Recommender Systems: Accurate Recommendation

Alistair Grom
964398

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Batchelor of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

May 2021

Declaration

This work has not been previously accepted in substance for any degree and is not being con- currently submitted in candidature for any degree.

Signed .....  ..... (candidate)

Date **04/05/2021**.....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ....  ..... (candidate)

Date **04/05/2021**.....

Statement 2

I hereby give my consent for my thesis, if accepted, to be made available for photocopying and inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ....  ..... (candidate)

Date **04/05/2021**.....

Abstract

Recommender systems are a massively valuable application of machine learning, both to corporation's and users themselves. This technology is now ubiquitous amongst the products we use on a daily basis and shapes the interactions we have with these products for the better or the worse. This project aims to create an algorithm that gives the user accurate recommendations based on the content that they enjoy and that alone.

By creating a recommender system that is entirely content based hopefully we will be able to provide users with more of the content that they actively consume, as opposed to what is most popular or people like them like the most.

This paper will cover various approaches towards recommender systems, how I developed my recommender system product and the outcomes from this project.

Table of Contents

Contents

Table of Figures

Chapter 1

Introduction

A recommender system is a tool that suggests content to users based on their usual online habits. Due to the exponential growth of information on the internet these tools have never been more important. Recommender systems are information filtering systems that deal with the problem of information overload [1]. These systems have application across a multitude of mediums, from music recommendation to targeted advertisements. It is safe to say that these machine learning algorithms have become ubiquitous throughout the internet, so much so that we often open an app and interact with what it suggests not the other way around.

By way of data collection everything you click on, how long you spend on that item, what you rated it is tracked, along with many other increasingly invasive factors. Google has tracked android users' location when they 'check in' to a place [2]. This presents a huge business potential. These systems are worth a huge amount money to these tech giants. In 2013 Amazon was stated to be generating %35 of its revenue from the recommendations alone [3]. In 2019 Amazon created in revenue $280.52 Billion %35 of that results in $98 Billion.

What is accurate? And how do we define that. This metric is validated by the user choosing, buying, or interacting with that medium. The advanced deep learning algorithms used by the likes of Google [4], that learn only from driving the highest possible engagement rates, seem to think so. But is this pure accuracy based on a current item or simply what is going to drive the most engagement? This is a key distinction that often is lost amongst use of this type of system at the hands of large corporations.

This project will aim to only consider specific content that user has interacted with rather than generating homogenised feeds of information to generate the largest engagement rates.

1.1 Motivations

The main motivation for this project is to create a recommender algorithm that can accurately recommend content to users. In addition to this the algorithm will be a content-based system, meaning that new recommendations will only come from the actual content the specific user has interacted with before.
This decision was made as a response to the widespread use of recommender systems to drive clicks and draw users into what they want them to see, opposed to recommending content that is most like that of which they interact with most.

1.2 Overview/ Project Objectives

- **Create an accurate recommender system** – Creating an accurate recommendation system that can accurately recommend a user an item from a dataset based on their preference of another item/s. The algorithm that I have created is using a content based[1] system that looks for similarities between movies based on their metadata. This could be the keywords used to describe the movie the actors that are in the cast and the genre of movie or the top cast in the movie. The way that I have done this is through Natural Language Processing[2] to give values to the terms in the metadata of each movie.

- **IMDB Dataset** – The dataset chosen to perform the recommendations was the IMDB dataset from this dataset. In this dataset there are three files that I have used to form the data in which my algorithm is built on.

    - 'movies_metadata.csv' – This is the main data that the movie data is taken from. It contains fields such as title and overview which are key to the natural language processing techniques used.
    - 'credits.csv' - This contains data about the cast and crew for each movie in the dataset.
    - 'keywords.csv' – This contains data of keywords of movie, such as 'robbery', 'detective' to describe a movie.

    From this data we merge into one dataset from which contains all the data we need to generate recommendations.

[1] Content-based filtering uses item features to recommend other items like what the user likes, based on their previous actions or explicit feedback. [7]
[2] Natural Language Processing the application of computational techniques to the analysis and synthesis of natural language and speech.

- **Provide an interface for a user to interact with –** To provide users recommendations that are useful to them the users will need something to enter their preferences and receive their specific recommendations. The method I have chosen for this is through Jupyter Notebooks, a web-based environment for interacting and running code. It is very commonly used in the world of data analytics and machine learning.

Chapter 2

Background Research

The field of recommender systems is a vast section of machine learning with a large amount of research into different approaches to creating accurate recommendations for a large array of problems. There are many papers that I have used to form the basis of my approach on a more basic level as many are complex past the bounds of this project. In addition to just academic papers, I have used online resources to explain what some of the key concepts in recommender systems are.

2.1 Related Work

2.1.1 Research Papers

A large focus of my recommender system was to include an item focussed approach, an area of content-based filtering.

A paper that I looked at 'Recommendations Without User Preferences:
A Natural Language Processing Approach' [5] This paper looks at how a dataset of movies can be categorised into genres by a standard word-space similarity metric using a cosine similarity. This method does not require human preferences which is something that I felt was very useful for the type of system that I was intending to build, as opposed to the more popular collaborative filtering[3] this develops similarities from the plot summary of each movie.
The algorithms mentioned are 1st a words space similarity which is used to form a cosine similarity and then the similarity of the two films is the distance between those two feature vectors. While this algorithm does not perform that well in this paper, I have spotted a few areas that I will be able to improve this a large amount.

Another paper I used in the research phase of my project was, "DRN: A Deep Reinforcement Learning Framework for News Recommendation" [6] While this paper is looking at very complex applications of Reinforcement Learning and Deep Q Learning applied to Recommender Systems there were still some things that I was able to take from this paper when developing my approach to the task.

[3] Collaborative Filtering uses similarities between users and items simultaneously to provide recommendations [10]

This paper identifies that traditional content-based news recommenders use a method of TF-IDF[4] and user profiles based upon previous articles that the user has interacted with.

2.1.2 Dataset Exploration

A recommender system is nothing without data to drive the algorithm. Using the correct dataset for your algorithm is key and can drastically alter the performance of the system.

Throughout this project the exact dataset that I have been using to create a recommender for has changed many times. The dataset can have a large impact on the techniques used to implement the algorithm, for example more subjective data that is not as easily defined needs more metadata to allow the machine learning algorithms to accurately detect what data it is currently reading.

In the early stages of my project, I was looking towards a music recommender system and was using the Spotify 1922-2021 dataset which looks at around 600k songs in this period. I quickly discovered that a content-based system for music was not going to be viable as a lot of music is not described well in terms of metadata so the approach that I was looking to use was not to suited towards this.

The next dataset I looked at was the MovieLens dataset. This is probably the most used dataset for people learning about recommender systems. This dataset comes in many sizes to suit the computational power of those who intend to use it, from 100K to 20M. It includes both data about the content of the movies themselves and both user data containing ratings. This dataset would be more suited to an approach to recommender systems that take into account the ratings of users, such as Collaborative Filtering rather than my chosen approach of Natural Language Processing for a content-based system.
This led me to find a dataset that had a larger description of each film, beyond a one-word genre value.

The dataset that I have used for the development of this recommender system is the IMDB Movie dataset. This dataset contains around 45,000 unique movies with information on title, popularity, ratings, and description of the movies themselves. Alongside this dataset file there are additional auxiliary files that add more information to the movies, more in depth cast information, the ‘credits’ file and the ‘keywords’ file that is particularly useful in making the cosine similarity function as effective as it is.

[4] Term Frequency-Inverse Document Frequency (TF-IDF) algorithm which assigns an importance value to a keyword.

2.1.3 Approaches to Recommender Systems

Recommender Systems have different methods of achieving accurate recommendations of content to users these are not all solely based on the content as my system is. These can be split up into 3 main types of systems.

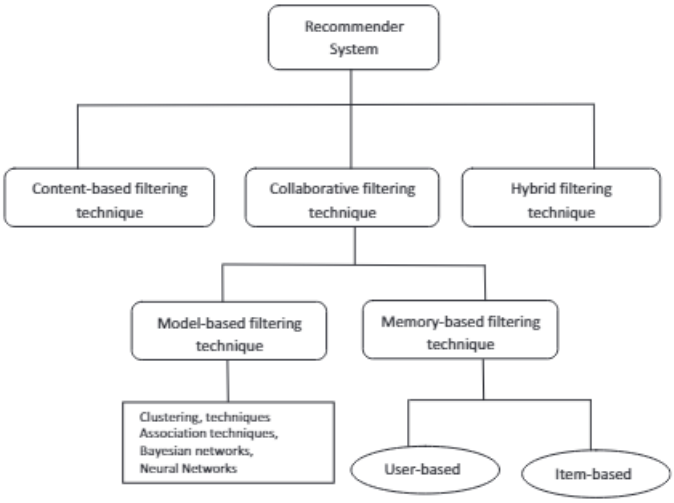


Figure 1 Recommendation Techniques *[19]*

Content-based Recommendation

Content-based Recommendation or Content-based filtering is a system that uses item data about categories to recommend the user items that are similar to items they 'like' or 'are interested in' based on previous search activity or selected user preferences. The items are assigned values to which category that they fall under. This should recommend items that are relevant to the user, based on a similarity metric that we can calculate using the dot product. Using a matrix representation can be useful.

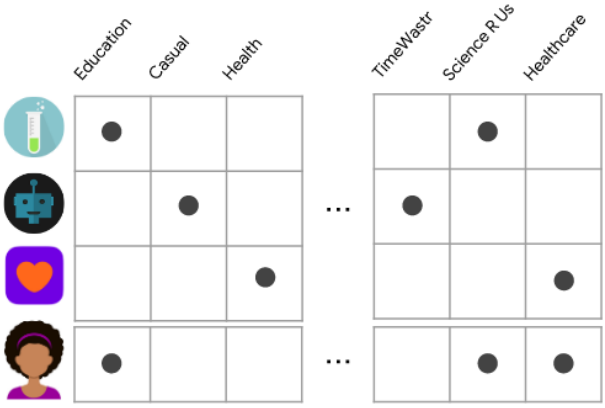


Figure 2 Dot Product as a Similarity Measure *[7]*

This image shows how the dot product $\langle x,y\rangle = \sum_{i=1}^{d} x_i y_i.$ is calculated. If a feature of an app appears in both the user's preferences (x) and the apps description

(y) then the value will be a 1. The higher the value of the dot product the higher probability that the user is likely to be interested in this app. [7]

One advantage of this method over a Collaborative filtering system is that the cold start problem[5] only applies to knowing what to specifically recommend to a user, not the entire system. A content-based system can generate a similarity value between all items in the dataset through, in the case of this project natural language processing or other similarity metrics. Whereas a collaborative filtering system does not use the features of the data itself, but the interactions of users like them.

Item-based or User-based Content Filtering

Item-based filtering focusses upon the similarity between the items in the dataset to rank them in terms of similarity to an item. This often depends on the quality of the description as that is how the items will be classified into their respective points in the vector space[6].

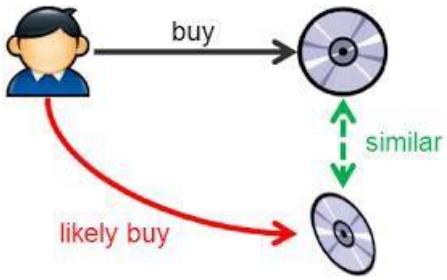


Figure 3 Item-based Filtering *[21]*

User-based filtering requires a user to build up a profile of a few things that they like, by either interacting or explicitly stating they 'like' or want to 'see more like this'. Then using the items data of those that they 'like' can be used to form a profile and recommend items that are more suited for a person of their tastes.

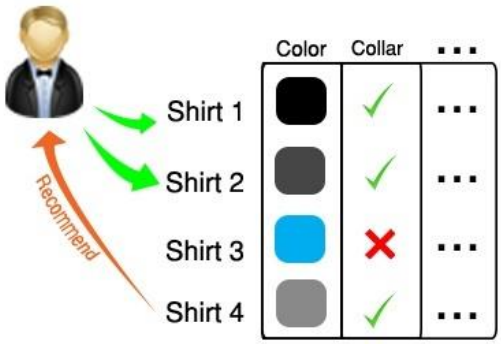


Figure 4 User-based Filtering *[21]*

[5] The cold-start problem, which describes the difficulty of making recommendations when the users or the items are new. [20]

[6] The mathematical representation of the features extracted from the datapoints. This can be used to calculate data similarity.

There are two types of feedback that involve the user giving their opinion of the item. Explicit feedback such as rating scales, provides users with a mechanism to unequivocally express their interests in items. [8] The other style is Implicit feedback, where the feedback is generated by the RS itself, through inferences it makes about the user's behaviour. [8] This can be in the form of time spent listening/watching, adding to a list, or simply clicking on the item. Both of these approaches have their merits, implicit being less accurate but more abundant and explicit meaning more concrete but less prevalent.
This approach may sound like it will have a cold-start problem which it will to a small extent, but this can be formed with as little as 2 items. Obviously, the more data about a user will result in better recommendations. Comparing this to the cold start in collaborative filtering we need multiple users with multiple likes to form this type of recommender.

Natural Language Processing
Natural Language Processing the application of computational techniques to the analysis and synthesis of natural language and speech. It is a branch of AI (artificial intelligence). The goal is simple, for the computer to understand a piece of text and make sense of it as a human would.

Term Frequency-Inverse Document Frequency (TF-IDF) is an algorithm which assigns an importance value to a keyword. Like the name suggests it measures the importance a phrase or word appears in a piece of text. In the context of my project the most important words would define the movie and be categorised as feature names, such as war or drama. These can then be used to calculate similarity values between movies.
Another method of feature extraction, one that I have used in my algorithm, is the CountVectorizer function which is in a package from sklearn. This does a similar task to the TF-IDF function of feature detection.

Vector Space Model
To compare the similarity of items with respect to a given keyword or set of keywords there are different methods, the one I have chosen for my project is Cosine Similarity. Where x and y are term frequency vectors for comparison.

$$sim(x,y) = \frac{x \cdot y}{||\, x \,||\, y \,||}$$

The closer to 1 that the value of sim(x, y) is then the more similar the two items are [9] This value allows us to determine what items to recommend based on the users preferences a threshold of what is considered similar can be implemented to meet the specific needs of the application this system is being used for. This use of Natural

Language Processing and the Vector Space Model is the basis for content-based recommendation.

```
[[1.         0.09534626 0.1        ... 0.12909944 0.1        0.        ]
 [0.09534626 1.         0.         ... 0.         0.09534626 0.        ]
 [0.1        0.         1.         ... 0.12909944 0.1        0.11952286]
 ...
 [0.12909944 0.         0.12909944 ... 1.         0.12909944 0.        ]
 [0.1        0.09534626 0.1        ... 0.12909944 1.         0.        ]
 [0.         0.         0.11952286 ... 0.         0.         1.        ]]
```

Figure 5 A screenshot of my 2nd cosine similarity function's output

This shows one of my cosine similarity functions in matrix form. We can see that where the movie is being compared to itself it is 1 and the varying values are the level of similarity of the metadata about the movie.

Collaborative Filtering

An alternative approach to the one that I have chosen to implement is the Collaborative filtering recommender system. This approach uses similarities between users and items simultaneously to provide recommendations. This allows us to recommend items to user 1 based on interests of a similar user 2 [10] As this approach relies on user feedback, we need a method we can use an explicit or implicit method, as mentioned in the User-based filtering section.

To show how this could work we can map users on feedback matrix as to how much they enjoy a certain genre of movie.

Figure 6 Example of a 1D matrix representation of user preferences *[10]*

We see here that not all users' preferences are clearly represented. Looking at the two users with small values within preferences and user profile. From this we can increase the dimensions such as another vector, possibly action or drama.

The power of collaborative filtering lies within its ability to learn the embeddings of users. This allows it to apply this same matrix of what these users watched to other similar users.

From my research into Recommender Systems, I did find this approach to be more popular and its use to be more prevalent amongst products that we use in everyday life such as Spotify, Netflix, and YouTube.

2.1.4 Use of Recommender Systems

Netflix

Recommender Systems are currently used in a vast array of software products from online shopping retailers to media streaming services to social media.

The Netflix homepage is full of recommendations. These are all calculated in each position for each user based on their history or chosen preferences at the start of their membership (avoiding the cold start problem).

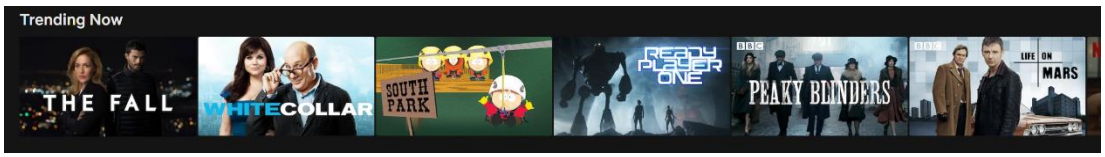


Figure 7 A screenshot of one of Netflix's content rows which are chosen based on your preferences.

Each of these rows is ranked with the most relevant show being placed furthest left and the most relevant categories further up the page. In 2006 Netflix organised a contest named ’The Netflix Prize’ to see if anyone could improve the recommendation system that they were using by %10 for a prize of $1 million. This contest involved a dataset of over 100 million ratings of 17,770 movies from 480,189 customers [11] This likely was a large factor in growing the popularity of recommender systems throughout these emerging tech giants circa. 2006.

YouTube

YouTube has a particularly good hybrid content-based collaborative filtering recommender system for picking videos that grab our attention and lead us down a 'rabbit hole' of videos that we hadn't initially intended to watch. It is with the use of Deep Learning, which Google uses across its products that this is achieved.
'Our system is built on Google Brain which was recently open sourced as TensorFlow. TensorFlow provides a flexible framework for experimenting with various deep neural network architectures using large-scale distributed training.' [12]

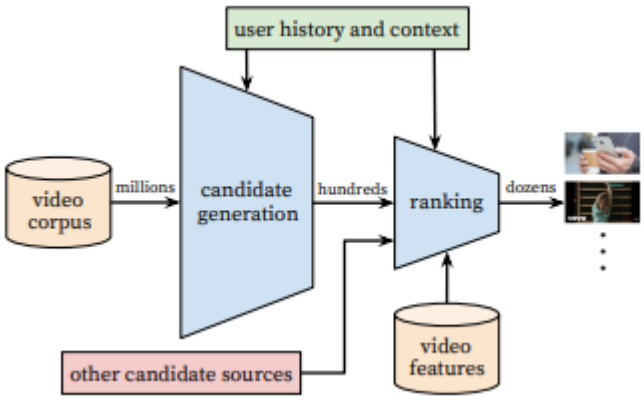


Figure 8 Recommendation system architecture demonstrating the "funnel" where candidate videos are retrieved and ranked before presenting only a few to the user. *[12]*

The recommender system is split into two parts. The candidate generation part which narrows down a large number of videos from the user's history and gets a small subset of relevant videos to the user. This process uses collaborative filtering. Users' similarity is calculated by which videos they watched, demographic and other searches. The second part the Ranking works by 'assigning a score to each video according to a desired objective function using a rich set of features describing the video and user.' [12] This is achieved by application of deep neural networks using logistic regression.

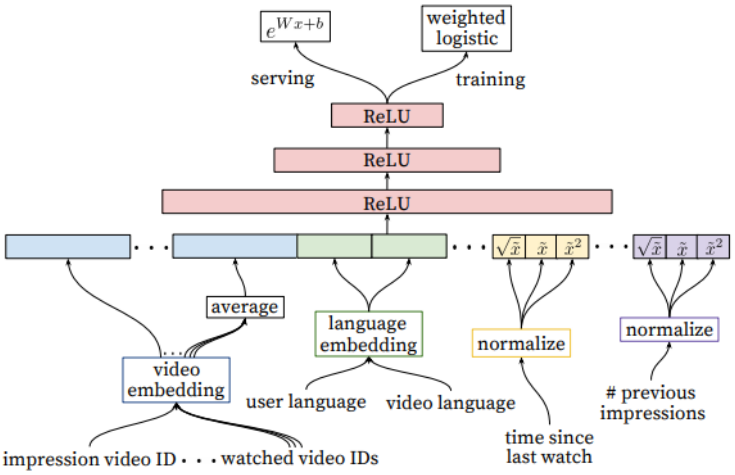


Figure 9 YouTube's Deep candidate generation model architecture. *[12]*

Criticisms of YouTube's Algorithm

Criticisms of YouTube's algorithm are not from the perspective of this technology being flawed or not working correctly, but on a human level [13] These algorithms that YouTube uses tend to attract us towards shocking and attention-grabbing videos that drive more clicks over a slower more considered video. Since the algorithm learns from our decision to choose an exciting video then we will get recommended more of the same. However, these videos may not just be recommended to us via simply our preference, but YouTube's internal bias to drive activity.

Guillaume Chaslot, an ex-Google software engineer, developed a program to scrutinise YouTube's algorithm. [4] Chaslot created a database of more than 8,000 videos collected by a program in the 'Up Next' section. The program searched 'Hillary Clinton' and 'Donald Trump' the exact same number of times. From these 8,000+ videos the 1,000 top recommended videos were chosen for further analysis. Of the remaining 643 videos, 551 were videos favouring Trump, while only 92 favoured the Clinton campaign. [4] The reasoning behind this 'bias' boils down to the videos supporting the Trump campaign tended to contain more sensational titles on average, for example "WHOA! HILLARY THINKS CAMERA'S OFF. . . SENDS SHOCK MESSAGE TO TRUMP" from an account called Next News Network.' Tufekci, the sociologist who several months ago warned about the impact YouTube may have had on the election, tells me YouTube's recommendation system has probably figured out that edgy and hateful content is engaging.' [4]

2.2 Chosen Approach

The product that I am making is at heart a recommender system that will make recommendations from what information the user provides to it.

For this project I have chosen to use a content-based system generating recommendations from both items and the user's profile.

The reasoning behind this method over collaborative filtering was that the collaborative filtering process derives its recommendations from users who are like you, not based on the actual preferences you have. I personally do not like this as much as recommending purely from similar content that the users interacting with One of the reasons for this is because collaborative filtering is susceptible to negative effects, such as popularity bias and over-specialisation. Even if this can lead to recommendations that are less novel and reduce content discovery.

2.3 Implementation Language and Tools

- **Python**

Throughout my research into recommendation algorithms and how they are implemented a common theme arose. Python is by far the most popular for this application and therefore provides the most support for people developing this type of system, such as myself. The use of Python in the Big Data and Machine Learning module CSC345 will help my understanding of coding machine learning algorithms in Python. Python has a large number of libraries currently available for use such as scikit-learn or pandas with large and detailed documentations.

- **Jupyter Notebook**

Instead of coding this in a text editor such as VSCode which I use for many of my programming using a notebook has many benefits for a task such as this. As we are working with quite a large amount of data, storing the data we have loaded in will greatly speed up the compile times during development. We are also able to only run parts of code which is helpful when we only make small changes at a time. Another great feature of Jupyter Notebooks is the ability to have markdown cells to explain what code blocks do.
As I will have the output inside the code these notebooks allow the user to see that content more clearly than just in a terminal window.

Chapter 3

Project Management

This section will cover the decisions I made about what Software Development Life Cycle I chose for my project and will evaluate my performance compared to my initial documents risk analysis and timeline. This section contains content from the initial document.

3.1 Software Development Life Cycle

The exciting model to use would be the **Agile** Methodology as this allows for little detailed planning and starts the development phase of the cycle earlier than other methodologies. The benefits of using agile lie with the ability to continuously adapt to market changes and set realistic targets for where the product will be within the next sprint (around 2 week) period [14] However in my case since we have no market to get input from and the requirements are not changing, we do not see these benefits that agile offers. Also being a 1-person development project having to be also the Agile leader and the coder will reduce the planning effectiveness.

To contrast to agile the next model I looked at was the **Waterfall Model**. This model has been around for much longer and was used in many projects where deadline and budget were fixed, and oversights could not be afforded. This is a much more structured model which follows a linear-sequential pattern [15].

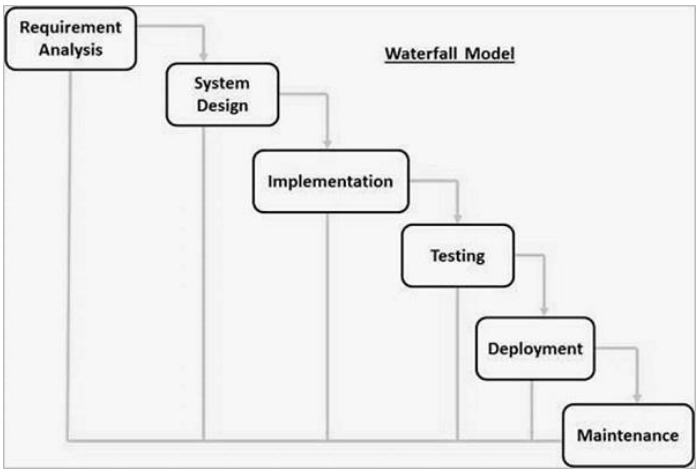


Figure 10 The Waterfall Model *[15]*

This model places a much higher importance on planning and documentation. When looking to applying this to my project it makes a good fit. The requirements will not change and gives me strict deadlines throughout the development to stick to.

However, the drawbacks of this model are relatively worrying to me. No working software is produced until late in the cycle [17].
Creating a recommender system is a task that I have never attempted before therefore exacerbating the risk of the software not working possibly past the risk assessed time, cutting down the time I will have to rectify this. Another drawback for me is while the requirements will not change the method in which I look to implement could change down the line.

The final software life cycle I researched was the Iterative Model. This model starts with the complete system requirements being clearly defined and understood. Then from there the first iteration of the project is built. This is then tested and checked for validation of requirements. [16]

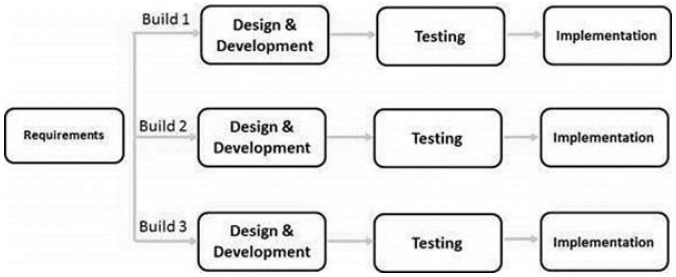


Figure 11 The Iterative Model *[16]*

A benefit of this model over the other is that if the developer needs to learn a new technology this can easily be incorporated in the next iteration. In addition to this when there is a high-risk feature, we can mitigate some of the damage caused by quickly changing iteration, as opposed to what would happen in the Waterfall model. A negative aspect of this model is that issues can arise in early (first) iteration as not all architecture is built meaning the first iteration could be a much greater amount of work than the others.

The Software Development Life Cycle that I have used is the Iterative Model. With the implementation of layers of systems such as having a bare bones front end and a simple content-based recommender for the first iteration. Where the iterations add more features to the algorithm this suits the direction, I would like to take this project in. An addition to that if I must change any features this should not set be as far back as the Waterfall method would. The risks that I have will be planned for and identified in my risk assessment to minimise the chance that they derail a whole iteration.

3.2 Project Milestones and Deliverables

To allow myself to keep on schedule and have something to refer to throughout my project to check that I was progressing at an appropriate speed I created a Gantt Chart.
The dates are slightly off as some of the dates that planned were moved throughout the year, but the time periods were roughly the same.

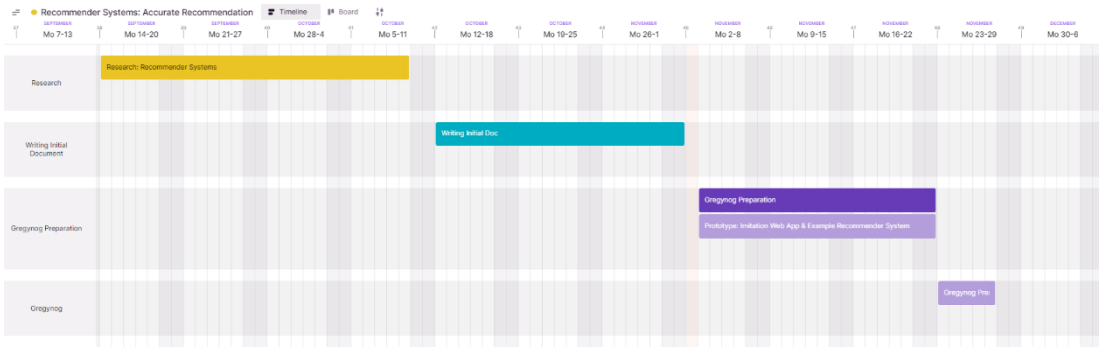


Figure 12 Gantt Chart image 1

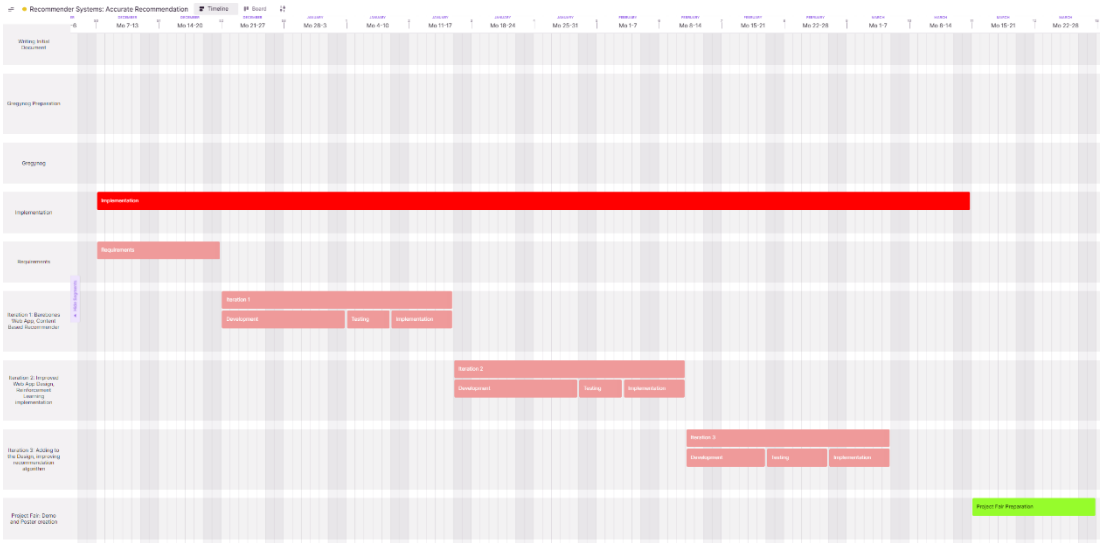


Figure 13 Gantt Chart image 2

The Gantt Chart software I have used is much better suited to a computer screen, so I have provided a link to it here.

3.2.1 Timeline Reflection

At the start of the project I had kept good time allowing the schedules two weeks for defining the Project Requirements. Moving into the first iteration was when I started on my algorithm which at the time was a music recommender. Quickly into the first iteration, around 1 week, I decided after some difficulty in finding the correct dataset for my content-based system and after starting to find a way that music's content can

be described through data, I had decided to pivot the project into a movie recommender. This decision was due to the widespread adoption of the MovieLens and movie datasets alike that have much more in-depth metadata about each movie, this is largely to do with the nature of movies being less vague and having descriptions that fully describe them.
This set me back around two to three weeks while I went through another week of defining a new set of Project Requirements.
In the final project requirements, I had defined my dataset, the IMDB dataset with enhanced cast and keywords is the one I have chosen. The approach I was using was a content-based recommender using natural language processing, with both item-based filtering and user-based filtering.

Iteration 1
In this iteration I developed a basic method of natural language processing, using the TfidfVectorizer() function from sklearn's feature extraction package and using a cosine similarity matrix function to generate a similarity value between the movies in the dataset. This was calculated based upon a movies overview. Then I defined a get_recommendation() function which allowed a user to find the most similar movies to the one they searched for by the title.
This system worked as a base, which was the goal of the first iteration.

Iteration 2
This iteration intended to refine the method of natural language processing. This was done by using a soup of information about each movie (more information in the Implementation section). This made use of all 3 datasets I have implemented merging the main dataset and the auxiliary keywords and cast datasets.
Then using a different method of feature extraction, CountVectorizer() also from sklearn, and again using a cosine similarity matrix function to generate a similarity value between the movies in the dataset.

Iteration 3
This iteration was cut quite short and ended up altering the final product. The original aim was to create a basic website that allowed users to search, add to a list and then receive recommendations of what movies are like their selections. As this Iteration was cut down to less than one week before I began writing this dissertation and preparing for the project fair. I decided to implement what I had intended to in a website in a Jupyter Notebook instead. I arrived at this decision because when starting to use Flask[7] I found that I could create something very basic but making things more complex was going to take too long and felt that it would not improve overall functionality of the recommender system. The whole point of the project.

[7] Flask is a lightweight Python web framework.

3.2 Risk Analysis

Risk	Chance	Severity	Prevention
Falling behind planned schedule	Low - The schedule I have allocated myself is quite lenient and allows for flexibility also	Low-Medium - This is very dependent on which part over runs and by how much	Regularly checking my Gantt chart timeline to keep on top of where I should be in the project
System does not perform as expected	Medium - The intention is to enhance my algorithm with Reinforcement Learning, this is quite a difficult concept to understand, one which I am not familiar with.	Medium-High - Creation of a recommender system is common there are many resources that can help, the difficult task lies with improving the base algorithm. However if this does not work the project is not going to be a success	If one method of recommending does not work there are many others to implement. Using online resources to help with this implementation
Compatibility Issues	Low - I am aware of issues relating to computer hardware and database size, also compatibility of browsers	Low - Can easily be rectified in most cases. Potentially Medium if this is an algorithm issue	This is quite resolvable with reducing database size and many browser issues are handled by web development frameworks. Also testing on a laptop is something that I can do as opposed to a desktop PC.
Lack of background knowledge	Low - I have spent a large amount of time researching which methods to use and what my product should look like	Low - Can always learn while developing the product	Spend time before implementation making sure that the methods I have chosen are the correct ones for this project. Meeting with supervisor for guidance

Risk	Chance	Severity	Prevention
UI is not up to standard	Low - While I have not coded an interface before I have designed several web pages	Medium - since the user will only see the front end this is key to allow them to interact with this software effectively	Online tutorials/resources such as templates to improve the UI
Software is broken to an irreversible state	Low - Since I am working with completely new software to me this could happen upon implementing more features, however with modern IDEs and use of version control stops this in most cases	Medium - If I was unprepared then this would cause a massive issue later on in the project	Using git to store working versions of the code on a site such as github.
Gold Plating	Medium - I have chosen some quite ambitious implementations such as django or flask and reinforcement learning this could lead me into having to add features I did not even intend to	Medium - These being too difficult could derail the project causing major implications to the final products quality	Making sure my requirement phase of implementation is realistic

I am going to explain some of the key risks, whether I was able to prevent them through my suggested methods of prevention and if not then what was the outcome of them. I will also explain if there were any other risks that I did not identify at the start of this project that occurred.

The first risk I will look at is the risk of falling behind. I initially rated this as having a low chance of happening, however it did. I think my prevention strategy of regularly look at my Gantt chart was effective in letting me know I was behind however, the actions I had to take because this happened were also more sever than the ‘low’ rating I gave it. For example, having to alter the front-end presentation of the recommender system.

The second risk I will examine is going to be Gold Plating. I rated this as medium which looking back probably was a mistake as both technologies that I mention here did not end up making it into my final program. As for reinforcement learning this came down to not having enough time with a working system where I could model user interaction and therefore not generate an action and feedback loop between the user and the program which is a key part of Reinforcement Learning. Another reason for reinforcement learning being omitted was simply I had not anticipated how difficult this was going to be to understand, much of my reading had been based upon theoretical sources not resources detailing how these systems are built in Python. I also did not create a website as I was behind and felt that a sufficient job could be done in a notebook.

The final risk I will look at is Compatibility Issues, I rated this as a low-risk issue. This was initially going be an issue as the first system that I created used a very large amount of RAM, over 23GB on my system. The prevention strategies that I have included here worked. By only considering movies with a certain amount of user votes this allowed the ram usage to be at around 3GB for the top 20%.

However, there were some risks that I had not anticipated. I had not considered the large workload of my 3rd year modules to jeopardise the amount of time I could put into the project, but I potentially should have considered this when creating my Gantt Chart.
Illness was a risk that I did not feel would be a barrier to this project however this did impact the final iteration of my project, which was disrupted. This was a risk that I felt would not affect me, having no issues like this in the past however building more time before the end of the project could have mitigated this.

3.3 Review of Project Aims

In my initial document I had defined aims for my project.

- **Create an accurate Recommender System.**

This aim was completed fully. As this was the title of the project this aim is pretty much the whole purpose of the project. I feel that this was completed sufficiently with the implementation I have provided.

- **Implement a machine learning algorithm to aid recommendation.**

The second aim was completed, albeit with a slightly different method that the one intended. When writing this aim in my Initial Document I specifically mention “a sophisticated machine learning algorithm such as reinforcement learning.”.

However, my project makes use of the arm of machine learning known as Natural Language Processing.
I would consider this aim as mostly complete. I state that it is to “aid recommendation” but it really is the whole system rather than just aiding it.

- **Provide a front-end interface for a user to interact with.**

The final aim was completed. Much like the previous aim was adapted along the development cycle. This originally was going to be a website but is complete to the same effect with a Python Notebook.

I have developed new aims for the project at the start of this document. I can see how they have been updated and adapted to the various obstacles and different directions I have taken this project.

Chapter 4

Implementation

This section will describe how the recommender system was developed, with examples of code and my explanation of the development choices throughout.

4.1 Program Architecture

This project is using Python, specifically Python 3. I think this decision is justified since Python is very much the most popular programming language for Data Science and Machine Learning. This is largely due to the concise syntax which is understandable to those who do not necessarily have a background in programming. Additionally, there is an extensive selection of packages and libraries specifically designed for the type of task that is currently at hand.

4.2 Data Preparation

Obviously for a project of this kind good data preparation is going to be key in the success of the ability to create a system that can give us an output that is valuable to us.
To process my data from the datasets I am using pandas package to aid with data manipulation. This package helps with data preparation both from the dataset files, which are .csv files in this case, but also when using dataframes to manipulate data and extract relevant features from the data where necessary.

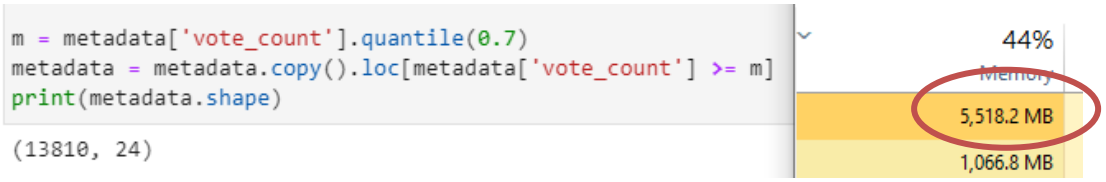
Loading the data from the dataset into the program is done via the pandas function .read_csv().

```
metadata = pd.read_csv('movies_metadata.csv', low_memory=False)
credits = pd.read_csv('credits.csv')
keywords = pd.read_csv('keywords.csv')
```

[Code screenshot of loading the data from the dataset into the notebook]

The option to set the low_memory to False (from the default True) is to prevent mixed type interference.

The next step of data preparation was to select the movies with the top 30% of votes, this is 13810 movies. This provides a large sample of the data while only omitting movies that many users will not be familiar with.



[Screenshot showing the 30% sample size of the dataset and the RAM usage based on 4 movies selected by the user]

Taking the top 30% any having the user select 4 movies the recommender requires 5.5GBs of RAM to run. While this is not an unreasonable amount, I did decide to alter this to the top 20%. This amount still gives 9151 movies which sufficiently covers many peoples tastes. Which as we can see uses much less RAM at 2.6GB which is more suitable for people who's computers only have 8GBs of RAM.

[Screenshot showing the 20% sample size of the dataset and the RAM usage based on 4 movies selected by the user]

```
# convert ids to ints, to merge datasets
keywords['id'] = keywords['id'].astype('int')
credits['id'] = credits['id'].astype('int')
metadata['id'] = metadata['id'].astype('int')

# use merge to add the credits and keywords datasets into the main dataframe
metadata = metadata.merge(credits, on='id')
metadata = metadata.merge(keywords, on='id')

features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    metadata[feature] = metadata[feature].apply(literal_eval)
```

[Code snippet showing how the auxillary datasets are merged with the main dataset]

While during the first iteration I only utilise the films overview to generate the features of each film during the second iteration I make use of other datasets to enhance the description of the movies to lead to a more detailed and accurate description of each movie. Here this code shows merging the additional data about each film into the main dataset so I can form a "soup" of information about each film from the four features listed here.

4.3 The Recommendation Algorithm

4.3.1 Feature Extraction and Cosine Similarity Generation

This section will describe the key features of the recommender system, the code that extracts the features and generates similarity values between the movies in the dataset.

Iteration 1

To extract features, I am using the TfidfVectorizer from sklearns feature_extraction package. This is a simple technique that calculates the importance of words based on the amount that they appear in the text.
I apply this to a movies overview which allows the algorithm to identify key features that each datapoint should be classed by.
The next step after generating the features for each film is to compare how similar these overviews are to each other. This is done by using the linear_kernel function from sklearns pairwaise package.

```
# create the td-idf vectorizer
# use stop words to remove redundant words
tfidf = TfidfVectorizer(stop_words='english')

# replace empty fields with an empty string
metadata['overview'] = metadata['overview'].fillna('')

# create the tfidf matrix and fit to the overview data
tfidf_matrix = tfidf.fit_transform(metadata['overview'])

# array mapping from feature integer indices to feature name.
tfidf.get_feature_names()[5000:5010]

# cosine similarity matrix for the tfidf
cosine_sim_tfidf = linear_kernel(tfidf_matrix, tfidf_matrix)
```

[The algorithm used in the first iteration of the system. The TF-IDF extracted features are used to create a similarity matrix]

This then produces an output similar to this. However, the numbers maybe slightly different as this was calculated using al different algorithm which I will cover.

```
[[1.         0.09534626 0.1        ... 0.12909944 0.1        0.        ]
 [0.09534626 1.         0.         ... 0.         0.09534626 0.        ]
 [0.1        0.         1.         ... 0.12909944 0.1        0.11952286]
 ...
 [0.12909944 0.         0.12909944 ... 1.         0.12909944 0.        ]
 [0.1        0.09534626 0.1        ... 0.12909944 1.         0.        ]
 [0.         0.         0.11952286 ... 0.         0.         1.        ]]
```

[A screenshot of my 2nd cosine similarity function's output]

Iteration 2

While I was able to create a reasonably accurate system using a movies overview using the technique I outline above, I knew there was some missing data about films such as the cast of the film, the director or genre keywords.

This led me to investigate the Bag-of-words model. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity [17]
This was perfect for the dataset that I was using since it contains enhanced metadata from separate datasets.

```
def create_bag_of_words(data):
    return ' '.join(data['keywords']) + ' ' + ' '.join(data['cast']) + ' ' + data['director'] + ' ' + ' '.join(data['genres'])

metadata['bag_of_words'] = metadata.apply(create_bag_of_words, axis=1)
print(metadata['bag_of_words'].head(1))
```

```
0    jealousy toy boy tomhanks timallen donrickles johnlasseter animation comedy family
Name: bag_of_words, dtype: object
```

[My function to define the multiset of information about movies in the dataset. This adds a new field to each of the movies.]

We see an example bag of words for movie at index 0. I think you can tell from its description that it is one of the Toy Story films. Comparing this to the overview we get a better way of detecting movies with similar datapoints such as the same actors, genre, or director.
This greatly improves the Natural Language Processing and Cosine Similarity algorithm that I have implemented for iteration 2.

```
# using CountVectorizer to get the counts of each keyword
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(metadata['bag_of_words'])

# generate similarity matrix
cosine_sim_count_matrix = cosine_similarity(count_matrix, count_matrix)
```

[The algorithm used in the second iteration of the system. The CountVectorizer is used on the bag of words to form the cosine similarity matrix]

This makes use of more sklearn packages, CountVectorizer from feature_extraction and the same cosine similarity function as used in Iteration 1.
The reason for the change from TFIDF is that we do not now need to extract words based on the frequency, all terms in the bag_of_words should be significant (if not we also have the stop_words flag). This removes this need for any inverse document frequency.

By this point we have a cosine matrix for each movie. We can find the points that have the most similar then feed them to the user.

4.3.2 Creating Recommendations

Now we have similarities between each movie we need a way of searching for the movies that we want to see recommendations for. To make this useful for the user the ability to search via text is going to be needed. Also, for the population of the user's profile and their preferences also searching via the index is something I found I needed.

```
def get_recommendations(title, cosine_sim=cosine_sim_tfidf):
    # find the index for the film searched by title
    idx = indices[title]

    # get similarity values of movies with current movie
    sim_values = list(enumerate(cosine_sim[idx]))

    # sort by highest sim value
    sim_values = sorted(sim_values, key=lambda x: x[1], reverse=True)

    # get top 10 most similar
    # 1 is going to be the movie its self
    sim_values = sim_values[1:11]

    # get the movie indices
    movie_indices = [i[0] for i in sim_values]

    # return the titles of the 10 most sim movies
    return metadata['title'].iloc[movie_indices]
```

[Code screenshot showing my get recommendation function]

This function gets the top 10 most similar movies and returns the titles of those. This implementation allows the user to search by the movie's title.
From the title we are able to find the index which allows for more direct targeting of the particular record.
The sim_values variable contains the decimal value of similarity then is sorted having the highest at the to signifying that it is the most similar. Using the range only getting the movies indexed at 1 to 11 gives us the top 10 most similar movies, starting at 1 to not include the movie searched for as it will have a similarity of 1.

This function also takes a cosine similarity parameter which is useful as later when creating a user row, we will have to use a cosine similarity with a different dimension.

This is very basic implementation, just searching for 1 film at a time does not provide the depth needed to create a particularly useful system on it's own.

Users Movie List

Inside a similar function to the one above I have added the ability for users to build up a list of movies that they like.

```
movies = []

searched_movie = idx

picked_movies.append(searched_movie)

print()
print("Your List")
for i in picked_movies:
    print(metadata[(metadata['index'] == i)]['title'].to_string(index=False))
print(f"\nMovies similar to {title}")
```

[Code for implementing the user movie list]

When a user selects a movie, they add it to their list of movies they like. This is done simply by adding the index of the movie to the list. The contents of the list is then printed out to the user so they can keep track of what movies are in their list.

```
Your List
 Drive
 Pulp Fiction
 Fight Club
```

[An example of how the user's movie list is displayed to them]

User Input

To allow this system to be interactive the user must be able to type their own preferences in and for the program to handle all the generation of recommendations and addition to the users List without them having to do any of this manually.
I created a method that handles all the users I/O[8].

```
def user_input():
    #cont is true for the program to keep looping
    cont = True
    while cont:
        print(f"Selection")
        user_input = input("Enter a movie (type 'done' when finished): ")
        #when the user is done they can end the loop and recieve output for all their preferences
        if (user_input.lower() == 'done'):
            cont = False
        else:
            #gives the top 5 most similar and adds the searched for movie
            print(f"{get_recommendations_top_5(user_input, cosine_sim_count_matrix)}\n")
```

[User input method, takes user input and passes this to the recommender system]

This allows for the user to enter as many movies as they wish to both add to their movie list of preferences and receive the top 5 most similar. I chose only 5 as having 10 for each made the output quite long.

```
Selection
Enter a movie (type 'done' when finished):  Fight Club

Your List
 Drive
 Pulp Fiction
 Fight Club

Movies similar to Fight Club
 The Curious Case of Benjamin Button
                                Stone
          The People vs. Larry Flynt
             A River Runs Through It
                                Babel
```

[An example of the what the user will see when inputting in a movie. In this case the user is entering their 3rd film of Flight Club]

This method is then called by the main method where various errors are handled, showing readable error messages that safely allow the code to continue when mistakes are made.

[8] I/O – Input and Output

Creation of a User Profile

```
def main():
    while True:
        try:
            user_input()
            print()
        except KeyError:
            print('Oops this movie is not in the database.\n')
            continue
        except ValueError:
            print('There has been an error with getting this movie.\n')
            continue
```

[Main method handling errors and calling the user input method]

So far, all the implementation has only considered Item-based filtering by only providing recommendations based on singular items. While the system is entirely proficient at doing this having User-based filtering adds some of the features used by recommender systems that you encounter every day.

Just as we used the multiple column sources to form the bag of words in Iteration 2 of the natural language processing and cosine similarity, we form the user profile recommendations in the same way.

```
joint_bag_of_words = []

print("-------------------------------------------------------------")
print()
print(f"Your List, generating recommendations from the following...")
for i in picked_movies:
    joint_bag_of_words.append(metadata[metadata['index'] == i].bag_of_words.item())
    print(f"    {metadata[metadata['index'] == i].title.item()}")

joint_bag_of_words_str = " ".join(joint_bag_of_words)
```

[Gathering the bag of words data from the users selected data]

To create this user profile, we create a joint bag of words array which we populate with the bags of words of the user selected movies.
Then we use this to print the users list of movies they have selected as a status update so the user can understand what the program is doing.
Finally, so that the joint bag of words is usable by the recommendation function I join the array values and separate them with a space to form a string.

To generate similarity between movies and a user I have created an additional data row. This is essentially a movie so that we can use the same technique to generate the most similar movies to those of which the user has selected.

```
this_index = metadata.shape[0]

new_row = {'index': this_index, 'title':'USER', 'bag_of_words':joint_bag_of_words_str}
metadata_2 = metadata.append(new_row, ignore_index=True)
```

[Creation of a new user row in a copied data frame]

Displaying Data to User

The final part of the program is to show the user their recommendations. In the User Profile section, we see how the program handles showing the user their movie list, however I will be showing the Item-based and User-based recommendations.
This is an idea that Netflix uses by having “Because you liked” and "Recommendations for You based on your User Profile".

```
if (len(picked_movies) > 1):
    print()
    print("------------------------------------------------------------\n")
    print("Recommendations for You based on your User Profile")
    print(get_recommendations_for_user(cosine_sim_3))

#Because you liked, the individual movies most similar
for i in range(len(picked_movies)):
    print()
    print("------------------------------------------------------------\n")
    print(f"Because you liked{get_movie_by_index(picked_movies[i])['title'].to_string(index=False)}")
    getrec(picked_movies[i])

print("------------------------------------------------------------")
```

[Showing how the data is shown to the user]

When the user only has indicated they like 1 movie we do not have enough data to implement the User-based, so we only look at the single movie they have interacted with.

```
 Selection
Enter a movie (type 'done' when finished):
```

[The user input section where the user can enter a movie to add to their list and get recommendations from]

```
------------------------------------------------------------

Your List, generating recommendations from the following...
    Drive
    Taxi Driver

------------------------------------------------------------

Recommendations for You based on User Profile
                        title                        genres
                        Drive  [drama, action, thriller]
                  Taxi Driver             [crime, drama]
                 Mean Streets             [drama, crime]
            Only God Forgives   [drama, thriller, crime]
                    Paparazzi  [action, drama, thriller]
       The Limits of Control   [crime, drama, thriller]
                        Trash  [adventure, crime, drama]
                       Casino             [drama, crime]
                   GoodFellas             [drama, crime]
        The Long Good Friday     [crime, drama, action]
 The United States of Leland             [crime, drama]
                      Bronson     [drama, action, crime]
                Gangster Squad     [crime, drama, action]
                   The Bag Man   [crime, drama, thriller]
              Falling in Love          [drama, romance]

------------------------------------------------------------
```

```
------------------------------------------------------------

Because you liked Drive
                    title                        genres
                Paparazzi  [action, drama, thriller]
      Only God Forgives   [drama, thriller, crime]
              14 Blades  [drama, action, thriller]
                 Dragon  [drama, action, thriller]
          Age of Heroes  [drama, action, thriller]
             Final Girl  [action, drama, thriller]
 The Limits of Control   [crime, drama, thriller]
       The Killing Room          [drama, thriller]
        Seeking Justice  [action, drama, thriller]
               Catch.44  [drama, action, thriller]

------------------------------------------------------------

Because you liked Taxi Driver
                        title             genres
                  Mean Streets     [drama, crime]
                       Casino     [drama, crime]
                    GoodFellas     [drama, crime]
                Falling in Love   [drama, romance]
                  A Bronx Tale     [drama, crime]
                   Raging Bull            [drama]
 The Last Temptation of Christ            [drama]
   Once Upon a Time in America     [drama, crime]
                The Deer Hunter       [drama, war]
                      Cape Fear  [crime, thriller]

------------------------------------------------------------
```

[Example user output based on the movies Taxi Driver and Drive]

While I had initially planned to create a web page to display this data, I feel that this provides the user with the same amount of information.

4.4 Testing

For this software there are not too many moving parts that the user could break meaning testing was not a massive task. However, like every program there are things that can cause errors such as incorrect inputs not being handled correctly or unacceptably high resource usage.

4.4.1 Performance Testing

This program is more RAM intensive than CPU intensive, this is likely to do with the size of the dataset and the values we need to calculate between each datapoint. Something to consider is not how this runs on my machine but also that of a common laptop computer in 2021. I estimated that this is probably a dual core processor with 8GBs of RAM so made my goal to allow this program to run easily on a system of these specs.

Initially my program was using the whole ~46000 records when looking for similarities. This used pretty much all available RAM on my machine (around 25GB) which was clearly unacceptable.
This was rectified by taking a sample of movies with the top 30% of votes on IMDB. Then further cut to top 20% as it had minimal effect on the quality of the system.

4.4.2 User Input Testing

As the user will be entering data themselves there needed to be some error handling which I realised during the testing period.
The way my implementation works the user does need to enter the name exactly as it is in the database, down to the capitalisation. This was something I would have liked to have more intuitive but had only realised this quite late on into the project.

When a user entered a movie incorrectly the algorithm would just end with a KeyError error. This was fixed using error handling in the main method.

A similar error arose due to the dataset having some malformed data such as having the same movie twice even after duplicate removal. This error was resolved also by error handling to stop the program abruptly ending due to a ValueError.
This is likely due to there being multiple records with some missing details with the exact same title so the algorithm cannot get the single index value.
The fixes also resolve the issue that when a user made a mistake their profile was reset.

This is evidenced in the User Input section of Implementation.

4.4.3 Output Testing

This part of the program did not have many issues while I was testing, mainly because it is quite simple. Issues with this component mainly were around improving readability and showing the data to the user in a what that would make sense.

Improvements and fixes included making sure the output was consistent no matter how many selections the user made.
Also, when testing I decided to show the genre of the recommended movies which adds some more context about each film.

Chapter 5

Results

This chapter I will evaluate the project as a whole. How the final product compares to the product that I had envisioned at the start, comparing the capabilities of the product to similar services and state if there are any improvements I could make to the final product in the future.

5.1 Project Achievements

The main aim of this project was to create a ‘recommender systems to make accurate recommendations to users based on their profile, exploration history, etc’ as per the project description. I feel that my program fulfils this with the application utilising machine learning methods of natural language processing and cosine similarity matrices to distinguish and compare data from a dataset accurately.

In my initial document I use the example of reinforcement learning as an algorithm that I could implement. I think my ideas towards this were slightly grandiose as when I started coding the project, I realised this may not be possible with the content-based approach that I had chosen and with my level of experience in this area of machine learning.
I also state that for the front end I will be developing a webapp. I think this would have been achievable with more time, however my main goal was to create the recommender system that created useful recommendations. This program will allow for that in a web app I simply just ran out of time to implement it.

5.2 Comparison

Comparing this project to existing uses of recommender systems in products we use is a slightly tricky task, many products use vastly different datasets and have much, much more sophisticated systems working behind them. Also, as many systems are not looking to generate their recommendations just from the content but also from what others have rated the content i.e., collaborative filtering they are not easily comparable to my system.

Comparing to the IMDB ‘More Like This’ section which states it uses “titles listed in the 'More like this' section are generated from a variety of information, including genres, country of origin, actors, and much more” [18]

We see quite a lot of different films recommended, this could be due to my algorithm having more bias towards the director, actors or specific genres ignoring some content which I feel discourages/encourages people from choosing films such as user rating, date released and popularity.

I think largely the success of this algorithm is, can it recommend a movie that has similar content to the movie that is searched? Yes. It does this in a different way to other products and that is something that I am very happy about.

I will say that the features are a little lacking in the product I have produced, when comparing to what is available on the market right now, however many of these algorithms are created by teams of highly experienced individuals compared to my solo project with little background knowledge before this academic year.
The features could be improved by embedding the functionality into a web app with user profiles and a better visual interface.

5.3 Future Work

Following this project, I would like to improve and tweak this algorithm when I have had feedback on its performance from others. As I can see some features that could be slightly over tuned in the current version.

Another feature I would like to implement is instead of a webapp is an API. This was something I came across when looking into developing a webapp for a machine learning project using flask. I feel that this perfectly suits the direction I would like to take this project. This would allow people to implement this algorithm without having to use a webapp that I have created increasing the application of this algorithm.

Next the option to apply this algorithm to a music dataset is something that I want to do. In my opinion music apps such as Spotify rely too heavily on collaboratively filtered playlists which causes many people to get recommended music they do not like and causes them to be categorised by the algorithm. This is the perfect application of a purely content based system.

Finally, this project has inspired me to pursue other avenues of machine learning and data science. During the research phase seeing a lot of interesting insights that can be generated from data has made me curious about what I can achieve in this field.

Chapter 6

Conclusion

To conclude, I deem this project a success. The task at hand, creating a content-based recommender system program that can accurately recommend users movies when given user preferences, was definitely a success in my opinion.

Throughout this project my proficiency in Python have increased by a lot, only having used it in two university modules before this. The experience with working with multiple machine learning packages has been useful, as well as using Jupyter Notebooks which are very common and useful in the field of data science and machine learning.

This project has opened my eyes to how much content users consume and the vast amount of that is recommended to us via these recommendation algorithms. Every time I open an app, I can now recognise the small questions and how my actions really shape the experience and the content I see.

Chapter 7 References

[1] J. K. a. J. Riedl, ““Recommender systems: from algorithms to user experience”,” *User Model User-Adap Inter 22,* p. 101–123, 2012.

[2] D. NIELD, “All the ways google tracks you—and how to stop it,” 2019. [Online]. Available: https://www.wired.com/story/google-tracks-you-privacy/. [Accessed 2020].

[3] C. M. S. N. S. N. Ian MacKenzie, “How retailers can keep up with consumers,” 2013. [Online]. Available: https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers. [Accessed 2020].

[4] E. M. Paul Lewis, “How an ex-YouTube insider investigated its secret algorithm,” 2018. [Online]. Available: https://www.theguardian.com/technology/2018/feb/02/youtube-algorithm-election-clinton-trump-guillaume-chaslot.

[5] E. H. Michael Fleischman, “Recommendations Without User Preferences: A Natural Language Processing Approach,” USC Information Science Institute, California.

[6] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie and Z. Li, “DRN: A Deep Reinforcement Learning Framework for News,” Pennsylvania State University; Microsoft Research Asia, 2018.

[7] Google, “Content-based Filtering,” 2021. [Online]. Available: https://developers.google.com/machine-learning/recommendation/content-based/basics. [Accessed April 2021].

[8] G. Jawaheer, M. Szomszor and P. Kostkova, “Comparison of Implicit and Explicit Feedback from an,” City University London, 2010.

[9] J. Han, M. Kamber and J. Pei, “Getting to Know Your Data,” 2012.

[10] Google, “Collaborative Filtering,” Google, 2021. [Online]. Available: https://developers.google.com/machine-learning/recommendation/collaborative/basics. [Accessed April 2021].

[11] D. Jackson, “The netflix prize: How a 1 million contest changed binge-watching forever,” thrillist, 7 July 2017. [Online]. Available: https://www.thrillist.com/entertainment/nation/the-netflix-prize. [Accessed 2020].

[12] P. Covington, J. Adams and E. Sargin, “Deep Neural Networks for YouTube Recommendations,” Google, Mountain View, CA, 2016.

[13] B. Popken, “As algorithms take over, YouTube's recommendations highlight a human problem,” NBC News, 19 April 2018. [Online]. Available: https://www.nbcnews.com/tech/social-media/algorithms-take-over-youtube-s-recommendations-highlight-human-problem-n867596. [Accessed 2021].

[14] “SDLC - Agile Model,” tutorialspoint, 2020. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm. [Accessed 2020].

[15] “SDLC - Waterfall Model,” tutorialspoint, 2020. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. [Accessed 2020].

[16] “SDLC - Iterative Model,” tutorialspoint, 2020. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm. [Accessed 2020].

[17] Wikipedia, “Bag-of-words model,” *Wikipedia, the free encyclopedia,* 2021.

[18] IMDB, “What is the 'More Like This' section?,” IMDB, [Online]. Available: https://help.imdb.com/article/imdb/discover-watch/what-is-the-more-like-this-section/GPE7SPGZREKKY7YN?ref_=cons_tt_rec_lm#.

[19] F. Isinkaye, Y. Folajimi and B. Ojokoh, “Recommendation systems: Principles, methods and evaluation,,” *Egyptian Informatics Journal,,* vol. 16, no. 3, pp. 261-273, 2015,.

[20] X. Zhao, “Cold-Start Collaborative Filtering,” UCL (University College London), 2016.

[21] K. Luk, “Introduction to TWO approaches of Content-based Recommendation System,” 3 February 2019. [Online]. Available: https://towardsdatascience.com/introduction-to-two-approaches-of-content-based-recommendation-system-fc797460c18c. [Accessed 2020].

[22] A. Sharma, “Beginner Tutorial: Recommender Systems in Python,” datacamp, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/recommender-systems-python.