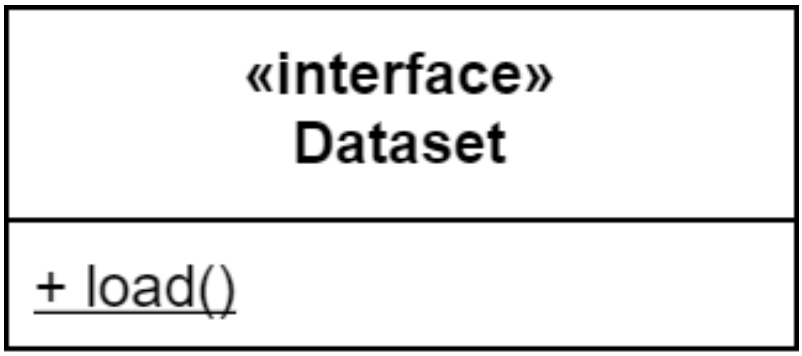

Figure 5.2 Dataset Interface

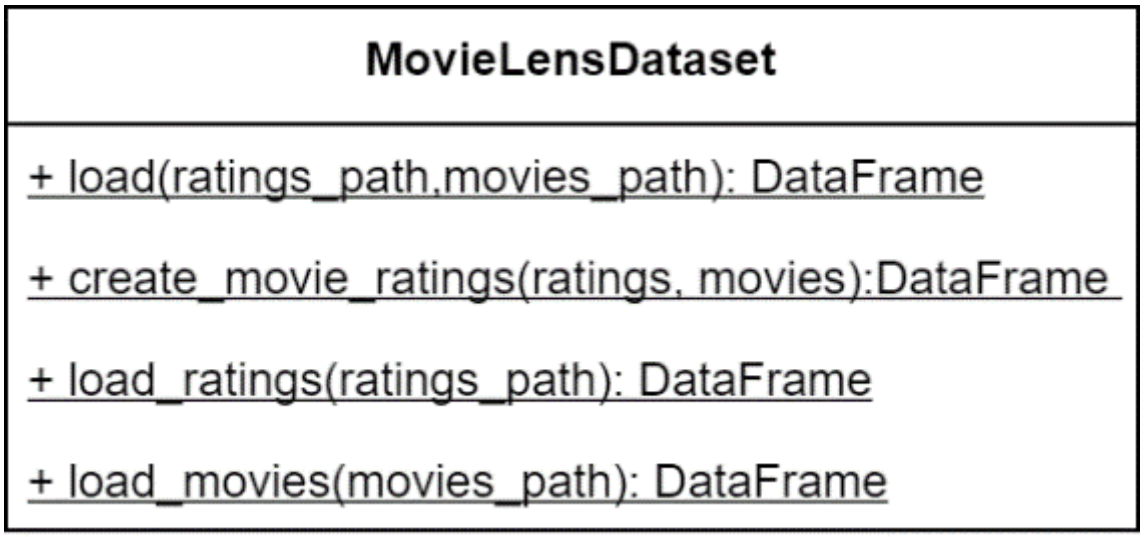

Figure 5.3 MovieLensDataset Class

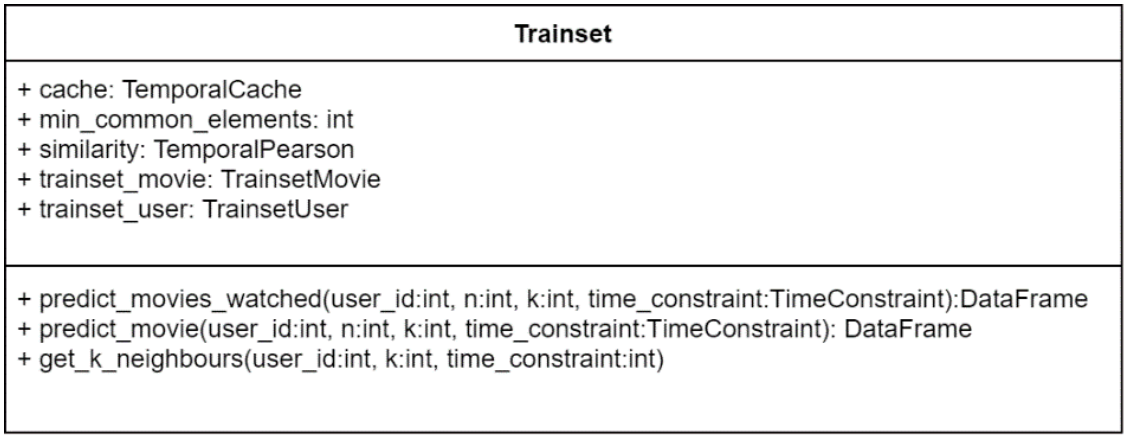

Figure 5.4 Trainset Class

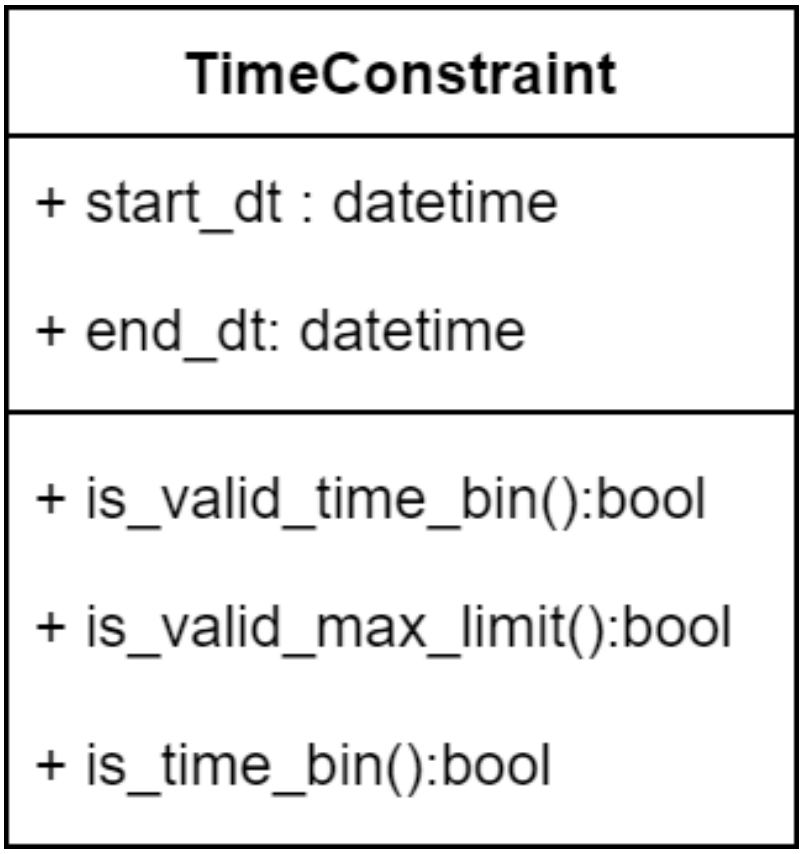

Figure 5.5 TimeConstraint Class

TimeConstraint Class The class which represents 2 types of time constraints on the any process that uses temporal aspects of the datasets. The class provide Max Year and Time Bin time constraints depending on where and how it is used (Figure 5.5).

Cache Class The class where we store data that have high calculation times. Provides basic utilities for caches and superclass of the TemporalCache class (Figure 5.6).

TemporalCache Class The class where we store data that has time constraints on it. This class also provides utilities in order use to store bulk results for those evaluation methods where we bulk run some methods and calculate same correlations over and over (Figure 5.7).

TemporalPearson Class The class where we calculate pearson correlations in between users and if required create caches of these correlations in the given cache parameter (Figure 5.8).

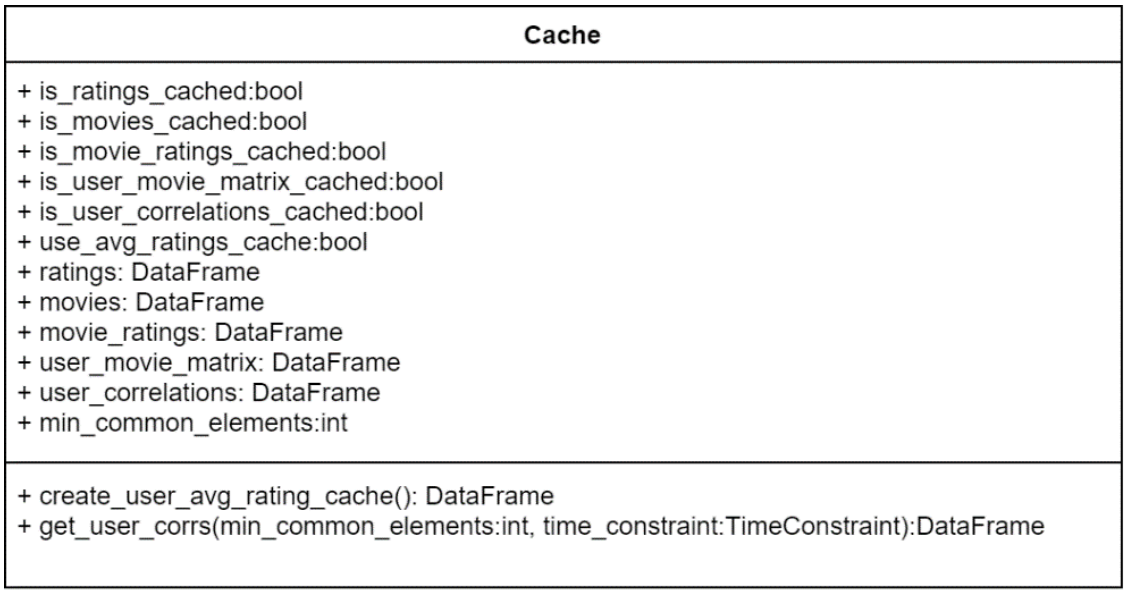

Figure 5.6 Cache Class

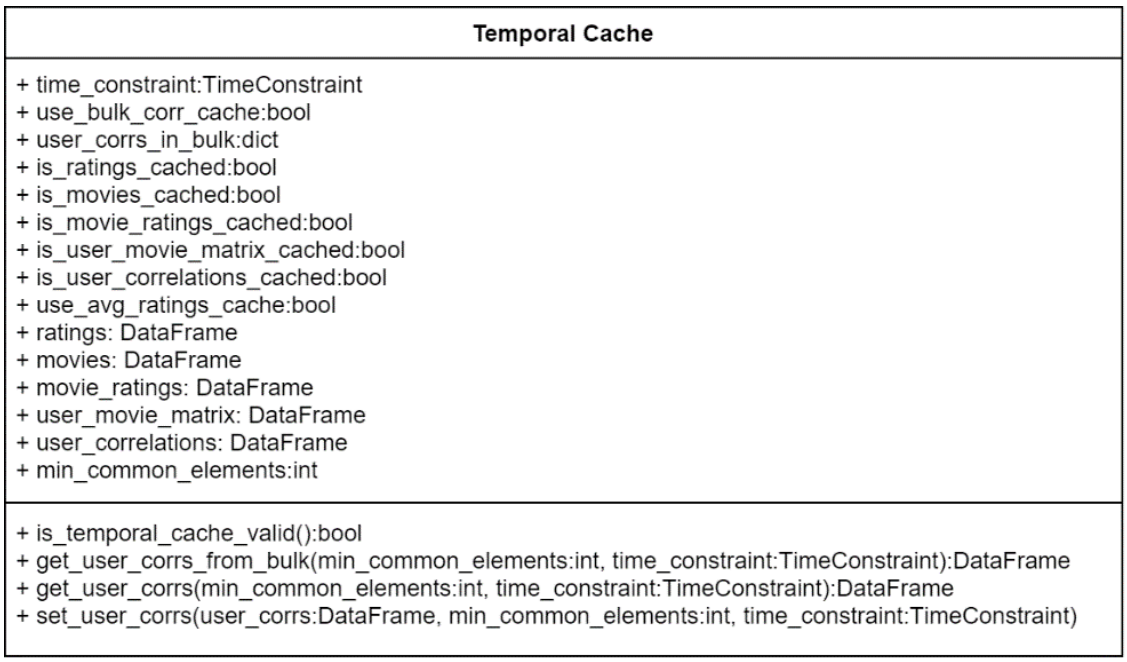

Figure 5.7 TemporalCache Class

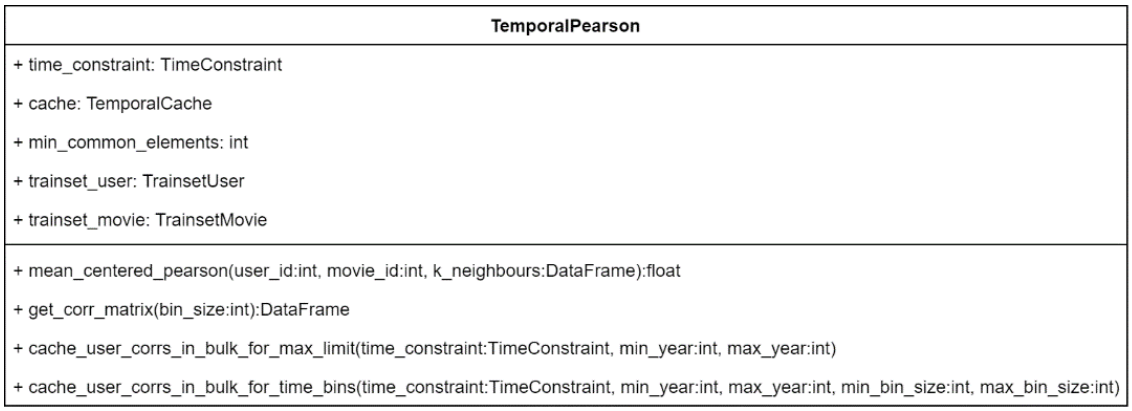

Figure 5.8 TemporalPearson Class

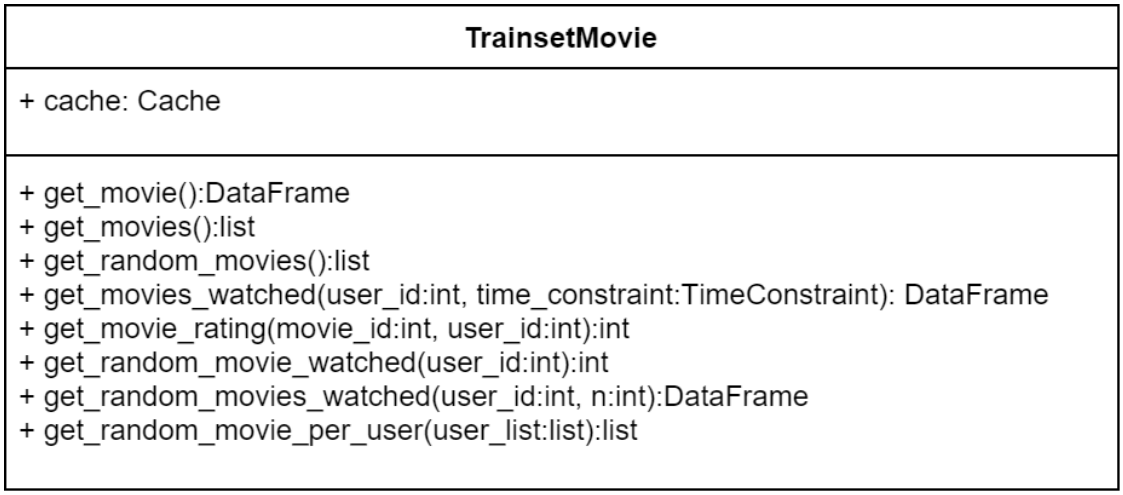

Figure 5.9 TrainsetMovie Class

TrainsetMovie Class The class where we extract data about movies and choose random movies to make rating prediction on it (Figure 5.9).

TrainsetUser Class The class where we extract data about users and choose random users in order us to make predictions on the movies that the users have watched (Figure 5.10).

Accuracy Class The class where we calculate accuracy of the predictions we make (Figure 5.11).

Evaluator Class The class where we evaluate the temporal aspects of the dataset and find hyperparameters (Figure 5.12).

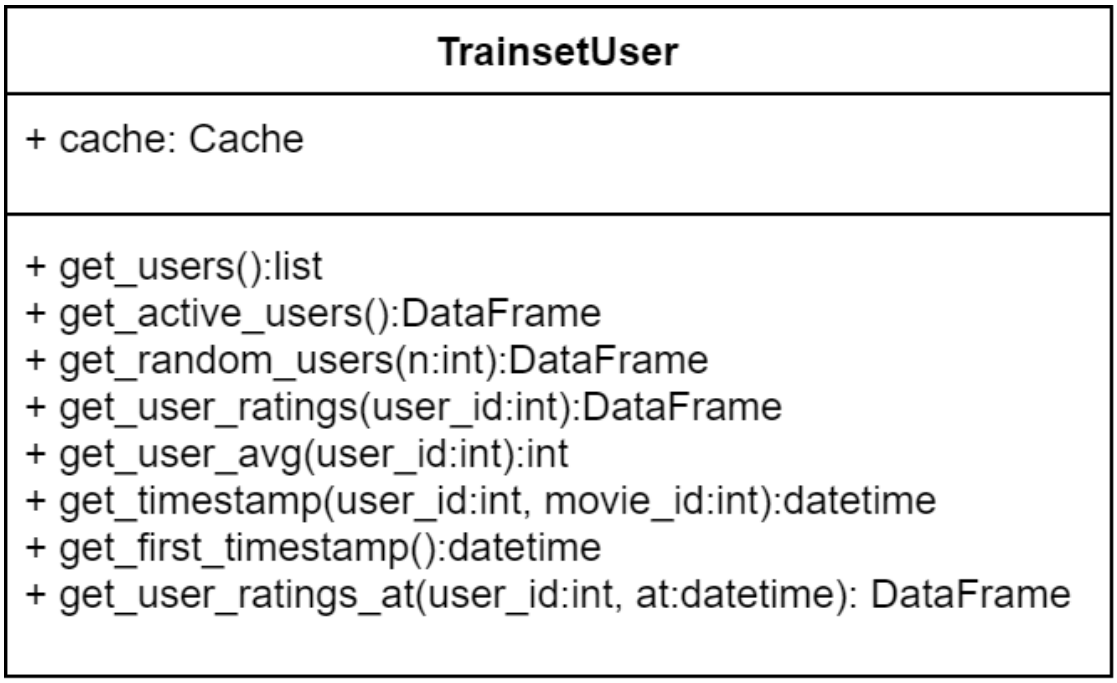

Figure 5.10 TrainsetUser Class

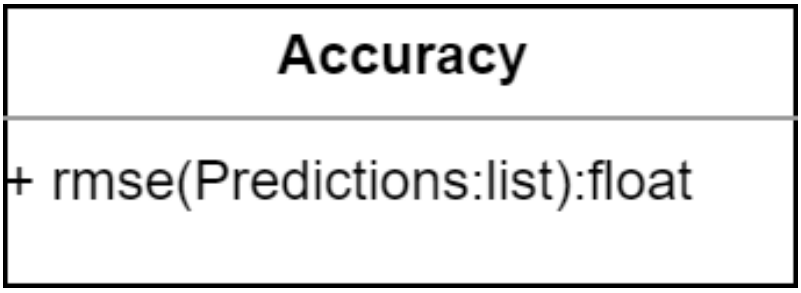

Figure 5.11 Accuracy Class

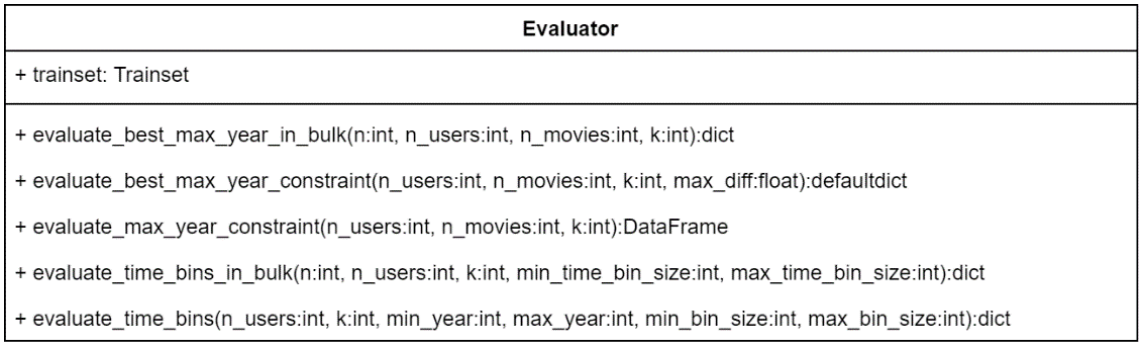

Figure 5.12 Evaluator Class

5.2 Evaluating Time Bins

When we think about human life, repetitive events tend to occur as we live. As these events occur, they make somewhat similar impact on the person who lives it. And we assume that when we feel something similar, we tend to act similar. For example, a person that falls in love in two different time frames may have same feelings and the person may want to watch romantic movies in both time frames. Also, some of the following events tends to have big impact on one's life: Loosing somebody you care, having an accident, having an important exam etc. If we can catch these temporal intervals, we can surely provide better recommendations to the users.

In the search of these hidden repetitive time intervals, we design Time Bin architecture where we look for different time bins and try to find most similar bins and make predictions according to the actions that the user has taken in these time bins.

5.2.1 Time Bins

As we have discussed in the previous sections, *The Time Bin* architecture is about finding the most similar time intervals that have high accuracy and make predictions by using only that intervals data. The way we search these intervals is that we take our Movie Lens dataset and partition it into different time frames. For example, a Time Bin of size 2, starts from the first movie ratings that have given in the dataset and collects data about every partition of size having equal to the time bin. In our dataset, first rating is given at 1996 and last rating given at 2020. We start scanning from 1996-1998, 1998-2000, ..., 2018-2020. But as it can be seen, some of the intervals are gone missing. So here, we shift our starting point by using the shift count formula that has been shown in equation 5.1. In this example, for time bin size of 2, we just shift once and also scan the dataset 1997-1999,1999-2001,...,2017-2019.

$$Shift\ Count\ Formula \quad ShiftCount = BinSize - 1 \tag{5.1}$$

After determining the time bins, we take each time bin and make movie prediction for random users by just using the data that has been collected using this time bin. So when we are finding Pearson correlations in between users, the only data that we use, is the data that has been given in this time bin.

By isolating these time bins, we aim to obtain the accuracy of each time bin so that when we compare a time bin with other bins with different sizes, we can be sure that the ones that have high accuracy, is really related to the users' current trend in this period of time.

5.2.2 Best Time Bins

As we start to create time bins of different sizes, number of bin sizes increase. For example, bin size of 2 only results in 24 bins. As we increase the bin size, we get redundant amount of time bins. In order to decrease the load of the calculations, we need to discard time bins that have low accuracy rates. So, the aim of the best time bin search is to prune the time bins that had low accuracy rates and reveal the ones that had reasonably high accuracy.

The procedure that evaluates and collects data for the purpose of detecting best time bins is shown in algorithm 5.2.2.

1: **procedure** EVALUATE TIME BINS($nusers, k, binSize_{min}, binSize_{max}$)
2: $userList \leftarrow GenerateRandomUsers(nusers)$
3: $userMovieList \leftarrow MatchUsersWithRandomMovieWatched(userList)$
4: $year_{min} \leftarrow GetDatasetMinYear()$
5: $year_{max} \leftarrow GetDatasetMaxYear()$
6: $binResults \leftarrow list()$
7: **for** $binSize \leftarrow binSize_{min}$ to $binSize_{max}$ **do**
8: **for** $shift \leftarrow 0$ to $binSize - 1$ **do**
9: $year_{curr} \leftarrow year_{min} + shift$
10: $predictions \leftarrow list()$
11: **while** $year_{curr} + binSize \leq \text{year}_{max}$ **do**
12: **for** $userId, movieId$ in $userMovieList$ **do**
13: $tc \leftarrow TimeConstraint(year_{curr}, binSize + year_{curr})$
14: $prediction \leftarrow Predict(movieId, userId, k, tc)$
15: $actual \leftarrow GetMovieRating(movieId, userId)$
16: $predictions.append(prediction, actual)$
17: **end for**
18: $year_{curr} \leftarrow binSize + year_{curr}$
19: **end while**
20: $binResults.append(binSize, predictions)$
21: **end for**
22: **end for**
23: **return** $binResults$
24: **end procedure**

5.3 Evaluating Max Years

In Time Bin architecture, we take the most similar time bins and give recommendations from that time bin only. But similarity does not always come with high accuracy. In order to get higher accuracy, we look for a max year for our time bin to locate the best time interval out of all years of the dataset. For example, lets say for a random user x, we have found that time bin of size 6 gives most accurate results for this user's current mood. But where does the best time bin is located in whole timeline of the dataset? Is it in in between 2000 to 2006 or 2007 to 2013. Which one of the time bins with size 6? Here, max year calculation comes to rescue us and we use a max year, lets say 2013 (which has been found after Max Year calculation), then we can make sure the time bin is the one that is in between 2007 and 2013. We also move one step ahead and not just take the interval in between 2007 to 2013 but all ratings from start up until this max year(2013 in this case), but we apply some kind of decay to this interval by giving higher chance of effecting by giving higher priority to the ratings that have been given in the 2007-2013 period.

5.3.1 Best Max Year

Best Max Year calculation is made by taking n random users and calculating accuracy on the movies that the selected user watched before. When calculating accuracy, we first put no constraint but just take the accuracy, later we put a time constraint and calculate the accuracy. If the difference of the accuracy of the both results are lower than some threshold value, then we add one vote for this year to be best. This voting mechanism is repeated by putting every year in between first and last rating timestamp and determined best years.

The procedure that finds the best max year constraint is shown in algorithm 5.3.1.

```
1:  procedure FIND BEST MAX YEAR(nusers, nmovies, k, diff_max)
2:      userList ← GenerateRandomUsers(nusers)
3:      noConstraint ← dictionary()
4:      for userid in userList do
5:          rmse ← PredictMoviesWatched(userid, nmovies, k)
6:          noConstraint[userid] ← rmse
7:      end for
8:      minYear ← GetDatasetMinYear()
9:      maxYear ← GetDatasetMaxYear()
10:     votesForYears ← dictionary()
11:     for year ← minYear to MaxYear do
12:         timeconstraint ← TimeConstraint(year)
13:         for userid in userList do
14:             rmse ← PredictMoviesWatched(userid, nmovies, k, timeconstraint)
15:             if abs(rmse − noConstraint[userid]) ≤ diff_max then
16:                 votesForYears[year] ← votesForYears[year] + 1
17:             end if
18:         end for
19:     end for
20:     return votesForYears
21: end procedure
```

5.3.2 Max Year

Once we calculate the best max year, and apply the max year constraint, we can be sure that the prediction accuracy any calculation will be somewhat similar to those predictions that has been given without any time constraint. From this point on, by combining best time bin size with best max year, we can make predictions and evaluate the results.

The procedure that evaluates the max year constraint is shown in algorithm 5.3.2.

```
1:  procedure EVALUATE MAX YEAR CONSTRAINT(nusers, nmovies, k, timeconstraint)
2:      list ← list()                          ▷ Init a list where we can store errors
3:      for i ← 1 to nusers do                 ▷ Iterate over random users
4:          userid ← RandomIntInRange(1, 610)
5:          rmse ← PredictMoviesWatched(userid, nmovies, k)
6:          rmse2 ← PredictMoviesWatched(userid, nmovies, k, timeconstraint)
7:          list.Append(rmse, rmse2)           ▷ Append result to the list
8:      end for
9:      return list
10: end procedure
```

5.4 Optimizations

When it comes to calculate best time bin and best max year, the normal techniques that we use to find predictions becomes almost impossible to apply. Since the calculation of user correlations in between users takes order of seconds, when we try to make predictions, even a few of them are taking runtime at the order of minutes.

In the evaluation of the time bins and max year we have set the minimum number of iteration as 1000 so that all calculations have to repeat 1000 times.

Since the max year calculation consists of making movie prediction on n users and taking m number of movies, number of prediction per iteration becomes m * n where in our case n = 10, m = 10. So with these specification, in total, we have to make 100.000 movie predictions.

With the best time bin calculations, time bins with size in between 2 to 9 is is being inspected. Since we scan the whole time interval that has been taken movie rating in database, in total we have to predict more than 200 time bins in each of which we predict movies on n users and m number of movies, in our case n=100, m=10, which leads us to more than 200.000 movie predictions.

As we can see, while normal predictions takes at the order of minutes, when we need to validate the hyper parameters, we need days.

The problem is being solved by using parallel processing and caching.

5.4.1 Caching

In order to overcome these problems that we have just mentioned, we have come up with the idea of using caching where storing each user correlations in between users. Whenever we make a bulk calculation, we first calculate the possible user correlations matrix and store the result into cache so that in the upcoming bulk calculation, we never calculate the user correlations again.

5.4.2 Caching on Time Bins and Max Year

Caching on time bins and max year cases requires us to use different caching scheme since the max year calculation requires only cache for number of years in between first and last year of the dataset which is 24 in our case, time bin calculation requires much more number of cache stored in our case more than 200.

When we make best year calculations, we first calculate user correlations by putting time constraint one by one in our case starting from 1996 up until 2020. We store all the user correlations in the cache and any further k nearest neighbor calculations is carried out by using these caches.

In time bin case, we first determine the time bins that needs to be calculated and according to that we calculate each of the user correlations matrix by putting time constraints of length that is equal to the time bin size, and store all the results inside the cache.

This way, from this point on, all calculations are being carried out by using stored cache, and each calculation takes order of milliseconds instead of seconds or minutes. This way, we have reduced to time required from order of hundreds of thousands seconds into a few thousands seconds.

5.5 Database Design

In this research, our main concern is to provide proofs for our analysis rather than building a full-fledged application. That is why we do not need to design a database but rather take small parts of our dataset and work on it. This gives us flexibility to make fast analysis and lower the complexity by not using a database.

5.6 Input-Output Design

As the input of our project, we are using readily available Movie Lens dataset.

Since our research is more scientific analysis oriented, there is no specific output to extract out of this project but rather scientific evaluation results that uncovers the temporal drift of the users. That is why, we do not have any output design.

6
Application

In this section, we have included some plots of the results that we have taken from our analysis and evaluate the temporal ascepts of our findings.

6.1 Evaluation on DataSet

In this section, in particular we evaluate the time bins, max years and their similarity to each other in their criterion. Before proceeding further, we should remind ourselves the fact that we do not exactly look for personalized results at this stage as we have discussed in our road map. First and foremost, we seek general answers and intuitions that could lead us to more appropriate measures that we can take on the users so that could possibly result in high accuracy predictions. That is why the analysis on time bins and max years in this section is comprised of whole dataset rather than particular users.

In any evaluation we make in this section, we use the results that have been collected upon running the related algorithm 1000 times. In each of these runs, we use different set of random users so that the results that have collected is scattered upon the dataset rather than specific to some sample of it.

6.1.1 Evaluation on Time Bins

When we think about time bins, the first question we come up with is what should be the size of the time bins that we search for? That is why we start by investigating the appropriate bin sizes for the whole dataset.

As we see from the Figure 6.1, as the time bin size increase, the error rate tends to go lower except some consecutive intervals where the error rate stays so close or almost same to each other.

When we take a look at the Figure 6.2, we see the similarity of time bin sizes that has

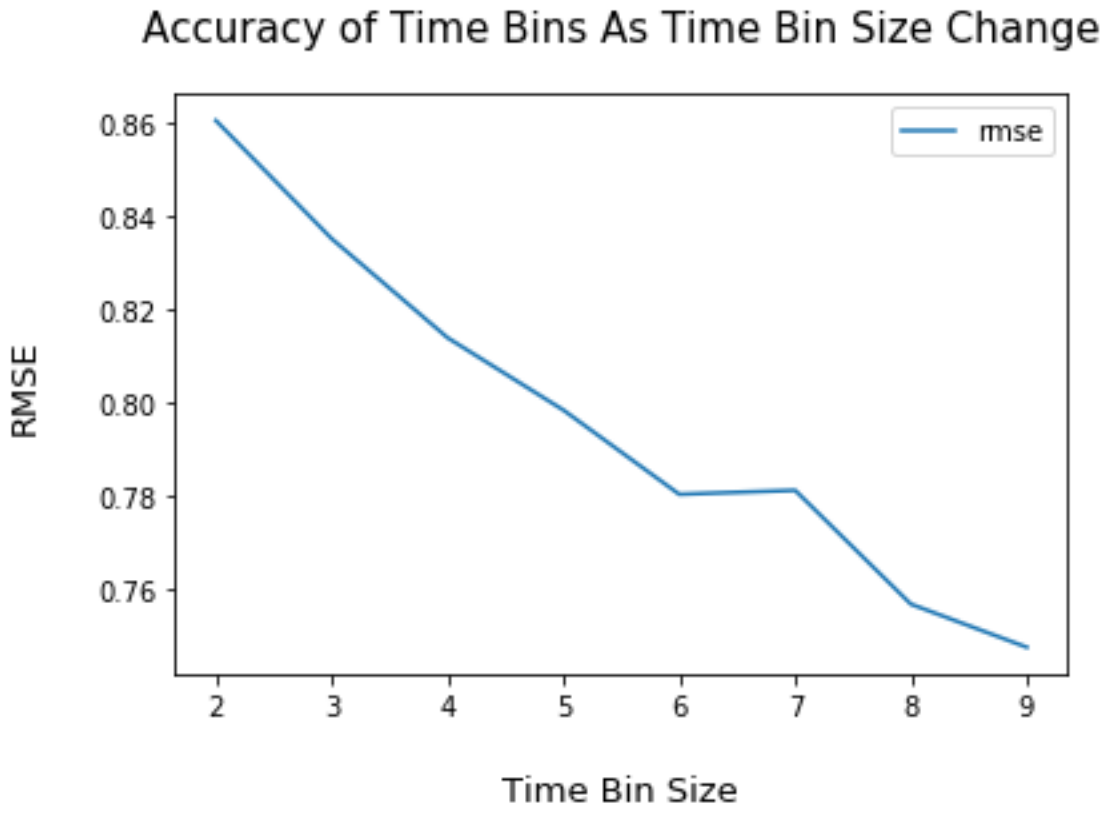

Figure 6.1 Comparing Accuracy Time Bin Sizes

been calculated using Pearson correlation. The first point that catches our eyes is that low time bin sizes tend to similar to time bins with high sizes. Another remarkable point we can see is that the time bin of size 6 quite similar to time bin of size 9.

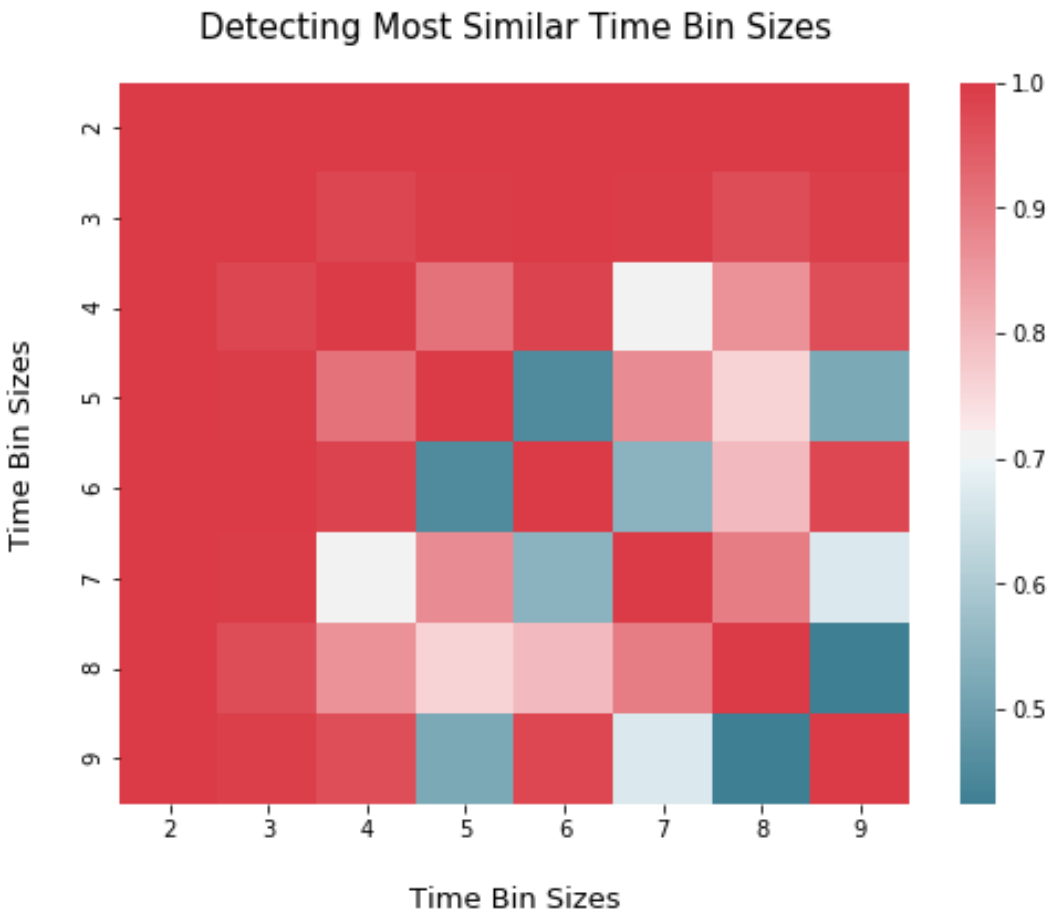

Figure 6.2 Similarity of Time Bins (Pearson Correlations Ranging From 0 to 1)

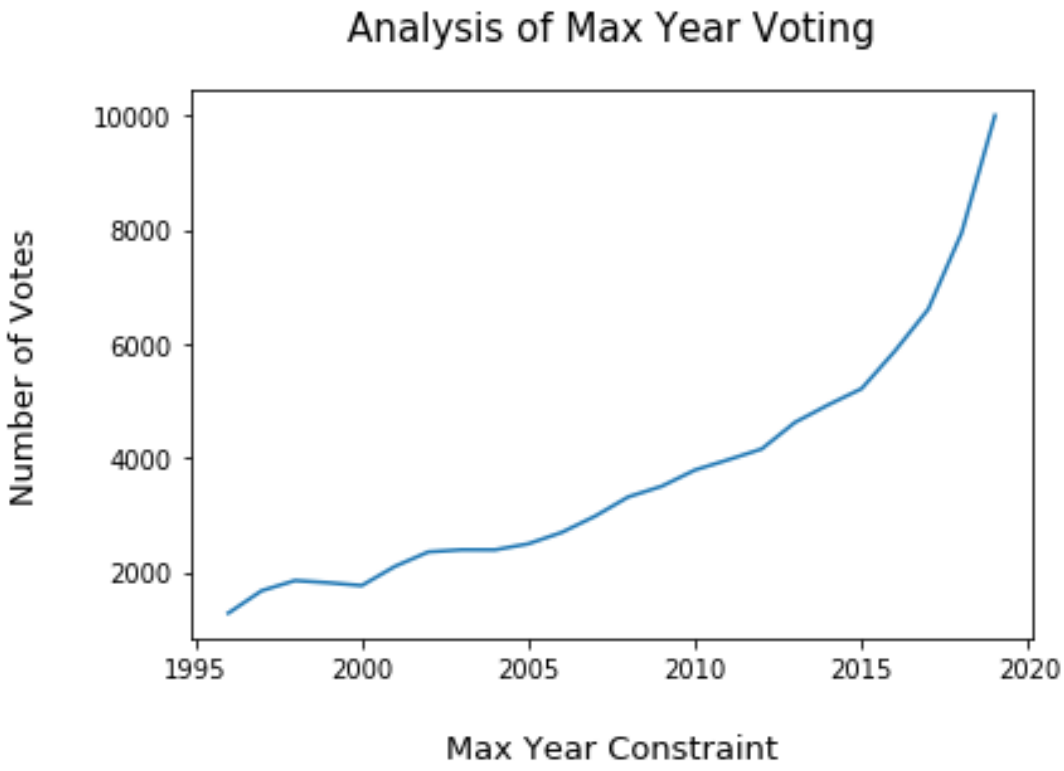

Figure 6.3 Results of Max Year Voting

6.1.2 Evaluation on Max Years

In the search for the best max year constraints, we have decided to make a voting on max years and evaluate the candidate max years that has been taken high number of votes in the voting but also has similar characteristic to most other candidates so that when it has been placed as the max year to our dataset, it provides a high level of accuracy on different ranges of years rather providing quite high accuracy in some sections and having low accuracy in other sections.

The results of the voting is measured as shown in the Figure 6.3. The results we see in the Figure 6.3 are kind of expected since as we get more and more data about users, we get better and better at making recommendations.

When we take a closer look the results of each max year constraint, we get the figure 6.4. From this figure, We can clearly see that as the max year constraint is getting closer to the maximum year(in this case 2019), the number of votes the year gets per run gets higher and higher. But interestingly enough, the results of 2013-2014 is seem to have quite have a high similarity to years like 2017 and 2018.

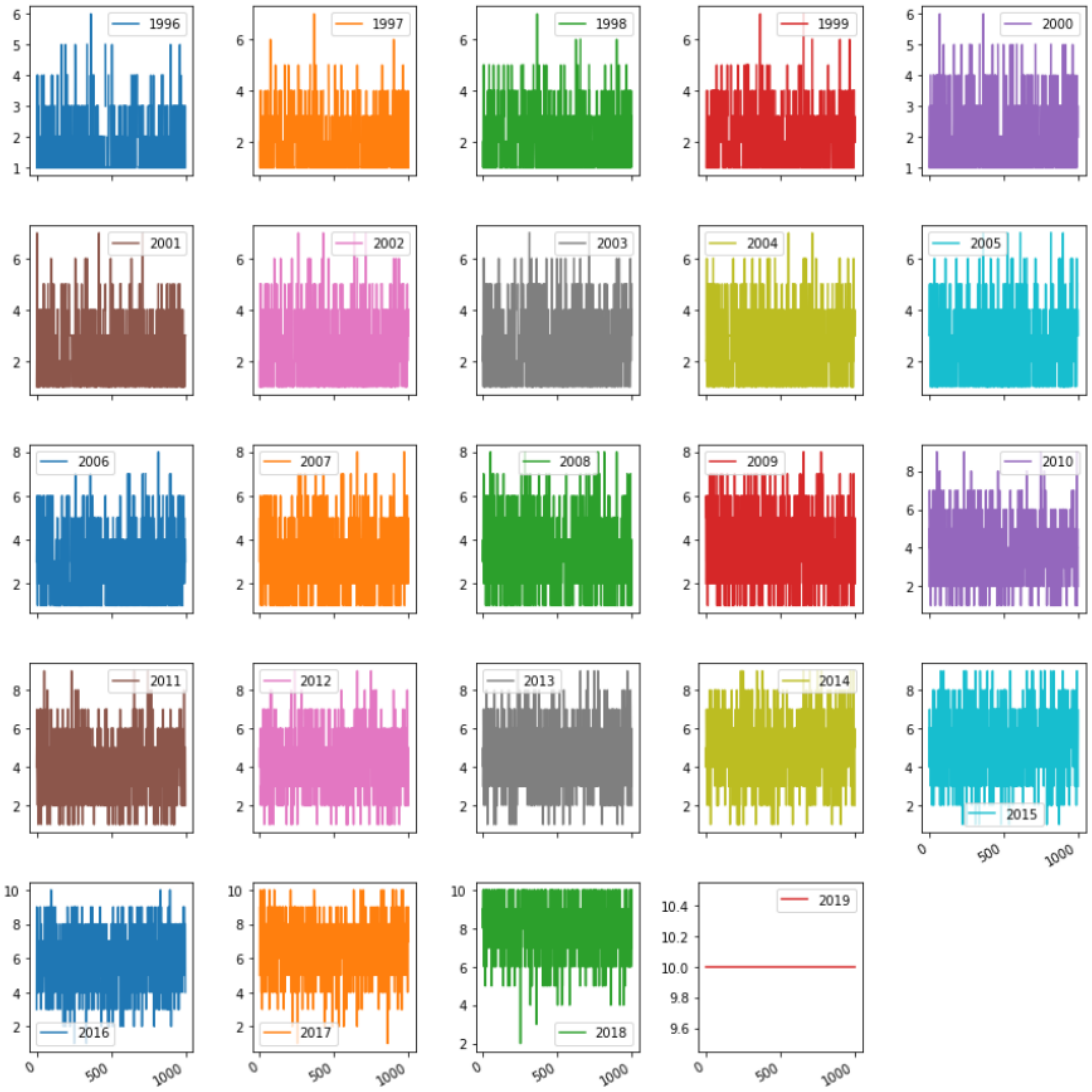

Figure 6.4 Average Votes For Each Max Year Candidate

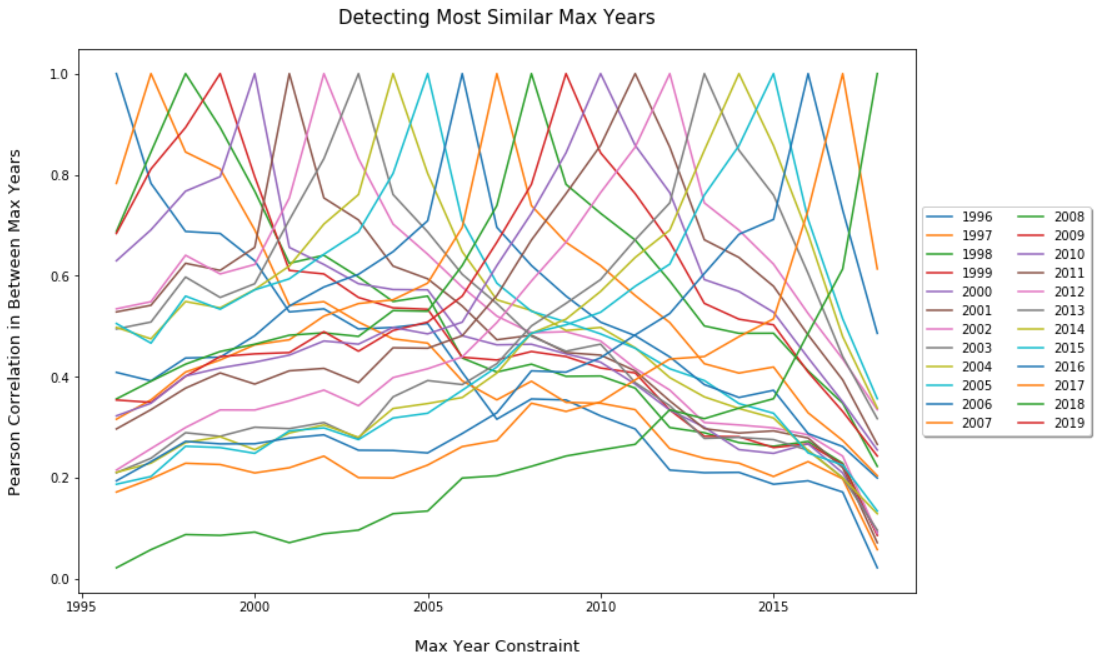

Figure 6.5 Detecting Most Similar Max Year Candidates

Last but not least, we study on Figure 6.5 in order to better understand the similarity between the max year candidates. Even though the figure seems to be complicated at first sight, it actually is not as complicated as it seems. In this figure, we have the years as the x axis, and the similarity in between any point as the y axis. Simply we choose a max year from x axis and a color that is also representing another max year from the figure, and then the y values of the intersection of x axis and the color shows us the Pearson correlation in between both of them. As expected, every year has a Pearson correlation of 1 with itself that is why the figure seem to have lots of triangular roofs.

Here we are searching for max years that have high correlations in general and also if possible, the range that the max year have should consist of high correlations. As in the case of Figure 6.4, the 2013 and 2014 is seem to be qualified for our analysis.

6.2 Interpreting The Findings

Here we want to sum up the outcomes of the evaluation on time bins and max years. And, we also try to interpret the outcomes we get out of our evaluations.

In the evaluation of max bin sizes, the results were quite similar to each other and we could not exactly get a winner with clear evidences. But, we are still able to intuitively approximate the winner to time bin size of 6.

Like the ambiguity of the best time bin size, the max year candidate is also seem to be kind of ambiguous since we could not get any distinctive results. Again, we are able to intuitively approximate the winner of max year constraint voting as the 2014.

As a result of our evaluation, we could approximately say that, when we try to make personal recommendations, if we ever take 2014 as the max year constraint and time bin size of 6 as the high importance part of the time scale, we can pretty much provide as high accuracy as possible on average. But of course these values will not reflect any best point for users, but rather when we take average on whole dataset.

References

[1] N. Polatidis and C. K. Georgiadis, "A multi-level collaborative filtering method that improves recommendations," *Expert Systems with Applications*, vol. 48, pp. 100–110, 2016, ISSN: 09574174. DOI: 10.1016/j.eswa.2015.11.023.

[2] Y. Koren and R. Bell, "Advances in collaborative filtering," *Recommender Systems Handbook, Second Edition*, pp. 77–118, 2015. DOI: 10.1007/978-1-4899-7637-6_3.

[3] R. M. Bell and Y. Koren, "Lessons from the Netflix prize challenge," *ACM SIGKDD Explorations Newsletter*, vol. 9, no. 2, p. 75, 2007, ISSN: 19310145. DOI: 10.1145/1345448.1345465.

[4] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 426–434, 2008. DOI: 10.1145/1401890.1401944.

[5] G. Dror, N. Koenigstein, and Y. Koren, "Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy," *RecSys'11 - Proceedings of the 5th ACM Conference on Recommender Systems*, pp. 165–172, 2011. DOI: 10.1145/2043932.2043964.

[6] Y. Koren, *Collaborative Filtering with Temporal Dynamics*. ACM, 2009, p. 1406, ISBN: 9781450355520.

[7] N. Lathia, S. Hailes, L. Capra, and X. Amatriain, "Temporal diversity in recommender systems," *SIGIR 2010 Proceedings - 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 210–217, 2010.

[8] F. Ye and J. Eskenazi, "Feature-based matrix factorization via long- and short-term interaction," *Knowl. Eng. Manag*, pp. 473–484, 2014.

[9] I. A. A. Q. Al-Hadi, N. M. Sharef, M. N. Sulaiman, and N. Mustapha, "Review of the temporal recommendation system with matrix factorization," *International Journal of Innovative Computing, Information and Control*, vol. 13, no. 5, pp. 1579–1594, 2017, ISSN: 13494198.

[10] A. Töscher, A. Graz, and R. Legenstein, "Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems,"

[11] N. Developers. (2020). Bsd 3-clause, [Online]. Available: https://github.com/numpy/numpy/blob/master/LICENSE.txt (visited on 04/18/2020).

[12] P. D. Team. (2020). Bsd 3-clause, [Online]. Available: https://github.com/pandas-dev/pandas/blob/master/LICENSE (visited on 04/18/2020).

Curriculum Vitae

FIRST MEMBER

Name-Surname: Mustafa Katipoğlu
Birthdate and Place of Birth: 15.07.1998, Hatay
E-mail: mustafakatipoglu98@hotmail.com
Phone: 0538 529 48 10

Project System Informations

System and Software: Windows İşletim Sistemi, Python
Required RAM: 16GB
Required Disk: 256GB