Movie Recommendation System

Anas Djebbari

2852680

Submitted in partial fulfilment for the degree of
B.Sc. in Computing Science

Griffith College Dublin

April, 2017

Under the supervision of Eoin Carroll

Disclaimer

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the Degree of Bachelor of Science in Computing at Griffith College Dublin, is entirely my own work and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed: _____ **Date:** 17th April 2017

Acknowledgement

I would like to sincerely thank my supervisor Eoin Carroll for all his help, support and patience throughout this project. I would also like to thank all the lecturing staff for the past four years of guidance special thanks to Faheem.

My parents whom without all of this wouldn't be possible thank you for the support and patience. I would especially like to thank my family for their love and encouragement. Thank you, finally, to all my friends for their backing and friendship over the years. Cesar & Haider thank you for the great memories and laughs, you made my college experience an unforgettable one.

AJay, thank you for the support, faith and always seeing the good in me, thank you for being there for me during the tough times. You are the reason I am who I am.

A special thanks to my good friend and brother Abe, we come a long way from when we were kids, love you man.

Thank you all, I am today because of you.

Table of Contents

List of Equations

List of Figures

Abstract

Recommendation systems are becoming more and more popular, with every major website incorporating them into there system, they are at an all time high. With this in mind, giving users quick but accurate recommendations, is more important than ever.
This document describes a movie recommender system that is designed to provide users with recommendations to help them decide what movie(s) to watch next.
It will look at previous and existing solutions and assess each system, and use this information in order to formulate a new app to help users get movie recommendations.
These recommendations will be achieved using a sophisticated maths algorithm.

Chapter 1 Introduction

1.1. Recommendation System

This project is a movie recommendation system, which would, with the help of mathematical equations give a list of movies which the user might enjoy to watch. At times we always find ourselves wanting to watch a movie, end up asking friends and family for suggestions, the goal of this app is to give the user a choice of movies which they might want to watch. How these movie recommendation are done is by using complex maths equations to try and identify which movies the user will most likely enjoy and give a high rating to. The project is an easy to navigate app which would allow the user to search for movies, rate movies and get movie recommendations based on certain criteria.

1.2 Project Aim

The aim of this project is to develop an recommendation system that can learn about its users and give them a set of movies that they will most likely enjoy. The aim is to provide both user based recommendation and movie based recommendation to the user. The goal is to have an easy to navigate application which allows users to navigate effortlessly.

1.3 Overview Approach

This project will retrieve data from SQL, convert the data to panda data frame then apply pearson's r. The system will be using pearson's r to calculate the correlation between movies, this is important, it's due to this correlation that we will be making recommendations. This project will be python based, so the majority of the calculation will be in python. It will be using Sql to store and retrieve data.

1.4 Software Development

The application will be developed using a Rapid Application Development (RAD) system development approach. Furthermore, it is predicated that the requirements of the project will change over the development of the project so by adopting a RAD approach, it is easier to change requirements with very minimal disruption to the final product. Additionally, as the approach makes use of prototyping, the app can be extensively adjusted and polished before being finalised.

1.5 Document Structure

The rest of this document is as follows. Chapter Two provides a literature review of the area of recommendation system and the sources consulted in accomplishing my project, in addition to related work. Chapter Three describes the methodology and high–level design of my project structure. In Chapter Four, I discuss the system design, and requirements/ specifications including any hardware and software used. Chapter Five provides the implementation details of my project/system/etc. In Chapter Six, details of the working prototype of my project/system/etc are provided, including my testing and evaluation technique. I also discuss results including any revisions to the overall design and implementation that were deemed necessary. Finally, Chapter Seven presents conclusions and future work.

Chapter 2 Background

This part of the report will aim to summarise the work carried out in the background research period of the project development. This will include my understanding of the problem which this project is trying to solve. It also looks at what solutions already exist, and what research has been done on this area. Also looking at tools which can be implemented into the project.

2.1 Literature Review

To begin with, some research was conducted when looking at what algorithms to use when we start implementing our recommendations. Understanding how the algorithm worked on paper was crucial for the development of this project. Some interviews were also set up with friends who study statistics were sought in order to better my understanding on how these algorithms worked.

Since learning a new frame work, the book “Flask Web Development”, help in the understanding and how to implement certain things. This was great asset in building the web app we had built.

Also reading a master paper on recommender systems gave a vivid idea on how they really worked.

Another resource which I also used was youtube, which helped me to understand how to work out the maths formulas, how to use python as this was a new technology. A few flask videos were also visited. Actually the author of the book mentioned below has a detailed youtube video which was helpful to get a good foundation.

2.2 Related Work

Of course, when ever the topic of movie recommendations is brought up, Netflix is mentioned. Netflix's movie recommendations algorithm is without a doubt one of the best, and does exactly what its advertised to do. Netflix conceal the algorithms that they used for how they recommend movies to its users, which is understandable seeing that they offer a million US dollars for anyone who improves on it.

Netflix uses two approaches in order to recommend movies, [1]**Content-Based-Filtering** and [2]**Collaborative-Based-Filtering**. The main difference between Netflix's app and mine is that, we are mainly concentrating on the content based filtering aspect. Netflix to disclose any of what they use or how their system works which is understandable, their system is great and works better than advertised, I would like to maybe try and recreate the functionality from testing their system and see if I get the similar results.

My web application will be recommending movies to users based on users likes and dislikes just as Netflix does. Additionally, the system is going to be a looking at how to implement a recommendation on a smaller scale than how Netflix does it.

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.130.4520&rep=rep1&type=pdf [Accessed November 2016]

[2] http://www.fxpal.com/publications/FXPAL-PR-06-383.pdf [Accessed November 2016]

Chapter 3 Methodology

This section looks at the technologies which were used and why they were chosen of others. This will briefly explain what was needed in order to implement this project.

3.1 Algorithm

When looking into what algorithm I would use in my web application there were endless choices including the [3]**Euclidean Distance** and [4]**Pearson's r Correlation** which kept popping up.
As a starting point I built prototypes for each algorithm in python and ran them separately using the same data sets, most of the time they gave very similar results but as I changed the data sets around I started to notice changes. The algorithms giving me different results didn't make any sense at the beginning yet with research I found out that pearson's r tends to give better results in situations where the data isn't well normalised, and given that my dataset isn't very well normalised, this made my choice easier. The Pearson's r formula is more complicated than the Euclidean distance, but accuracy is more important.

3.2 Language

I started the development using **PHP** and **R**, seemed like the obvious choices at the time. During development, certain things were giving me problems, I realised early on that integrating PHP and R wouldn't be the easiest task. From research point of view, there wasn't a lot of material on the integration of these languages. Reason why I didn't use PHP on its own is because it didn't offer much in terms of mathematical support, which is needed for the core functionality of the web app. This leads to more research, to find a language which would be able to do analytical analysis on the data I collected. I found **Python** was very popular in the world of data analytics, also offering great support for mathematical calculations.

---

[3] - J.C. Gower Euclidean distance geometry Math. Sci., 7 (1982), pp. 1-14 [Accessed January 2017]

[4] https://www.researchgate.net/publication/277324930_Pearson%27s_Product-Moment_Correlation_Sample_Analysis [Accessed January 2017]

3.4 Panda

The reason [5]**Panda** is chosen for this project is as the project progressed there was limitation to how the data could be formatted, so panda was one of the solutions which was used. More detail will be discussed in chapter 4.

3.5 Framework

Now that I decided to use Python I needed something to project my calculations onto html, [6]**Flask** and [7]**Django** where two frameworks which came up quite a bit. I had never used any of these frameworks in the past so I would have a major learning curve, but it would be worth the investment in the long run when actually incorporating the algorithm from python.

The main difference between these two was that flask doesn't offer much fancy add-ons when compared to Django, but for the application we are building I don't need any of there add-ons, it was a trade off that didn't have much of an impact.

Flask was easier to start with and for someone who didn't know much about python it was a great way to build my app and also learn python along the way. Also flask offers more control of the components I would use (such as type of databases I want to use and how I want to interact with them). Also flask is very adaptable to changing requirements, which was a good fit with the RAD approach, as mention in chapter one, which was being used to develop this application.

[5] http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html [Accessed March 2017]

[6] http://flask.pocoo.org [Accessed December 2017]

[7] https://www.djangoproject.com [Accessed December 2017]

Chapter 4 System Design And Specifications

This section of the report will be looking at how the system was designed. It will discuss how the data was collected and stored. It will discuss the tools which I used throughout out development. Tools which are used to get the data to appear in a certain format. We look at the languages that were chosen and why they were chosen. Essentially what the app is doing in the background is as follows:

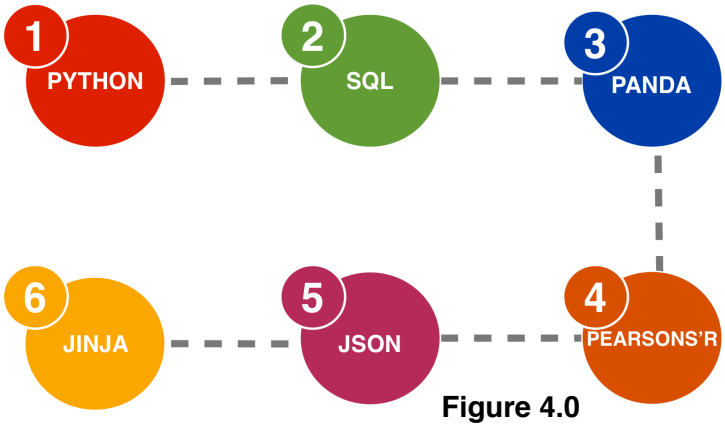

Figure 4.0

- Python is going to collect data from the SQL database,
- Data is then converted into panda data frame (matrix),
- Pearson's Algorithm is then applied to the matrix,
- More Data is then collected for each of the results from a JSON file,
- Once data is collected it is displayed on HTML using JINJA templating.

4.1 Language

It is important to choose a programming language which would offer the most for the development of the app. The main aspects which were considered when choosing a language were:

- how familiar I was with that language. Given the limited time I had for this project, choosing a new language would be time consuming but also a risk of not completing this project in time.

- The ability to work with the data formats which I will be passing it, (primarily SQL and Json files).
- Also the compatibility of this language (mainly when passing variables to **JavaScript**).

After some research, based on the above aspects, it decided to use python as the primary language to build this app. Though, python was new to me, yet not absolutely starting from scratch, I have a background in Java, some of the same concepts would carry over. In terms of data science it seemed like the most appropriate choice, offering extensive libraries in order to achieve the calculations that are needed for a recommendation system.

4.1.1 Version Selection:

The initial development began with python 3.5, reason being that was the version I initially started building my python prototypes. It was decided to keep this version as there was no reason to change, everything worked fluidly.

4.2 Panda

	MOV 1	MOV 2	MOV 3	MOV 4	MOV 5	MOV 6	MOV ...
USER 1	4	1	0	4	4	0	...
USER 2	2	0	0	4	1	1	...
USER 3	5	4	0	3	5	3	...
USER 4	1	0	2	2	3	3	...
USER 5	5	0	5	0	0	0	...
USER 6	3	1	0	4	0	1	...
USER ...	...	...	...	...	...	...	...

Figure 4.2

This is a data frame which is a library written for python, it is for data manipulation data and analysis. It offers data structure and operations for applying mathematics. It is actually very popular in the world of data science. Originally I really I wanted to use R as it was one of the best languages for data analysis, this is similar with panda, each row of these grids correspond to a value and a measurement, which each of the column is a vector which holds the data for a specific variable. Unlike **SQL**, this means a data frame's row doesn't need to actually contain, but it can contain, the same type of values.

This is going to be great for when we need to execute calculations, because we need the data frame to be a certain format which SQL doesn't support. The main motivation for choosing Pandas is to get a format that looks like the table above, and with panda this can

be easily achieved, where as with SQL this is a much bigger task. This is what it looked as follows:

```
M1 = ratings2.pivot_table(index=['user_id'],columns=['movie_id'],values='ratings')
```

This give back a matrix of users and movie ratings, meaning we convert the ratings2 (panda data frame which I collected from the ratings table in SQL) to a matrix with a user per row and a movie per column.

4.3 Data Set

First thing first I needed to collect data before starting anything, So I found a website known as **GROUP LENS** which is an open source website which contains 100 thousand ratings from 968 users on 9126 movies. This data is going to be the main component when starting the implementation of the web app.

The dataset provided was in CSV format. Before I could go any further I first had to do some data manipulation while it was in CSV. The data looked as follows:

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

Excel has a simple column splitting, reason for this is because we will be needing the year a movie was released of the course of the project development. Once the data was split, it was as easy as creating an SQL database and importing the data I collected. Now that I had all that data on movies, I come to realise that it only had information based on the movies, the only information available for the user was a user id. I needed more data on the user, I came across another data set in group lens that had information (user id, age, sex, occupation, zip code) based on the users who rated the movies, the drawback was that the movies didn't have any genres which I will also need for the project.

The solution was to keep tables I already had and create and add a new table called users and copy that data from the new dataset in, so now the primary key from the users table will link to the users id from the ratings table which already existed, now we have a bit more information about the users who rated the movie.
Now that we have all of our datasets and tables how we want them and are almost ready to be used, almost, because we still need to give this users names and passwords in order

for us to view each user from our web application when we start developing our web app. Since I'm going to be using python I decided to add the data into our database using python to get familiar with it. This was achieved by downloading a text file that had the most popular names in the world, I then ran this python script which will populate the new column we created in our users table, figure 4.3.1. Its also going to identify weather the user we are putting in is either a male or female and inserting the user accordingly.

```
with open("names") as f:
    content = f.readlines()
# inserting my names into a list
#list got changed after I run the first loop
#so first i uploaded the males then the femaels
content = [x.strip() for x in content]


x = random.randint(0,2940)
i = 0
while i < 486: #the amount of male users i have
        a.execute("UPDATE users set name = %s WHERE sex = 'M' AND user_id = %s", (content[x]
        config.commit()
        i += 1
        x = random.randint(0,2940)  #pick a ranom name from the list
        print(i)
```

Ran this again for Females

Figure 4.3.1

Another variable which we'll be going to need is the poster for each movie, today no one wants to look at a list of titles, so having pictures is essential. First I had to find a way of getting these posters, I came across **OMDBAPI**. This website was perfect for what I needed, I could use the IMDB id from my database and insert it in the middle of the url they have, this will give me a **json** file which was related to just that movie itself

http://www.omdbapi.com/?i= + IMDB ID + &plot=full

The original plan was to extract the poster url for the movie every time and displaying on the my app. This worked for a while pretty well but had a lag, especially if it were to do this for multiple movies. The solution was to build a simple python scraper which would run this url for every movie IMDB id on the database and inserting the poster url accordingly into a new column which was added to the movie table that will hold the urls. This is a more efficient way of using the posters on the app. Figure 4.3.2, is how the column was populated.

```
import requests
from dbconnect import connection

a, config = connection()


a.execute('SELECT imdbid from links')
x = a.fetchall()

for imdbid in x:
    url = 'https://www.omdbapi.com/?i=tt0'+str(imdbid[0])+'&plot=full&r=json'
    response = requests.get(url)
    results = response.json()['Poster']
    a.execute('Update movies SET img = %s',(results))
    config.commit()
```

Figure 4.3.2

4.4 Prototypes

Using the RAD approach, a number of prototypes were built before the development began. The motivation behind the prototypes was to get familiar with the algorithms that will be used, to see how they really worked independently. A number of sketches were made before hand to understand the algorithms and how they really worked, from the sketches, a prototype was created to do the calculations using python.

This allowed for an understanding on how the algorithms were going to be implemented into the system. This allowed the algorithm to be visualised, and also to collect feedback and find drawbacks to each algorithm early in the developmental cycle.

The specifics of how these prototypes were developed will be discussed in the following chapter. There weren't any compatibility issues when combining these technologies. Everything worked smoothly, my initial thought was that I might have an issue with flask and SQL, but all the doubted faded once development started, thanks to **SQLAlchemy** (Python SQL toolkit and Object Relational Mapper which allows for the flexibility and full power of sql)

4.5 UML

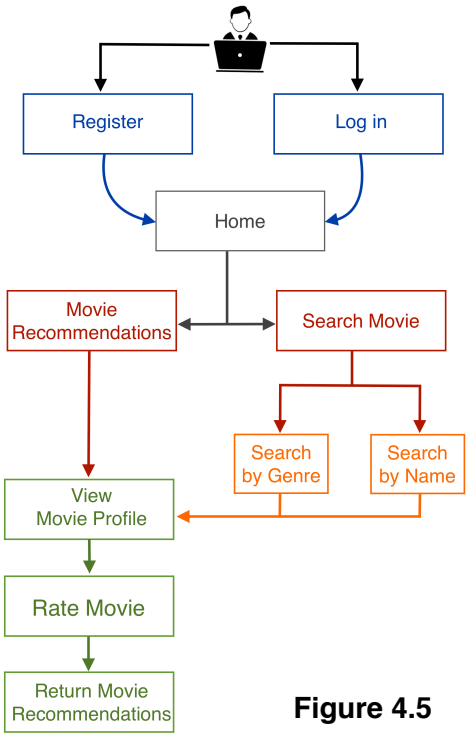

Figure 4.5

Chapter 5 Implementation

With the background reading, the functionality and the interface design finalised, next step is putting all this together into the movie recommendation system. As mentioned, RAD will be used for the development of the application, reason being a number if iterative prototypes are developed, each one is tweaked and improved for the next iteration. This section will look at the 5 iterations that were developed.

Pearson's r Correlation

This algorithm is given by Equation(1.0),

$$r = \frac{N\Sigma XY - (\Sigma X)(\Sigma Y)}{\sqrt{[N\Sigma X^2 - (\Sigma X)^2][N\Sigma Y^2 - (\Sigma Y)^2]}} \quad \textbf{1.0}$$

Where:

N = number of pairs

XY = product of XY (multiply)

ΣXY = multiply each X times each Y, then sum the products

Equation 1.0, is a statistical formula which measures the linear strength between two variables. In order to determine the strength of a correlation, the above formula will produce, what is known as a coefficient value. This value can range between -1.00 and 1.00. If the coefficient is in the positive range, this means that the relationship between the variables are positively correlated, signifying that both values increase and decrease together. On the other hand if the coefficient is in the negative range, this is a negatively correlated relationship, meaning that as one value increase, the other decreases.

5.1 Iteration One

The first iteration was just really to get a working algorithm on python, to test it with random arrays of ratings and to incorporate the dataset instead of a random array. It consisted of a matrix that looked like figure 4.2. The web aspect at this point wasn't a concern, the focus was on executing the calculations.

5.1.1 Implementation

First a function was created which would be calculating pearson's r, refer to equation 1, this is the python equivalent of the maths formula refer to figure 5.1.1. That worked smoothly and gave expected results. This is keeping with the RAD approach, meaning we develop the core functions first adding on non-essential functionality. We are using pearson's r as our core algorithm. This is what it it looks like in python:

```
def pearson(m1, m2):
    m1_c = m1 - m1.mean()
    m2_c = m2 - m2.mean()
    cor = np.sum(m1_c*m2_c)/np.sqrt(np.sum(m1_c ** 2) * np.sum(m2_c **2))
    return cor
```

figure 5.1.1

Next thing was to try and make a connection with our database from python, we need to extract the ratings. Since python has a library for almost anything, they had library called [8]**pymysql**. Which allows for executing sql queries from python. Now that we have an algorithm working and our dataset extracted from the database, we can run a calculation on it. The issue, is trying to get the data collected from the sql table to look in a certain way (matrix) like figure 4.2. Various different test were
done in order to achieve this, they all didn't have the result which we desired. Since a lot of our resources were already spent to that point, it was decided to put that issue on hold.

5.1.2 Observation

The algorithm worked as planned with the data we gave it and a connection has been made between the python script and the SQL database. The issue we face is now, the data collected from the database is difficult to arrange into a matrix form. Which a crucial component for the algorithm to do its calculations.

[8] https://media.readthedocs.org/pdf/pymysql/latest/pymysql.pdf [Accessed November 2016]

5.2 Iteration Two

Following from the last iteration, issues are identified and given the time frame present, we couldn't delegate more time to solving that issue, so for the time being we will put that issue on hold. In order to change the representation of the results to a much more user friendly format, it was deemed necessary to output the python console to something like a HTML page. Next thing was to make a flask app and pass through the python results.

5.2.1 Implementation

The second prototype was focused more on launching a flask app and solving the issue of the data being in a matrix format. Creating a flask app, the Flask documentation was used to help in the development of the app. In the final version of the app, it will consist of a number of linked python programs and function. These will be discussed as we go on.

Flask

By default flask creates its own local web server(default address http://127.0.0.1:5000/)) which acts as the url for the web app itself, meaning thats what you enter in your browser to run the app after the server starts.

After following the documentation from the flask documentation, a *"main"* program was created which had code in order to run the app on the web server. Went an extra bit and created a login function, by putting what was learnt from the previous iteration of pymysql I extract the users name and password and let them login.

```
@app.route('/', methods=['GET', 'POST'])
def index():

    if request.method == 'POST':
        a.execute('SELECT user_id FROM users WHERE name = %s', (request.form['username']))
        userid = a.fetchone()
        userid = int(userid[0])
        a.execute('SELECT password FROM users WHERE user_id = %s', (userid))
        password = a.fetchone()
        passw = str(password[0])

        if request.form['password'] == passw:
            session['user'] = userid
            session['password'] = request.form['password']

            return redirect(url_for('protected'))

    return render_template("index.html")


if __name__ == '__main__':
    app.run()
```

figure 5.2.1

This will in turn link to another app.route which is another function. In this case its protected, protected would be our home page and only a user who has logged in with an active session can enter this page.

In flask the render_template is where you data will be displayed when the browser is run. Using what was learnt about python from the last iteration, the **jinja2** template engine is used to incorporate the python code into HTML.

Figure 5.2.2 shows a code snippet of how python code within the {} is being incorporated. This is simple getting a list of movies (which will be cover later on, how we get these variables) and printing them each into a HTML document.

```
<div class="container">
    <ul class="gridder">
        {% for item in imdbid %}
        <li class="gridder-list" data-griddercontent="#gridder-content-{{loop.index}}">
            <img src="{{item[2]}}" height="200px"  class="img-responsive" />
        </li>
    {% endfor %}
    </ul>

    {% for item in imdbid %}
    <div id="gridder-content-{{loop.index}}" class="gridder-content">
        <div class="row">
            <div class="col-sm-6">
                <img src="{{item[2]}}" height="600px"  class="img-responsive" />
            </div>
            <div class="col-sm-6">
                <a href="/movie/{{item[0]}}"><h2>{{item[1]}}</h2></a>
                <p>{{item[3]}}</p>
            </div>
        </div>
    </div>
    {% endfor %}
```

figure 5.2.2

To this point we have created a login page in HTML and we validate that user in the function from the python script, where that user is then directed to the protected (home) page where there is an empty html page.

Panda

Lets jump back to the python function which calculating the pearson's r correlation, the issue from the last prototype was that couldn't get the dataset to appear in the format we need it to be in (matrix). After some research, it seemed that panda would be the solution to the problem with the app is facing. So to begin with, panda documentation was used to aid in the understanding of how to implement this to the web app. To do this panda needed to be imported, once imported an sql statement of which data I wanted to be put into my panda df (data frame) was copied into a panda data frame.

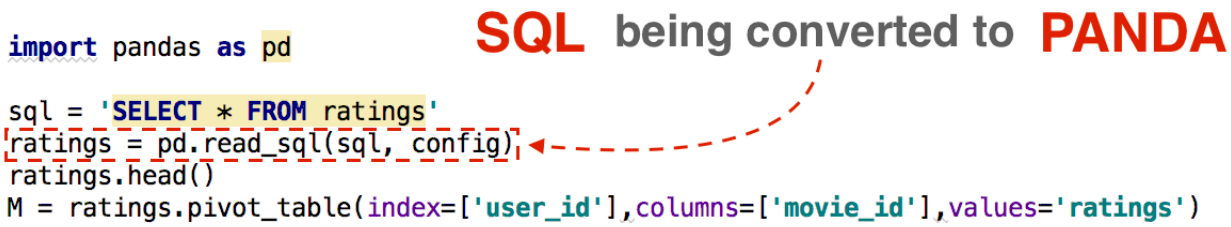

figure 5.2.3

Now we have a working algorithm and now have the data in the format we want it to be in, we can start to play with the what we have. If you refer to diagram figure 5.2, you can see, that by using the column movie_id, it would use the values(ratings) attached to that movie.

Movie Correlation

So now we are going to call the pearson's function we created previously and test it with the data set we have. To do this, we take our matrix M and give it a column to target (*which is going to be movies*).

Example lets compare toy story (*which has a movie_id of 1*) and toy story 2 (*movie_id of 3114*), this gives back 0.356288650363, which suggests they are similar, meaning people who watched and liked toy story also liked toy story 2. Reason for it being a good correlation is because its close to 1 (*the closer to one the better, the closer to -1 the least likely to like the movie*), even though 0.3 isn't very close to 1 but since we have such a massive dataset, the correlation isn't going to be as sparse. Lets have one more test to see if this algorithm is working, lets test toy story (*id=1*) and the god father III (*id = 2023*) gives back a correlation of -0.05288521250, so here we see that users who liked toy story will most likely not like the godfather. In python this is how we called the persons function refer to the figure 5.2.1, also refer to figure 5.1.1 to see how it is calculating persons.

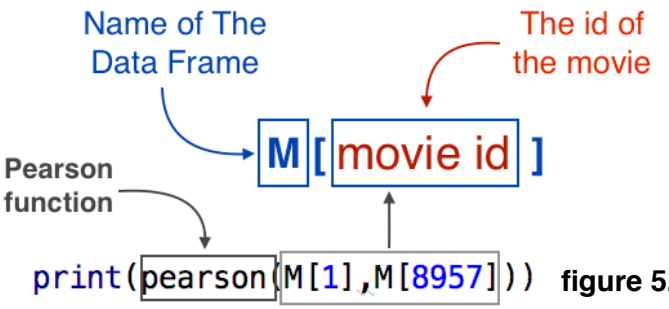

figure 5.2.4

5.2.2 Observation

The second iteration is much more closer than the first iteration, the web app is staring to shape into the final app. For this prototype we didn't face any major issues. Additions were made, we now have a log in page using flask and makes sure the user exists in the database before the directing them to a protected page, that page doesn't display anything as of yet but that is something we will cover in the next iteration. Also a major progress was achieved, introducing panda solved the issue of our data not having the desired format. We also tested that the algorithm worked with our dataset, by comparing a movie with two others and the results were as predicted.

5.3 Iteration Three

Taking in the observations from the previous two iterations, this version, is going to concentrate in give the results we collect some meaning and present the data in a form a user can interact with. In the previous iteration we based out recommendation based comparing two movies at a time, which isn't very efficient, this iteration will have a movie compared to all the movies in the database.

5.3.1 Implementation

User Recommendation

To this point we get a correlation between two movies, what we would like to achieve is give a user back a list of movies which are sorted by their correlation, and this would be there recommendation for whatever movie they decided to like.

First we start by creating a new function which will take in two parameters, movie id (movie which we want a recommendation for) and M (the panda data frame we created in the previous iteration). Essentially what is going to happen in this function is we should get back an array of movie id which will have been through pearson's with and compared against what the user has given, they should also be sorted according to there correlation. To do this we create an array inside the function, which will be populated within a for loop, that goes through all the columns (movies) inside M. Inside the loop it will call the pearson's function which will give us back a correlation. We then save the correlation and the id and add them to the array. Once we have the array populated by the id's and the correlation, we sort that array based on the correlation, finally we say give back the first 10 movies. The code equivalent of what was explained can be seen as follows:

```
def get_rec(mid, M):
    reviews = []
    for title in M.columns:
        if title == mid:
            continue
        cor = pearson(M[mid],M[title])

        reviews.append((title,cor))

        reviews.sort(key=lambda tup: tup[1], reverse=True)
        return (reviews[:10])
```

figure 5.3.1

If we ran this it will only give us back the movie id's for each movie, which has no meaning to the user. This is the output you would get, if we were to get the movies similar to toy story, the output would give you back a movie id and its correlation.

For now this doesn't make any sense, all we can see that the correlation is decreasing the further we go down the array, but the movie id doesn't tell the user what movie it is, we would like to get back the movies title, while we do that we might as well extract the img url and the IMDB id also. This is relatively easy enough to implement. In our get_rec function we just need sql statement to collect the extra variables. So before we append to the reviews array, we add a the sql statements and append the results into the reviews.

```
a.execute('SELECT title from movies WHERE movie_id =%s', (int(title)))
name = a.fetchone()
a.execute('SELECT img from movies WHERE movie_id =%s', (int(title)))
img = a.fetchone()
a.execute('SELECT imdbid from links WHERE movie_id =%s', (int(title)))
imdb = a.fetchone()

reviews.append((title,cor,name[0],img[0],imdb[0]))
```

figure 5.3.2

Now here we can see that we have extracted the relevant information for each recommendation, and appended it to the reviews array. So if we were to print out the array now we would get the title, correlation, img url and the IMDB id

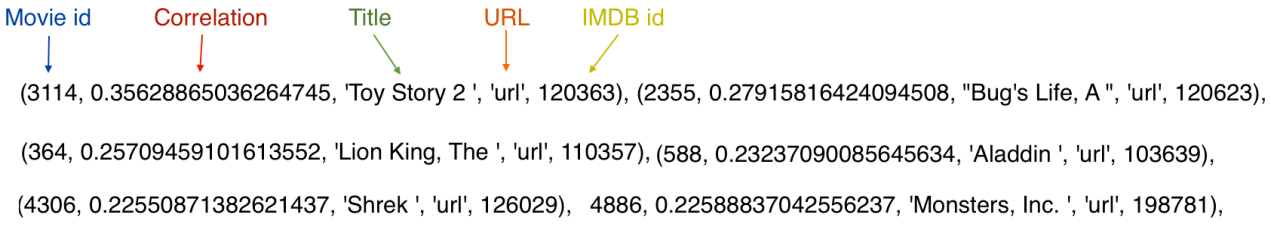

figure 5.3.3

Now we have a movie recommendation for any movie the user chooses, we also get back more information based on each movie, now the user can identify the movie based on a the poster(since the url was very large I just display it as 'url' just for the diagram purpose) and the title, there is also an IMDB id which will allow us to collect more data later on the development. All of this data is being displayed in pythons console, we would like to present the results to more of a user- friendly format. Thats what we will look at next, how we can display our data in our app.

Displaying Results in Flask:

In the previous iteration we created a simple web app which allows a user to login and gets directed to an empty home page, we will try and populate the home page, so we get the user started with their recommendations. So to view the movie recommendations we need to go to the movie's profile, so in the home page we create a search bar, which is actually a select tag which has a search functionality within it. Its a quick search bar that allows the user to very quickly get the movie they are looking for, figure 5.3.4. Once a movie is selected the user gets directed to the movies profile page, where they can view that movies recommendation.

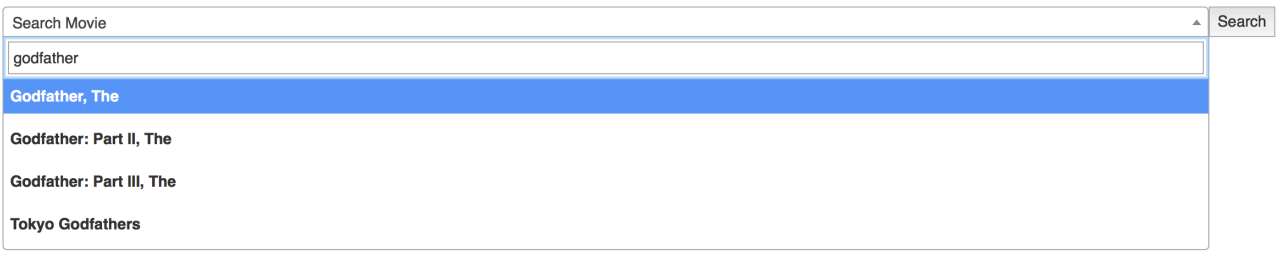

figure 5.3.4

This was achieved by going into the protected function in the flask app in python and pulling out all the movie titles with their corresponding IMDB id and appending to an array. That array would then be rendered to html as the options (title) in the search bar and would have a value of (IMDB id) for the select. Reason behind this is so that a user can only search for movies that exist. This is done in figure 5.3.5.

```
@app.route('/protected')
def protected():
    search = list()
    sqlsearch = 'SELECT DISTINCT movies.title, links.imdbId from movies, links WHERE links.movie_id = movies.movie_id'
    a.execute(sqlsearch)
    srchdata = a.fetchall()
    search.append(srchdata)
    if g.user:

        imdbid = get_user_rec(int(g.user))


        return render_template('regout.html',search=search, name=g.user, imdbid=imdbid)

    return redirect(url_for('index'), )
```

```
<select name="search" style="...">
    <option value="" disabled selected>Search Movie</option>
    {% for item in search[a] %}
    <option value="{{item[1]}}">{{item[0]}}</option>
    {%endfor%}
</select>
{%endfor%}
<button type = "submit" value="search">Search</button>
```

figure 5.3.5

Now that a user can view a search bar and can navigate and see all of the available movies, we are going to need some sort of movie profile page which allows the user to view the movies details and to get the movie's recommendations. So back in our flask app, we are going to receive a post variable (holding the movie id the user chose) which would be sent once the user clicks search. In this function srch, we just send that variable to the movie function which we haven't yet created. Figure 5.3.6, we see that we are taking in a variable and sending it to the movie function.

```
@app.route('/srch', methods=['GET', 'POST'])
def srch():
    select = request.form.get('search')
    return redirect(url_for('.movie', movie_id=str(select)))
```

figure 5.3.6

Next task, create a movie function which would take in a move id. It's in this function were we will be using the algorithm to get back a recommendation for the movie the user chose. First, html for this profile needs to be created, so we can send it information. Like before we are going to pass a movie id only this time its passed in from the search, and the data frame it will work with.

This will give us back an array like we saw already, now its as simple as displaying all that data inside the movie html page, just as demonstrated in figure 5.3.7.

```
<div class="movierec">
    {% for item in rec %}

        <a href="/movie/{{item[4]}}">
        <div class="effect">
            <img src="{{item[3]}}">
          <div class="overlay">
            <div class="text">{{item[2]}}</div>
          </div>
        </div>
        </a>
    {% endfor %}
</div>
```

```
@app.route('/movie/<movie_id>')
def movie(movie_id):
    if g.user:
        rec = get_rec(movie_id, M)

        return render_template("movie.html", rec = rec)

    return redirect(url_for('index'))
```

figure 5.3.7

We would like to just add a bit extra functionality to the profile page, we could allow the user to know which movie they have clicked on and show them information about that particular movie. This can be achieved by calling a JSON file from the OMDB url which was mentioned in chapter 4. This was as simple as just saving all the relevant data into an array, which would then be past in the movie function, then just rendering it out in the movie.html.

5.3.2 Observation

The third iteration was a success in terms of what was planned to achieve, this iteration is a closer representation of final app we are building. This prototype, allows user to login, once logged in they have the option to search for a movie, they are then directed to that movies profile were they can view information about that particular movie, and they are also presented with a list of movie recommendations based on the movie they picked. Major progress has been made yet an issue arises, movie page to load takes 1 mins and 10 seconds, which is very slow and a drawback from a users point of view. This is something which will be addressed in the next iteration.

5.4 Iteration Four

From the previous iteration we can see that our web app is coming along quite well we just need to address some issues before moving onwards, we identified that the speed for loading the movie recommendation was taking too long, in this iteration we will look at how we can speed this up. Another thing to consider in this iteration is giving the user the ability to rate a movie. A function which will be added to this iteration is that users to get recommendations based on the rating they give a movie. This iteration will also, give user recommendations, instead getting a recommendation based on what you have watched rather than getting it based a single movie. Distinguish between a new user or an existing user is also crucial.

5.4.1 Implementation

Speed

The first part of trying to solve this issue, is to ask why is the recommendation function taking so long to execute, answer is that it is executing millions of calculations its literally comparing every movie with the movie the user choses. So one solution which was under taken is limit the amount of movies we compare a movie to. How this was done was we only considered movies with 20 ratings or above, this is not only going to speed up our calculations but it's also going to give us more accurate recommendations.

After limiting the data set to movie which have been rated above 20 times, there was another issue, some movies which the users can view won't be in that dataset which gives an error. Example since godfather has 203 ratings it will show up as it will be included in the new data set but if the user clicked on Sarfarosh(movie in the dataset) which has 4 ratings it won't exist when it goes to do the calculation. To solve this issue, instead of having 1 data frame, a second was created which would be filtered. So the algorithm now is going to grab the ratings from the users choice from the first data frame which contains all the movies and compare it to every movie from the second data frame. This solved that issue, now lets see if it made a major difference in our speed. After running the new algorithm the speed of the load had dropped from a 1:10 to 36 seconds which is a great improvement over the last test.

Note that 36 seconds that of an great improvement, just in terms of user usability it is still a significant wait. After to some research it was suggested by the supervisor to use multiprocessing which should speed up the calculation significantly, especially with the machines now most holding more than one core. Thanks to pythons libraries this can be achieved relatively easy, there is module which does the multiprocessing, figure 5.4.1.

```
def recmov(mov,set):
    p = mp.Process(target=get_rec, args=(mov, set))
    p.start()
    p.join()
```

figure 5.4.1

This brought the load of a page to 7 seconds which isn't that long of a wait for the user.

Movie Rating System

To get ratings on the movie page, the first thing was to add 5 links which will present stars (stars were icons from [9]fontawsome), each linking to a function, ratings which we will create shortly. Each link will have a rating depending which link the user clicks on it will will send the rating to the rating function.

Now let's create our rating function, as mentioned this is going to take in a rating, but how would the function know rating to give that movie, so it will also take in a movie id. So the user will click on a star which will collect the movie id and the rating the user gave and send those variable to our function rating, from there we will identify if a user has rated this movie before or not, this is important due to the sql statement as inserting and updating will follow different queries. This quite straightforward, we have a query which will check if that rating exists for that user, if it does update the table, if it doesn't exist insert rating into the table, figure 5.4.2.

```
@app.route('/rate/<rating>/<movie_id>')
def rate(rating, movie_id):
    if g.user:
        rate = rating
        timestp = int(time.time())
        movie_id = movie_id.replace("(", "").replace(",", "").replace(")", "")
        a.execute('SELECT movie_id from links WHERE imdbId =%s', (movie_id))
        m_id = a.fetchone()

        a.execute('SELECT * from ratings where user_id = %s and movie_id = %s', (g.user, m_id[0]))
        new = a.fetchone()

        if new == None:
            a.execute('insert into ratings VALUES (%s,%s,%s,%s)', (g.user, m_id[0], rate, timestp))
            config.commit()
        else:
            a.execute('update ratings set ratings = %s,timestamp = %s where user_id = %s and movie_id = %s',
                      (rate, timestp, g.user, m_id[0]))
            config.commit()

        return redirect(url_for('.movie', movie_id=movie_id))

    return redirect(url_for("index"))
```

figure 5.4.2

[9] http://fontawesome.io

We then redirect them back to the movie page. We also would include some css to highlight the stars the user gives, so if they rate it a four it would highlight 4 stars, this done through javascript, we pass a python variable holding how many stars that movie got and it would run if statements changing the colour of the stars depending on the rating.

Improve Movie Recommendation

To improve on the recommendation that the user is getting back we just need to look at the function one more time, previously we were only taken in 3 variables, the movie (which will be used for the algorithm), the entire data frame, and a filtered data frame. A new variable will be introduced and will be called like. This variable is either going to be a 1 or 2, so before we sort the data we will have an if statement asking to see if the user liked or disliked the movie, and according to their rating we will return the first 10 movies or the last 10 movies of the sorted array, figure 5.4.3.

```
if like == 1:
    reviews.sort(key=lambda tup: tup[1], reverse=True)
    return (reviews[:10])

if like == 2:
    reviews.sort(key=lambda tup: tup[1], reverse=False)
    return (reviews[10:])
```

figure 5.4.3

User Recommendations

This won't be very different to how we computed the recommendation for movies, here we will take the list of movies which the user would have rated, sum the correlations of those movies with all the other ones. We then return an array of these movies sorted by the total correlation with the user.

The difference here is we are going to take the list of movies they have rated as the input.

```
def get_movie_recommendations(user_movies):
    movie_similarities = np.zeros(corr_matrix.shape[0])
    for movie_id in user_movies:
        movie_similarities = movie_similarities + get_similar_movies(movie_id)


        similar_movies_df = pd.DataFrame({
            'title': movie_index,
            'sum_similarity': movie_similarities
        })

    similar_movies_df = similar_movies_df.sort_values(by=['sum_similarity'], ascending=False)
    return similar_movies_df
```

figure 5.4.4

In figure 5.4.4. you can see that we have a function which will give us a user recommendation. It's taking the list of movies the user has watched and getting the correlation of those movies, and it's being added into a matrix. Like before this function is just going to give us back a list of movie id's, so like before we'll have an array which we will append relevant variables to it.

Now that we are getting user based recommendations we need to display them somewhere. So we create a home page, and render the movies in the home page. Also adding a link to each movie poster so that from there they are directed to the movie page to get more recommendations and view the profile for the movie where they can also give a rating.

New User

In order to get user recommendation the user must at least have one movie rated, thats how the algorithm works, so to solve the issue of a user not getting any recommendations, as the user logs in, before they are directed to the home page (user recommendation page), to run a query to see if they rated a movie or not, depending on the output direct them to the search page and prompt user to rate at least one movie, otherwise direct them to their recommendations.

5.4.2 Observation

This iteration contains nearly all of the core functionality, user logs in, depending weather the user is new or existing they are directed to rate a movie or are presented with a list of movies which is recommended based on what they rated. For each movie there is also recommendations based on users rating. Also the time of the recommendation has improved drastically. This iteration is almost becoming the final app.

5.5 Final Iteration

For this iteration we will be looking at the smaller functions, since the heavy lifting has already been done previously in the last iterations. We will be looking at styling and user interface. Also to add a tab in order to see genres and their top movies. We will also include a profile page to see what movies a user has rated.

Style

Styling the home page I used an open source template, [10]gridder, which allowed for the movies to be displayed from the poster, once clicked it would open up a section displaying the poster and the plot of the movie. Also the menu was a template which from an open source website, [11]w3schools, it was a simple black side menu which looked slick and simple. Throughout the app, all the posters had a hover effect, where the movie wouldn't display the title unless it was hovered over, this was achieved by using CSS, figure 5.51.

figure 5.5.1

This added a bit of an interaction to the web app, to make it more user friendly. I had this through out app where a list of movies appeared.

Profile

For the profile page it was as simple as executing a query which extracted all the movies that user has rated and appends all of the movies details in an array and render it on profile.html, each movie is then linked to the movie page, in order to get more information.

[10] https://github.com/oriongunning/gridder

[11] https://www.w3schools.com/howto/tryit.asp?filename=tryhow_js_sidenav

Genre

For the genre function, an sql query was executed to retrieve all the unique genres, once again it was saved into an array, which would be sent to a genre. Each genre would be displayed in a block which once clicked would execute a python script which would retrieve the top 20 movies for that genre and them in a html page

```
@app.route('/genre')
def genre():
    if g.user:
        gen = get_genre()
        return render_template('genre.html', genres = gen)

@app.route('/genremovies/<mgenre>')
def genremovies(mgenre):
    if g.user:
        x = str(mgenre)
        gen = gen_movies(x)
        print(gen)
        return render_template('genmov.html', movgen = gen)
```

figure 5.5.2

From figure 5.5.2, get_genre is a function which is going to select all the DISTINCT genres from the movies table in SQL and render them onto the HTML page. This will look like figure 5.5.3.

figure 5.5.3

Each genre will link to a page which will display the top movies for the genre the user selected. This will also do another simple SQL query to collect all the movies which contain the genre the user clicked on, then order by number ratings and the movies rating. This was achieved by executing a PANDA query.

```
movie_stats = lens.groupby('title').agg({'ratings': [np.size, np.mean]})
movie_stats.head()
atleast_100 = movie_stats['ratings']['size'] >= 30
movie_stats = movie_stats[atleast_100].sort_values([('ratings', 'mean')], ascending=False)[:20]
movie_stats.head()
```

figure 5.5.4

Figure 5.5.4, is Panda filtering out the movies which have been rated less than 30 times, it then sorts it based on the rating and the mean, and renders the top 20 movies.

Chapter 6 Testing & Evaluation

This chapter describes the different issues that were found during the testing analysis phase followed by the testers that raised the issues and the solutions that were implemented to solve them. The system's testing and evaluation analysis performed consisted of assigning testers to test the application and ensure that it was working as it was intended to. In addition. Moreover, the app was tested upon every iteration by the supervisor.

6.1 Testing

ISSUE #1 *Raised by: Haider Bari (tester 2)*

It was suggested that to give the web application a more interactive feel, to give users a profile picture.

SOLUTION

It was agreed to have a smart default picture on registration. This was achieved by using the sex of the user and their age (6 age groups, are available in the dataset). Once the pictures were gathered, if statements were created to first identify weather the user was a male or female, then the age would determine which profile picture would get selected.

ISSUE #2 *Raised by: Cesar Velasco (tester 1)*

Quick search box is not visible in all web pages.

SOLUTION

This functionality already was built, the issue here was that it was only visible in the home page, it was as simple as including on all the other pages.

ISSUE #3 *Raised by: Cesar Velasco (tester 1)*

The top movies of the genre page should display the poster of the movies.

SOLUTION

Once the movies have been extracted and save in an array, by the movie's title. That array is then iterated through using a for loop. A new array is created which, instead of holding just the movie's title, it will hold link to the poster and the IMDB id. After this step it's as easy as rendering this array onto the HTML page. This can be seen in figure 6.1.1.

```
topgenmov = []
for index, row in movie_stats.iterrows():
    a.execute('SELECT img from movies WHERE title =%s', (str(index)))
    img = a.fetchone()
    a.execute('SELECT movie_id from movies WHERE title =%s', (str(index)))
    mid = a.fetchone()
    a.execute('SELECT imdbid from links WHERE movie_id =%s', (int(mid[0])))
    imdb = a.fetchone()

    # print(imdb[0])
    topgenmov.append((index,img[0],imdb[0]))
```

figure 6.1.1

ISSUE #4 *Raised by: Haider Bari (tester 2)*

The movie profile page was dull, add some sort of a reviews section, to see what rating other users gave the movie.

SOLUTION

Here, an SQL statement was executed which would collect the users name, the date and time they rated the movie, and the rating they gave for that movie. This was then added to a list and rendered, figure 6.1.2.

```
notification = list()
sql = 'SELECT users.name, movies.title, ratings.ratings, ' \
      'FROM_UNIXTIME(ratings.timestamp), users.sex, users.age ' \
      'from ratings,movies, users WHERE ratings.movie_id=%s ' \
      'and movies.movie_id = ratings.movie_id and ratings.user_id = users.user_id '
a.execute(sql, (id))
notidata = a.fetchall()
notification.append(notidata)
```

figure 6.1.2

ISSUE #5 *Raised by: Eoin Carroll (Supervisor)*

A suggestion which was made by the supervisor was to have the users watched movies in the profile page to be ordered alphabetically. Reason behind doing this, for ease of locating movies. This suggestion wasn't specific to only the movies in the profile page but also in the quick search bar.

SOLUTION

Adding this functionality was carried out by, ordering adding an extra query in SQL, before the result was saved into the array. An order by was added to the end of the query, as highlighted in figure 6.1.3.

```
SELECT movies.title as mov,links.imdbId from movies, ratings,
links WHERE user_id = 90 AND movies.movie_id = ratings.movie_id
AND ratings.movie_id=links.movie_id ORDER BY mov
```

figure 6.1.3

6.2 Evaluation

This sub section will evaluate and review the overall and finished web web app.

6.2.1 Aim

The original aim of this project (Section 1.2) was to "*provide both user based recommendation and movie based recommendation to the user. The goal is to have an easy to navigate application which allows users to navigate effortlessly*"

In this respect the project was a success, the web app is a fully working app which was tested and had accurate recommendations for both user based and movie based recommendations, with an average of 8 movies out of the 10 being accurate recommendations

Additionally, the design of the app was simple and also spacious with the movie icons being boxy and easy to navigate, once clicked it gives a quick overview of the movie before you giving you the option to go to the movie profile page, allowing user to get exactly what the need without the clutter and hassle. The motivation behind the clean and simple design was to keep the mobile users in mind.

6.2.2 Speed

Moreover, the speed in which the recommendations are being calculated is relatively fast. Initially the speed in which it took to compute a recommendation was 1m 10s, which was a very long waiting time which at times resulted in a timeout.

By reducing the data set to only movies which are rated by at least 20 users, which will be considered for the calculation, this improved the speed of the computation by almost twice as fast. The time here was 36s.

The speed was improved more by introducing multiprocessing and allowing us to keep the entire dataset with still achieving a much faster time, time here was 7s. This was a drastic improvement to be made, as the load time would have caused timeouts and would have effected the user's experience.

6.3 Conclusion

Overall, the application met the requirements and deemed as a success. From the testing we improvements are noticed. Furthermore, feedback from the testers was extremely positive. The web app works as advertised with extra functionality added to better user experience.

However, the web app is not perfect, there are areas which still need improvement, which will be discussed in the next chapter.

Chapter 7 Conclusion & Future Work

The original issue that this web application was attempting to solve, remove the need of asking people for what movie to watch next. Also to get a real time recommendations for movies based on the ratings that were given for movie(s). Create an app that learns about the user the more involvement there was. The recommendation system app built has a high potential of becoming a commercial product. It uses a sophisticated maths algorithm to predict movies for each individual user. Both movie and user based recommendations are calculated for the user. A unique feature which the app provides is that once a user changes a rating it gives a different recommendation based on the rating given.The user interface was easy to navigate with a lot of recommendations ensuring the user finds a movie of interest. The design is clean and boxy, ensuring that users visiting from smartphones won't have difficulty navigating through the app. A major drawback was the time in which the calculations were taking to execute, which was improved quite noticeably. The following are some of the extras which would have been a great addition to the app.

7.1 Algorithm

Though the algorithm used was sophisticated and gave back recommendation, it still had multiple limitations. Such include as:

- The algorithm needs at least one rating for a user to get a recommendation.
- It doesn't take into account any other constraints, example genre, actors, studio, etc

These limitations could make the recommendations the user gets back much more accurate.

The major draw back to this algorithm is that since its a type of collaborative filter, it suffers from the cold start issue, this is were new users get recommendation unless they've reviewed a movie.

One way to over come this is to incorporate more of a hybrid system, meaning you use both collaborative and content filtering, that way even users with not rating get reconditions and movies with no ratings get recommended.

7.2 Cloud Platform

Another issue is that in order to use the app you need to have both python and flask installed in your machine. It would have been ideal to have it run in the cloud. Though this isn't a big issue in terms of users this could be a problem for less technical users.

A way around this issue is to host the app as a commercial solution. This would ensure that as long as Flask was installed on the web server of the company which is hosting the app, then the app's user don't need to worry about installing anything onto there devices. This can be as simple as just entering the url for the site in their browser.

7.3 Include Facebook

A really interesting functionality which would have been a great addition to the app, would be incorporating Facebook with the app. It would have been easier for users to register to the app through there Facebook profile. Also linking users together through their friends list.
Even including Facebook for the movie recommendations, grouping friends together finding movies a user might like based on what there friends have watched. Even taking a certain group of friend's movie ratings and identify a movie that group should watch together.

Bibliography

I. [Online] http://onlinestatbook.com/2/describing_bivariate_data/calculation.html

II. [Online] https://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2010/rapporter10/eliasson_anna_10123.pdf

III. [Online] https://www.youtube.com/watch?v=SxsAIh8vpEE&ab_channel=Mathified

IV. [Online] http://pandas.pydata.org

V. [Online] Available at: http://www.avonmore.ie/

VI. [Online] http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

VII. [Online] http://flask.pocoo.org

VIII. [Online] J.C. Gower Euclidean distance geometry Math. Sci., 7 (1982), pp. 1-14

IX. [Online] https://www.researchgate.net/publication/277324930_Pearson%27s_Product-Moment_Correlation_Sample_Analysis

X. [Online] https://www.djangoproject.com

XI. [Online] https://media.readthedocs.org/pdf/pymysql/latest/pymysql.pdf[Book] Flask Web Development by Miguel Grinberg,

XII. [Paper] Linden, Greg, Brent Smith, and Jeremy York. ”Ama- zon.com recommendations: Item-to-item collaborative fil- tering.” Internet Computing, IEEE 7.1

XIII. [Paper] Esteves, R. M. and Rong, C., “Using mahout for clustering wikipedia’s latest articles: a comparison between k-means and fuzzy c-means in the cloud,” in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011, pp. 565–569.

XIV. [Book] Koren, Y., Bell, R., and Volinsky, C., “Matrix factorisation techniques for recommender systems,” Computer, vol. 42, no. 8, pp. 30–37, 2009.

XV. [Book] Flask Web Development by Miguel Grinberg,