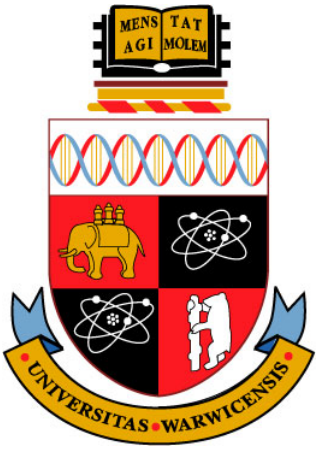


Revolvr: A Crowd Sourced Content Aggregation and Suggestion System

CS310 Computer Science Project

Project Report

Jordan Wyatt

Supervisor: Dr. Matthew Leeke

Department of Computer Science
University of Warwick

2015-2016

Abstract

The modern world is vastly connected, with close to half of the population having access to the internet. With such a rapidly evolving digital landscape also comes an influx in the quantity, accessibility and consumption of online media. Given such vast quantities of data how can user identify relevant and desirable content tailored to their tastes? This paper outlines the development of a web-based media aggregation and recommendation system named Revolvr, which strives to be a hub for discovery encapsulated in a uniquely accessible and comprehensive system. Via the collection of preferred media from multiple third-party sources the system aims to create a *media footprint* from which multiple models of inference can be used to generate meaningful recommendations, with a focus on *asymmetric feedback*. The developed system provides promising results as a result of a lengthy testing and evaluation process and via a combination of academic, technical and research focussed discussion this paper aims to summarise the incarnation of a flexible recommender system.

Acknowledgements

I would like to gratefully acknowledge a number of individuals who have shown support throughout the development of the project. Firstly, I must give my utmost thanks and an enormous debt of gratitude to my project supervisor Dr. Matthew Leeke. His guidance, patience, encouragement and insight both throughout the project and my time as a student have been invaluable. I believe that without his help I would not be in the position I am today. I have been extremely lucky to have a supervisor who takes so much pride in the work and personal development of his students, and working alongside him in academic and personal ventures has been an absolute pleasure. I must also extend my thanks to Dr. Nathan Griffiths for the time taken in marking my work, and providing valuable insight during the presentation. I would further like praise the willingness and insight from my colleagues and friends especially Naqash, Isheanesu, Josh, Luke and Cem for their suggestions, feedback and general support throughout the last 9 months. Furthermore, an application such as Revolvr cannot exist without its users and appropriate test data and as a result I thank all the individuals who signed up the system and took their time to provide invaluable data and general evaluations.

Finally, I thank Hollie for her endless reassurance, support and care – I couldn't have done it with you and will always be grateful.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The modern world is vastly connected, with close to half of the population having access to the internet. As of January 2015 there are 3.010 billion active internet users, with 2.078 billion of these users having active social media accounts [6]. These figures are the product of a year-on-year growth, which saw a 21% increase in the number of active internet users, and a 12% increase in the number of active social media accounts in the space of 12 months. With such a rapidly evolving digital landscape also comes an influx in the quantity, accessibility and consumption of online media. However, with such easy access also comes the problem of over saturation. Given such vast quantities of data how can a user identify relevant and desirable content for their tastes?

1.1 Motivation

As discussed in Christopher Chamberlain's previous work on Revolvr, consumers of online media are conscious of their *online footprint* [4]. With monumental developments in mobile technology in the past decade mobile and tablet devices now boast a 38% share of worldwide web traffic [6]. As a result, access to the web and its media is easier than ever before and the development of an online footprint is almost unavoidable. The sources of such content vary, from general purpose social networks such as Twitter and Facebook to format specific platforms such as Spotify and Netflix, a user has a wide range of options and discovering content they find relevant often feels like finding a needle in a haystack. The focus on online consumption is only emphasised via industry support and developments with Kanye West recently releasing his highly anticipated album 'The Life of Pablo' exclusively on TIDAL, a streaming service aiming to push *'high fidelity music streaming'*. Furthermore, 70% of North American 2015 downstream traffic was streaming of audio and video in 2015, with 37% of this traffic belonging to Netflix [7]. It is clear both the consumer and the creators are invested in this model, but as a result an abundance of both media sources and items exists. However, this ocean of content can often seem like a flood, with users having

to manage their media on multiple platforms and jump between providers. With so much time being spent organising there is often no time for discovery, and although individual providers attempt to recommend content internal to their service they all seem to ignore a user's whole taste profile, instead only working within their respective closed ecosystem. Revolvr was developed as an attempt overcome this problem, with its documentation stating *'why settle for what you've already seen, when technology can steer you towards something you may love, perhaps before you know it?'* [4]. However, limited development time and ensuring the delivered product met the proposed requirements left the software more as a proof of concept, as opposed to an effective content aggregator and recommender. The existing work provided a good basis to allow not only improvements to the recommendation algorithm and UI, but also research opportunities to evaluate variations of recommendations algorithms and explore contrasting areas of recommender systems and machine learning.

1.2 Problem Statement

Web based media consumption is at an all time high with no signs of slowing down. With an overproliferation of media being available to users, and providers competing with one another to achieve user loyalty as opposed to collaboration consumers can often feel lost. Whilst existing services offer recommendation techniques, this only accounts for part of a users taste profile. Spotify may know a user enjoys Taylor Swift, but YouTube may have no knowledge of this and therefore the profile they aim to tailor for is often not representative of the whole user. Existing work on the system provided a playground for further exploration of the ever important topic of recommender systems, and although it provided a well rounded taste profile it did not use this to its full extent. By analysing the downfalls of the existing system, utilising the existing foundations, and exploring a number of successful recommendation algorithms explored in academia the project thrives to be a hub for discovery encapsulated in a uniquely accessible and comprehensive system. Instead of users relying on multiple systems trying to find that needle in the haystack, Revolvr aims to provide one in which the media revolves around them.

1.3 Project Stakeholders

Since the projects incarnation a number of individuals have provided their interest and personal investment within the system and can therefore be considered as stakeholders. Firstly, the project supervisor, Dr. Matthew Leeke, has played a crucial role in the project offering honest insight and suggestions regarding all aspects of the project. The previous developer, Christopher Chamberlain, and his work on Revolvr has also been valued and his opinion on the new vision Revolvr encapsulates is extremely valuable. The users also trust the system will keep their information safe, and for the developer not exploit the trust they had in the system with regards to their aggregated content being identifiable to the developer. Finally the developer himself, Jordan Wyatt, has devoted his time and effort to ensure that Revolvr is a success not just as an academic project but as a personal piece of work and investment.

1.4 Report Structure

This report can be viewed as a combination of market research, academic exploration, technical development, and project management. As a result the report will consist of three major areas: preliminary research, development, and reflection.

Preliminary Research

Sections 2-4 can be seen as preliminary research and considerations before development. Section 2 gives an introduction to recommender systems, considering a number of different recommendation models as well as examples of existing systems. Section 3 highlights the legal, social, ethical and professional issues specific to Revolvr which ensure that the product itself and those working on it act both in an expected and professional manner. Moving on from this Section 4 sets the system requirements which are constructed based on the existing foundations as well as the research undertaken in Section 2.

Production

The production of the system will be the main focus of the report, with both academic and technical content. Section 5 focuses on the design of the system, highlighting appropriate design choices and provides the ground work for a consistent and planned development cycle. Section 6 realises the content in Section 5, giving a step by step explanation of the development process in appropriate technical detail. Following this Section 7 discusses the testing of the implemented features in Section 6 via unit, aggregation, system and user acceptance testing.

Reflection

In the final sections an evaluation and discussion of the system and the development process behind it are given. Section 8 outlines the management of the project as a whole and the approach taken. Section 9 compares the initial requirements and proposals to the final product, ensuring that the functionality of the envisioned system fulfils these requirements. Finally, Section 10 concludes the project with suggestions for future work that the developer sees as beneficial.

Revolvr hopes to combine technical, academic and evaluative perspectives to become a system of streamlined discovery, in which a user is at the centre of an experience in which the media revolves around them.

CHAPTER 2

Research

In the last decade consumption of online media has sky rocketed, yet discovery becomes harder. Users are exposed to more media, and as a result are often less invested in taking the time to form a personal collection and instead focus on breadth over depth. As a result, the development of dedicated and integrated recommender systems have become an important topic in both research and revenue generation whilst also playing a large role in user opinions. Before work on Revolvr could begin it was important to consider the ways in which recommendations are commonly made, the differences between various recommendation methods, and existing solutions used by successful media providers.

2.1 Recommender Systems

Recommender systems are defined in Resnick and Varian's seminal article as follows: *'In a typical recommender system people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the systems value lies in its ability to make good matches between the recommenders and those seeking recommendations.'* [8]. They have proven to be useful means for users to cope with overload of information and are used extensively in e-commerce. Recommender systems are able to be classified according to the knowledge sources they use, as seen in Figure 2.1. This taxonomy proposes three types of available knowledge to a recommender system [1].

- **Collaborative (Social) based knowledge:** knowledge concerning other users
- **User (Individual) based knowledge:** knowledge of the individual user
- **Content based knowledge:** knowledge about the items being recommended

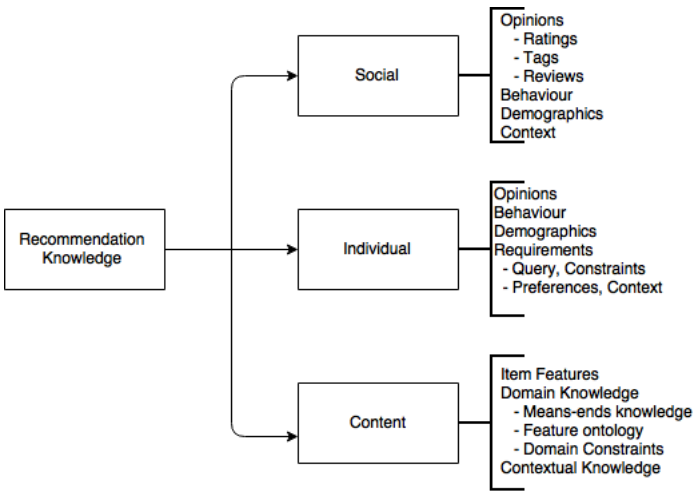


Figure 2.1: Taxonomy of knowledge sources in recommendation [1]

A recommender system may use a combination of these knowledge bases, or focus solely on a single one. This taxonomy also helps understand different recommendation techniques.

2.1.1 Recommendation Techniques

As defined in Felfernig and Burke's 2008 paper, a recommendation technique is *'a set of knowledge sources and an algorithmic approach to generating recommendations using those sources'* [1]. The most common recommendation techniques are collaborative recommendation, content-based recommendation and hybrid recommendation. Collaborative filtering is based on gathering information on a user's activities and preferences, and from this predicting what they may like based on similar users. This method benefits from not requiring prior knowledge on an item, so can generally be used without any prerequisite information. In contrast, content-based filtering uses a description of the item and a user's preferences and from this decides if the user will like the suggested item or not. In layman's terms collaborative filtering stems from the idea if two people agree they are likely to agree again in future, and content-based implies a user is likely to seek items similar to what they have liked before. Hybrid recommender systems combine collaborative filtering and content based filtering in a number of possible ways. One approach is to make predictions with each approach separately then combine the results, while in contrast you can design model which unifies the approach of both. Studies have suggested that combining collaborative and content-based filtering can be more effective in a recommender system. Netflix is a successful real world example of hybrid recommender system, collaborative filtering is implemented by comparing watching and searching habits of similar users and recommending content based on a user's previous ratings is content-based filtering.

The techniques discussed above are not exhaustive when talking about types of recommender systems, Robin Burke's 2002 paper distinguishes the recommendation techniques shown in Figure 2.2 [2]. Here **I** is the set of items over which recommendations may be made, **U** is the set of

Table I: Recommendation Techniques

Technique	Background	Input	Process
Collaborative	Ratings from **U** of items in **I**.	Ratings from **u** of items in **I**.	Identify users in **U** similar to **u**, and extrapolate from their ratings of **i**.
Content-based	Features of items in **I**	**u**'s ratings of items in **I**	Generate a classifier that fits **u**'s rating behavior and use it on **i**.
Demographic	Demographic information about **U** and their ratings of items in **I**.	Demographic information about **u**.	Identify users that are demographically similar to **u**, and extrapolate from their ratings of **i**.
Utility-based	Features of items in **I**.	A utility function over items in **I** that describes **u**'s preferences.	Apply the function to the items and determine **i**'s rank.
Knowledge-based	Features of items in **I**. Knowledge of how these items meet a user's needs.	A description of **u**'s needs or interests.	Infer a match between **i** and **u**'s need.

Figure 2.2: Recommender Techniques [2]

users with known preferences, **u** is the user who requires recommendations, and **i** is some item for which we would like to predict **u**'s preference. Further into the project exploration of the more niche methods will be discussed, but a high level understanding of common approaches taken by recommender systems is key to discussion.

2.1.2 Cold Start Problem

Recommendation systems often suffer from what is known as the *cold-start* problem. This is defined as when *'recommenders cannot draw inferences for users or items for which it does not have sufficient information'*, an issue that can effect both users and items. A new user will be given non personalised recommendations until they have rated enough items to build a profile on their tastes, and a new item in the system will not have any rating information and it is unlikely to be recommended to a user. For a user the initial recommendations are crucial on the first impression the system leaves on them, and a poor performance could reduce the appeal of the system.

Two common ways to solve the cold start problem are *association rules* and *clustering techniques*, as discussed in Hridya Sobhanam's 2013 paper [3]. Association rules focus on expanding a user profile so it contains more ratings, and clustering groups new and existing items based on similarity measures to make predictions for the item. Figure 2.3 shows one example of a flow of processes that can be performed to attempt to combat this problem. The cold start problem is a complicated problem which although detrimental is often hard to avoid, with many papers solely focusing their efforts on alleviating its effects as opposed to eliminating it as a whole.

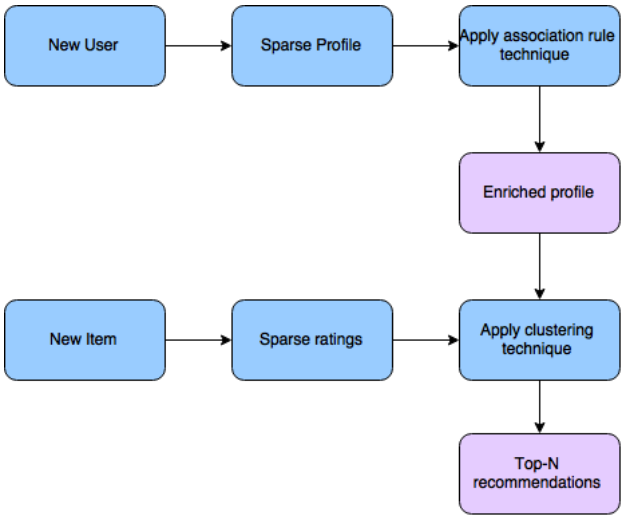


Figure 2.3: A possible cold-start problem solution [3]

2.2 Media Providers and Recommendation

As discussed in Section 1, the number of online content creators, providers, and distributors is rapidly growing. Video streaming is beginning to overtake conventional viewing methods such as cable TV with many turning to services such as YouTube and Netflix to provide video content. The same can be said for audio streaming services such as Spotify and Apple Music, with physical copy revenues declining from 60% of income in 2011 to 46% in 2014 [9].

With the commercial viability of streaming services growing, companies are making an effort to integrate recommender systems into their products. One prime example is Netflix, who in 2006 announced a machine learning and data mining competition known as the Netflix prize. This was an open competition to develop a collaborative filtering algorithm that could predict user ratings for films, with a $1 million dollar prize up for grabs. The winning algorithm offered a 10% improvement over the in house algorithm at the time and was developed by a team known as BellKor's Pragmatic Chaos [10]. The scale and rewards offered in the competition highlight the significance recommender systems play within profitable companies. Neil Hunt, Netflix's chief product officer recently stated that *"the user either finds something of interest [within the first 60 or 90 seconds] or the risk of the user abandoning our service increases substantially"*, further stressing the importance companies place on providing strong recommendations [11].

Spotify have also placed an emphasis on recommendations, with their new 'Discover Weekly' feature receiving praise. Discover weekly deploys a collaborative filtering algorithm known as matrix factorisation every week to present a 'mixtape' of 30 songs to users on a weekly basis. It works by developing a *taste profile* based on songs a user listens to / saves, Spotify then identifies similar songs that appear on the billions of playlists already on Spotify. Finally, it combines these to find songs that fit a user's profile but that they have not listened to yet. During a presentation by Spotify engineer Chris Johnson in November 2015, he discussed how since its launch in June 2015

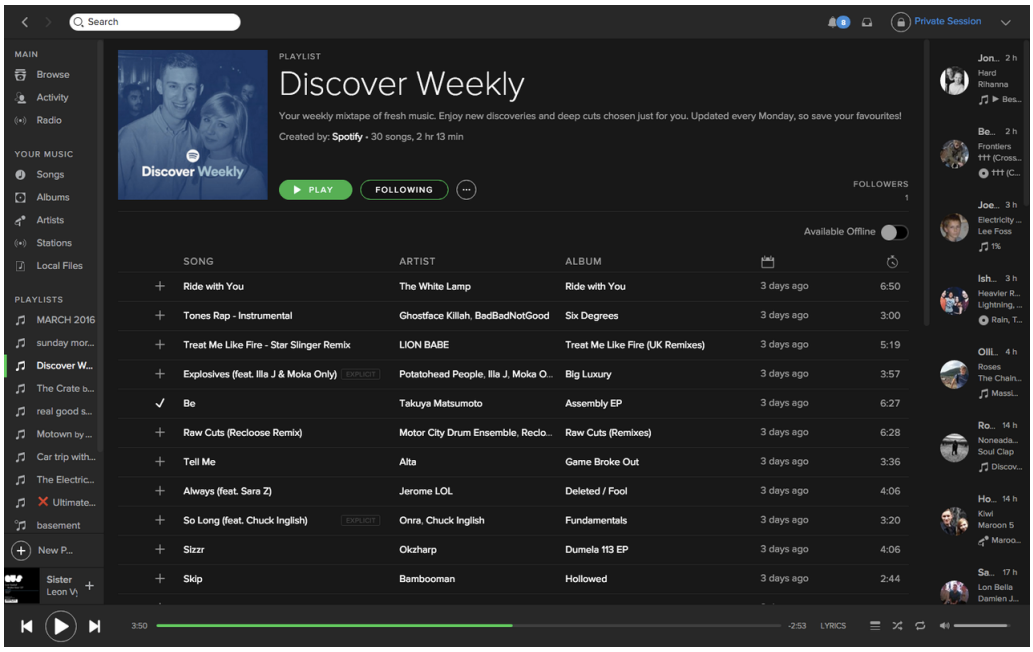


Figure 2.4: A screenshot showing the developers Discover Weekly playlist [1]

the playlists had been streamed 1.7 billion times, he also notes the performance of the algorithm and general reception by users has been extremely positive [12]. Figure 2.4 shows an example of a Discover Weekly playlist and its integration into the existing Spotify application.

These examples provide a clear indication that recommendations are important not just from a personal perspective, but also a business perspective. Providing fresh and interesting content can be the make or break of a user continuing to use your service, and in turn generating revenue. However, it is important to note the wide number of use cases that recommender systems offer. E-commerce, social networks, online dating and search engines are all examples of areas where recommender systems are deployed, even networks of people could be seen as a collaborative filtering system where the decisions and tastes of your peers often influence and guide your own.

2.3 Types of Feedback

Having considered how the recommendations are made it becomes clear collaborative filtering based on a 'taste' profile is a popular approach with Netflix, Spotify and other providers such as Soundcloud taking similar approaches. But, defining how this profile is constructed is an important consideration especially with the limitations that a system based on aggregation puts forward. One term often used to describe recommender systems that respond to a users input such as those often found in media based setups is interactive recommender systems. In his 2013 paper based on interactive personalised music recommendation, Xinxi Wang states *"A successful recommender system must balance the needs to* ***explore*** *user preferences and to* ***exploit*** *this information for*

recommendation", an idea that is employed in many large existing systems [13]. In order to make any recommendations a system requires data on user preferences, and there are two common ways to gather this. The first method is to ask for **explicit** ratings from a user usually in the form of a rating scale, a common technique used by Netflix and previously Amazon. For example, if a user enjoys action movies they may rate *Die Hard* a 5 out of 5 rating, implying movies of this style appeal to their taste profile. On the other hand this user may rate *My Little Pony* a 1 out of 5, allowing a contrast and explicit notion of what the user does and does not like. An alternative approach is to monitor user behaviour, gathering data and attempting to make deductions on preference based on this without effort on the users part, this is known as **implicit** feedback. The former can be directly interpreted as user preferences and is therefore easier to work with. However, the data collection process becomes the responsibility of the user who is likely to not want to take the time to enter ratings [14].

One example of implicit feedback being used in a production system allows discussion to return to Spotify, who have been vocal in their use of implicit feedback as a way of garnering user preferences. In a 2014 presentation Spotify data engineer Chris Johnson highlights there is a difference between "The Netflix Problem" and "The Spotify Problem" [15]. At the time of publication Netflix relied on users explicitly rating movies, a form of explicit feedback, from which preferences could be modelled. In contrast Spotify attempts to build a taste profile through streaming behaviour, using the notion that if a user has streamed a song (perhaps multiple times) this may indicate they prefer this song over another with no streams. Collection of this data can be carried out without the user diverging from their normal behaviour of consumption, and although it may be less descriptive than explicit feedback allows a layer of abstraction to be introduced allowing the user to focus on their listening experience [15]. In the presentation Johnson describes an algorithm known as *'Implicit Matrix Factorization via Alternating Least Squares'*, proposed by Hu, Koren and Volinsky in their 2008 article *'Collaborative Filtering for Implicit Feedback Datasets'*. In this paper data is treated as *'indication of positive and negative preferences associated with vastly varying confidence levels'*, and in turn propose a factor model which is tailored for implicit feedback recommenders [16].

Through the consideration of existing media recommendation systems and the approaches they take both in generating recommendations and collecting preference data the evolution of Revolvr hopes to take inspiration from its peers. By aggregating the content of multiple providers into a homogeneous store, creation of this taste profile aims to be reflective of the whole user and as a result create recommendations tailored directly on these tastes.

CHAPTER 3

Legal, Ethical, Social and Professional Issues

As with all software products, especially ones of such a personal nature, it is important to ensure the product follows standards and can be trusted by users. This section of the report will discuss the possible issues that could arise during the development of Revolvr in the legal, ethical, social and professional domains without careful planning and consideration. To assist the developer the British Computing Society Code of Practise was taken into account, the issues discussed are not exhaustive but instead are the most applicable [17]. The previous developer also ensured the work that was left adhered to these standards [4].

3.1 Legal Issues

Due to the nature of the system, Revolvr will be processing and displaying media from third party sources. A clear concern here is the topic of intellectual property and licensing. In regards to licensing, Revolvr will only be aggregating content from trusted and established sources that themselves adhere to the correct licensing laws and terms and conditions. As a result of this the system must also comply to the same standards that the third party providers do. Intellectual property will be preserved in a number of ways, firstly the API's for the aggregated services all provide watermarks on the embedded players to show the source of the content, and by following the link these watermarks provide the original uploader receives credit too. Use of official API's also ensures that the method of aggregation and processing will be in line with what the provider agrees to provide in terms of accessibility of their data. Each service also provides the ability for uploaders to disable embedding, meaning Revolvr would not display content if a creator wishes not to do so. A software license agreement will also be developed which states where the products obligations lie. Furthermore, due to using the appropriate third party players and not an embedded player all revenue streams such as advertising and stream counts will not be effected.

Although legal issues are an important topic in all products, in this case the developer and

the project supervisor are confident that by using trusted and reputable aggregation sources such problems can be avoided.

3.2 Ethical Issues

The key ethical concerns with user oriented sites is data privacy, and with media oriented products also comes the task of filtering explicit content. The former will be guaranteed by encrypting any sensitive user data, and Revolvr's philosophy is concerned with respecting the user so their data will not be passed to third parties in any circumstances. Furthermore, the use of Facebook login leaves management of sensitive data such as passwords to Facebook, meaning Revolvr never has access to a users password in the first place. The system also has no plans of introducing user based monetisation in the form of transactions or subscriptions, meaning the management of financial details need not be considered. Regarding suggested content of explicit nature the platforms Revolvr uses will themselves provide filtering of explicit content (i.e YouTube's age gate feature). This obviously is not fool proof, however this is again a situation where the system can rely on the third party content providers to do their jobs by being reliable and trusted platforms. Although this may come across as a lack of planning or carelessness, Revolvr will ensure it only aggregates from sources who the stakeholders believe fit the mould in regards to the provided content and its suitability for the system.

3.3 Social Issues

Social issues are often the ones that can make the user feel victimised, as opposed to approaching from a business or financial perspective. Revolvr will thrive for inclusivity, ensuring that perspectives such as cultural, social, gender and disability are taken into account into the refreshed design. This will be enforced by ensuring the technology is available to all, and does not cater to a specific race or gender.

Regarding race, Messing and Westwood have proposed that recommender systems often are biased and clustered to certain demographics [18]. Although this is often a naturally occurring phenomenon due to cultural differences regarding both media availability and taste, Revolvr will not take such factors into account and only provide recommendation based on the items and users themselves.

In terms of disability the system will take into account colour blind and disabled users, by ensuring multiple queues are available to promote the functions of the product and making the UI clear and intuitive. Although multiple languages are likely to not be supported due to time constraints and lack of resources the design of the UI will be approached such that both visual and language prompts are provided. Care will also be taken to ensure information on a users preferences and recommendation will only be viewable by them, as taste in media and the corresponding profile can be of a personal nature. Users should not be able to view other users profiles, or be identifiable themselves within the public facing system.

3.4 Professional Issues

Being a user oriented service it is important to value the user to create a sense of worth for the platform and a relationship of trust. As previously mentioned all data provided by a user will be kept private and only used for the service they receive. The previous developer also ensured all services included used secure channels and session tokens, this means that the connections to the providers cannot be exploited by malicious third parties if an attack were to occur. The developer will act according to the aforementioned BCS code of conduct by showing public interest, professional competence and integrity, duty to relevant authority and showing duty to the profession [17]. Regarding development practises, a professional approach will be taken ensuring that code is documented both in-line and also in the form of a system overview as given in this report. Furthermore, the development and testing of recommendation algorithms requires inspection of backend data which can often be traced back to a user. The developer promises to avoid identification of users where possible, aiming not to break the bond of trust with the test users who have agreed to assist with the project.

Following the clear definition of these issues and research into the core concepts of recommender systems the goals of the system are able to be defined. To do so, the developer has constructed a set of requirements which highlight the functional and non-functional criteria the resultant system aims to meet.

CHAPTER 4

System Requirements

Within software development and academic projects clear requirements are vital for tracking progress and providing clear development goals throughout. To ensure the simple yet engaging environment Revolvr thrives to promote a number of *functional* and *non-functional* requirements have been developed. These can be used to directly measure the success of the project by comparing the work carried out with the work proposed.

4.1 Refinement of Requirements

Over the course of the project, an agile methodology has allowed leeway with the direction of the project. As work commenced the initial vision outlined in the specification (see Appendix) slowly transformed into the system discussed in this report. As a result, the following requirements differ slightly from those proposed in the specification to reflect the projects current state and vision.

4.2 Functional Requirements

F1: *The system must access and store third-party data*

1. The system must connect to the given media services
   (a) *Facebook*
   (b) *Twitter*
   (c) *Vimeo*
   (d) *YouTube*
   (e) *Soundcloud*
   (f) *Spotify*
2. The system must obtain media information from each service on a per-user basis

F2: *The system must store and convert data from selected third parties in a common format*

1. The system must aggregate media from the following sources directly using a users liked / saved items

   (a) *Vimeo*
   (b) *YouTube*
   (c) *Soundcloud*
   (d) *Spotify*

2. The system must utilise connected Twitter accounts to extract and resolve any media links for services in **F2.1**
3. All media items must be stored in a common format
4. All media items must be identifiable as belonging to a specific third party provider

F3: *User aggregated data in the system must belong to a group of individuals, or a single individual as a minimum*

F4: *Credentials for the media sources an individual wishes to use data from must be stored*

1. A user must be able to log in to the system using Facebook
2. A user must be able to connect the services in **F1** via authentication token exchange
3. The tokens received in response to service authentication must be stored for re-authentication
4. A user must be able to revoke access to previously authenticated services

F5: *The system must provide a non-user based aggregation of the most popular content of the sources in **F2.1***

1. This content must be refreshed and viewable on a daily basis

F6: *Inference must be made to enrich input data obtained from third-parties for semantic analysis*

1. Titles must be decomposed and stored for semantic analysis
2. Similarities must be inferred between these titles to produce links / relations between users

F7: *Multiple models of inference must be available to the user*

1. The following three models must be available:
   (a) Semantic based analysis
   (b) Feature based analysis
   (c) Social based analysis

F8: *Suggestions should be personalised based on an individual‘s tastes*

F9: *The system should be able to group users, and provide suggestions based on the preferences of these similar grouped users*

F10: *The system should be able to recommend to new and unauthenticated users whilst attempting to reduce the severity the cold-start problem*

- Unauthenticated users must be able to demo the system and receive random recommendations
- Popular items in the Spotlight area must be immediately viewable
- Users must be staged in order to provide recommendations as quickly as possible for appropriate algorithms
- If recommendations cannot be generated to a new user, they must be informed why and provided alternative media

F11: *User feedback must be fed into the system to refine the inference mechanism*

F12: *The media must be consumable within the application by the user, without deducing from the experience provided by the original source platform*

F13: *Aggregated media and media subject to internal feedback must be available to explore and reconsume*

F14: *The system must provide an engaging, focused and informative way of delivering recommendations to the user*

1. Titles of media items must be provided
2. Descriptions of media items must be provided if possible
3. The viewing experience must be focused on one item at a time whilst still providing multiple recommendations

F15: *The user interface must provide appropriate functionality for a user*

1. Authenticated users must be able to view their recommendations
2. Authenticated users must be able to view popular media
3. A user must be able to save / rate media easily
4. A user must be able to log in and out
5. A user must be able to select between recommendation algorithms
6. All users must have access to the privacy policy

4.3 Non-Functional Requirements

Such a user driven, engaging system relies heavily upon a reliable and smooth user experience. Therefore, a set of nonfunctional requirements relating to both user experience and maintenance of the system are vital. This allows further work to continue due to modularity and strong system design whilst providing a seamless, enjoyable experience for a user.

NF1: *The system must be modular, to allow extensibility*

1. Components must be easily accessible, editable and replaceable
2. Each component must address a particular function

NF2: *The system should be maintainable, commented and provide strong documentation for future work*

1. A detailed description of the system and its workings must be provided
2. Complicated / non-trivial areas of the codebase must be heavily commented to promote future work and aid understanding
3. Version control system commit messages must be concise and accurate to provide further reference of functionality and system additions
4. Standard and technology specific coding practises must be adhered to
5. The production server must be kept up to date

NF3: *The system should provide a seamless user experience, more so than the original implementation*

1. The system will provide AJAX loading on the major components and interactions, avoiding page refreshes
2. Discovery of media must be more focused, allowing exploration and engagement

NF4: *The system should be designed so as to allow scalability*

NF5: *The system should be preferable for discovery as opposed to using the individual services separately*

NF6: *The system should be secure*

1. User credentials must be kept secure
2. User information must only be available to the authenticated user
3. A user must not be personably identifiable to other users, and vice versa

NF7: *The system promote concepts of usability providing an intuitive user experience*

1. The user experience must be consistent
2. A user must never see a page they are not intended to see
3. The system must respond in a timely fashion with minimal delay
4. The system must be easy to navigate

Table 4.1: Server Specifications

	Original Server	**Current Server**
Processor	Single Core	Dual Core
Memory	512MB	2GB
Disk Space (SSD)	20GB	20GB
Transfer	1000GB	3TB

4.4 Technical Constraints

With web based systems it is often the case that performant and eloquent software design is bottlenecked by the hardware it is deployed on. Table 4.1 shows the original hardware specification of the deployed environment, however towards the end of the project the setup shown in the right column of the table was instead adopted. This increase in computing power was a requirement due to the new recommendation algorithms in the system, which put more strain on the system. Although the new server specifications provide more power than the predecessor it is always optimal to obtain as much computing power as feasibly possible, and as a result the effects of bottlenecking should be taken into account.

Following a clear definition of requirements and sufficient preliminary research design of the system could occur. These preliminary steps are important to ensure that work is focussed and that there is an *end goal*, providing direction for the design of the system with succinct goals.

CHAPTER 5

Design

Following preliminary research and the construction of a set of system requirements to guide development, a process of designing and implementing required components may occur. It is important to note that Revolvr is a continuation project, and therefore the previous system provided a base implementation to work from. However, to provide a consistent narrative and representative view of the whole system this report will discuss all components in terms of a preliminary design followed by a corresponding implementation. This section will be discussed following the order of processes the system follows. Firstly, the act of data collection and aggregation will be outlined which will be followed by data processing, recommendation algorithms, and finally user interface design used to display content to the user. Due to the adoption of an agile development methodology each component will be designed and implemented before moving onto the next task, which also allows continuous testing to take place as well as the processes described in Section 7

5.1 System Architecture

The system's overall architecture and design can be found in Figures 5.1 and 5.2 respectively. The former provides information on the flow of processes and data within the system to understand how the components of the system will interact together, whereas the latter aims to provide a high level overview of the interaction between modules and technologies. The operation of the system can be split into four main stages: data collection / aggregation, data processing, recommendation, and data display. Data collection will be responsible for the aggregation of media from each user's authenticated third party sources and storing this for later use, as well as aggregating popular content from each source not belonging to a specific user. This information is then stored and undergoes processing to transform the data into a standard format to avoid discrepancies across services. The data is also semantically analysed and broken down into tag based components used in semantic analysis which form a graph of users based on similarity. Each of the 3 recommendation algorithms

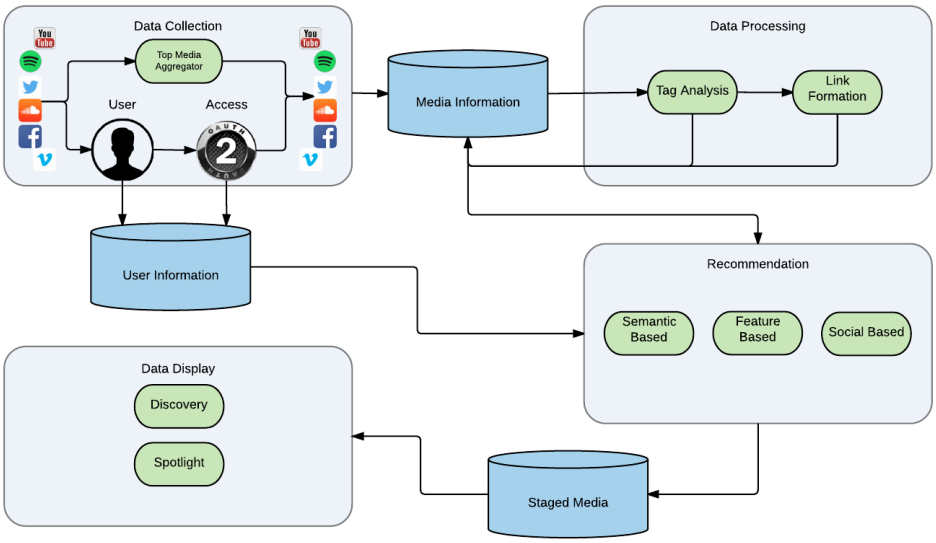


Figure 5.1: Proposed System Architecture

are then ran on the resulting media data, each using a different approach to generate varying recommendations each stored in their own respective data store. Finally, these recommendations are displayed to the user via a web based user interface.

5.2 Data Collection

Without appropriate data work on the system can be deemed pointless, as it is a vital component within data oriented processes such as those utilised by recommender systems. As a result, the design of data collection must be carefully considered to ensure the system has appropriate information to perform inference on and generate recommendations.

5.2.1 Choice of Services

Data lies at the heart of a content aggregator, whilst services such as Netflix and Spotify provide fixed datasets they have full control of a system such as Revolvr relies on user-information to construct a dynamic, ever-growing store of media. As a result it is important to consider the services used in the aggregation process, and whether they are suitable for the system. There are a number of factors that come into play, but four questions lie at the heart of the selection process:

- How popular is the content provider?
- Is a flexible API available?
- Will the inclusion of the provider entice users to use the system?

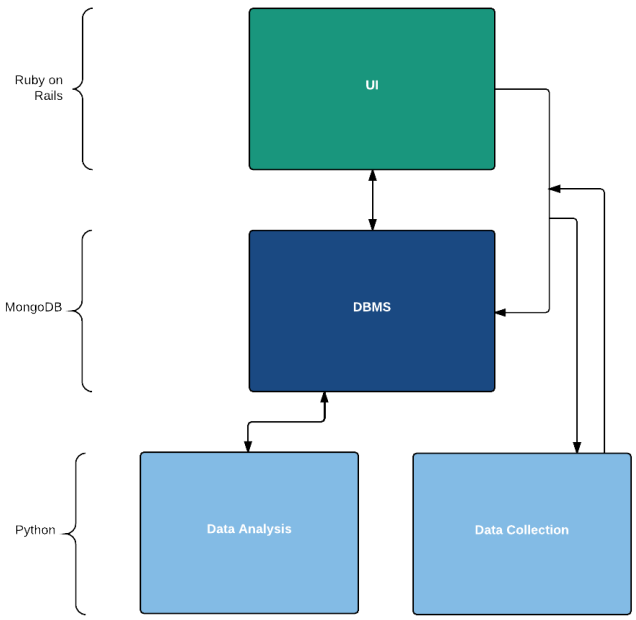


Figure 5.2: High Level System Design

- Does the selection of services represent a true cross-section of user interests?

With these questions in mind, the services chosen to be included within the system both for media aggregation and social connection are *Facebook, Twitter, Spotify, Soundcloud, YouTube, Vimeo*. According to Global Web Index's Q1 2016 social report, the average digital consumer now has close to 7 social media accounts. This number is likely to be skewed, with those aged 16-24 usually having many accounts and the older generation less. Due to this, 6 services was deemed an appropriate number. Furthermore YouTube, Twitter and Facebook were listed in the top 8 social media platforms, with Facebook holding the top spot regarding membership and active usage and YouTube dominating in visitation [19]. Image sharing platforms such as Instagram, and in general more 'social' platforms were considered for selection, however due to the often personal nature of the content services falling under this category were omitted due to privacy concerns. Each of the selected services also offer well documented API's, which will ensure aggregation of appropriate media can be performed with little difficulty as well as adhering to each services specific terms of service. Furthermore, the developer believes media consumption offers a great level of interactivity and immersion as opposed to images and social media posts which are likely to be looked at for a short period of time compared to audio or video. Media of a social nature also often omits filtering, causing issues to arise concerning ethical issues as discussed in Section 3.

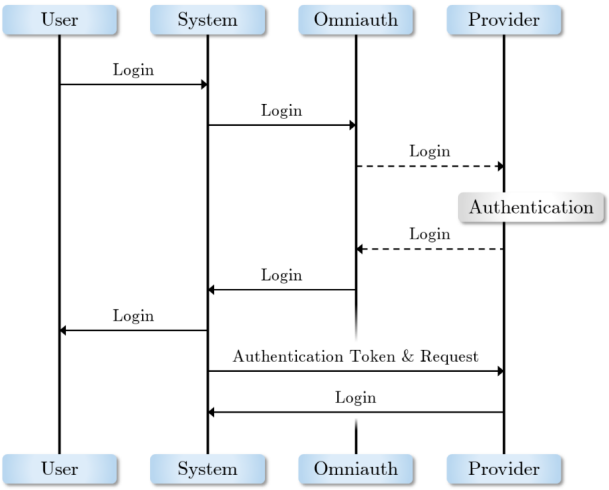


Figure 5.3: Omniauth authentication flow [4]

5.2.2 User Authentication

User login will utilise Facebook's *'Facebook Login'* feature, which provides a *'secure, fast and convenient way'* for people to log into an app or website [20]. With 1.59 billion monthly users as of December 2015, the mass adoption of Facebook provides a simple and easy way for users to login and therefore is the only method to do so. The inclusion of a username and password option was considered, but maintaining a personable, social aspect and avoiding login forms allows the user to feel valued and jump right into receiving media to consume without having to jump through hoops to get there. For connecting to Facebook and the other services the OAuth 2.0 standard is utilised. OAuth is an authorisation framework that *"enables a third-party application to obtain limited access to an HTTP service on a user by user basis, or as a client application as a whole"* and avoids the storage of user credentials [21]. It works by delegating user authentication to the service that hosts the user account, and authorises a third party application (Revolvr in this case) to access the user account. OAuth will be used throughout the application via a Ruby implementation known as Omniauth. Omniauth is a library that *'standardizes multi-provider authentication for web applications'*, and makes use of *strategies* per provider which contain functionality required to connect via the OAuth protocol. [22]. Facebook, Twitter, and each of the aggregated media sources each have their own respective OmniAuth strategies for authentication via the OAuth protocol. A simplified diagram in Figure 5.3 taken from the existing Revolvr documentation outlines the process of authentication common across services.

A mechanism known as *refresh tokens* are used in the OAuth authentication flow, which are used to access user information offline when the user is not explicitly granting it. These are permanent tokens that are valid until the user revokes access to the application. Access tokens provided by the initial authentication only have a limited lifetime for security purposes, however a refresh token can be exchanged to allow access tokens to be granted again when the original token expires.

Unfortunately, the Omniauth solution for Spotify does not include a infinite length access token or automatic token refresh, therefore a script will be developed to do this at the appropriate point in time to ensure access is not denied [23].

5.2.3 Aggregation

Aggregation of media will be essential for obtaining information used to generate suggestions and provide media for user consumption. Two types of aggregation will be utilised in the system, and further information can be found in Sections 5.2.3.1 and 5.2.3.2

- **User-media Aggregation**: This involves aggregation of *user-media* on a user by user basis, that is, media that users have shown preference for in each respective provider.
- **Spotlight Aggregation**: This is not concerned with a particular user, and instead will receive the most popular media from YouTube, Vimeo, Spotify and Soundcloud.

5.2.3.1 User-media Aggregation

Once a user has registered and connected a number of services, a server side algorithm will access a users information (within granted permissions) to retrieve *'liked'* content from each service. Media will be taken from each service's representative *like* store, such as a user's liked videos on YouTube or saved songs on Spotify. All media will be stored directly, as well as in a user-media table which will contain a list of users who have shown preference for that item. This idea remains largely unchanged from the original implementation asides from some API updates to account for updates since the last iteration. Algorithm 1 highlights the operation of user-media aggregation.

Algorithm 1 User-media Aggregation Function [4]

```
1: for all users do
2:     media ← []
3:     for each service provider do
4:         if authentication token for this provider then
5:             for 25 most recent items the user has shown a preference for do
6:                 media[provider] ← aggregated_media
7:             end for
8:         end if
9:     end for
10:    database ← media
11: end for
```

5.2.3.2 Spotlight Aggregation

Spotlight is a term used for an area in Revolvr which will contain the most popular media from its aggregation sources, namely Soundcloud, Spotify, Youtube and Vimeo. For each service a script

will be written to retrieve the most popular media on a daily basis, which will then be displayed for consumption categorised by provider. Specific implementation details will complicate the process depending on how each providers API allows this, however, Algorithm 2 gives a high level proposal for this algorithm.

Algorithm 2 Spotlight Aggregation Function

```
1: media ← []
2: for each service provider do
3:     for each item in top 20 popular items do
4:         media[provider] ← item
5:     end for
6: end for
7: database ← media
```

5.2.4 Scheduled and Repeated Aggregation

Automation of the aggregation of both user and popular media is required on a daily basis. This will allow the system to keep an updated representation of the media footprint / taste profile for each user, as well as provide new media to recommend to users. Cron, a time-based job-scheduler in UNIX environments, will be used to run the above aggregation algorithms on a daily basis.

Repeated aggregation poses the problem of duplicate entries being added to the system. For example, if a user's 25 most recent items have not changed since the previous day then the system should not consider these items for full aggregation to avoid both duplicates and computational overheads. As a result during aggregation explicit checks must be made to check whether an item considered for aggregation already exists for a user, and if so terminate the iteration early. Again, this process existed in the existing implementation and an outline of the process can be found in Algorithm 3.

5.3 Processing

Once the user-media has been aggregated into the system via Algorithm 2 a number of significantly different recommendation algorithms will be applied to the data set resulting in recommendations for a user. Three different algorithms will be selectable by the user to provide contrasting recommendations. Firstly, the semantic algorithm utilised in Revolvr's original implementation will remain in place aiming to provide predictions based on semantic deconstruction and analysis of titles. Following this two new algorithms will be designed, one will provide recommendations via an algorithm known as matrix factorisation in which a definition of a hybrid feedback type (see Section 2.3) will be utilised. Finally, a social algorithm which makes suggestions based on Facebook and Twitter connections will be implemented. These three algorithms do not all necessarily aim to provide the *best* recommendations as a whole, because using that logic they would all recommend the same content to the user and therefore the differentiation becomes pointless. Instead they

Algorithm 3 Duplication Prevention Function [4]

Require: *media* ← *list of aggregated media*
Require: *user* ← *given user*
for each media item **do**
  if media item exists in database **then**
    if user has previously been associated with the item **then**
      return True
    else
      return False
    end if
  else
    return False
  end if
end for

aim to provide different approaches to allow varying content, to promote diversity amongst the results and promote exploration of media from all three options for a user. Sections 5.3.1, 5.3.2, and 5.3.3 provide descriptions of the proposed algorithms and 5.3.4 highlights how the sample of recommendations will be chosen.

5.3.1 Semantic Based Recommendation

The semantic based algorithm was present in the former implementation of Revolvr, but for consistency will be explained in terms of its design and its implementation details for clarity of discussion. The discussion will mirror that included in the original documentation to provide a whole overview of the system [4]. Despite its questionable performance (see Section 7.4.1) it was kept in the system to provide 'curve ball' recommendations to provide diversity, and also as a benchmark for comparison of quality. A *direct link* between users defines when two users express an interest in the same piece of media, and a *similar link* is the notion of sharing an interest in the same phrase belonging to a title. The former can be seen as a direct related interest, whereas the latter is only aims to model an indication of similar preference.

This algorithm aims to process media via the decomposition of media titles into key terms, before analysing commonly coupled terms and identical interests between items to create *links*. These links, known as *direct* and *similar* links can be used to create a graph representation of the user-base from which relationships can be deduced as a basis for recommendation.

5.3.1.1 Identifying Similarities

Similarities between users will be modelled via direct links and similar links. A direct link will be produced when two users share an identical piece of media, which can be deduced during aggregation as highlighted in Section 6.2.2.1. To account for stringent approach modelled by direct links, similar links will be constructed via simple semantic analysis of media titles. This analysis

will create similar links based on frequently coupled pairs of terms in a title. Media titles will also undergo pre-processing by comparing the title to a corpus of common words as well as removing punctuation, in an aim to reduce the title to a more meaningful representation of the media. The corpus will contain common words related to media such as *official, music, video*, etc. For example if a title was *'Taylor Swift - Bad Blood Official Music Video HD'* the title would be stripped to become *'Taylor Swift Bad Blood'*. Algorithm 4 recreated from Revolvr's existing documentation highlights this process.

Algorithm 4 Removal of Unnecessary Terms from Media Title [4]

Require: *old_title*
1: *new_title* ← " "
2: **for** term in old_title **do**
3: Remove all punctuation
4: **if** stripped term not in common corpus **then**
5: **if** stripped term is not a single letter **then**
6: **if** term not in custom corpus **then**
7: *new_title* ← *new_title* + " " + *term*
8: **end if**
9: **end if**
10: **end if**
11: **end for**
12: **return** new_title

When terms in a title frequently appear alongside others, their likelihood of being related increases and when this likelihood reaches a threshold a similar link will be created. During aggregation, all individual terms of a media item's title will be stored as well as a list of associated terms for each individual term along with a frequency of occurrence. Related terms do not have to appear in the same media item, meaning that if two different media items shared the term *'Taylor'* terms from both titles would appear in the *'Taylor'* association list.

5.3.1.2 Relating Users

Following the processing of direct and similar links after aggregation, users will be related by producing a similarity rating for each user-user pair where there is a link. Direct links will carry a higher similarity weighting than similar links as they show direct commonalities between users. The links can be seen as a graph (V, E) which represents the relational network. V is the set of all users of the system, and E is the set of similarity ratings between users. E is constructed as the combination of all ratings from D, the set of direct links, and from S, the set of similar links [4]. The strongest relationships between users are then viewed as users who are more likely to enjoy similar media, which is used to inform the recommendation process during staging. The computation of similarity will be based upon the existing implementation and is described by Algorithm 5.

Algorithm 5 Proposed Computation of Similarity Ratings [4]

```
1: s ← []
2: for all related users do
3:     if user has similar media then
4:         if user has directly shared media then
5:             s[relateduser] ← |similar| + (|direct| × weighting)
6:         else
7:             s[relateduser] ← |similar|
8:         end if
9:     else if user has directly related media then
10:        s[relateduser] ← |direct| × weighting
11:    else
12:        s[relateduser] ← 0
13:    end if
14: end for
15: return sorted(s)
```

5.3.2 Feature Based Recommendation

Feature based recommendation will utilise the process of matrix factorisation via alternating least squares, based on research by Hu et al. The following subsections provide information into the motivation, adaptability and implementation of the concepts put forward in their 2008 paper *Collaborative Filtering for Implicit Feedback Datasets* via the definition of a hybrid feedback type [16].

5.3.2.1 Motivation

When exploring the existing documentation and implementation of Revolvr it became clear that only aggregating media with prior knowledge of it being preferred posed a limitation. The system was designed to use user preferences to create a taste profile, and create recommendations on this using a semantic analysis algorithm. However, aggregating media that users have shown a preference for only allows the system to create a preference profile as opposed to a taste profile, meaning content users dislike is not modelled. Originally a rework of the system was considered, however due to API restrictions the developer quickly realised that to do so aggregation of user content would no longer be needed and voids the underlying point of the project. What originally appeared as a limitation opened up a door for academic exploration into an area that has not been extensively discussed, providing a combination of academic and technical content for the project. Instead of backtracking, the foundations provided by Revolvr could be used as a playground to explore this problem.

When researching feedback types used in recommender systems as outlined in Section 2.3, it became clear that implicit feedback is popular among many current recommender systems. The developer noticed the following similarities between Revolvr's dataset and those constructed via

implicit feedback:

1. **Preferred media is indicated:** Aggregated media in the system implies it is preferred, and in implicit feedback a high 'value' (number of streams, page clicks, minutes viewed) often provides an indication of preference.

2. **Little or no information on disliked media:** In both Revolvr's aggregation and implicit feedback absence of an item / value for a user does not distinguish between disliking the item or simply not knowing about it. For example, if the stream count of a song was zero it may be the case the user strongly dislikes the song, or does not know about it yet.

Despite these similarities, there is an area of uncertainty with regards to Revolvr's feedback. In each respective third-party the rating or saving of a media item can be seen as explicit feedback as this is *'explicit input by users regarding their interest'* [16]. However, once aggregated this uncertainty is introduced and the context of the rating is lost, only knowing that the item is preferred. As a result Revolvr's dataset may not be considered as implicit feedback necessarily, however will be described by the term **asymmetric feedback**. By this we mean there is an explicit notion of preference, however, deductions cannot be made about items lacking this explicit preference. For discussion within this project this will be a binary scale, where 1 implies preference and 0 denotes an unknown. This definition could be extended to include the concept of confidence in which a higher value suggests an item is more preferable, but for discussion the binary approach will be taken.

During research, a 2008 article named *'Collaborative Filtering for Implicit Feedback Datasets'* by Hu et al. was referenced during a number of Spotify presentations relating to the operation of their recommender system [12] [15] [16]. The problem domain of missing preferences for unknown items appeared analogous to the problem the developer was attempting to tackle. In this paper, Hu, Koren and Volinsky, the latter two ultimately being winners of the Netflix Prize, identify unique properties of implicit feedback datasets and propose an algorithm to make recommendations based on such data. The authors begin by identifying some *'unique characteristics of implicit feedback'* which are as follows:

1. **No negative feedback:** It is hard to reliably infer which items a user did not like, and therefore *'missing data'* must be addressed as this is where negative feedback is expected to be found

2. **Implicit feedback is inherently noisy:** Passively tracking user behaviour can only allow you to guess a users preferences or true motives. For example, a user purchasing an item online may be doing so for someone else and not reflect their tastes.

3. **The numerical feedback of explicit feedback indicates preference, whereas the numerical value of implicit feedback indicates confidence:** Numerical values of implicit feedback usually describe the frequency of actions, e.g. how much time a user watched a certain show. A larger value does not necessarily indicate a higher preference, however the numerical value tells us about the confidence that can be had in an observation.

The developer believes these properties can be extended to the concept of asymmetric feedback via the following comparisons:

1. **No negative feedback:** Asymmetric feedback such as Revolvr's data only allows positive preferences to be known, with no information on negative feedback.

2. **Implicit feedback is inherently noisy:** Asymmetric feedback in Revolvr's case actually offers improvement over implicit feedback, as a notion of preference can be taken with a higher confidence. This is because the aggregator aims to take media from a users store of preferred media from each external source. A user is unlikely to add an item to their likes on a service such as YouTube by mistake as this is an explicit action requiring user effort. However, you cannot be certain of this this assumption. This scenario may still occur by accident, be used simply to *'save'* media as opposed to prefer it, or be due to user misunderstanding. With these scenarios in mind the dataset will still contain some level of noise. Furthermore, tastes in media such as audio and video can change. Liked items from a recent time period are more likely to represent a users current interests whereas items liked a year ago may no longer be relevant due to a change of tastes.

3. **The numerical feedback of explicit feedback indicates preference, whereas the numerical value of implicit feedback indicates confidence:** In asymmetric feedback assume a 1 implies preference and 0 is unknown. Here the presence of a 1 does indeed indicate preference, however, little judgement can be made about a 0. The statement *'numerical feedback indicates preference'* does not hold for both a 1 and 0 in asymmetric feedback. A 1 can be seen as a confidence **and** an explicit preference, whereas a 0 can only be seen as a confidence. Therefore, both scenarios in asymmetric feedback can be confidence values but only a subset can be seen as explicit preference. Table 5.1 highlights the similarities and differences between the defined feedback sets, and displays a higher level of similarity between asymmetric and implicit feedback than between explicit and asymmetric. The term confidence is used due to the possibility of noise as discussed in the previous point.

Table 5.1: Comparison of Feedback Types

Feedback Type	**Value High**	**Value Low**	**Value Missing**
Asymmetric	Confidence item is preferred (binary)	Confidence item is preferred (binary)	Unknown
Implicit	High confidence that item is preferred	Lower confidence that item is preferred	Unknown
Explicit	Item is preferred	Item is disliked	Unknown

With these comparisons in place the developer believes the notion asymmetric feedback Revolvr promotes collects is suitably commensurable to implicit feedback, and as a result the algorithm discussed in Hu et al.'s paper was selected for the system. There is no guarantee that the results will

be reliable until implementation and user testing, however the above arguments and equipollent nature of the feedback types provides confidence and a motivation to explore the area. For consistency the algorithm will be referred to as the Implicit Matrix Factorisation algorithm despite the discrepancies between the datasets.

5.3.2.2 Preliminaries

The following notation will be required for discussion of the algorithms operation:

- u and v will be used to denote users.
- i and j will be used to denote media items.
- Input data will be denoted via r_{ui}, which is user $u's$ observed value for item i. In this case if an item is in user $u's$ aggregated items then $r_{ui} = 1$, and if it is not present then $r_{ui} = 0$ implying an unknown preference. These will be known as *observations*.

5.3.2.3 Setup

In Hu's article, a notion of confidence which r_{ui} measures is introduced as a set of binary variables p_{ui} derived as follows:

$$p_{ui} = \begin{cases} 1 \text{ if } r_{ui} > 0 \\ 0 \text{ if } r_{ui} = 0 \end{cases}$$

However, this definition is based on implicit feedback where r_{ui} is not binary. The idea is that if a user u consumes an item and r_{ui} is non zero, then they may prefer it and $p_{ui} = 1$. When extended to the notion of asymmetric feedback p_{ui} and r_{ui} are simply equal to one another as varying confidence levels do not occur due to the binary nature. Despite this difference the idea still holds that zero values of p_{ui} are associated with low confidence, as *"not taking any positive action on an item can stem from many other reasons beyond liking it"* [16]. The confidence values for observing p_{ui} is given as $c_{ui} = 1 + \alpha r_{ui}$ which provides a minimal confidence in preference for every user item pair. For asymmetric feedback r_{ui} does not vary, and therefore the confidence value just becomes a scaling factor, however tuning of this value via variation of α can increase performance and is data dependent.

The algorithm proposed utilises *Latent Factor Models*, which is an approach to collaborative filtering with the goal of characterising both items and users via the discovery of latent features that explain observed ratings [24]. Hu et al. specifically focuses on Singular Value Decomposition (SVD) of what is known as a user-item observations matrix, which aims to factorise the matrix into user and item latent factor vectors [16, 25]. A user-item observation matrix is simply a $n * m$ matrix representing n users and m items in a recommender system, as shown in Figure 5.4. In such a matrix a column represents an item and a row represents a user.

The values occupying the matrix are the values of p_{ui}, and with implicit and asymmetric datasets are likely to result in a sparse matrix. The vectors produced from factorisation aim to characterise

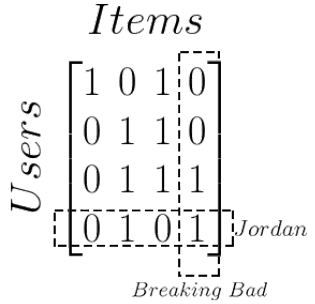


Figure 5.4: User-item Matrix

both items and users by vectors of factors inferred from item rating patterns. Figure 5.5 shows an example of this factorisation where f is the number of factors, X a matrix of vectors for users and Y is a matrix of vectors for items. This is known as a joint latent factor space with dimensionality f. Each item i is associated with a vector y_i and each user u is associated with a vector x_u. y_i measures the extent to which the item possess a factor, and x_u measures the extent of interest a user has in items. The dot product of any user vector and item vector aims to capture the interaction between user u and item i, which is essentially the user's overall interest in the item's characteristics [26]. An approximation of the rating, or in this case preferences, is assumed to be the inner product of a respect user and item vector: $p_{ui} = x_u^T y_i$. As a result of this, the algorithm aims to deconstruct the user-item matrix into latent factor vectors that minimise the RMSE (root mean squared error) between the actual values and the predicted values. Algorithm 6 provides a high level description of how this matrix will be constructed from the aggregated media data and user store.

Algorithm 6 Construction of User-Item Matrix

Require: n = number of users, m = number of items
Require: $matrix = n * m$ matrix of 0's
for each user, u **do**
    Map u to row index in matrix
end for
for each item, i **do**
    Map i to column index in matrix
    for each user u in item i's list of users **do**
        row = User u's row index
        column = Item i's column index
        matrix[row][column] = 1
    end for
end for

5.3.2.4 Operation

Factors are computed by minimising the following cost function:

$$min_{x_*,y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u ||x_u||^2 + \sum_i ||y_i||^2 \right) \tag{5.1}$$

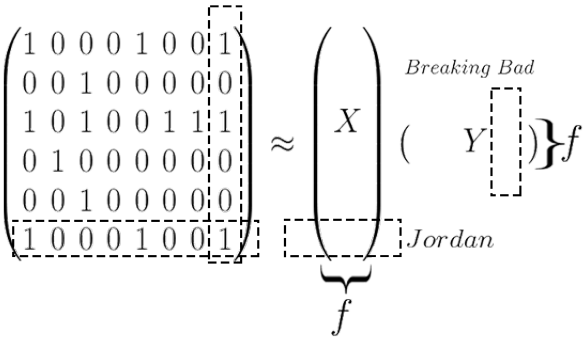


Figure 5.5: Matrix Factorisation

The $\lambda \left(\sum_u ||x_u||^2 + \sum_i ||y_i||^2 \right)$ is a regularising term which ensures it will not overfit the dataset. λ is data-dependent and determined via a cross validation procedure similarly to α therefore placeholder values for both λ and α will be used until after implementation. $p_{ui} - x_u^T y_i$ can be viewed as the actual value of the user-item matrix minus the predicted value, therefore this term squared over all predictions gives the root mean squared error which is what the algorithm aim to minimise. Both the user and item vectors are initialised to random noise, and the authors observe that when either the user-factors or item-factors are fixed the solution to the cost functional becomes quadratic, and therefore the global minimum can be readily computed. This leads to an alternating-least-squares process, in which the algorithm alternates between re-computing user-factors and item-factors, and each step is guaranteed to lower the value of the cost function [16]. When one set of factor vectors is fixed, the derivative of the loss function with respect to the other set of factor vectors is taken, set to zero to give the minimum, and then solved. The paper also provides some mathematical rearrangement and equality that allows the computation of these vectors to be performed more efficiently, but discussion of this will be omitted due to the focus of this section being on the process of the algorithm not the theory. Equations 5.2 and 5.3 give the solutions for each of the latent-factor vectors once the derivative has been taken and these optimisations have been applied. This process of fixing one set and solving for the other is repeated until convergence as it can be shown that the

only minimum is the global one [26]. Finally, following the computation of these vectors they can be used to provide a 'user-item' score via computation of $\tilde{p_{ui}} = x_u^T y_i$ where $\tilde{p_{ui}}$ represents the predicted preference of user u for item i. User u would then be recommended the K available items with the largest user-item score. This will be implemented within the system by simply accessing the user-media table and constructing the appropriate matrix and then applying the algorithm discussed above. Finally, to provide fresh recommendations items the user has already shown a preference for will be discarded from the top K values to ensure all recommended items are new and then the items presented to the user. Algorithm 7 shows a high level overview of this process that will be implemented to attempt to extend Hu et al.'s work to the idea of asymmetric feedback [16, 26, 15].

$$x_u = (Y^T Y + Y^T (C^u - I)Y + \lambda I)^{-1} Y^T C^u p(u) \tag{5.2}$$

$$y_i = (X^T X + X^T (C^u - I)X + \lambda I)^{-1} X^T C^u p(u) \tag{5.3}$$

5.3.3 Social Recommendation

The final algorithm that will be implemented can be considered simpler than the previous two, but aims to select content based on social aspects as opposed to the item itself. By promoting login via Facebook and offering the ability to connect to Twitter a graph of connected users may be constructed, and connected users can be recommended each others content. This approach will be based on the fact that two users are connected and therefore likely similar in some way. Although being connected to someone does not *necessarily* mean you share tastes, it does provide another perspective for recommendation in an aid to diversify results from each algorithm.

The social algorithm will iterate through each user and check for two pieces of information: their Facebook friends using Revolvr, and people they follow on Twitter using Revolvr. These users will be known as connections, each connection's preferred media which is gathered during aggregation will then be combined into one media set. This set of media will then be processed to remove any media than the user requiring recommendations has already shown a preference for, before taking a random selection of the final subset for recommendation. Algorithm 8 summarises this process.

5.3.4 Selecting Appropriate Media

Once the three algorithms have generated their recommendations, a subset must be selected for display to the user. For featured based and social based algorithms the generation of predictions is explicit within the algorithm. In the former, a top K list of recommended items ordered by predicted preference will be stored in an *implicit staged media* table. When it comes to displaying the media via the user interface a random selection of media will be taken from this table and shown to the user. A similar case will occur for the social algorithm, the media will be stored in a *social staged media* table before a random subset is selected for consumption. As for the semantic algorithm, again a random subset of the media stored in a *semantic staged media* table will be suggested. However, it is important to ensure the recommendations will not be dominated by the most similar user. Similarity will be computed using the existing algorithm which uses the graph

Algorithm 7 Recommendations via Matrix Factorisation with Alternating Least Squares

Require: $matrix$ = matrix returned from running **Algorithm 6**
Require: $iterations = 30$
Require: $factors = 40$
$uservectors = n * f$ matrix of random noise
$itemvectors = m * f$ matrix of random noise
$predictions = n * m$ matrix of zeroes

for i in range(iterations) **do**
    $uservectors$ = Fix item vectors and solve for user vectors
    $itemvectors$ = Fix user vectors and solve for item vectors
end for

for each user, i **do**
    for Each item, j **do**
        $predictions[i][j] = uservectors[i]^T itemvectors[j]$
    end for
end for

for each user u's prediction p in predictions[u] **do**
    if user has already liked this item **then**
        $prediction[u][p]$ = extreme negative value so item is not recommended
    end if
end for

for each user, u **do**
    Recommend top K items from predictions[u] for recommendation
end for

Algorithm 8 Social Recommendation

```
1: twitter_users = {}
2:
3: for each user u in the system do
4:     if user u has a connected twitter account then
5:         Add user u and their authentication details to twitter_users
6:     end if
7: end for
8:
9: for each user u in the system do
10:    follows = []
11:    fb = []
12:    for each of user u's Facebook friends do
13:        if friend exists in user table then
14:            append friends user id to fb
15:        end if
16:    end for
17:    if user u is present in twitter_users then
18:        u_follows = list of users u follows on Twitter
19:        for each user v in twitter_users do
20:            if user v is in u_follows then
21:                add user v to follows
22:            end if
23:        end for
24:    end if
25:    friends = follows ⋃ fb
26:    media = media that users in friends like
27:    user_media = user u's liked media
28:    recommendations = media \ user_media
29:    Recommend to u a random select of recommendations
30: end for
```

produced in Section 5.3.1 and is shown in Algorithm 9. As a result the system will randomly select fixed quantities of media from the most similar user, as well as media from less similar users to ensure the selections vary in similarity. The developer does not necessarily think that the semantic algorithms selection technique promotes as much content diversity as the other two algorithms, but will be kept in its original form for comparison purposes in Section 7.4.1.

Algorithm 9 Computation of Similarity Ratings [4]

```
1:  s ← []
2:  for all related users do
3:      if user has similar media then
4:          if user has directly shared media then
5:              s[relateduser] ← |similar| + (|direct| × weighting)
6:          else
7:              s[relateduser] ← |similar|
8:          end if
9:      else if user has directly related media then
10:         s[relateduser] ← |direct| × weighting
11:     else
12:         s[relateduser] ← 0
13:     end if
14: end for
15: return sorted(s)
```

Allowing random selection of the recommendations in all three algorithms will allow the user the chance of seeing something new, unexpected and of a spontaneous nature. The generation of recommendations will be computed server side daily using a cron job similarly to content aggregation. This means instead of straining the server upon a user request, the recommendations for the whole system will be processed once every 24 hours to reduce computational strain. The selection of a subset of possible recommendations will also improve performance when displaying items, as embedded media players are often costly. Finally, this subset will also provide *'bite-sized'* selections, intriguing the user to keep refreshing the selection and increasing user engagement.

5.3.5 Training

Recommender systems often include the ability to train the system, which allows users to provide the system explicit feedback to further tailor their recommendations. Restricting the system to only using third party providers for preferences can cause limitations, and may result in poor recommendations for users who do not utilise these platforms extensively. As a result, it is important to include an element where usage of the system itself influences recommendations and furthers the development of a taste profile.

Previously, Revolvr offered a training system and an evaluation system. The training system showed the user items they had not seen before, offered them the chance to like or dislike the media,

and edited their taste profile accordingly. Evaluation worked on a similar basis but instead the user could explicitly rate the recommendation on a scale from 0 to 100. However, these instances of user feedback were not utilised fully and added little to the system, as well as being intrusive and breaking the idea of streamlined consumption and discovery. Furthermore, the system offered no way to reconsume media the user had liked before, and therefore its long term appeal and ability to offer media organisation suffered. The new system will offer a simple 'like' button within the embedded player that will also act as a saving mechanism, placing the media in an area of the site where all aggregated and liked media can be reconsumed. Liking an item will explicitly create a preference in the user media table, which as a result means the item will be included in all three algorithms due to the modular design. This new item will be available for semantic analysis, add a 1 in the user-item matrix without editing its dimensions, and also be included for that user in the social algorithm. The principle here is that as more and more users train the system via internal feedback instead of aggregation the sparsity of preferences decreases, and the accuracy of recommendations can increase. Furthermore the ability to recall media adds an element of media management, further fulfilling requirements **F11** and **F13** defined in Section 4.

There is a concern that inclusion of this feature moves away form the domain of implicit and asymmetric feedback, and instead could be seen as a form of explicit feedback raising the question of why a dislike option is not included. This format of a combined like and save is present in many services including Soundcloud, Spotify and Vimeo. The developer believes that the media a user is recommended perhaps would have not been discovered without the help of Revolvr, and if the user does like a recommended item it is important to develop their taste profiles based on this.

5.4 User Interface Design

In a user focused web based application such as Revolvr, a strong User Interface (UI) is a leading factor for its success. The UI must provide a platform for users to view recommendations in a fashion which is engaging, easy to understand, yet does not deter from the viewing of media. Standard navigation and design principles must be employed to ensure the user does not feel lost, and pages must have a clear function to avoid ambiguity. The growth of mobile browsing via phones and tablet devices has also put a great emphasis on responsive design. With 60% of digital media time spent on mobile devices it is clear the majority audience is shifting [27]. As a result, a responsive design that functions n a number of different screen sizes will be critical for multi-platform use. These developments in consumption habits provide new challenges in both UI design as well as human-computer interaction, but promotes mass appeal of the system regardless of device [28]. The following designs of the critical pages within the system are purely mockups, but aim to be replicated as accurately as possible during the implementation phase. Changes will be made to the existing UI to follow current UI trends, and address visual issues in the existing implementation. Areas of the website in which media can be consumed will utilise AJAX loading, to provide seamless loading of dynamic content without the need to refresh the page.

5.4.1 Previous Work

Revolvr, in its current form, provides a strong basis to work from developed using the responsive web framework Bootstrap [29]. Via discussion between the developer and the supervisor the decision was made not to reinvent the wheel, and instead use the existing UI as a framework to build upon. Due to this it is important to distinguish between existing work and proposed changed, therefore each figure will provide a screenshot of the original design followed by the proposed changes as a tool for comparison.

5.4.2 Data Representation

The pages of the system can be split into two distinct categories: dynamic pages an static pages. Dynamic pages will interact directly with information stored in the systems database, as outlined in Figure 5.6. This is a simplified representation to convey the idea of how general components and data stores will interact. Static pages on the other hand will be displayed uniformly to all visitors of the site. When a user is logged out generic non-personalised pages will be displayed, and once a user is registered these pages will be tailored to an individual user to promote a one-on-one personal experience. The number of pages compared to the original design has been reduced, aiming to reduce the breadth of the system and instead focus on depth.

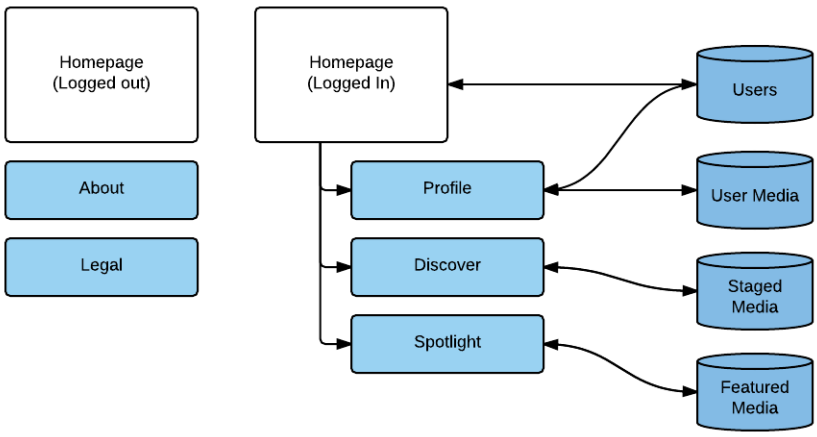


Figure 5.6: Flow of Information

5.4.3 Navigation and Information

A simple navigation bar will be used to move around the pages within the website, with the key pages being present at all times and less important pages being available via a dropdown menu. When logged out this bar will simply provide the options to learn about Revolvr via the *About* page and also offer the option to sign up. When logged in, the navigation bar will focus on media

consumption by containing links to the *Discover* and *Spotlight* pages. An image of the user taken from Facebook will be placed at the top right and also act as a dropdown menu, providing user-focused options such as the *Profile* page and the option to sign out. This navigation bar will be common across all pages of the website and act as a hub to navigate the system, and will only change depending on whether a user is logged in and out. Figure 5.8 shows the proposed format of the navigation bar. A footer will also be provided across all pages, providing information on the developers, a link to the about page, privacy policy and finally appropriate social media links.

Figure 5.7: Old Navigation Bar

Figure 5.8: Proposed New Navigation Bar

5.4.4 Home

The home page will be fundamental in captivating potential users attention, aiming to advertise Revolvr and entice users into signing up. A simplistic and focussed design will be deployed via a minimalistic colour palette, and key information about the service will be provided. A quick overview of the functionality of the system and the selected third party providers will be displayed, and further content is dependent on whether or not a user is logged in. The designs for this page were made by editing the existing HTML, as the developer liked the design of the page and felt only minor colour and logo changes were in order.

- **Logged Out:** When the page is visited with no knowledge of a user, the top portion of the webpage which is most visible will offer the user the option to connect using Facebook. By placing this content higher up the page it is the first thing the user is likely to see, and will aim to entice users to sign up. The lower portions of the home page will provide information on what services are available to connect after a user has signed up. Logos, graphics and buttons will be prioritised over text as a technique to captivate the user and provide familiarity and a sense of trust. Figures 5.10 and 5.11 show the design for this page, with Figure 5.9 showing the old design. There will also be an option to view a random set of media as a *'trial run'* of the system, to provide a look into how media is presented and allow the user to experience the system before committing.

- **Logged In:** When a user has registered and logged in the home page view will largely reflect that of the logged out view, however a focus on personalisation and beginning a journey of discovery will be present. The aforementioned list of services will now be paired with connect buttons, meaning a user is immediately provided with a way to begin building a taste profile without the need to navigate further. A personalised message will also greet the user based on their user profile information, further adding a layer of personality to the website. A figure of this has not been included due to the subtle changes.

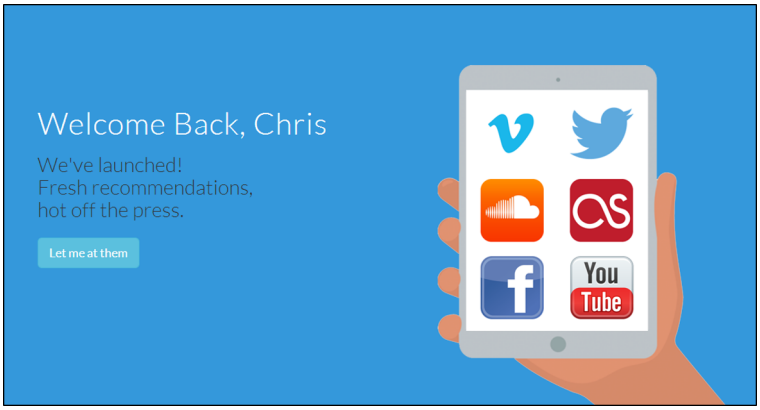


Figure 5.9: Old Home Banner

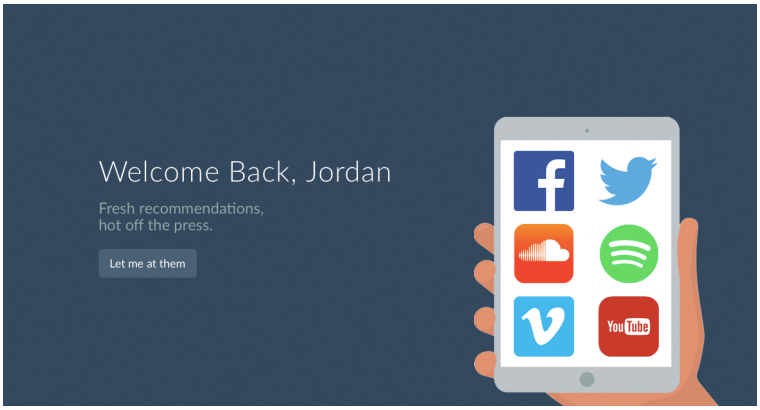


Figure 5.10: Proposed New Home Banner

By reducing the number of options available in the navigation bar, Revolvr aims to become a streamlined experience of discovery for a user. This allows more of a focus on receiving and enjoying recommendations, as opposed to constantly promoting static and purely informative pages.

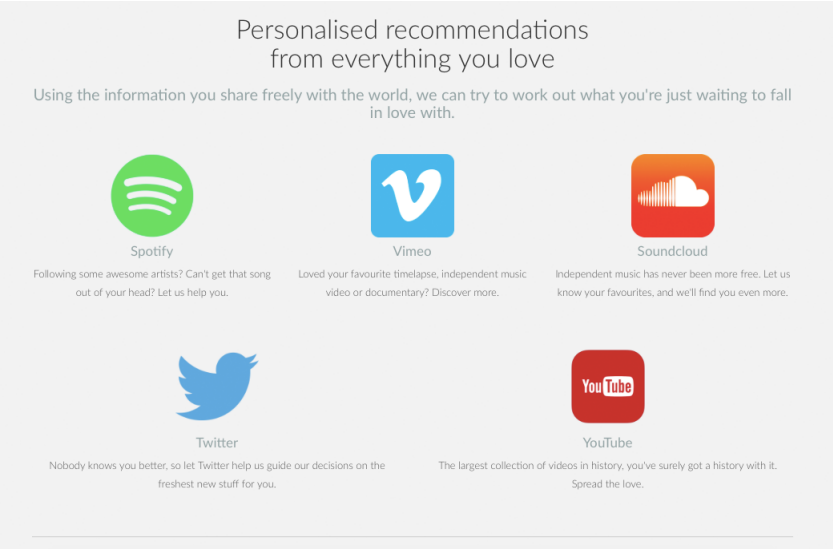


Figure 5.11: Lower Portion of Proposed Home Screen

5.4.5 Discover

Discover will be the home of a users recommendations, providing the ability to view and save the recommended items as well as view information such as titles and descriptions. This is an area that will be focussed on strongly as the original vision was lacking and muddled due to placing multiple recommendations on the screen at once, with no information or ability to interact with the items apart from consuming them. The new discovery view will be based on the idea of a *revolving* wheel of media consisting of 10 items, which can be refreshed via AJAX to display more recommendations. The term revolving is used loosely here, with recent web design trends focussing on flat UI's a cover-flow style using 3D rotation would not be appropriate, and instead a sliding mechanism will be used. In each slide one item will be the focus of the page with the item itself, its title, and a description available. Beneath each media item a small heart button will be present, allowing the user to *like* the item both evolving their taste profile and saving the media with a simple click of a button. Figure 5.12 shows a mock up for the discovery view, compared to the old design shown in 5.13

5.4.6 Spotlight

Spotlight will be the hub of popular media, aiming to provide a concise summary of what media is popular on the web on a day by day basis. Displaying the media will use an identical player to the Discovery section, however, additional tabs will be displayed above the player to allow users to select which service they would like to see media from. For example, clicking the Soundcloud tab would provide the 20 most popular songs from Soundcloud in the revolving player, within an identical player to provide consistency in design amongst pages. Design of this component can be seen in Figure 5.14.

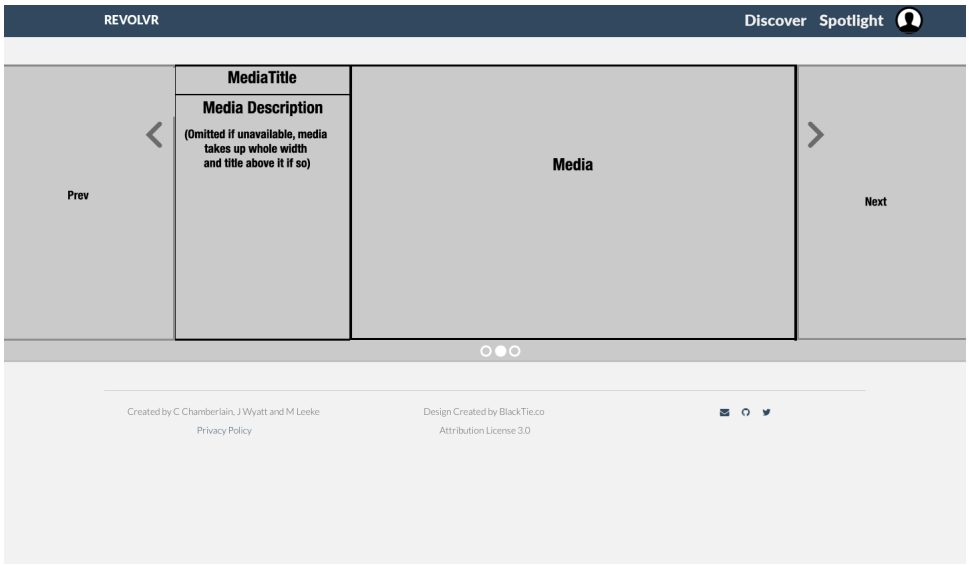


Figure 5.12: Proposed Discover View

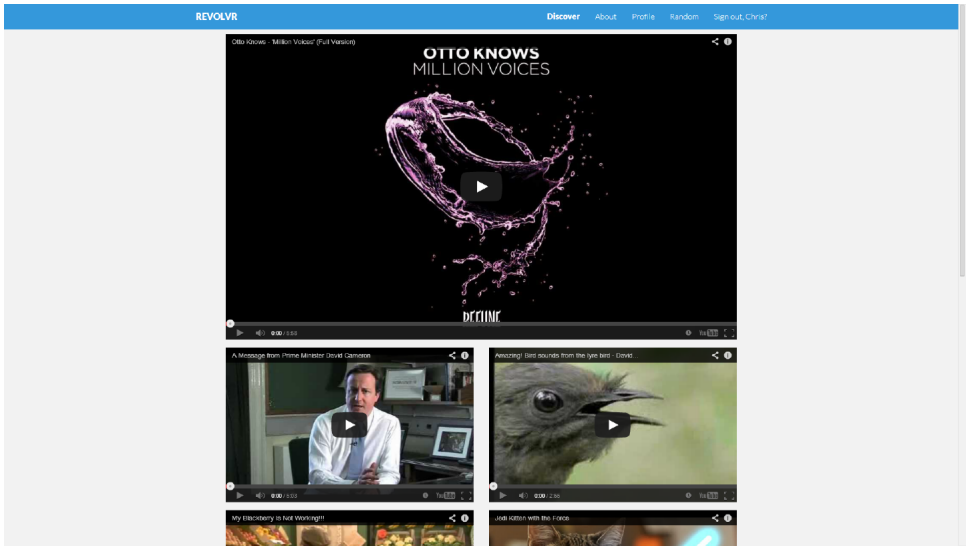


Figure 5.13: Old Discover View

5.4.7 Profile

The profile page will act as a hub for personal content and preferences. The design remains largely the same as the existing implementation, however a focus has been made on providing the ability to select from multiple algorithms as well as view media the user has liked. The profile page will provide the following information:

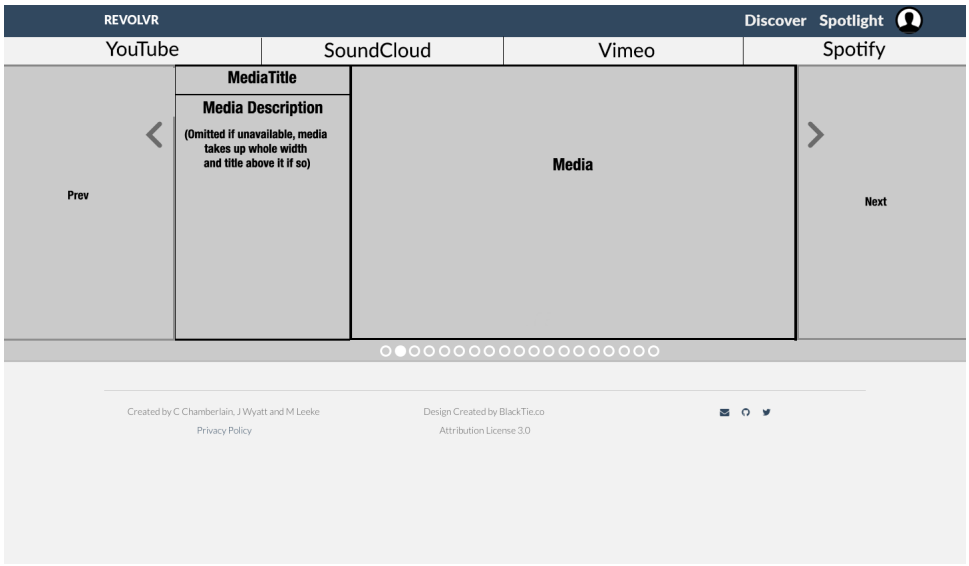


Figure 5.14: Proposed Spotlight Design

- The ability to select one of the three recommendation algorithms, with brief user friendly descriptions on how they operate
- A list of services the user has connected
- A list of services the user has not connected, with the ability to do so
- Lists of videos and audio that the user has been responsible for either liking internally or aggregated from externally. Clicking on the title will allow an embedded popup player to be displayed to consume the item once again
- A timeline showing when the user performs actions such as signing up and connecting a service
- Buttons and instructions on how to remove any connected services

Figures 5.15 and 5.16 show the proposed designs of the algorithm selection and popup player, with the remaining portions of the profile page staying largely the same.

5.4.8 About

The *About* page will outline the intentions of Revolvr and what it hopes to achieve. This page will remain unchanged from the original design, as it covers the core ideals of the system and its operation well enough in its current form.

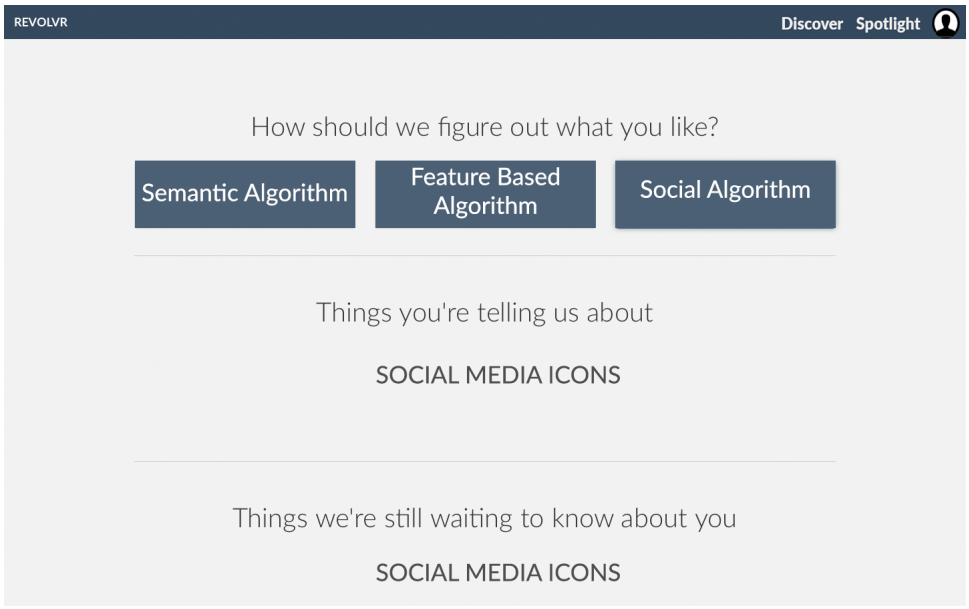


Figure 5.15: Profile Algorithm Selection Design

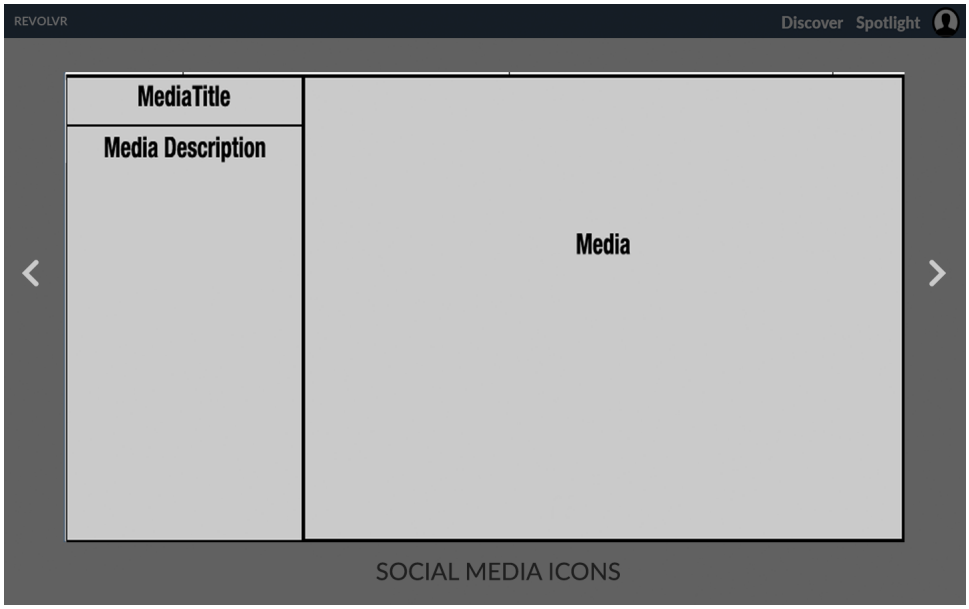


Figure 5.16: Embedded Player Design

5.4.9 Privacy Policy

To ensure that the legal, ethical, social and professional issues outlined in Section 3 are abided, a *Privacy Policy* page is available for users to view. Again, this static design will remain unchanged

as the information provided can already address any privacy concerns a user may have. It gives information on what data is collected, how the data is collected, how the data is used, what control the user has over their data use, and how Revolvr aims to protect users information.

The detailed, clear design of a system offers the chance for a smooth implementation process. By providing detailed implementation plans in the form of a design proposal the process of implementation becomes streamlined and can be seen as the realisation of defined components. Without such a design process issues can often occur further into the development cycle, and with such a project this is important to avoid. The implementation of the specified designs will be discussed in the following section.

CHAPTER 6

Implementation

Following the development of detailed design plans, this section highlights how the system evolves from a proof-of-concept design to a functional system.

6.1 Development Technologies

In a web based system such as Revolvr, a number of technologies are responsible for the processing, storage and display of data. Figure 5.2 highlights the technologies used for each core component of the system. Data analysis and collection is performed server-side using Python and stored in a NoSQL MongoDB database. The web based presentation of recommendations deduced from these processing elements uses a web framework known as Ruby on Rails, which incorporates a number of web stack technologies as described below.

6.1.1 Data Processing

Python is a popular, general-purpose, high level programming language [30]. With a design philosophy based on readability, development in Python allows developers to write fewer lines of code compared to a lower level language such as C++ or Java. Since its release in 1991 Python has become increasingly popular, ranking second in the PYPL PopularitY of Programming Language Index [5]. Figure 6.1 shows Pythons increase in share of Google searches relating to language tutorials, growing 5.8% over the last 4 years and only beaten by Java. As well as its large standard library, the mass adoption of Python has lead to 78,000 third party packages being developed and provided by the Python Package Index (PyPi) [31]. These packages include official and community-based API's for interacting with the third party providers, and importantly the *pymongo* package allows interaction with a MongoDB database. The emphasis on readability will allow fast development to occur, further complimenting the agile nature of the project. Furthermore, the extensive

selection of packages provides platforms for interaction with third party providers as well as matrix manipulation for feature based recommendations using the *NumPy* and *pandas* packages. All of the following data processing techniques are performed server-side due to their computational demands, aiming to provide a smooth-loading web experience for the user where possible.

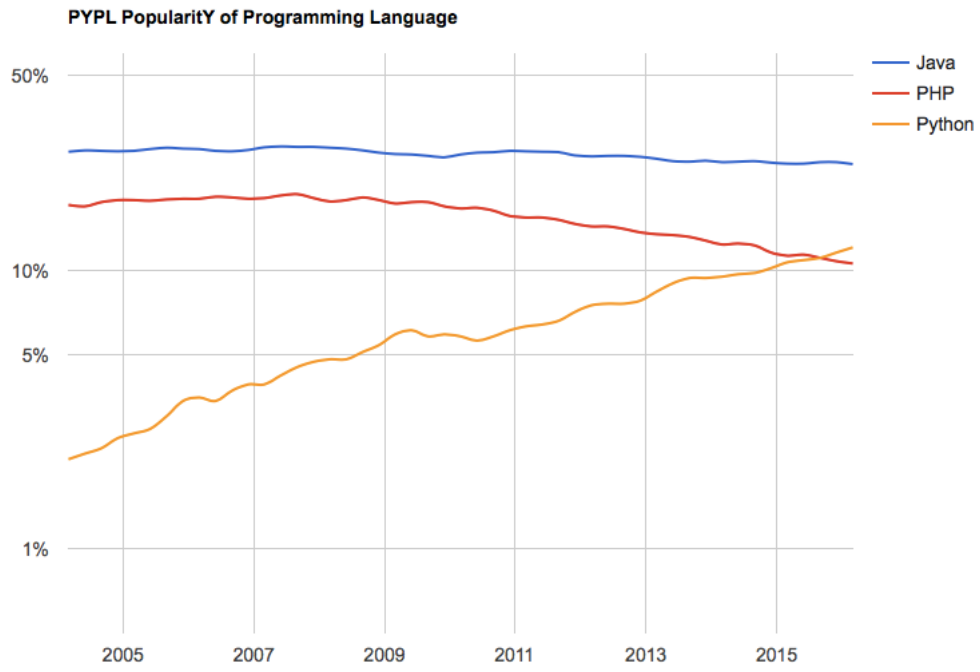


Figure 6.1: Programming Language PYPL Share Growth [5]

6.1.2 Data Storage

Data storage is handled by an open-source database known as MongoDB [32]. MongoDB is an example of a NoSQL database, which is a data store used for storage and retrieval of information in a non-relational manner. It features dynamic schemas, a rich querying language, and simple administration abilities making it a fully fledged alternative to a relational database. It takes a document-oriented approach which constructs a database via a collection of documents, with each of these documents having key/value attributes. Documents are also not constrained to a database schema, meaning design choices can be flexible and suit an agile methodology. The ability to evolve a schema as the application evolves without requiring an expensive migration process is vital in such a design methodology. Furthermore, the lack of a strict schema allows information from varying third party sources to be added quickly and easily regardless of format.

6.1.3 User Interface

The web-based user interface will utilise Ruby on Rails, also known as Rails [33]. Rails is a framework and software library that extends the Ruby scripting language, with an emphasis on building web based applications. Via a combination of HTML, CSS, Javascript and the design principles behind both Rails and the Model View Controller (MVC) framework [34], web apps can be created with ease. Functionality is abstracted from the interface, which allows modular design not only resulting in cleaner code but also clear separation of required elements. Ruby also provides what is known as the RubyGems package manager, which provides third party libraries similar to Python's PyPi service known as gems. These will be useful for third party interaction and also smaller required functions in general, aiming to focus on the task at hand instead of building core functionality with little return in terms of Revolvr's operation [35].

6.2 Data Collection

Data collection is fundamental in a system such as Revolvr, and without it the application would serve no useful purpose. The aggregation of both user media and popular media as outlined in Section 5.2 provides a basis to make recommendations, and therefore the correct implementation and resulting functionality will be paramount to the success of the project. The follow subsections will highlight how this information is collected, from a user authenticating to the end point of their media being aggregated from third party sources.

6.2.1 User Authentication

When a user visits the home page, they are offered the option to sign up to the site using Facebook's login system. This process utilises principles of the OAuth 2.0 protocol as described in Section 5.2.3.1, and is realised via the use of the OmniAuth Ruby gem. OmniAuth provides multi-provider authentication for web applications, providing a number of third-party *strategies* that allows OmniAuth to connect to disparate systems [22]. During login the Facebook OmniAuth strategy makes use of the OAuth 2.0 authentication protocol to connect to a users Facebook and provide user information based on their Facebook profile, which is stored in a *user* object in the database to represent a user. The initial user model following this process has the following fields:

id A unique ID generated by the DBMS

provider A text field containing the provider this token belongs to

uid A user's id on the appropriate service, usually their chosen username or numerical id

name The full name of the user, as given by Facebook

oauth-token The authorisation token given by the service

secret-token The secret authorisation token given by the service

oauth_expires_at An ISODate formatted date-time stating the date at which the given token expires

algorithm The algorithm to be used to generate predictions, initially set as semantic

updated-at The time that the user last logged in

created-at The date and time that the user last used the system

tokens This is a list of embedded token documents, containing information on third party service the user has connected

image A link to the profile image on the user's Facebook profile

email A user's email address linked to their Facebook profile

Due to the use of Facebook to login explicit usernames and passwords are not defined, instead all authentication is handled via token exchange. A similar process occurs when connecting to third party providers, but instead of a user model being created an entry is made within the corresponding users tokens list as defined above. The fields are a subset of those in the user model (see Figure 6.29) but it is important to note fields such as *uid* are specific to the service the user is connecting to. For example, if the provider was Spotify the *uid* is their username, whereas on Twitter it is a unique user id normally hidden from the user. Storage of these tokens in such a way may seem excessive, but treating each service as a *token* including the Facebook influenced *user* model allows all tokens to be treated identically.

OmniAuth operates as a flexible blackbox, simply relying on the strategies implemented to provide correct functionality. Users are redirected to `/auth/:provider` where `:provider` is the name of the strategy. OmniAuth then takes the user through the necessary steps to authenticate them based on the provided strategy. An *authentication hash* is then returned when authentication has finished and Revolvr is returned to via a redirect to `/auth/:provider/callback`. This hash *'contains as much information about the user as OmniAuth was able to glean from the utilised strategy'* [22]. This callback URL then redirects to a Ruby controller method, which allows the information from this hash to be accessed and stored. The operation for simply connecting a service or registering using Facebook differs at this point:

1. **New user registration:** The user information is taken from the Facebook authentication hash returned by OmniAuth and stored in a new *user* model. A session is then created through Rails' session handler.

2. **Existing user login:** The user token is updated to extend the expiry period. A session is then created through Rails' session handler.

3. **User adds service:** The new or updated token is stored in the appropriate users token list found in the user's *user* model.

6.2.1.1 Spotify Token Refresh

For each service in the system apart from Spotify, either a secret token is automatically used to reauthenticate the user when their original token expires, or the tokens are offline tokens with

Table 6.1: User Media Aggregation Quantities and Locations

Service Name	**Aggregation Method**
Facebook	No media is aggregated from Facebook.
Twitter	25 most recent Tweets by user checked for media links for the below services, if any such links are found they are resolved and the item is stored
YouTube	25 most recent videos in a users 'liked' video playlist
Vimeo	25 most recently liked videos
Soundcloud	25 most recently liked songs
Spotify	25 most recently saved songs

no expiry date. However, when implementing the authentication protocol for Spotify (*omniauth-spotify*) it became apparent that refresh functionality in the gem was not automatic via inspection of the appropriate GitHub page [23]. As a result, a custom token refresher was implemented and the main functions for this are shown in Figure 6.2. When the system requires access to Spotify using user specific authentication, this refresher is called with the user's Spotify token and their system user id. The user's id is used to update and store information about their new access token. The script uses a simple date time check to see if the token has expired, and if so makes a call to the Spotify API for a new token using HTTP requests. As seen in the *get_access_token* method the script sends a base64 encoded hash of Revolvr's public and secret tokens to authenticate the identity of the requesting service. Once Revolvr itself has been authenticated, the requesting user's refresh (secret) Spotify token is sent to the Spotify API authentication endpoint via a POST request. If successful, Spotify then returns a new access token which is used to replace the existing expired token for the user, and the expiry time is updated accordingly.

6.2.2 Aggregation

The system performs two types of aggregation as described in Section 5.2.3: user media aggregation and spotlight aggregation. User media aggregation is responsible for connecting and retrieving media items from each service a user has connected to within Revolvr via OmniAuth authentication. In contrast, spotlight aggregation is not user specific, and instead aims to collect the most popular media from Spotify, YouTube, Vimeo and Soundcloud. Tables 6.1 and 6.2 highlight the areas in each service which media is aggregated from, as well as how many items are aggregated for both aggregation types

6.2.2.1 User Media Aggregation

Once a user has authenticated both their Revolvr account and the respective services they wish to provide information from, authentication tokens are now available to request personalised information from these services. All algorithmic processes including aggregation occur once every 24 hours,

```
    def __init__(self, oauth_token=None, user_id=None):
        self.client = MongoConnector()
        self.oauth_token = oauth_token
        self.user_id = user_id

    def check_token(self):

        if self.oauth_token[u'oauth_expires_at'] <= datetime.datetime.now():
            print 'Requesting new access token'
            return self.get_access_token(self.oauth_token['oauth_token_secret'])
        else:
            print 'Access token valid'
            return self.oauth_token['oauth_token']

    def get_access_token(self, refresh_token=None):

        payload = {'grant_type': 'refresh_token', 'refresh_token': refresh_token}

        headers = {'Authorization': 'Basic '
                    + base64.standard_b64encode(client_public_token
                    + ':'
                    + client_secret_token)}

        r = requests.post('https://accounts.spotify.com/api/token', data=payload,
    headers=headers).json()
        token = r['access_token']
        expires_at = datetime.datetime.now() + datetime.timedelta(0,r['expires_in'])
        print 'replacing \n' + self.oauth_token['oauth_token'] + '\n with \n' + token
        self.update_token(self.user_id, token, expires_at)

        return token

    def update_token(self, user_id=None, token=None, expires_at=None):

        self.client.current_db.users.update(
            {
                "_id": user_id,
                "tokens": { "$elemMatch": { "provider" : "spotify"}}
            },
            {
                "$set": {"tokens.$.oauth_token": token, "tokens.$.oauth_expires_at":
    expires_at}
            }
        )
```

Figure 6.2: Spotify Token Refresher (spotify_token_refresh.py)

Table 6.2: Spotlight Aggregation Methods and Quantities

Service Name	**Aggregation Method**
YouTube	API call to retrieve top 20 most popular videos of the day
Vimeo	API call to retrieve 20 most recent videos from the staff picks playlist
Soundcloud	Custom HTML parser written to retrieve top 20 song links from `https://soundcloud.com/charts/top`
Spotify	API call to retrieve 20 items from official daily Spotify playlist consisting of most popular UK songs

and are batch processed instead of on a user specific basis. Aggregation occurs by considering each user sequentially and the service tokens available in their *user* model entry, and for each valid token aggregating a selection of media which indicates preference. When a service is found to be present in a users list of tokens a service specific media connection and collection function is called. These functions take a user's information and the corresponding access token and collect their liked media from the appropriate service.

YouTube, Vimeo Soundcloud and Vimeo (i.e the direct media providers) all operate similarly. Figure 6.3 shows a code snippet of a service specific connector which in this case connects and aggregates from Spotify. This example uses a Python library known as *spotipy* which provides interaction with the Spotify API and simplifies the process.

This method firstly authenticates the user using their stored Spotify access token returned from OmniAuth. Following this, it retrieves and stores their 25 most recently saved tracks into a list of entries containing the track name, a blank description, and the Spotify URI which is used for embedding the item within the user interface. A blank description is used for consistency, as all other services offer a form of an item description apart from Spotify. The general idea is the same for each service specific connector, however API restrictions causes differences in how each one was implemented. For example, the YouTube API has no direct Python library for interacting with the API. Instead the *request* package must be used to make explicit calls to the API endpoints. As a result the code is more convoluted and raw JSON data returned from the requests had to be manipulated, and authentication also had to be explicitly requested as opposed to a simple constructor method. Figure 6.4 shows the `get_media` function for the YouTube connector. This snippet of code is around half of the whole process of connection and aggregation, but highlights how service specific differences have effected what originally appeared to be identical functionality.

Aggregation of media from Twitter works slightly differently due to the fact Revolvr aims to promote 6 services, 4 of which are the utilised media providers. Originally all media from Twitter was going to be entered into the system, and the original implementation extracted all images and links from a users tweets. However, this media was never displayed once aggregated and often offered little insight into a user's preferences. As a result, Twitter was instead used as a secondary source for media from the four content providers and also for the social based algorithm. In terms

```
import spotipy
from spotify_auth import client_public_token, client_secret_token

class SpotifyConnector:

    # Initialiser checking for authentication
    def __init__(self, oauth_token=None):
        if oauth_token is not None:
            self.spotify = spotipy.Spotify(auth=oauth_token)
        else:
            self.spotify = spotipy.Spotify()

    # Gets info on latest 25 tracks from a users 'liked' songs on spoify
    def get_media(self):

        tracks = self.spotify.current_user_saved_tracks(limit=25)
        self.media = []

        for item in tracks['items']:
            track = item['track']
            description = ''
            self.media.append({'name' : track['name'], 'link' : track['uri'], '
    description': description})

        return self.media
```

Figure 6.3: Spotify Aggregator / Connector lines 1 to 25 (spotify_connector.py)

of aggregation a call is made during aggregation to authenticate access to a users Tweets using their respective token, and the most recent 25 tweets are extracted. Any links embedded in these tweets are then obtained and processed to see if they contain YouTube, Soundcloud, Vimeo or Spotify in the URL. However, a link containing these provider names is not necessarily a link to a media item for all services. Spotify and YouTube explicitly state the item ID in the hyperlink, which can simply be obtained via string manipulation and passed into the appropriate API to retrieve the item information. Soundcloud and Vimeo links do not contain item ID's as an argument and instead go for a `.com/user/item-name` approach, however their API's offer methods to resolve non-API formatted links to and retrieve information about the item like a normal API call. If the link is not a valid media link an exception is thrown that can be handle this case. As a result, a users most recent 25 Tweets can be analysed and media items can be retrieved in an identical format to the other service aggregators.

Once each content aggregator has been run for a user the general aggregation algorithm now has a media list consisting of `(provider,[item])` pairs, where `[item]` is a list of aggregated media information. Each of these items then undergoes processing (see Section 6.3) and storage. Media items are stored in two locations: firstly the *media* collection and then the *user_media* collection. The first insertion into the *media* collection is simply to contain information about the item within the following fields:

```
    def get_media(self):

        media = []
        access_token = self.get_access_token()
        headers = {'Authorization' : 'Bearer ' + access_token}
        payload={
            "part" : "contentDetails",
            "mine" : "true",
            "maxResults" : 25
        }

        r = requests.get('https://www.googleapis.com/youtube/v3/channels', headers=
    headers, params=payload)

        try:
            l_id = r.json()['items'][0]['contentDetails']['relatedPlaylists']['likes']
        except IndexError:
            return media

        likes = requests.get('https://www.googleapis.com/youtube/v3/playlistItems?part
    =contentDetails&maxResults=25&playlistId=' + l_id, headers=headers).json()

        for like in likes['items']:
            video_id = like['contentDetails']['videoId']
            url = 'www.youtube.com/watch?v=' + video_id
            r = requests.get('https://www.googleapis.com/youtube/v3/videos?part=
    snippet&id=' + video_id, headers=headers).json()
            if len(r['items']) > 0:
                name = r['items'][0]['snippet']['title']
                description = r['items'][0]['snippet']['localized']['description']
                media.append({'link' : url, 'name' : name, 'description' : description
     })

        return media
```

Figure 6.4: YouTube Aggregator / Connector lines 11 to 40 (youtube_connector.py)

id A unique item id generated by the DBMS

link A URL or service specific URI for the item

name The name of the item

description A description of the item taken from each service, i.e YouTube video description

Following this insertion, the id generated is returned and used as the id for the item as it is inserted into the user_media collection. This ensures that if an items id is x, it is referred to as x in other tables meaning there is a one to one mapping between the set of id's and items. The *user_media* entry has the following fields:

id The items existing id in the system

link A URL or service specific URI for the item

user_ratings A list of (u,id) pairs representing user ratings associated with a media item

Entries in user_ratings imply that user u has preference for the item the list belongs to, and the id is again generated by the DBMS. The important point here is that the *media* collection acts as a simple information store for media items, ensuring duplicates do not enter the system and that information about an item can be accessed. Entries in the *user_media* collection on the other hand serve to store which users are interested in which media items.

6.2.2.2 Spotlight Aggregation

In order to provide a common area of discovery for Revolvr, Spotlight provides an aggregated view of the most popular media from the systems key media providers. The motivation from this came when researching Hu et al.'s paper based on implicit feedback as discussed in Section 5.3.2. When comparing other models to their own they found that recommending shows based on their popularity was a *'surprisingly powerful'* naive measure, and that *'crowds tend to heavily concentrate on few of the many thousands available shows'* [16].

Similarly to user media aggregation, the algorithm iterates through a number of individual aggregators for YouTube, Vimeo, Soundcloud and Spotify. Each of these aggregators at a high level simply retrieve the 20 most popular items from the location stated in Table 6.2, however each provider varies in the operation of its aggregation. These aggregators will be known as *top aggregators*.

- **Spotify:** The Spotify top aggregator first requests access to make API calls via the client credential work flow. This authenticates Revolvr using its application credentials as opposed to accessing the API with user permissions. Once authorisation has been granted by the authentication end point the access token is used to access an official playlist of the most popular tracks in the UK for the current day. The top 20 of these tracks are then returned to be inserted into the system.

- **Vimeo:** Vimeo does not require authorisation to access public material such as playlists, and as a result a simple GET request for the video on the 'Staff Picks' playlist is made. Information regarding the first 20 of these videos is then processed for insertion.

- **YouTube:** This is identical to the process taken for Vimeo apart from authentication via Revolvr's client key must be presented to access the most popular videos within the last 24 hours. Retrieving the most popular videos is available via the API and therefore is again a simple procedure of requesting video information and selecting the appropriate details from the top 20.

- **Soundcloud:** The Soundcloud API unfortunately offered no way to simply retrieve the most popular songs due to its restrictive API. However, Soundcloud offers a chart of the top 50

most played tracks of the week via `https://soundcloud.com/charts/top`. Although this does not update daily, it still offers a partial solution to the problem and is better than not providing this information at all. As a result a HTML parser was implemented, which is used to extract the links from the charts page. BeautifulSoup is a Python library that allows the extraction of data from HTML and XML files and was used to perform the scraping of the raw HTML [36]. These links are then resolved via the Soundcloud API to return information about the track similarly to if the track id was passed into the API call. This parser is shown in full in Figure 6.5.

The result from this aggregation are then presented identically to the results of the user media aggregation in `(provider,[item])` pairs. However, these items are not stored in the *media* collection and are instead stored in a collection known as *featured_media*. This collection is dropped and reconstructed on a daily basis and therefore the media items do not permanently enter the systems media store unless they are inserted into the *user_media* collection via internal user feedback.

6.2.3 Repeated Aggregation

During aggregation it remains crucial to reduce computational overhead where possible, as well as ensure duplicate items do not enter the media table and effect the recommendation algorithms. The *user_media* table defined in the previous section allows the system to maintain not only what items have been imported into the system, but also what users have already shown a preference for them. When new media enters the system it is checked to see whether it exists in the system at all using the `media` store. If it does already exist the item is not inserted, but instead the ID of the item is taken from the appropriate `media` entry, and is then looked up in the `user_media` table. If the user has already liked this item, no action is taken in terms of noting a preference. Furthermore, as the 25 items per service are considered in chronological order from most recent to oldest if a preference already exists for that item and user the aggregation can be terminated early. The algorithm can then avoid work with no benefit and move onto the next service or user. Aggregation is repeated daily at midnight along with other processing elements. This ensures that a user's represented taste profile is always up to date whilst still avoiding excessive computation via the use of batch processing.

6.3 Processing

Once media has been aggregated via the implemented procedures detailed in Section 6.2.2, the system has a store of asymmetric feedback representing a users taste profile that can be fed into a number of algorithms to retrieve meaningful recommendations. The three implemented algorithms discussed in the following subsections are based around the idea of semantic analysis, matrix factorisation / latent factor models, and social media connectivity. Each of the algorithms implemented will contain an overview on both their methodology and technicalities aiming to provide a comprehensive summary of their operation.

```
import soundcloud
import urllib2
from bs4 import BeautifulSoup
import requests
from soundcloud_auth import client_public_token, client_secret_token

class TopSoundcloud:

    def __init__(self):

        self.client = soundcloud.Client(client_id=client_public_token, client_secret=
    client_secret_token)

    def discover(self):

        html = urllib2.urlopen('https://soundcloud.com/charts/top').read().decode('UTF
    -8')
        soup = BeautifulSoup(html, 'html.parser')
        links = soup.findAll('a', {'itemprop' : 'url'})
        media = []
        i = 0

        while len(media) < 20:
            link = ''.join(['http://www.soundcloud.com',links[i]['href']])
            song = self.resolve_song(link)
            if song is not None:
                media.append(song)
            i += 1

        return media

    def resolve_song(self,link):
        try:
            url = self.client.get('/resolve', url=link)
            return ({'name' : url.title, 'link' : url.uri, 'description' : url.
    description})
        except requests.exceptions.HTTPError, e:
            print (e)
            return None
```

Figure 6.5: Soundcloud HTML parser (soundcloud_connector.py)

6.3.1 Semantic Based Analysis

In semantic based analysis the system aims to create a graph of connected users based on similarity, where similarity is a measure based on the commonality between the titles of media items between users. The process of tag deconstruction occurs concurrently with the aggregation process to prevent unnecessary repeated access to the database and provide a logically ordered process. Once media items have been processed and the appropriate tags have been constructed and stored, the

user-base is represented as a network in which users have varying levels of similarity. This graph is used to recommend media to a user based on similarity values.

6.3.1.1 Identifying Similarities

Once an item is inserted into the appropriate *media* and *user_media* tables, its title is deconstructed into a number of elements that aim to describe the media item in a more meaningful way. A title can be constructed into either *phrases* or *tags* and these phrases will be used interchangeably, however, a tag only refers to a single term whereas a phrase can be multiple terms.

Title deconstruction occurs by considering the terms in a title sequentially, with each one firstly being processed undergoing a validity check to avoid ambiguity. Firstly, each word in the title is stripped of punctuation and is analysed to determine if it is a common English word. If so, the term is removed and the process continues onto the next term. If it is not a common word, the origin term is converted to lower case, stripped of punctuation asides from apostrophes, and also any numbers are removed. This new term is then compared against a custom `media` corpus. This is simply a developer defined list of words that should be ignored to increase the quality of tags and includes words such as *official*, *video*, *hd* and *preview*. All of these words are often included purely to inform the user about the media item, and do not offer a direct insight into preference like an artist or series name would. Figure 6.6 displays the operation this process using an example and was taken from the original Revolvr documentation. In this example the item is split into its individual terms, undergoes the described processing, and results in two terms remaining which will be stored and used in processing. Tags that remain after this process are stored in the database in a *tags* collection, along with the full resultant title as it is possible for two similar items to be broken down into the same title consisting of the same terms. Entries within the *tags* collection have the following attributes:

id A unique key generated by DBMS

phrase The term taken from a media title

tag_users A list of users who have shown interest in this phrase, denoted as *(u,f)* pairs

associations A list of phrases with which this phrase has been collocated to, denoted as*(p,f)* pairs

The *(u,f)* pairs within the *tag_users* field are *user-frequency* pairs which indicate how many times a user has shown interest in the given phrase. The *(p,f)* pairs in the associations list are *phrase-frequency* pairs which denote how many times a given phrase has been associated with its parent phrase. As a result of this decomposition, phrases with a high frequency value within the *phrase-frequency* pairs indicate a trends between terms. A frequent collocation of terms is likely to suggest a relevant between the two and result in the formation of a similar link.

6.3.2 Relating Users

Following the production of the set of tags, a number of deductions can be made. If a tag has no associations then it is a entire title being stored as a phrase, which can be used to create a

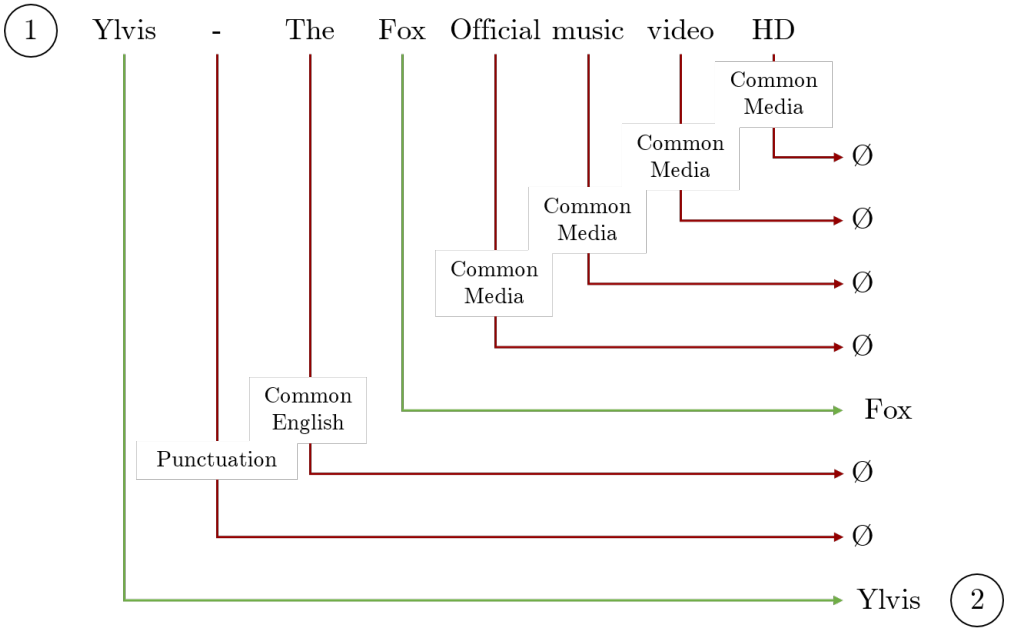


Figure 6.6: Example Tag Decomposition [4]

direct link. Otherwise, the tag has been associated with other tags a number of times and this may correspond to a similar link. These direct and similar links can be used to create an overall *similarity value* acting as a basis for similarity and therefore recommendation.

Firstly, a list of possible similar tags (known as *basic tags*) is constructed by sequentially checking each tag in the `tag` collection for associations. If associations are present for a tag, then the phrase itself and its associations are added to the list of basic tags. These are in the form `(phrase, (associated_phrase), frequency), users)` and are generated for every tag-association pair. The `phrase` is the tag in question, the tuple consists of an associated phrase and its frequency, and `users` is a list of *user-frequency* pairs. This is constructed per association, and results in a number of possible similar links where the endpoints are two individual tags. This basic tags list is then reduced to any tuples where the frequency for the association is more than a specific threshold, currently 1. The idea here is that repetition of collocation may indicate two terms are related and not down to just coincidence. As the system increases in size this threshold can be increased, but due to the small scale operation of the system a smaller value was required for appropriate recommendations. In leymans terms, this process simply looks at pairs of tags that have been associated known as t_1 and t_2, and when the frequency of association for t_1 and t_2 exceed some threshold they are treated as a similar link.

Direct tags on the other hand are simply taken to be phrases with no associations, implying its a media title stored in full. These are added to a list of direct tags each of the form $(phrase, users)$, where users consists of all users who have also shown a preference for that media item. Here the assumption is that user $u_1$and user u_2 both have shown interest in an item with title t, then a direct link should be produced between them.

Once these potential links have been established, each user in the system is considered for their specific link creation. These are generated on a user by user basis, with the following checks being performed for each user:

- **Direct Links:** To produce direct links for user u, every user u_i in the system excluding user u is checked against each direct tag d_j. If u_i is present in the list of users for tag d_j aswell as user u, then a direct link is created for user u between themselves, user u_i, and the tags corresponding phrase.
- **Similar Links:** For each user u in the system, a set of the similar tags for which they are present in the *users* list is created which will be denoted as S. S now contains a number of similar tags that user u has an interest in. Each user $u_i's$ set of similar tags is then intersected with user $u's$, which returns a set of similar links in the form of $(t1, t2)$. This indicates both user u and user u_i share an interest in the association $(t1, t2)$.

One these links have been determined and processed for each user, each link is added to a collection named *relations* which makes the use of a *relation* model and a *related user* model. These structures attributes can be described as follows:

id A unique reference ID generated by the DBMS

user The user ID of the user which this relation refers to

similar A list of similarly related users, determined by users who are present in the users similar links

direct_users A list of related directly related users, determined by the users who are present in the users direct links

user id The ID of the related user

phrase A list of phrases (direct or similar) which link this user and the containing user

The strengths of utilising a NoSQL document oriented database become clear here. By not requiring a schema, this allows users with only one of the two link types to be stored in the same database. Leading on from this, the free reign of allowing the user to store lists and other structures avoids splitting data over many strict and difficult to understand relations.

6.3.3 Selecting Appropriate Media

Following the creation of direct and similar links this information can be used to build the graph described in Section 5.3.1.2 and generate appropriate recomendations. This stage of processing occurs after tag analysis has completed and is the final stage related to semantic based recommendations. To begin with, the direct and similar tags stored represented via the relation model are utilised to construct numerical links, part of this process is shown in Figure 6.7. Here the `rank` dictionary object contains the fields *user*,*direct* and *similar* which contain the user ID, the number of direct links and the number of similar links they belong to. Direct links offer a greater insight

to similarity than direct links, and therefore receive a weighting via a *weighting* factor which is provided as a variable within the script and can be altered as necessary. By default the value is set to 10 meaning direct links have a much greater influence on the overall similarity rating than a similar link. These numerical links are then utilised within a *User Graph* model, which contains a single entry for each user in the system with multiple *Rank* models embedded within it. Ranks models for user u provide information on the similarity rating and its contributing factors to a user u_i. These models contain the following fields:

user id The ID of the user whose information this model contains

ranks A list of ranks associated with the user, defined by the *Rank* model

id A unique reference ID for this entry generated by the DBMS

total The total similarity rating

direct The number of direct links

similar The number of similar links

user The user ID of the user responsible for the link

Following these calculations a list of similar users ordered by the similarity rating are returned in reverse order, meaning the most similar users according to this process are found at the top of this list. This list is traversed and for each similar user media is collected. When selecting media for user u from each similar user u_s from this list, a subset of $u'_s s$ media is taken. To ensure items the two users have in common are not redisplayed, only items liked by u_s that user u has not shown a preference for (as indicated by the user media table) are selected. Selected media is inserted into one of three staged media collections, specifically *Semantic Staged Media*. These collections contain recommendations from each algorithm in the system, and allows multiple algorithms to make generations to consume from. These records are disposable and are updated each day when recommendation generation occurs, and acts as temporary store of the recently recommended media. By batch processing recommendations then simply retrieving them from the appropriate staged media table this reduces strain on the system as opposed to generating the recommendations on each request, which would be unscalable and cause delays. Instead, selections are precompiled and are available for display with a single request. The *Semantic Staged Media* collection's fields are described as follows:

id A unique reference ID generated by the DBMS

media The ID of the media item being recommended

similar_user The ID of the user this recommendations comes from

user The ID of the user that the item is being recommended to

similarity The total similarity rating between the two users

```
def rank_order(self, relation):
  #Create a list of all users which are at all related to this user
  ranks = [{'user' : x} for x in set([y['user'] for y in (relation['similar'] +
  relation['direct'])])]

  for user_links in relation['similar']:
    for rank in ranks:
      if rank['user'] == user_links['user']:
        rank['similar'] = len(user_links['links'])
        continue

  for user_links in relation['direct']:
    for rank in ranks:
      if rank['user'] == user_links['user']:
        rank['direct'] = len(user_links['links'])
        continue

  for rank in ranks:
    if 'similar' in rank.keys():
      if 'direct' in rank.keys():
        rank['total'] = int(rank['similar']) + (int(rank['direct'])*10)
      else:
        rank['total'] = int(rank['similar'])
    elif 'direct' in rank.keys():
      rank['total'] = int(rank['direct'])*10
    else:
      rank['total'] = 0

  return sorted(ranks, key=lambda rank: rank['total'], reverse=True)
```

Figure 6.7: Creation of numerical links (stager.py)

6.3.4 Feature Based Analysis

Feature based analysis utilises the concepts of latent factor models and matrix factorisation as outlined in Section 5.3.2, inspired by the work of Hu et al. [16]. The process will be discussed in three stages: the construction of the user-item matrix, the factorisation of this matrix into the appropriate item and vector matrices, and finally the generation of predictions. The implementation found within the system was based on Spotify's Chris Johnson's open source implementation provided on GitHub with appropriate changes where necessary [37].

6.3.4.1 Initialisation and Matrix Construction

The first function of this algorithm is to convert the collection of users (*users*) and user preferred media items (*user_media*) into a $n * m$ matrix where n is the number of users and m the number of items. These items and users are stored in a MongoDB document and required a mapping from this format to indices of a user-item matrix. Figure 6.8 highlights the method used to do this, where the variables *users* and *items* contain a MongoDB formatted collections of users and items

from the systems database. Counts is $n * m$ matrix initially full of zero values constructed based on the size of *users* and *items*. Firstly, all users in the system are iterated through in the order they were returned by the DBMS and their user id's are mapped to an integer index. *user_index* is a dictionary mapping users id's to indices, and *index_user* is simply the inverse mapping of indices to user id's. This results in each user u in the system having a respective index value of i which will be their corresponding row index in the user-item matrix. A similar process occurs for the items from the *user_media* table, mapping item v to column index j of the user-item matrix, however the matrix is also filled for each item at this point. Following item v being assigned an index, each user who has shown preference for this is looked up in the *user_index* dictionary via their user ID and a 1 is placed in matrix at $[u][v]$.

For the construction and general manipulation of matrices the *NumPy* and *pandas* packages were utilised [38, 39]. These packages allow matrix definitions and operations to be easily interpreted within Python, offering a number of standard matrix operators to allow computation of n-dimensional data. This allowed a faster development process with the ability to focus on the problem as opposed to technical constraints.

```
    def construct_matrix(self,counts):

        # Give all users an index for matrix
        for i,user in enumerate(self.users):

            self.user_index[user['_id']] = i
            self.index_user[i] = user['_id']

        # Give all items an index for matrix
        for i,item in enumerate(self.items):

            self.item_index[item['_id']] = i
            self.index_item[i] = item['_id']

            n = self.item_index[item['_id']]

            # For each user rating for this item, find the users index and put a 1 in
    matrix
            for user in (item['user_ratings']):

                m = self.user_index[user['user']]
                counts[m][n] = 1
```

Figure 6.8: Construction of the User-item Matrix (mf.py)

6.3.4.2 Matrix Factorisation

Following the construction of a $n * m$ user matrix, the asymmetric feedback presented in this format can be mapped to a latent factor space, employing implicit matrix factorisation as previously discussed. For a full reference of technical operation the file `revolvr/Processing/mf.py` should

be referenced, but the following section will provide a comprehensive discussion of key points. Firstly, the matrix of 1's and 0's is multiplied by the α value, as both the confidence format and preference format required later in the algorithm can be deduced from this representation. The selection of alpha in the system is currently set to 20, a value determined by a cross validation procedure discussed in section 7.5.2 which results in all 1's in the matrix becoming 20's. This can be explained by noting that our original matrix consists of values of r_{ui}, the rating user u has for item i. Confidence values are defined by $c_{ui} = 1 + \alpha r_{ui}$, and rearranging this equation gives $c_{ui} - 1 = \alpha r_{ui}$. This matrix is then converted into a sparse matrix where all 0 values are omitted to speed up future computation via the use of the *scipy.sparse* module [40].
The model is now ready to be trained via a predefined number of iterations of ALS, 30 iterations has been seen to be an appropriate value for convergence in many papers and was therefore the selected number of iterations [16, 41]. Figures 6.9 and 6.10 highlight the key elements of this process. To begin with the user and item vectors are initialised to random values and are of dimensions $n * f$ and $m * f$ which will be the starting point for the convergence to the optimal (minimal) solution. Following this, the concepts of ALS as outlined in section 5.3.2 are used to alternate between the sets of vectors and solve for the other, via equations 5.2 and 5.3 for the set number of iterations. An iterations consists of solving the appropriate equation depending on which vectors are being solved for, where again the NumPy and pandas packages were utilised for the computation of matrix products. To solve for x_u and y_i vectors SciPy's *spsolve* function was applied for every vector in the set of vectors [42]. This function takes two arguments `a` and `b` and solves the sparse linear system $Ax = b$ where b may be a vector or a matrix. This is equivalent to solving equation 5.2 or 5.3 depending on which vector was being solved for every entry in the set of respective vectors.

Once an iteration has completed the solved vector matrix is returned and the other set of vectors are solved for, and this process is repeated until convergence via 30 iterations. This results in a set of user vectors and a set of item vectors that can be used to make predictions of user-item preference score by calculating $p_{ui} = x_u^T yi$, where x_u is user u's vector and y_i is item i's vector. A $n * m$ matrix of predictions is then constructed using these products, as shown at the bottom of Figure 6.11.

6.3.4.3 Selecting Appropriate Media

Unlike the semantic algorithm which makes predictions based on stored information from the database, the matrix factorisation algorithm simply stores the appropriate media and everything else is temporary for processing. This is because unlike the user network the matrix cannot be stored and computed incrementally, it requires recompilation for every new item / user it wishes to process. Once the matrix of predictions has been constructed this results in each row representing a users user-item score, p_{ui} for every item in the system. Before predictions are made however, it is important to note that items the user already likes will dominate the predicted score. As a result, for any $[u][i] = 1$ in the original user-item matrix the predicted matrix has $[u][i]$ set to -1, to ensure items the user already likes are not recommended. To generate predictions for user u the row with index u is extracted from the matrix resulting in a $1 * i$ array, where i is equal to the number of items in the system. This array is then sorted based on the predicted scores it contains from highest to lowest, and the top 10% of results are taken and placed in the *Implict Staged Media*

```
    def train_model(self):

        # Initialise to random noise
        self.user_vectors = np.random.normal(size=(self.num_users,
                                                   self.num_factors))
        self.item_vectors = np.random.normal(size=(self.num_items,
                                                   self.num_factors))


        # For each iteration
        for i in xrange(self.num_iterations):
            t0 = time.time()
            # Fix item vectors and solve for user vector
            print 'Solving for user vectors...'
            self.user_vectors = self.iteration(True, sparse.csr_matrix(self.
    item_vectors))
            # Fix user vectors and solve for item vectors
            print 'Solving for item vectors...'
            self.item_vectors = self.iteration(False, sparse.csr_matrix(self.
    user_vectors))
            t1 = time.time()

            print 'iteration %i finished in %f seconds' % (i + 1, t1 - t0)
```

Figure 6.9: Training the Model (mf.py)

collection containing the following fields:

id A unique reference ID generated by the DBMS

media The ID of the media item being recommended

user The ID of the user that the item is being recommended to

This collection is similar to that used for semantic staged media, but does not have a notion of similarity as no connection between users is utilised. Although the implementation details could be considered simpler than those discussed for semantic analysis, the underlying design and operation of the algorithm as discussed in Section 5.3.2 is crucial to understanding, and must not be overlooked.

6.3.5 Social Analysis

The social algorithm can be considered the most simplistic recommender at a high level, simply generating recommendations based on the Facebook and Twitter connections a user has within Revolvr.

```
    def iteration(self, user, fixed_vecs):

        # Number of user / item vectors you are solving for
        num_solve = self.num_users if user else self.num_items
        # Size of fixed matrix
        num_fixed = fixed_vecs.shape[0]

        # Precalculate matrices they dont depend on u

        # Y^t Y calculated
        YTY = fixed_vecs.T.dot(fixed_vecs)

        # Identity matrix
        eye = sparse.eye(num_fixed)

        # LambaI
        lambda_eye = self.reg_param * sparse.eye(self.num_factors)
        # Initialise a vector to store results of recomputed factor vector
        solve_vecs = np.zeros((num_solve, self.num_factors))

        t = time.time()

        # For each item / user you need to recalcualte for
        for i in xrange(num_solve):
            if user:
                # if recomputing user vectors, retrieve their ratings
                counts_i = self.counts[i].toarray()
            else:
                # else, if recomputing item vectors, get all ratings for item i
                counts_i = self.counts[:, i].T.toarray()
            CuI = sparse.diags(counts_i, [0])
            pu = counts_i.copy()
            # setting preferences from c values.
            pu[np.where(pu != 0)] = 1.0
            # Calculate Ttrans
            YTCuIY = fixed_vecs.T.dot(CuI).dot(fixed_vecs)
            YTCupu = fixed_vecs.T.dot(CuI + eye).dot(sparse.csr_matrix(pu).T)
            xu = spsolve(YTY + YTCuIY + lambda_eye, YTCupu)
            solve_vecs[i] = xu

        return solve_vecs
```

Figure 6.10: An ALS Iteration (mf.py)

6.3.5.1 Discovering Connections

The final algorithm focuses on connecting user via social aspects to make recommendations, a contrasting approach to the semantic and feature based methodologies. As outlined in Section 5.3.3, the promotion of both Facebook login as a minimum requirement and also Twitter connectivity

```
    def predict(self,user_vectors,item_vectors):
        # For each index in reconstructed matrix, calculate using dot product
        predictions = np.zeros((self.num_users,self.num_items))
        for i in range(self.num_users):
            for j in range(self.num_items):
                predictions[i][j] = self.user_vectors[i].T.dot(self.item_vectors[j])

        return predictions
```

Figure 6.11: Generating Prediction Scores (mf.py)

allows users to be related within Revolvr via the connection and user-centric commonalities of these third party social media providers. The algorithm consists of three main stages: identifying which of users u's Twitter *follows* who also use Revolvr, identifying user u's Facebook friends who use Revolvr, and finally generating a set of recommendations based on what these common connections have shown preference for inside of Revolvr.

To begin with an empty dictionary called *twitter_users* is defined, which will act as a mapping from a user's Twitter ID to their Revolvr system ID and Twitter authentication details. This is achieved by first iterating through each user of the system and checking the list of services that user has authorised within Revolvr. If an authorisation token for Twitter is present this indicates they have connected Twitter to Revolvr, and a corresponding entry is added to the *twitter_users* dictionary. The keys of this dictionary are Twitter user ID's, and the values are embedded dictionaries containing their Revolvr user ID and the necessary Twitter authentication details, namely a public and private OAuth token. The construction of this mapping is necessary not only to indicate which users of the system have connected a Twitter account, but also for authentication purposes. To retrieve an individuals *'follows'* on Twitter an appropriate authentication protocol must be executed, therefore the stored OAuth access tokens are presented upon connection to the Twitter API for each user. Furthermore, the Twitter API operates in a way such that the list of users a user follows are returned as Twitter ID's [43]. By storing all Revolvr authenticated Twitter ID's in the keys of the *twitter_users* dictionary a simple set intersect can be made with the list of *'follow'* ID's to indicate which of these Twitter accounts is also registered with Revolvr. The initialisation and construction of this dictionary is shown in Figure 6.12.

Following this mapping, an iteration over users occurs again to not only generate a users set of Facebook friends who use Revolvr but also generate recommendations. Firstly, an instance of the Facebook Connector (also used in aggregation and referred to as the Facebook Aggregator) is created and authorised using the stored authorisation token for the user. A method in the connector named `get_friends` is called, which is responsible for firstly retrieving a list of Facebook friends for the user in question using calls to the Facebook graph API [44]. The graph API provides a way for Facebook registered applications such as Revolvr to read and write to the Facebook social graph, which can be viewed as a connected graph of Facebook users and their corresponding information. Due to Facebook being used to sign users up to the system, the Facebook ID is what dictates a users user ID in the system and therefore the two can be considered equal. This means when the API call is made to Facebook to request the ID's of the users Facebook friends they are already in

```
def get_twitter_users(self):

  for user in self.users:
    try:
      for service in user['tokens']:
        if (service['provider'] == 'twitter'):
          self.twitter_users[service['uid']] = {'id' : user['_id'], 'public_token' :
   service['oauth_token'],'secret_token' : service['oauth_token_secret']}
    except KeyError, e:
      print 'No service tokens for ' + str(user['name'])
      print (e)
```

Figure 6.12: Generating Twitter-User Mappings (social.py)

the correct format, and can simply be iterated through and passed into the database one by one to check if a user with such an ID exists. As a result, for a user u a list is returned of all users in Revolvr who are also friends with u on Facebook, meaning they should be considered connected. Once this list is constructed, a similar list for a user u is constructed using the *twitter_users* dictionary. Firstly, the dictionary is iterated through to check if user u's id is present within the tuple value for a given Twitter ID to tuple mapping. If so, this indicates user u has connected a Twitter account to Revolvr and a call to the Twitter API using the credentials stored within the tuple via the Twitter connecter takes place using the `get_follows` method. This method acts on an authenticated user, retrieving a list of Twitter IDs of users they follow whilst taking the dictionary `twitter_users` as an argument. These two lists of Twitter IDs are then compared for commonality, and any ID that appears in both corresponds to a user who uses Revolvr, has connected Twitter, and is followed by user u. If this case occurs, the users Revolvr ID which is stored in the key value pair within `twitter_users` is appended to a list of follows. When returned, this results in a list of Facebook friends and Twitter connections now exists for a user u in the form of Revolvr user ID's. The list of ID's for Facebook and Twitter connections is then subject to a union operation which results in a list of all Revolvr user ID's related to user u via Facebook or Twitter, where a relation is defined as a friend or a follow. Figures 6.13 and 6.14 show the respective connector methods for Facebook and Twitter.

6.3.5.2 Selecting Appropriate Media

Once discovery of connections has finished, the list of ID's can be used to query the `user_media` table to retrieve media that each respective connection of user u has shown a preference for. To again promote discovery and ensure the user is being recommended interesting and new media, any items that user u has already shown interest in are ignored when selecting media from each connection. Selected media is stored in the *Social Staged Media* collection, which contains identical fields to the *Implicit Staged Media* table discussed in Section 6.3.4.3.

```
def get_friends(self):

  fb_friends = (self.graph.get_object("me/friends")['data'])

  for friend in fb_friends:
        friend_id = self.client.current_db.users.find_one({'uid' : friend['id']})['
 _id']
        self.friends.append(friend_id)

  if len(self.friends) > 0:
        return self.friends
  else:
        print str(self.user['name']) + ' has no friends on Revolvr '
```

Figure 6.13: Retrieving Facebook Friends (facebook_connector.py)

```
def get_follows(self, twitter_users):

  self.twitter_users = twitter_users
  friends = []
  try:
    follows = self.api.friends_ids(self.uid)
    for key, value in self.twitter_users.iteritems():
      if int(key) in follows:
        friends.append(value['id'])
  except tweepy.TweepError, e:
    print(e)

  return friends
```

Figure 6.14: Retrieving followed Twitter users (twitter_connector.py)

6.3.6 New User Processing

The described processing is carried out once every 24 hours, meaning aggregation and recommendation via the three aforementioned algorithms can be considered a batch process. Batch processing refers to the execution of a series of programs on a set on inputs without the need for manual intervention, where in this case the inputs are the users and media items. This approach is required not only to ensure the computational overheard is reduced, but also due to APIs providing request limits meaning dynamic processing per request would be infeasible.

As a result, when a new user registers to the system they are entered into a *staging* collection, which maintains a list of users that remain to undergo the process of aggregation and suggestion. Consideration of how to handle this is important, as if a user is required to wait up to 24 hours for recommendations it is unlikely the system will captivate and persuade them to return. As a result a CRON job has been scheduled every 15 minutes to process users within this table and perform a custom aggregation, recommendation and media staging process specific to that user.

Due to the computational complexity and necessity of having to recalculate matrix factorisation recommendations from scratch, this prediction technique is only available after a batch process has occurred. However, the CRON job enables new users to see personalised predictions from the social and semantic algorithms shortly after signing up via the use of a modified *single* aggregator that only considers staged users. This ensures that not only the user feels engaged but also that computational strain and the likelihood of bottlenecks is reduced by not processing new users in real time. The CRON jobs for both staging and system wide processing can be expected to follow a number of conflicting trends: an increase in processing time with the size of the processing data (users and media), and a reduction in time with an increase in processing power.

6.4 User Interface

The existing implementation of Revolvr provided a responsive and simple means to provide recommendations, however, the end product did not necessarily live up to its full potential. Due to this, a number of changes were discussed as discussed in Section 5.4 in order to evolve the existing work into a stylish, immersive and uninterrupted viewing experience. The aim was to make Revolvr feel like a professional, production quality product as opposed to meeting the minimum requirements. With a system so focused on both the user and the idea of consumption, the user experience is paramount to its success and therefore a number of considerations have to be taken into account. Firstly, the growing share of mobile traffic emphasises a need for responsive design. Due to Revolvr having a limited timeframe frameworks were utilised for the implementation of the user interface, allowing rapid development and ideas to come to life without significant overheads. A full front-end framework known as Bootstrap was used for the UI implementation, which provides extensive JavaScript and CSS functionality for dynamic webpages as well as responsive design as standard. Via the use of Bootstraps responsive roots and further tweaking via CSS3 media queries the mobile experience does not make the user feel limited, but instead serves as another option to discover media they will love. Combined with Rails' MVC framework development of the user interface could be entirely abstracted from the data processing elements it depends on, allowing the developer to focus on displaying data in an immersive and intriguing fashion.
A major downfall of the existing system was the lack of dynamic content generation via AJAX technologies, meaning the generation of new recommendations required a page refresh. Furthermore, the existing method of displaying recommendations was extremely muddled as discussed in Section 5.4, flooding the user with an abundance of options instead of focussed presentation. As a result the developer has focussed providing a seamless user experience especially in relation to the loading and display of recommendations and media.

6.4.1 Common Elements

In order to provide a consistent, cohesive and familiar user experience, a number of elements within the user interface are common across all pages.

- **Design:** In order to be an improved iteration of Revolvr and not deviate from the original vision, the new UI has attempted to maintain the look and *feel* presented originally. The

design used is adapted from a theme named *Flappy* provided by Black-Tie [45]. This is a free to use theme with a provided attribution licence and therefore the required information and credit is provided within the footer of each page [4]. The theme promotes a bold, flat design in line with current web design trends. Although this may seem generic, it is important to provide a sense of familiarity for new users instead of scaring them away. To assist with design a number of external, free to use graphics and fonts were utilised. Firstly, the Font Awesome repository has provided a number of standard vector icons throughout the site, allowing time to be spent on the technical implementation as opposed to graphic design [46]. A strong, clean font known as *Lato* from Google's font repository was also used throughout [47]. Finally, a jQuery content slider named *bxSlider* was used to display both *Discovery* and *Spotlight* media, which will be discussed fully in the following subsections [48].

- **Navigation:** A navigation bar is present throughout every page on the site, acting as the key method of navigation as well as a reference to what page the user is currently on. This page contains links to various sections of the website. *Discovery* and *Spotlight* are immediately viewable, whereas *Profile* and *Sign Out* are located in a dropdown menu. This dropdown menu is a circular version of the logged in users Facebook profile image, adding a touch of personalisation to the website. The desktop variation of the navigation bar is shown in Figure 6.15. When the user visits the site via mobile the use of an image for the dropdown menu was not appropriate for smaller resolution displays. As a result, the user oriented functions of profile settings and signing out are embedded within the mobile drop down menu under the users full name. This version is shown in Figure 6.16. When a user is not signed in the navigation bar simply contains a link to the *About* page, and more importantly a link to sign up to the service due to the lack of personalisable elements for unregistered users.

- **Footer:** The footer is displayed across all pages of the site, and provides a number of links that although need to be included, are not necessarily key elements of the user experience and instead act as additional information. The footer contains: names of the creators of Revolvr, credit to BlackTie for the design inspiration, links to social media accounts, the Privacy Policy and the *About* page. The information included in this footer and the linked pages aim to provide information that for the user in an unobtrusive manner, streamlining the design and making the key functions clear to the user.

Figure 6.15: Desktop Navigation Bar

6.4.2 Home

The homepage of a website is undeniably one of the most important aspects of web design, with first impressions often being the deciding factor in whether or not a user will take the time to

Figure 6.16: Mobile Navigation Bar

explore the website. Studies have shown that people make snap judgement, and it takes only 50 milliseconds for users to form an opinion of a website based on a homepage [49, 50]. Elements such as structure, colours, spacing, symmetry and amounts of texts have all shown to leave an impression and effect a users judgement. This judgement in turn effects whether a user will stay on the site or leave, and therefore strong a strong homepage can be the make or break when enticing users to use a system. The homepage differs depending on whether a user is logged in or not, with the content dynamically being generated and differing for each case. If a user is logged in, the homepage aims to be personable to further emphasise the idea of familiarity and a one-on-one experience. Otherwise, it becomes vital to advertise the service and entice users to sign up, providing an appealing yet informative overview of what Revolvr does and why it will benefit a user.

Logged Out

When no user is logged in the aim of the homepage is to inform the user of what Revolvr does, and why it will benefit them. The majority of the homepage is taken up by a bold banner with a flat stylish graphic, which highlights the services that Revolvr utilises to provide recommendations. This banner is shown in Figure 6.17, accompanying the graphic is an option to sign up to the system and also the tag line *'Discover the undiscovered'* which highlights the discovery element of Revolvr. Whilst this tagline does not comprehensively explain Revolvr's functions, it aims to make the user curious and initially grab their attention with the aim of them wanting to seek further information about the system. Beneath this main banner is a brief description of what the system does, the services it utilises, and what information it uses from these services. As shown in Figure 6.18, these descriptions remain brief to allow the user to dive right in as opposed to reading excessive amounts of text. At the bottom of the homepage a small button will allow users to demo the system and receive non-personalised random recommendations, although this does not reflect the personalised nature of the system it allows the user to experience the system first hand and hopefully feel an incentive to use it. Finally a quote from *The Perks of Being a Wallflower* related to the element of discovery has been included above the footer, although a minor detail it nicely summarises the idea of sharing and discovery that Revolvr is built upon. This bottom section is shown in Figure 6.19 and concludes the discussion of the homepage for logged out users.

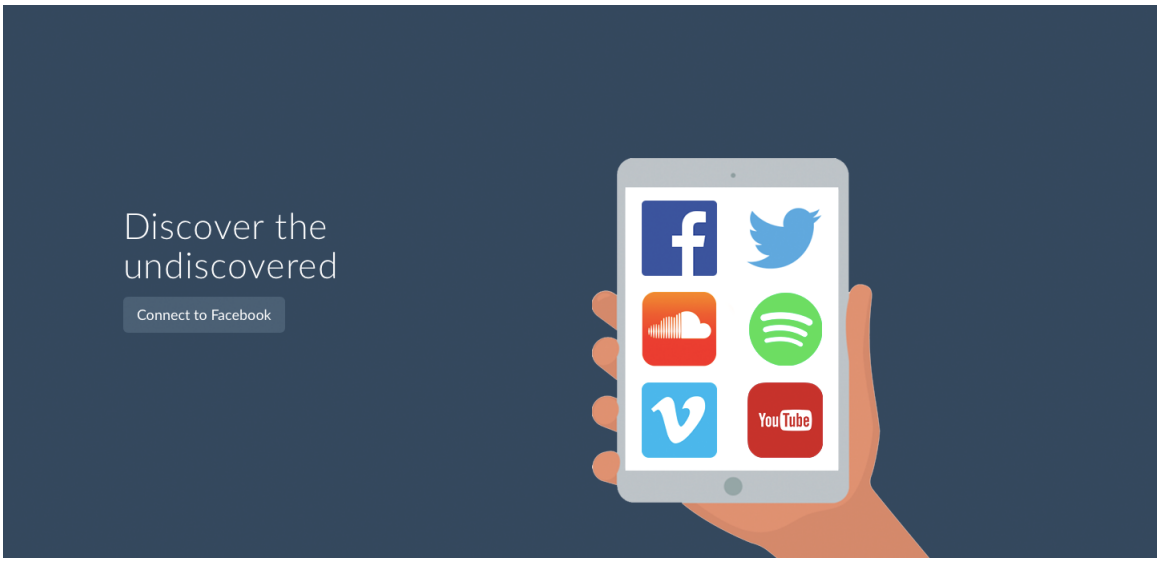


Figure 6.17: Homepage Banner

Logged In

When logged in the same sections of the homepage as the logged out state are shown. However, the tagline is replaced by a welcome back message which is personalised to the user, and a caption highlights that recommendations are ready and *'hot off the press'*. The signup button is also replaced with a button allowing the users to be redirected to discovery, meaning this banner acts as an influential element for users to explore their recommendations. The information about services is also accompanied by a connect button for each service urging users to connect their external accounts. Although this functionality is also present in the *Profile* page, placing it on the homepage makes the task of connecting services immediately available and therefore a user may be more likely to connect services immediately after signup and / or login. The quote and ability to trial the system are removed for logged in users, as by this point advertisement is less important and the homepage acts as more of a landing page for users to dive into discovery.

6.4.3 Discover

The Discovery view (formally known as the Gallery [4]) is where a user is able to view a subset of their staged recommendations which are selected via staging processes discussed in Section 6.3. As discussed in the design process, although the previous implementation allowed many items to be recommended on one page this resulted in a muddled and overwhelming recommendation process. Instead the new design presents items to a user in groups of ten, one item at a time via a plugin known as *bxSlider* and is shown in Figure 6.20. This slider is intended to allow users to cycle through HTML elements, however using Bootstraps column layout a generic template was made and embedded into a container within the slider allowing a title, description and embedded player

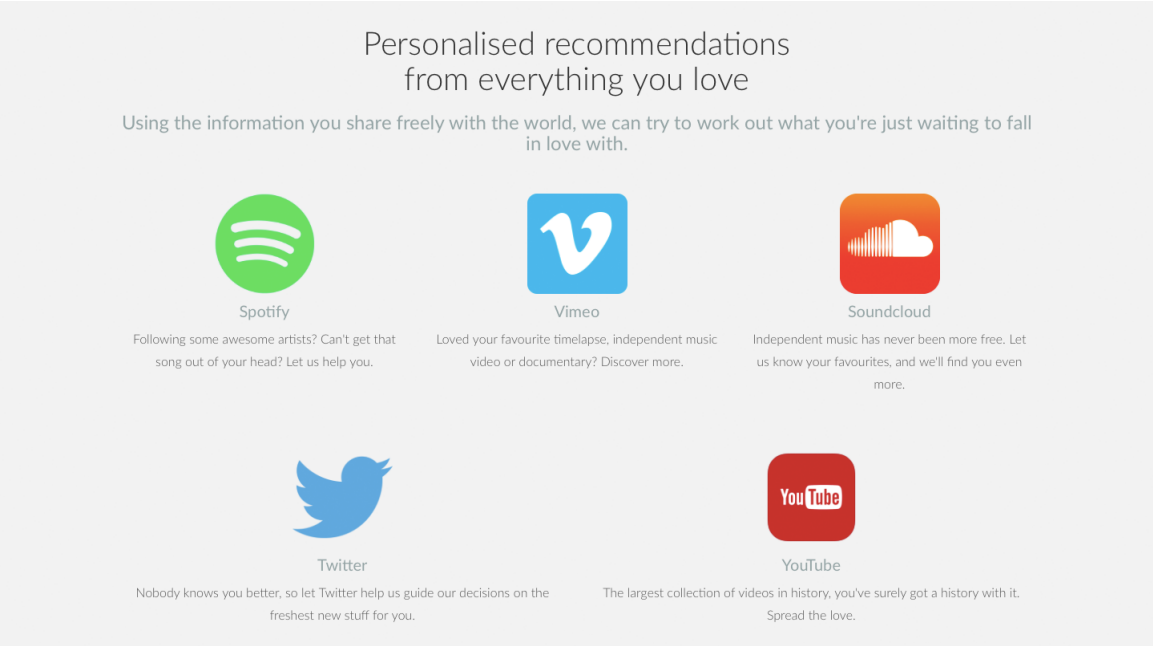


Figure 6.18: Information on Homepage

to be displayed on each slider. The arrows to the left and right of the player can be used to view the next and previous recommendations, as well as the slider being swipe responsive for mobile devices. The small dots at the bottom of the player also allow users to switch between slides without having to iterate through them one-by-one. The computation of media to display is dependent on the algorithm, but the general process can be abstracted for discussion. For the algorithm a user has chosen from the semantic, feature-based and social options the appropriate *staged media* table is queried for staged items belonging to that user. The computation of a subset of media upon a request to display recommendations is processed server side before being returned to the client. The semantic algorithm relies on a notion of similarity, and as a result all staged media is sorted into buckets of items with an identical rating before a random subset from each bucket is chosen, resulting in items with a mixture of similarity ratings. This avoids the recommendations being dominated by the most similar user. The feature-based and social algorithms instead simply select a subset of staged media, as there is no notion of one piece of media being more preferable than another in social and this information is explicit in the generation of feature based recommendations. Ten items are shown to the user, and a slider at the 11th index allows the user to load a fresh set of recommendations within the *bxSlider* instance via AJAX without refreshing the page. This means users have less of a delay between requesting and receiving recommendations is likely to increase the amount of time a user spends on the page. Originally more items were proposed to be shown, however iframes are expensive elements to render in vast numbers and ten was found to provide a good balance between quantity and performance. Figure 6.20 shows an embedded Soundcloud player, however YouTube, Vimeo and Spotify players are also supported. Due to restrictions in the

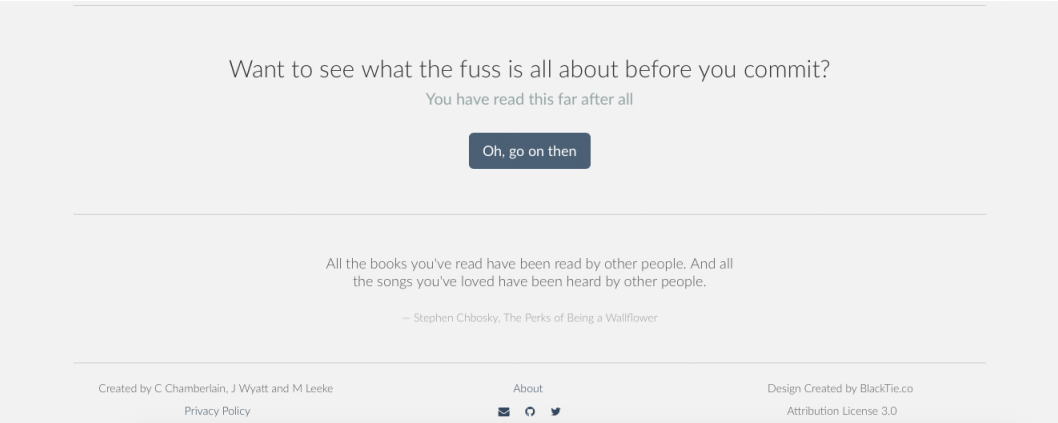


Figure 6.19: Homepage Banner

Spotify API the player unfortunately requires the user to launch the desktop or mobile application and have an account to consume media from the service. However, the tradeoff was deemed to be allowable as Spotify plays a huge part in modern day music consumption, and can be seen as a leader in the discovery and delivery of free audio material.

6.4.3.1 Feedback

At the bottom of each embedded player is a rectangular button with a heart on, which is used to save items and also provide asymmetric feedback within the system. The original implementation relied on separate web pages to provide feedback and also did not allow users to save items, however this streamlined approach reduces effort on the users part and is multi-purpose. This button also utilised AJAX functionality, when clicked the heart bolds to provide feedback to the user that the action has been completed and a server side call is made to add the user to the list of users who like this media in *user_media*. This button can be seen at the bottom of Figure 6.20.

6.4.3.2 Edge Cases

Due to discovery being a generic viewing platform it is important to consider a number of edge cases which may confuse a user if not dealt with. Firstly, unregistered users can access this page using the demo as discussed previously so they must be notified that the media they are seeing is random. Furthermore, there is also the possibility for each algorithm that recommendations cannot be made or are not ready in the following cases:

- **New Users:** When a user first signs up no recommendations are available until they are processed via the staging process discussed in Section 6.3.6. Whilst they are waiting for staging the user must be notified they need to wait for recommendations.
- **Semantic Staged Media:** If a user has no media they prefer within the system, no links can be inferred for this user and as a result no semantic recommendations will be available.

It is also possible the user has media imported into the system but they simply do not have anything in common with any although this is unlikely. As a result, a user must be warned if they have no semantic staged media due to either of these possibilities.

- **Implicit Staged Media:** Due to the nature of the implicit algorithm even if a user has no media imported into the system it is still possible to generate predictions despite their questionable quality. However, the computational demands of this algorithm means it is only run once every 24 hours and new users must be notified they need to wait until the next day for results from this algorithm.
- **Social Staged Media:** If a user has no connections via Facebook and Twitter in Revolvr they will have no staged media resulting from the social based analysis. If this is the case, the user must be notified of this.

For all these cases random media is shown in the player as opposed to recommended media, but a dismissible warning is displayed at the top of the page to let them know. These differ slightly for each case, but all forms explain the problem to the user and notify them that recommendations will be random for that case until further notice. An example warning is shown in Figure 6.21 for the case where no feature based recommendations are currently ready for a user.

By computing recommendations in batch requests to display media from the appropriate staged media table are low cost, and highlights the benefits of this approach as opposed to subjecting the system to strain by computing recommendations on each request.

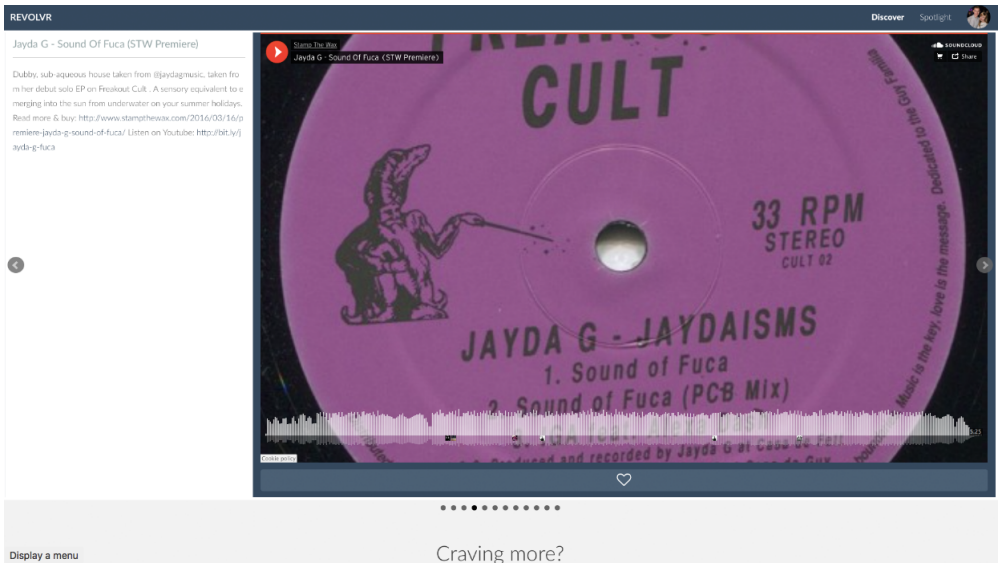


Figure 6.20: Content Delivery via the Discovery page

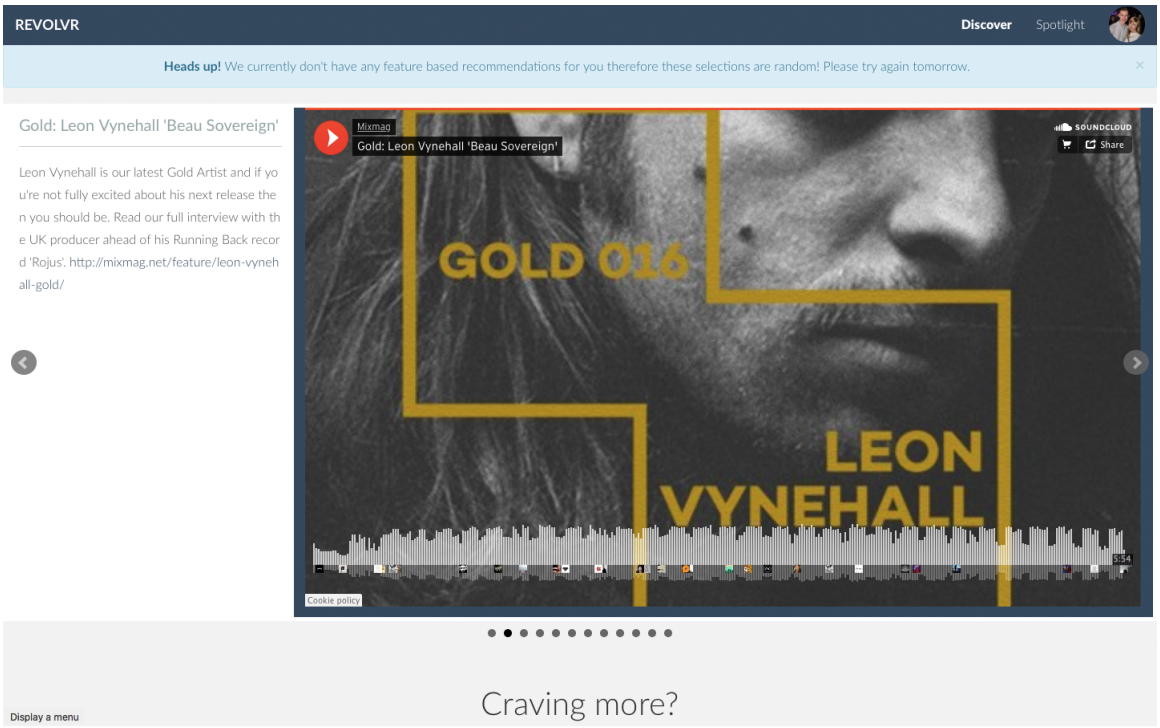


Figure 6.21: An Example Warning

6.4.4 Spotlight

Spotlight is a non-personalised portion of the site where users can view the 20 of the most popular items from each service, with selections being refreshed on a daily basis. The use of tabs above the embedded players allows users to select which service they wish to view content from, and when clicked will load the top items via AJAX from the *featured media* collection without the need for a page refresh. By separating the services users can have greater control over the media they view, and also avoids having to generate 80 embedded players on one page which is a costly operation. An example of the YouTube selections in the Spotlight view is shown in Figure 6.22. The layout is very similar to the discover view, this is an intentional design choice to have consistent design throughout the site to minimise confusion for a user.

6.4.5 Profile

The *Profile* page is the area of the website where a user can select their recommendation algorithm, see the services they have connected, connect any unconnected services, disconnect services, and view their liked items. The top of the profile page contains an image of the user and their name to name the profile page personable and indicate it relates to them. Javascript scroll effects are used on this element to make it fade as you scroll to make the page more dynamic and visually pleasing. Following this are three divs that act as buttons to select which algorithm the user would like to

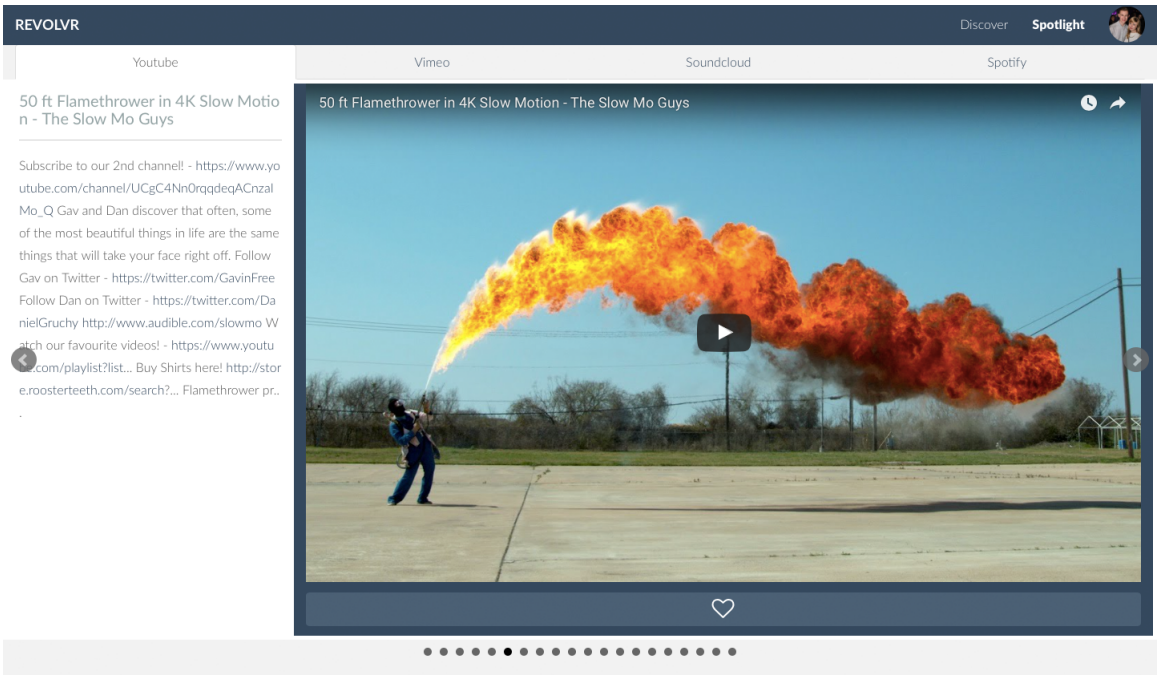


Figure 6.22: Spotlight View

utilise for recommendations. Each one gives a small user friendly description of the algorithm, and the selected algorithm is highlighted using a small glowing border around the appropriate divider. When clicked the user can send an AJAX request to the server to change their selected algorithm, meaning that this can be changed without the need for a page refresh. Following this, a list of icons belonging to connected third party providers are generated depending on the service a user has connected. Figure 6.23 shows the top portion of the profile page with the discussed features.

Following these sections is another icon bar providing the option to connect any service the user has not yet connected via Omniauth authentication. If the user has connected every service this section is not shown. Again, the icons shown here are generated based on information for that user from the database regarding what tokens are present in their list of connected services. Beneath this are two horizontal bars which can be clicked to display a list of any media a user has shown preference for via internal feedback or external aggregation. This is split into audio and video and entries are ordered chronologically to be able to find media without an issue. This is shown in Figure 6.24.

When the preferred media lists are expanded the user is able to click on any title to bring up a modal containing an embedded player to view the requested item. An example of this is shown in Figure 6.25, the player is slightly different to that used in the *Discover* and *Spotlight* pages due to being displayed in a popup. As a result *bxSlider* was not utilised here directly but a piece of Javascript was written to utilise similar behaviour to allow media to be cycled through without having to open and close the modal constantly. The arrow graphics from *bxSlider* were used in this

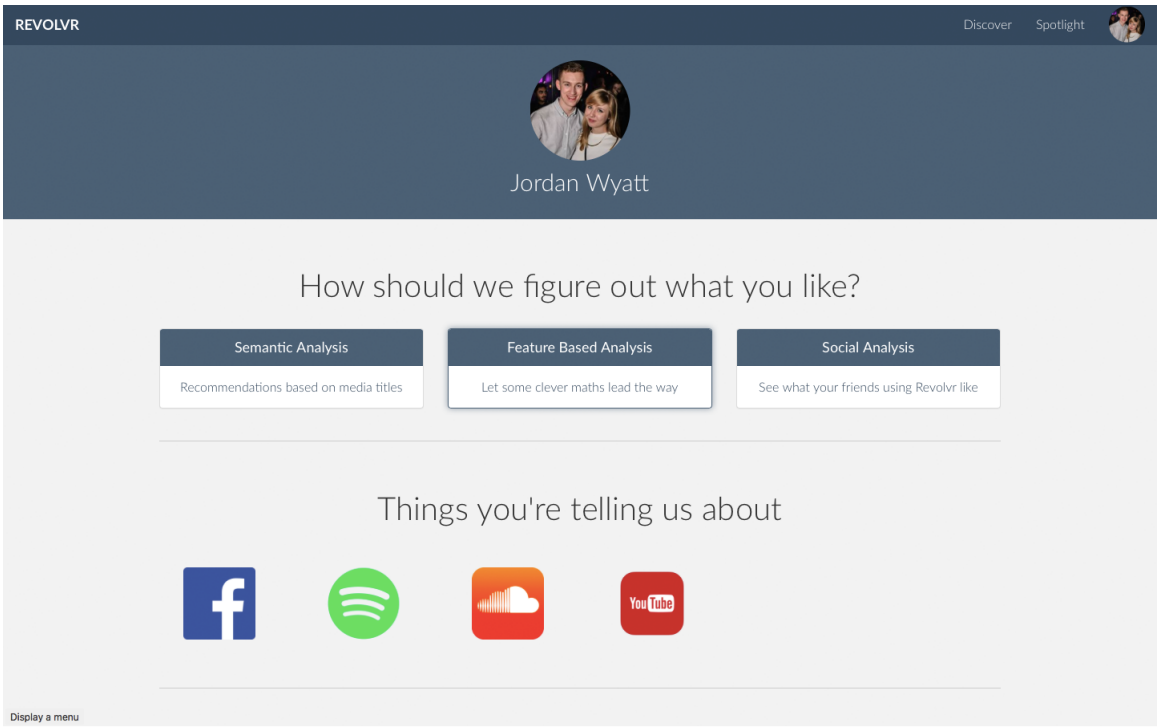


Figure 6.23: Top of Profile Page

custom modal player for consistency in design between players. In the list of media items the cross symbol can be clicked to remove this item from their preferred media which can be seen as 'unliking' an item both from the saved list and also for all variants of the recommendation algorithms. This removes the user in question from the items list of associated users in the *user_media* collection.

At the bottom of the profile page is a timeline and also the option to disconnect any connected services. The timeline is simply for novelty and to make the profile page more informative, displaying when the user has performed key actions in Revolvr such as signing up and connecting services and is shown in Figure 6.26. Finally, the bottom section of the profile allows users to revoke previously granted tokens in compliance with data protection and fair use of information, as shown in Figure 6.27. After the user confirms they would like to remove the service their user model is updated to remove the token from their list of services, and therefore media will not be aggregated in future. However, removal of a service does not result in the removal of any aggregated media from that user as information is not stored associatively.

6.4.6 Static Pages

The static pages of the site do not interact with any data in the database or server processes, and instead are simple HTML documents providing required information regarding the operation of the system. These remain identical to the original implementation as the focus of development was on

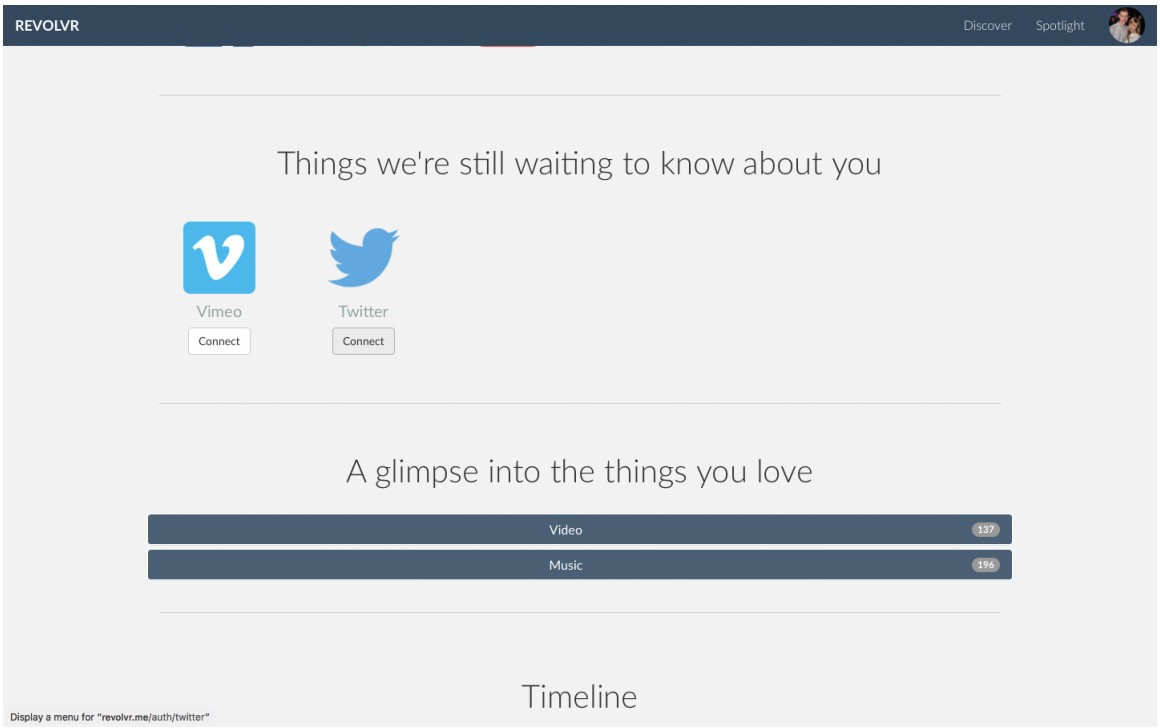


Figure 6.24: Middle of Profile Page

the data driven processes of generating and viewing recommendations. The two static pages used are the *About* page and the *Privacy Policy* page.

The *About* page highlights the three main goals of Revolvr: *entertainment*, *suggestion*, and *awareness*. They justify the need for such a system and outline Revolvr's goals in a concise and simplistic manner, as shown in Figure 6.28. The privacy policy does not require a full breakdown, but includes headings referring to the following questions a user may have:

- What information do you collect?
- How do you collect information?
- How do you use the information?
- What control do the users have over their personal data?
- How do you protect users' information?

Each of these headings is accompanied by a small body of clear textual explanations of how Revolvr goes about using protecting the data users grant it access to. By avoiding ambiguity issues regarding the understanding of how data is used and issues regarding the use of data can be avoided.

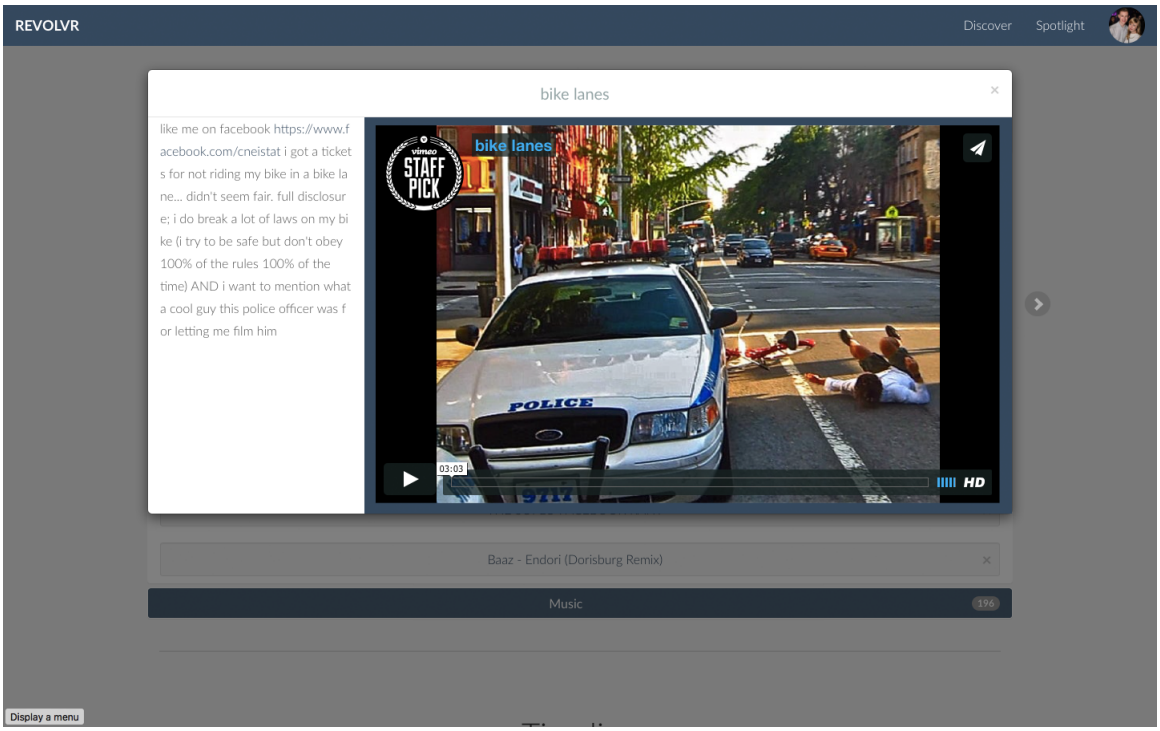


Figure 6.25: Modal Media Player

6.5 Overall Architecture

The final system encapsulates a process of data collection, processing and display with a simple goal in mind, to provide recommendations for users to enjoy with little effort on their behalf. With such quantities of data being imported and processed from many different sources it is vital that not only the relationship between data models and the order of processing remains clear, but the process pipeline itself.

6.5.1 Data Hierarchies

Due to MongoDb's schema-free approach, strict relations and normalisation of the database are not necessarily the prime focus. Instead, flexibility and rapid development are promoted but due to the lenient nature of the document-oriented model these relationships between elements must be clear. Figure 6.29 shows an overview of the core relationships between the implemented models, displaying a system that despite containing a number of documents still revolves around *Media* and *Users*. In this diagram, a model within a model implies that the documents are embedded with others.

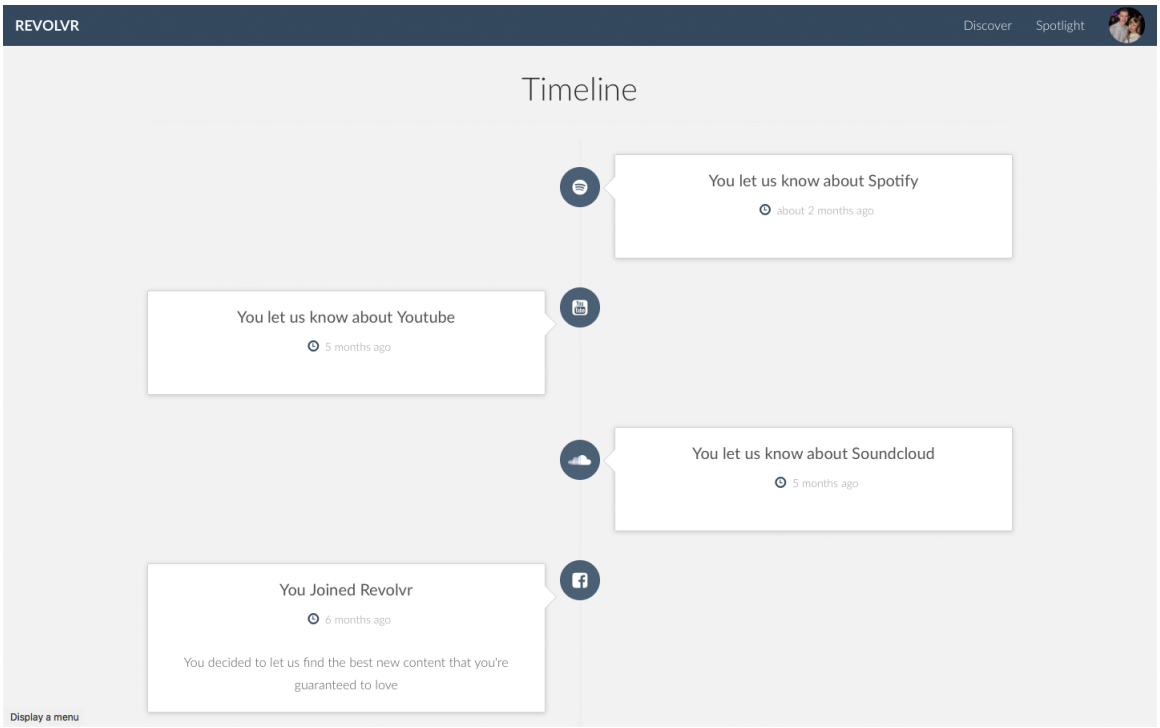


Figure 6.26: Timeline

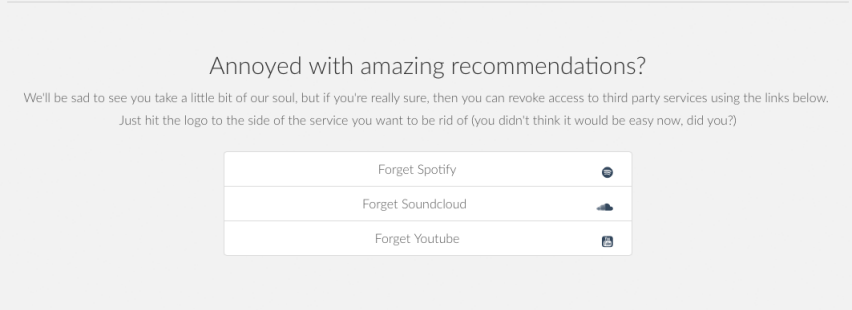


Figure 6.27: Service Disconnection Options

6.5.2 Sensitive Information

The previous developer designed the existing system in such a way that information has been protected from the start of this project. The use of Ruby on Rails parameter filters in the existing controllers limits the ability of attackers to inject [4]. Leading on from this the session handling ensures only authenticated users can access personalised pages, and in the case they view these pages the media will only belong to the appropriate user. Furthermore, all development and production environments were password protected including the production server, the production database,

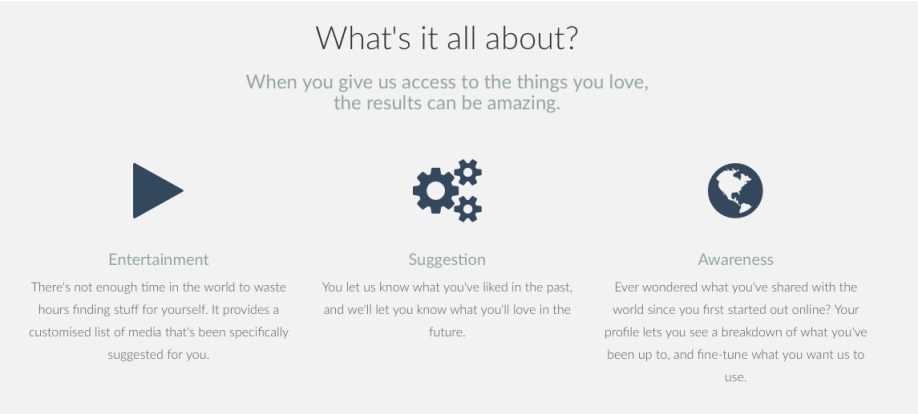


Figure 6.28: Contents of About Page

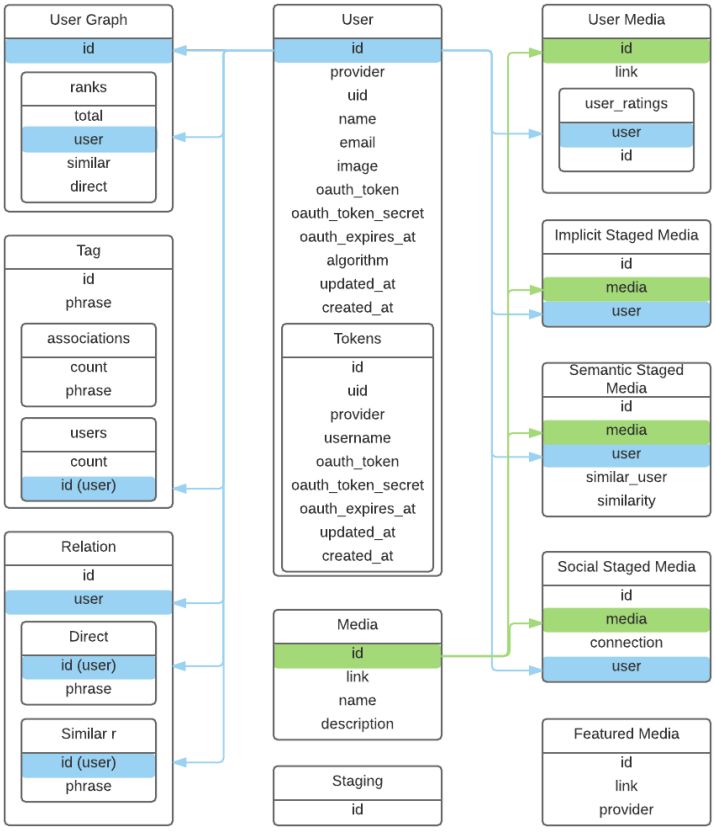


Figure 6.29: Document-Oriented Database Structure

and the local equivalents.

Following an involved and rewarding development process the developer believes the requirements for the project have been fulfilled, and via a combination of testing and personal use the system has proved to be an engaging and enjoyable experience. Testing was carried out continually during development, and following the implementation of major design components due to the adoption of an agile approach. The following section will discuss the various ways testing was conducted in relation to the proposed project goals, as well as an evaluation of the systems performance from both subjective and objective perspectives.

CHAPTER 7

Testing and Results

In a system such as Revolvr, where data processing, data storage, and the presentation of these processes in the form a full web stack are utilised, testing cannot be undertaken lightly. The modular design combined with the agile design methodology allows testing to take place at component level via unit testing, and during integration of these components via integration testing. Finally, when the system comes together as a whole functioning product system wide testing was carried out to ensure all respective components worked as a fully functioning system. Rigorous testing ensures that the systems functional and non functional requirements are met, which in turn can be directly compared to testing results to act as a measure of success for the project. Furthermore, assessing the general user experience and functionality separately from the requirements allows a general review to be taken to ensure Revolvr's user experience is satisfactory. Finally, user testing also occurred due to Revolvr's strong ethos and design focus on user experience. this includes numerical subjective assessment of quality of recommendations as well as objective user testimonials regarding the systems underlying potential.

7.1 Unit Testing

Unit testing involves systematically evaluating individual units of source code, a task that is promoted by Revolvr's modular design. By taking the smallest unit of testable software with an application and isolating it from the remaining components it allows the developer to observe if the unit of code is functioning exactly as expected without external influence. The unit testing can be broken down into three main subsets of system functionality, with each of these sets having their respective unit tests related to their respective processes. These subsets are directly related to the high level system design components outlined in Figure 5.2, namely data aggregation, data processing and storage, and finally the user interface. Both white and black box testing have been performed, to assess both the general results of processes as well as the internal operation and

general manipulation of data. Each set of tests aims to fulfil the functional requirements outlined in Section 4, and can be seen as a direct measure of success regarding implemented functionality. Tables 7.1, 7.2, 7.3, 7.4, and 7.5 in the following respective subsections show the key tests and the results for each of these components, and whilst not an exhaustive selection shows the approach taken and the key consideration during this process.

7.1.1 Data Aggregation Tests

For tests regarding aggregation of data from a service all services should obey the same general rules, therefore service specific tests have not been highlighted but the provided tests can be applied to any of the media providers. These tests were focused on ensuring information can enter the system and is stored appropriately for later processing and display stages, as well as a providing a heavy focus on user authentication. The results of these tests are shown in Tables 7.1 and 7.2.

7.1.2 Data Processing Tests

Following successful tests for aggregation and authentication, a user and media store is available for the data processing element to apply inference on to produce tailored recommendations. Due to Revolvr utilising three algorithms extensive testing was performed on this element, as the quality of recommendations is key for the success of the system in an academic and also commercial sense. Key tests include the decomposition of media titles to tags and the resulting construction of a relationship graph, matrix construction and factorisation, social network linking and finally generating predictions based on the resultant information from these algorithms. The storage of these recommendations was also throughly tested, to ensure that the recommendations are not only available, but also belong to the user in question. Tables 7.3 and 7.4 show the tests as well as the results of each test.

7.1.3 User Interface Tests

The UI cannot be overlooked during testing, as it is user facing and any errors in its design or functionality will not only be immediately noticeable but also effect a user's opinion on the site. Processed information must be easily viewable by the user via the *Discover* view, as well as popular media from *Spotlight*. User interaction is also critical and therefore tests were constructed to ensure users can comfortably interact with all areas of the system and that the general flow of the UI is simplistic without deducing from the viewing experience. The tests and their results are shown in 7.5.

7.2 Integration Testing

Following the successful testing of the functionality of individual units, integration testing was to follow to ensure these components could successfully operate together. Although unit testing provides confidence in the operations of individual components, often errors can occur when these components are pipelined to form a full process especially with regards to the formatting and

Table 7.1: Data Aggregation Unit Tests (1)

F#	Description	Input	Expected Output	Actual Output	Result
F4	A user can authenticate and register with Facebook OAuth 2	User requests to provide Revolvr with authentication credentials	Facebook user credentials returned to Revolvr	Response received correctly, user authenticated and appropriate information stored	PASS
F4	On reuse, Facebook authentication tokens are updated with a new access token	Aged authentication token on authentication request	New token valid for 60 days	Updated token returned and saved	PASS
F4	Third party services can be connected to via authentication token exchange using OAuth	User requests to authenticate Revolvr for the given service	A valid authentication token and user information	An authentication token with a valid expiry time and service specific user information	PASS
F4	Authentication tokens must be stored for future use	Received authentication token from service	Stored in the appropriate users token list	Expected token stored for user	PASS
F2	A valid authentication token for a service allows media to be aggregated	A valid authentication token for a service	List of relevant media	API response containing requested media	PASS

Table 7.2: Data Aggregation Unit Tests (2)

F#	Description	Input	Expected Output	Actual Output	Result
F2	Unique media from aggregation is stored	JSON formatted API response containing media items	Appropriate details stored in media table	Expected records added	PASS
F2	Appropriate media links from connected Twitter accounts must be retrieved, resolved and stored	Twitter authentication token	Resolved media items for retrieved links	Links resolved to API calls, records added to media table	PASS
F3	Each media item must be associated with a user	A list of identified media	Entries added to the UserMedia table associating a user with a an item	Expected records added	PASS
F3	Aggregation does not reprocess old media	A repeat collection process for a previously aggregated user	Only newest media is returned	Aggregation terminates early upon encountering previously seen media item, some media ignored	PASS
F5	The system must aggregate popular media from services	Revolvr credential token to service	A list of popular media from the service	API response containing a list of popular items in JSON format	PASS

Table 7.3: Data Processing Unit Tests (1)

F#	Description	Input	Expected Output	Actual Output	Result
F6	Titles must be reduced and decomposed into phrases	A media items title	A reduced title and respective tags	A number of tag entries each with associations	PASS
F6	Identical media produces a direct link	Two identically named items of media	A TRUE boolean	Media was identical	PASS
F6	Similarly titled media produces a similar link	Two songs by the same artist	A frequency pair of the artist name	Two instances of the artist name	PASS
F7/8	Users must be related via semantic links	A collection of direct and similar links	Relation collection entires for users related by these links	Relation entry for each user who is semantically related for another user	PASS
F7/8	A user-item matrix must be able to be formed based on the User and UserMedia collections	The User and UserMedia collections	A $n * m$ matrix of users and items with preference entries	An appropriately filled and indexed user-item matrix	PASS
F7/8	The matrix can be factorised via ALS	A $n*m$ user-item matrix	User-factor and item-factor matrices containing vectors for each item and user	User and Item factor matrices	PASS

Table 7.4: Data Processing Unit Tests (2)

F#	Description	Input	Expected Output	Actual Output	Result
F7/8	User and item vectors must generate predictions to be stored	User and item vectors	Matrix of predicted preference values to base predictions on	Matrix of prediction values i.e a vector of top N items per user	PASS
F9	Users must be relatable via social media	A users social media tokens	A list of related users based on Friends and Follows who use Revolvr	A list of related users	PASS
F7	These relations must be utilised to recommend media	List of related users for a given user	A list of all media which their connections are interested in	Collection as expected	PASS
F8	Recommended media should be staged for a user	The recommendation output of an algorithm	A stored collection of media for the chosen algorithm	Collection as expected	PASS
F8	New users should undergo staging	A new user to the system	User staged, aggregated and processed	New users processed reguarly	PASS
F8	Staged media must represent the appropriate algorithms recommendations	The recommendation output of an algorithm	A collection of all media in which every user may be interested in	Collection as expected	PASS

Table 7.5: User Interface Unit Tests

F#	Description	Input	Expected Output	Actual Output	Result
F10	Recommendations made to unauthenticated users	Unauthenticated user visits Discovery page	A list of randomly recommended media	A random list of recommended media	PASS
F10	Recommendations made to new users	New User visits Discovery page	A list of recommended media	A list of recommended media	PASS
F10	If personalised recommendations are not available, user must be informed and presented random selection	User visits Discovery page	A list of random media and an appropriate warning message	A list of recommended media and a notification	PASS
F10	User may rate (like) and save media concurrently	User 'hearts' media on Discovery or Spotlight page	Rating adds user to appropriate UserMedia collection entry	Actions as expected	PASS
F11	Likes must effect the recommendation process	User likes an item	Media added to UserMedia for use in algorithmic processing	Relevant media record added	PASS
F13	Liked media must be available for reconsumption	User rates media	Media viewable on profile page	Media available for viewing	PASS
F14	Presented media must provide appropriate detail	User visits receives recommendation	Title, description and the item itself available	Recomendations provide appropriate item detail	PASS
F15	A user can view their recommendations	Authenticated user visits Discovery page	Subset of recommendations displayed	Recommendations displayed	PASS
F15	A user can view the popular media	Authenticated user visits the Spotlight page	Popular media displayed	Spotlight media displayed	PASS
F15	A user can view the privacy policy	(Un)authenticated user visits Privacy Policy page	Privacy policy displayed	Privacy policy displayed	PASS
F15	A user can log in and out of the system	User attempts to log out	Session forgotten and user unauthenticated	As expected	PASS
F15	A user can select between recommendation algorithms	User clicks on profile page	User presented buttons to select algorithms	Buttons displayed	PASS

manipulation of data. Carrying out integration testing provides a level of confidence that the proposed flow of processes work together at a smaller scale before testing at a wide scale approach via system testing. Tables 7.6 and 7.7 show a subset of the tests performed, in this case aiming to outline a general flow through the system in a logical order. Again, these are not exhaustive tests and many cases can occur, but aim to provide confidence in the ability for units to work together. Due to the nature of the project similar tests to those in the existing documentation were performed with appropriate additions, therefore the original table has been modified to reflect the new version of the system.

7.3 System Testing

System testing is the process of assessing a complete, integrated system in compliance with its requirements. This portion of the process utilises a black box testing approach where the internal workings of the system are mostly ignored and instead considers real life use cases to identify problematic areas. System testing was not as calculated as the unit and integration testing phases, and instead the developer ensured the previous two test cases were conducted throughly and then simply used the system as a user would. The developer connected all services and on a daily basis ensured a variety of media was entered into the system under their account. Friends of the developer were also offered to sign up to the system and connect their services and although they were not asked to use it their media accounts allowed data to be aggregated and provide relations for the developers experience. This approach gave insight into how a user would use the system without the technical details and provided an alternative perspective which allowed the developer to not necessarily get too tied up with the internal processes but instead the user experience. The full process of aggregation, processing and recommendations was also designed so that each respective process outputs information about its results, such as the number of items aggregated, tags taken from media titles, items recommended from matrix factorisation etc. These logs were produced via the aforementioned CRON jobs and emailed to an email account purely used for logging. Daily inspection of these logs combined with the realistic use of the system provided both a subjective and objective insight into how the system was operating and flagged any issues that should be deal with.

7.4 User Acceptance Testing

Revolvr's user experience is an indispensable factor to the success of the system, with media consumption being a personal experience which can differ for every user. Furthermore, alternative perspectives from users who may not be as technically experienced or as familiar with the system as the developer are of great value, as developers can often overlook or be biased to errors in their product. As mentioned previously a number of users were asked to sign up earlier on in the development process to allow media to enter the system over the course of the development, aiming to provide an idea of performance with a reasonable number of media items. Users were asked to provide informal feedback on elements such as the user interface and general *feel* of the site, as well as report any errors they encountered to the developer. In terms of assessing the quality

Table 7.6: Component Integration Testing (1) [4]

Component 1	Component 2	Input	Expected Output	Actual Output	Result
User Interface	User Authentication	User requests authentication	Tokens stored and authenticated user session started	User successfully stored and updated	PASS
User Authentication	Content Aggregator	Tokens sent to media services	Personalised media information returned	Content aggregated per user	PASS
Content Aggregator	Media Storage	Aggregated media	Aggregated media filtered, processed and stored	Unique aggregated media stored	PASS
Media Storage	Similar Link Identification	Stored media	List of similar media	All applicable media related to other media	PASS
Similar Link Identification	Relation Identification	List of similar media	List of similar users	All users with relationships placed in storage	PASS
Semantic Relation Identification	Semantic Staged Media	List of similar users	List of suggested media per user	All new media suggested for a given user stored	PASS
Media Storage	Matrix Factorisation and Prediction	Stored Media	n*m matrix where row n contains a users predicted preferences for all m items in the system	n prediction arrays of length m	PASS

Table 7.7: Component Integration Testing (2) [4]

Component 1	Component 2	Input	Expected Output	Actual Output	Result
Matrix Factorisation and Prediction	Implicit Staged Media	List of top N predictions per user	Recommendations entered in staged media table	Recommendations successfully stored	PASS
User storage	Social Recommendations	List of users	List of socially related users for a given user	A list of connections for a user	PASS
Social Recommendations	Social Staged Media	List of connections for a user	List of unseen media for a user belonging to their connections	Store list of appropriate recommendations	PASS
User Profile Page	Discovery View	User selects recommendation algorithm	Discovery view displays media from appropriate staging table	Discovery view shows appropriate recommendations	PASS
User Profile Page	User Media Table	User requests to view their previously liked media	User retrieves a list of previously preferred media to consume	List of audio and video items shown with option to view media	PASS
User Media Storage	Profile View	List of user related media	Display of all media the authenticated user has used	The user can identify how many items of media they are associated with	PASS

of recommendations a subset of users agreed to being part of an objective study in which they rate recommendations to provide an idea of the quality of each algorithm. Section 7.4.1 provides some examples of how the informal, personal feedback was useful in the design process and refinement period, and in contrast Section 7.5.3 provides a numerical assessment of the quality of recommendations.

7.4.1 User Feedback

User feedback was provided in three important stages: the incarnation of the system, at a smaller scale during development, and upon completion to assess the system as whole. Comments were generally favourable as development progressed but in the earlier stages assessing what the existing system did wrong was invaluable.

- **Beginning of development:** Before any work was started the default state of the existing system was shown to close friends of the developer, most of which had development experience themselves. The majority of comments were related to the UI, which acted as a basis for the work carried out. Firstly, numerous participants said the colour scheme was too bright and some of the colours clashed, although this is a minor detail this was taken onboard and a darker more professional three palette scheme was chosen. Secondly, users stated the lack of AJAX especially when viewing recommendations hindered the experience and made it feel stagnated resulting in the motivation for dynamic content loading. The site was also described as *'confusing'* and *'unclear'* especially with regards to the *Evaluation* and *Training* pages, which users stated they felt were intrusive and therefore this process was streamlined into the heart button in the final product.
- **During development:** On a monthly basis earlier in development users with technical experience were asked about new features in the system, and generally this feedback was positive. However, an early implementation of a revolving cover flow style player for recommendations received overwhelming criticisms, and as a result the developer decided to implement the flat sliding player as seen in the final product. Again, this highlights the important of external opinions as otherwise this feature was likely to worked on further with little gain in terms of the user experience. A number of varying mobile devices belonging to users were also utilised to ensure the responsive design worked as intended across multiple platforms due to only Apple devices being owned by the developer.
- **Final Stages:** Once the system were near complete none technical users were also asked to explore the system further. This feedback provided an alternative view from less of a design perspective and instead a general outlook. The user feedback was essential during this stage as it became clear many features of the website were unclear and confused the user. One example of this is the algorithm selection, originally there was no description of each algorithm which users reported as intimidating and reduced the chance of them clicking the buttons as they were unaware of there functions. Other changes such as the titles of pages and positioning of links were also modified to ensure the website is as clear as possible to all potential users.

7.4.2 Testimonials

As well as tests and user studies it is important to gauge how Revolvr feels as product to users and professionals at a personal level. To vouch for the quality of the system user testimonials have been collected from both a MEng student and an industry professional to provide an informal view of Revolvr's potential.

> "Here is a chance for users to finally really influence what their media service is recommending them. Revolvr puts the recommendation methodology in your control. However manages to do so in a slick and easy to consume manner, I'm not forced into answering a dozen questions, or worried about the impact of a recent binge of 80s TV soundtracks. I'm sure it won't be long before an algorithm that identifies time signatures and breaks is added and Revolvr becomes the go-to discovery service."
>
> - H. Varatharasan, *Project Manager, Barclays Investment Bank*

> "Via a easy to use UI and smooth navigation system I enjoyed being able to modify how my suggestions were chosen. Although one might be *'better'*, alternating methods gave me the opportunity to discover more music, some of which I would not have discovered otherwise"
>
> - I. Gambe, *3rd Year MEng Computer Science Student, University of Warwick*

Varatharasan's quote highlights Revolvr's focus on free reign by allowing the user to switch between recommendation algorithms. He also brings attention to the fact that a user can relax and receive recommendations easily as opposed to having to provide mundane and often irrelevant information, praising Revolvr's passive approach to recommendation. His confidence in Revolvr is promising both as a general user and an individual with experience in many computing related fields, outlining the bright future Revolvr could have if it continues to evolve and grow.

Gambe's feedback focuses again on the multiple inference methods, praising the variety of recommendations a user is able to receive. Although he notes that some algorithms arguably perform 'better' than others, he instead sees this as a strength of the system. The ability to receive *curveball* recommendations exposes the user to media the perhaps would not even think of consuming, but provides a multitude of contrasting media for discovery. These testimonials outline Revolvr's strengths as an existing product, and more importantly the potential for future growth and potential market penetration due to the unique approach of aggregation and customisable suggestion.

7.5 Tuning and Assessment

This section of the report aims to provide statistical evidence to provide insight into the performance of the system. The main focus will firstly be describing the tuning the Implicit Matrix Factorisation algorithm for optimal performance on the asymmetric dataset, with details on both the procedure

and the findings. Following this, a small scale study with users will be discussed offering details into the objective performance of each algorithm supported by subjective data.

7.5.1 Parameter Tuning and Performance

As discussed in Section 5.3.2 and the original source material, the matrix factorisation algorithm utilised contains a number of tuneable parameters that vary based on the dataset. Furthermore, evaluation of such models is vital in the area of machine learning where problems such as overfitting can occur if a model is not trained, validated and tested on appropriate datasets with a suitable performance metric.

In Hu's paper on IMF, they collected viewing data from 300,000 set top boxes (users) related to 17,000 unique television programs. Over four weeks viewing data was collected for these 300,000 users and acted as the training data, that is the data used to decompose the matrix into the appropriate user and item vector matrices from which predictions cna be made. The values of this matrix r_{ui} corresponded to how many times a user watched a program based on the number of minutes. The resultant training set had around 32 million non-zero r_{ui} values [16]. The test set was constructed from a weeks worth of viewing following the 4 week training period the system was trained on, and the goal was to predict viewing habits for this week period. The list of predicted programs viewed was then compared to the actual test data and percentile ranking was used to compare the predicted desirabilities to the actual results. A clear issue was noticed early on here: Hu's data featured a **closed ecosystem**. By a closed ecosystem we are referring to the fact that the number of users of items does not change and therefore the resultant user-item vectors of a trained model can be used for new data. Revolvr's aggregation results in new items being added to the system regularly which results in the dimensions of these vectors and the original user-item matrix not matching,. This means a new user u cannot receive predictions without repeating factorisation, and in contrast a new item i cannot have its preference rating predicted for users until the same repeating factorisation occurs.

The lack of explicit negative feedback makes precision based metrics unreliable, as it is uncertain what a user dislikes and such metrics require knowledge of this [51]. The methodology in the paper could also not be used due to the ever expanding datastore of Revolvr, posing an issue with regards to the evaluation (and in turn tuning) of the model. Remaining work on the project led the the development of a custom procedure that attempts to measure the performance of the system.

7.5.1.1 Methodology

The developed evaluation exploits the idea of leave-one-out cross validation, which is where one observation of a dataset is used as a validation set and the rest as a training set. The algorithm works by iterating through every non-zero element i in the user-item matrix, and checking if another non-zero element exists in that column. If so, it sets i to zero, decomposes the matrix into the appropriate vectors, and generates the prediction array for the current user from the resulting prediction matrix. At this point there is knowledge the user does like this item however the system has no knowledge as the non-zero element was converted to a zero, and the prediction score for the item will reflect. The number of unique scores is then taken, where a one is the highest possible

score and zero is the lowest, and the predicted score for item i is compared to all possible scores to see where it ranks. For example, if the prediction for the zeroed item was 0.5 and the scores present in the users prediction array were [0.9,0.8,0.5,0.2,0.01], then the corresponding rank would be 3, as 0.5 is the third highest score. Items the user already likes are set to an extreme value of -1 as these should not be considered due to always having the highest score. Ideally, the item should have the highest score possible and hence a rank of 1, and in general the higher the rank the better the predictor operated. This process is computed on a row by row basis, and can therefore be seen as a per user basis due to each row in the matrix representing a user. For a user n, the rank of any item m which n likes and at least one other users likes is zeroed, predicted, and the rank of the score is calculated. An average of all the ranks for a user is then calculated, as well as the average number of possible ranks. This process occurs per user, and therefore results in a list of average ranks and average number of possible ranks per user meaning if there are n users, then you have n average ranks each corresponding to a user and also n average total number of ranks. The average, mean, and range of the ranks are then taken as well as the mean of the total ranks. This whole process is repeated for a number of alpha and lambda values, making it a computationally expensive process due to performing many matrix factorisations and is shown in Figures 7.2 and 7.3

This process is based on the premise that, if a user likes an item then a good prediction mechanism will be able to determine this when that information is hidden. By using this process for various parameter setups we can attempt to model the performance of the system by comparing the returned information for each combination of parameters. The algorithm only considers items with multiple rankings as two or more users liking a common item provides a relationship to try to infer on.

Algorithm 10 Cross Validation Algorithm

Require: $matrix \leftarrow user\ item\ matrix$
1: $alpha \leftarrow [1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$
2: $results \leftarrow []$
3: **for** a in alpha **do**
4: $lambda \leftarrow [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 5.0, 10.0]$
5: **for** l in lambda **do**
6: $result \leftarrow get\_rank(l, matrix)$
7: **end for**
8: **end for**

7.5.2 Cross Validation Results

The process described in section 7.5.1.1 was run for the parameter combinations described in Algorithm 10 to receive the mean, standard deviation, and range of the average rank for each user aswell as the mean number of possible ranks. The process was run on the same user-item matrix that is initialised and stored at the start of the process, meaning subsequent iterations were all running on the same snapshot of data. During this process there were around 18 users and 1500 items in the system, with only around a third of the users having a reasonable amount of items. Due

Algorithm 11 get_rank(l,matrix)

```
ranks ← []
total_ranks ← []
for each user, m do
    user_ranks ← []
    user_total_ranks ← []
    for each item, n do
        if counts[m][n] > 0 then
            if there exists an x, in column n with counts[x][n] > 0 then
                counts[m][n] = 0
                preds ← prediction matrix resulting from IMF
                user_preds ← preds[m]
                rankings ← unique scores in user_preds
                rank ← position of item n in rankings
                total ← number of unique ranks (len(rankings))
                append rank to user_ranks
                append total to user_total_ranks
            end if
        end if
    end for
    append mean(user_ranks) to ranks
    append mean(user_total_ranks) to total_ranks
end for
Return mean(ranks), mean(total_ranks), std_dev(ranks), range(ranks)
```

to this, the results although useful must be used with caution as the dataset is not representative of a full scale recommender system. Furthermore, the users invited to use the system are friends of the developer, and therefore genre bias may also be present. The cross validation process took approximately 3 weeks, due to the vast number of matrix factorisations it had to perform with each consisting of 30 iterations with a total running time of between 3 to 4 minutes. Figure 7.1 shows the mean ranks for each parameter combination, accompanied by Figures 7.2, 7.3 and 7.4 showing subgraphs for clarity. Accompanying this is 7.5 showing the standard deviation of ranks, again split into Figures 7.6, 7.7 and 7.8 for clarity. The mean number of possible ranks was 29.8 ± 0.1 and due to the lack of variation these values were not plotted on a graph. As well as providing information on optimal parameters, it was also observed that the models performance varies greatly when lambda is between 0.7 to 1 for all models. In the production system this sensitivity would have to be considered, with regular parameter tuning being performed given the appropriate hardware to ensure the model is performing as well as it can.

Based on these results the values of alpha and lambda chosen for the production process were 20.0 and 1.0 respectively, offering a mean rank of 6.54946886447 out of an average of 29.6019230769 possible ranks, meaning each zeroed item on average appears in the top 25% of scores. This setup carry a standard deviation of 4.36674203433 for the ranks, which although was not the lowest discovered offered a suitable tradeoff when considering the mean rank. Further parameter sweeps in the optimal range were likely to provide even better scores, but due to the lengthy processing time and lack of performant hardware options the decision was made to settle with these values and continue work on other elements of the project. Although not an established methodology this custom cross validation process does provide insight into how the model performs in suitable detail for a project of this scale, as well as for a data format (asymmetric data) not widely explored.

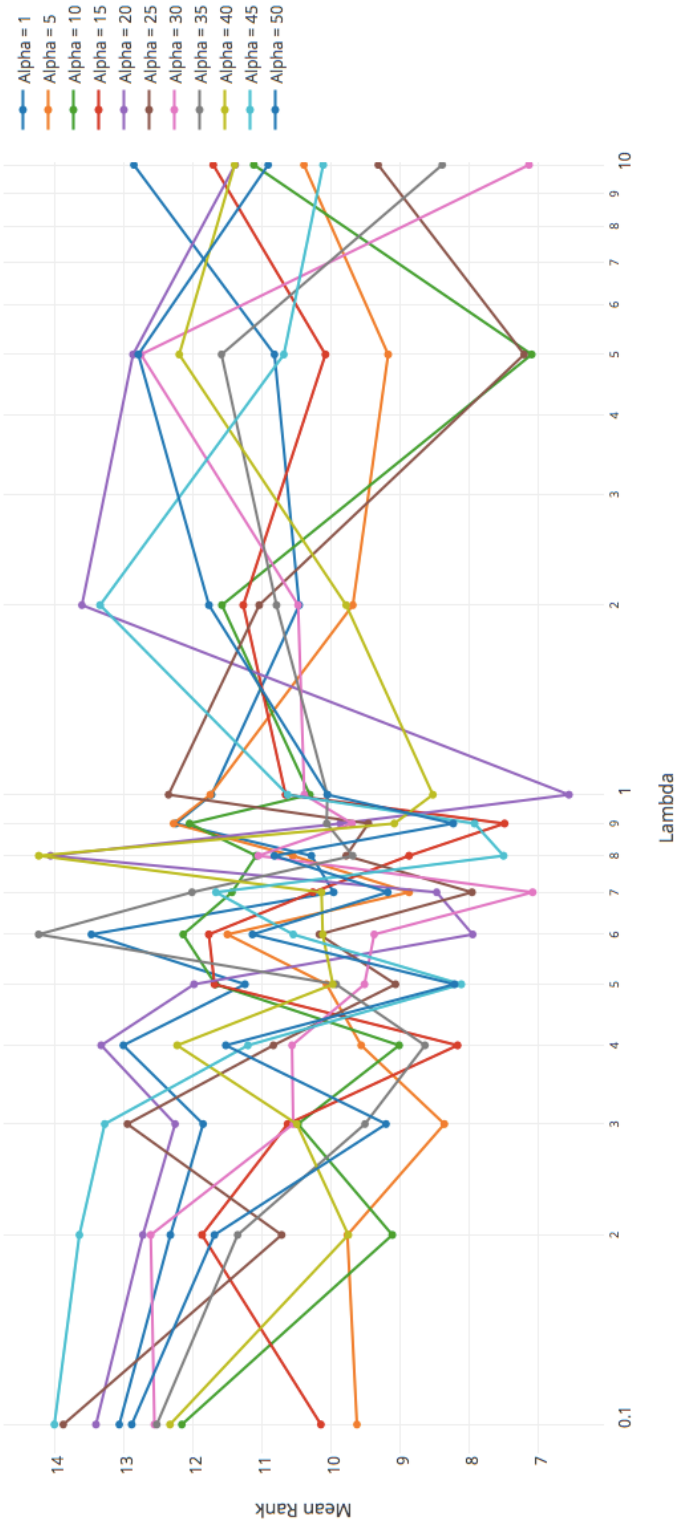


Figure 7.1: Cross Validation Results - Mean Rank

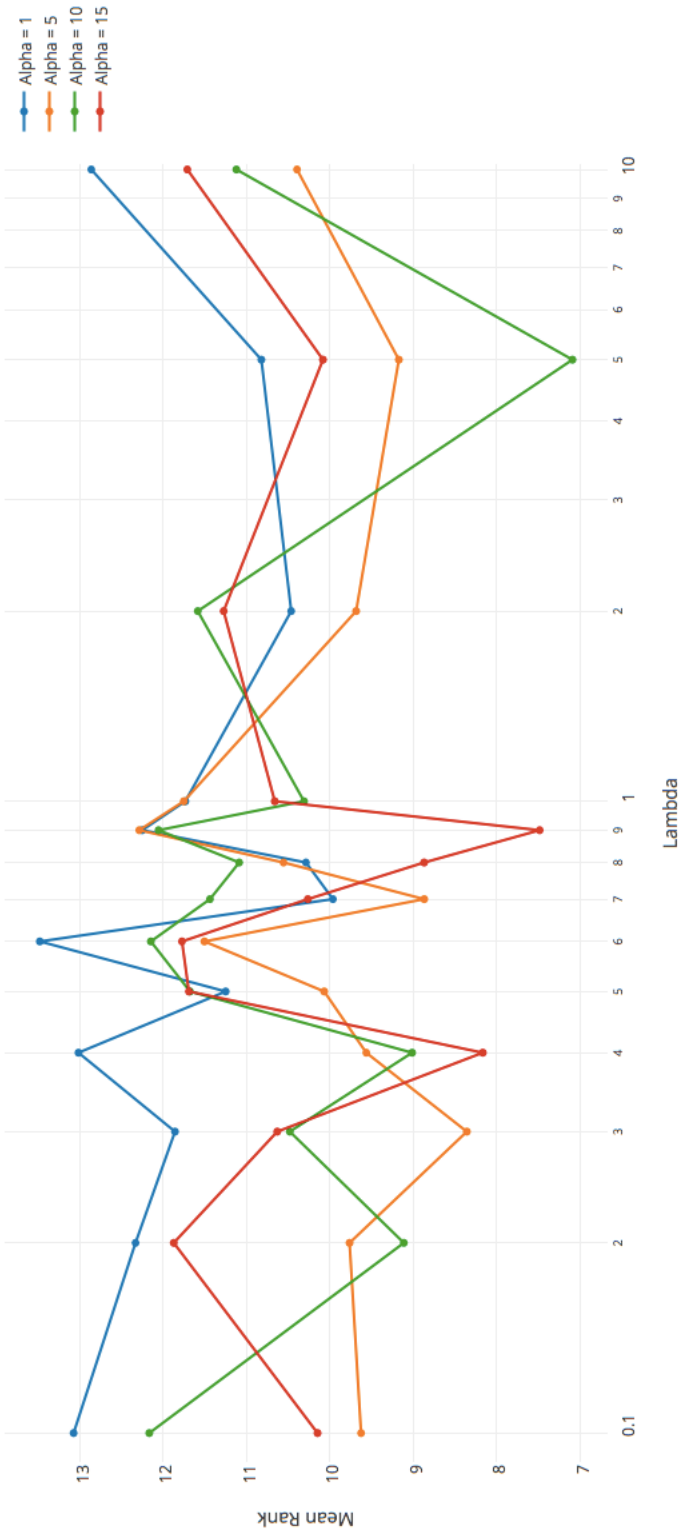


Figure 7.2: Cross Validation Results - Mean Rank (Alpha 1-15)

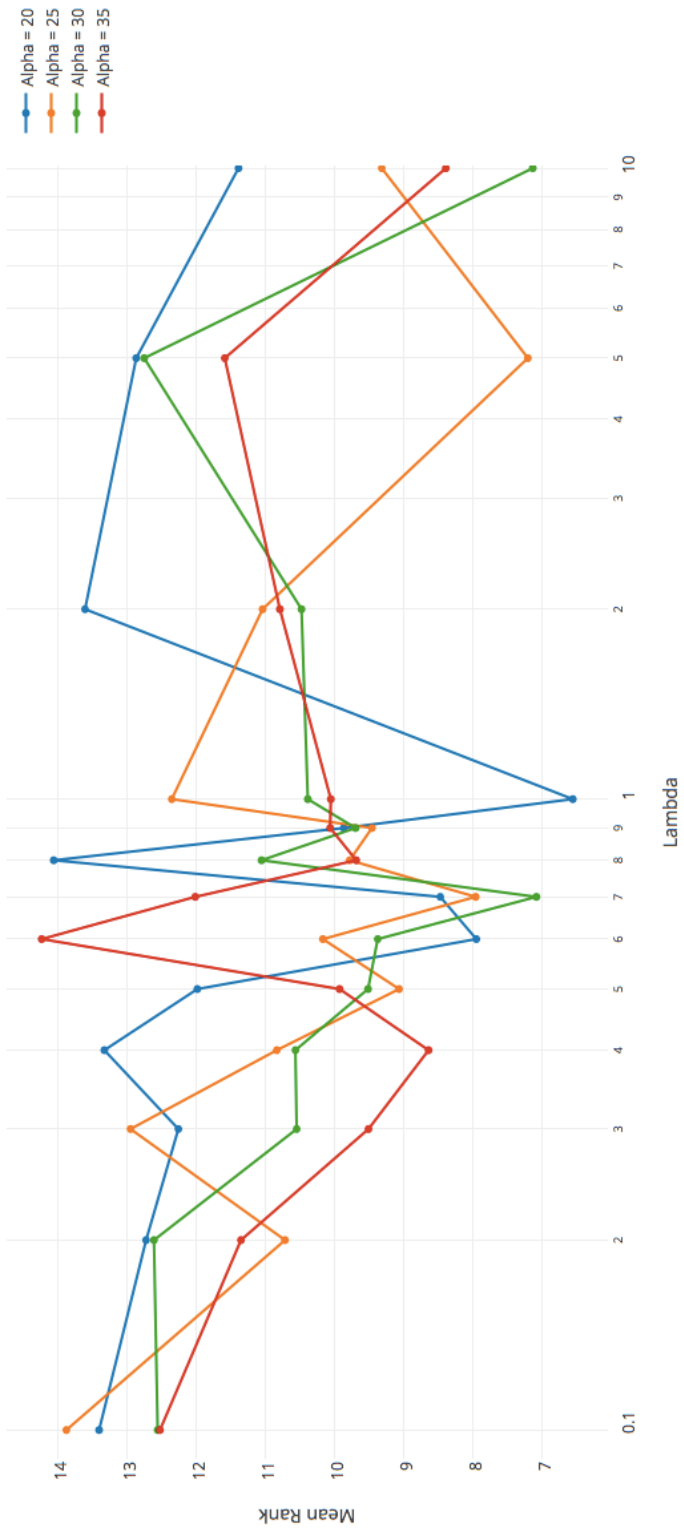


Figure 7.3: Cross Validation Results - Mean Rank (Alpha 20-35)

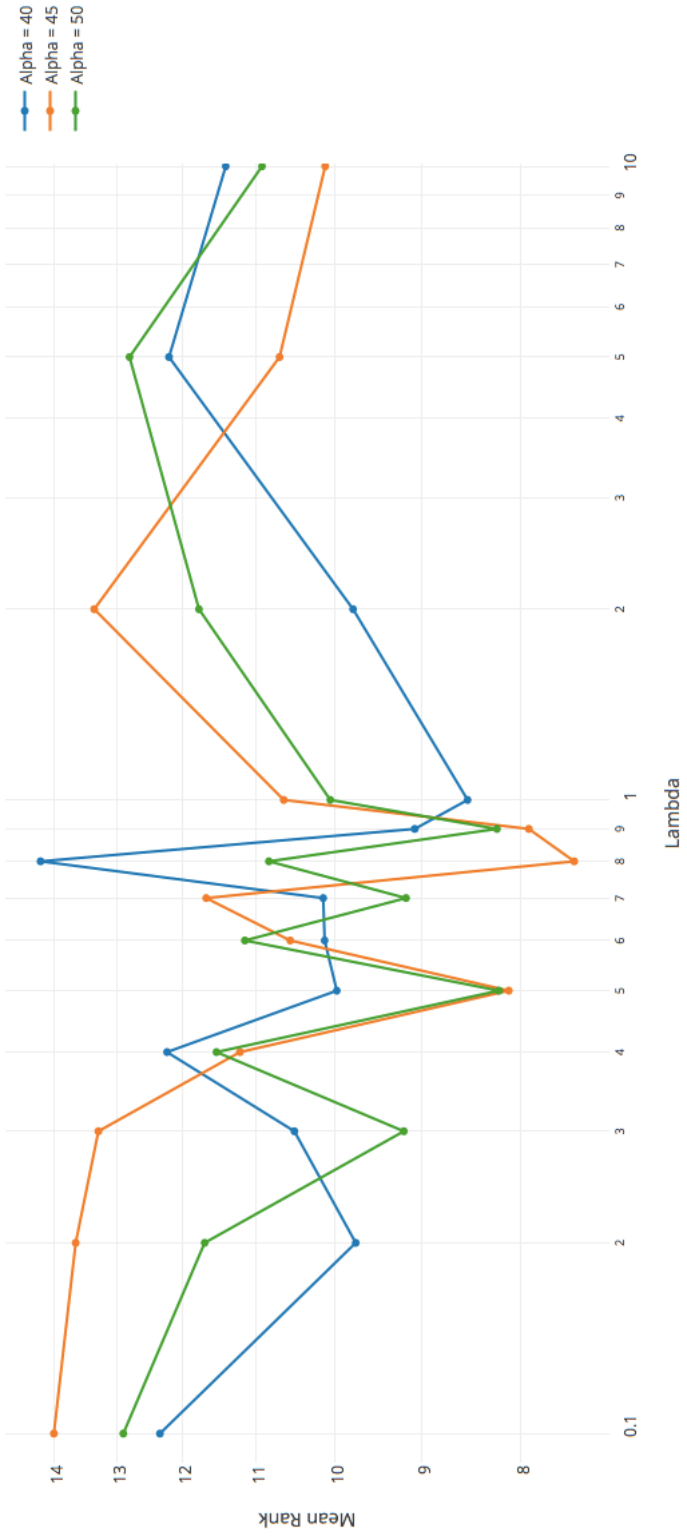


Figure 7.4: Cross Validation Results - Mean Rank (Alpha 35-50)

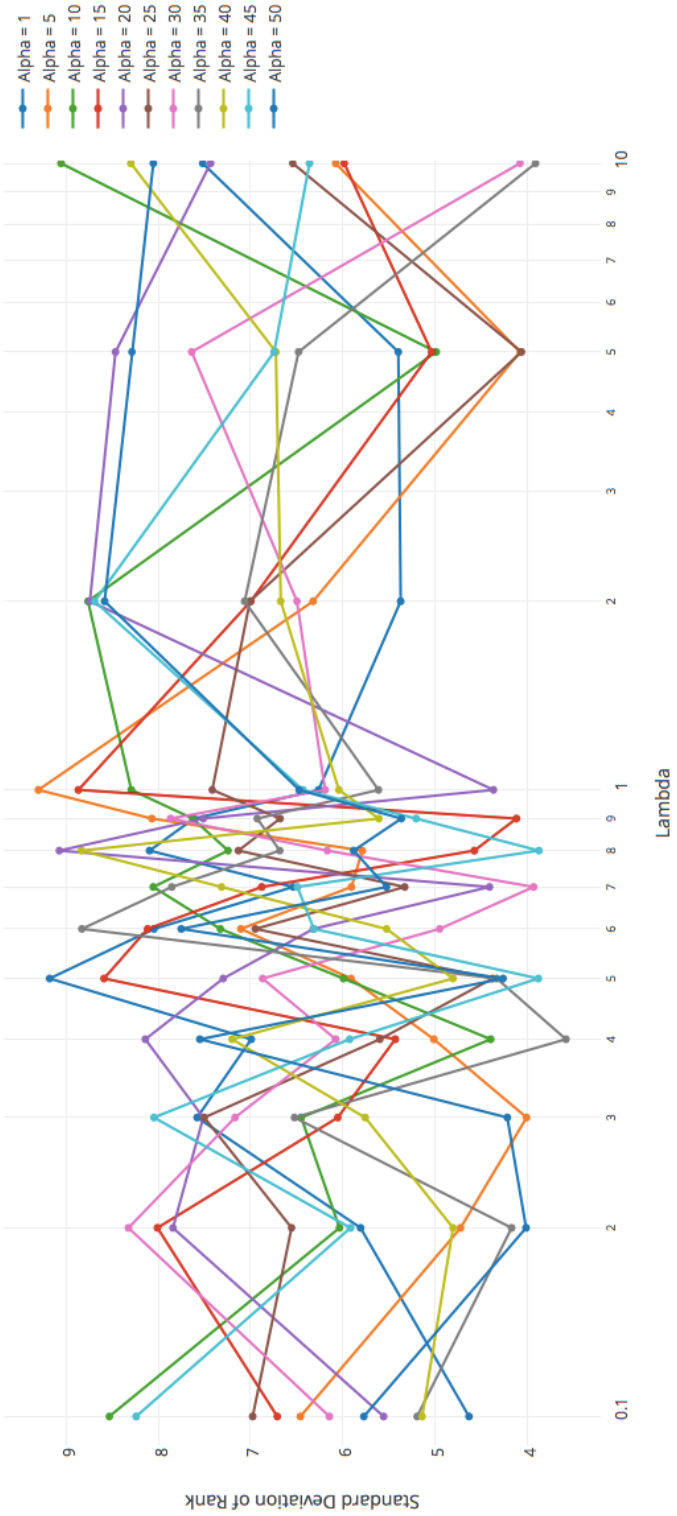


Figure 7.5: Cross Validation Results - Rank Standard Deviation

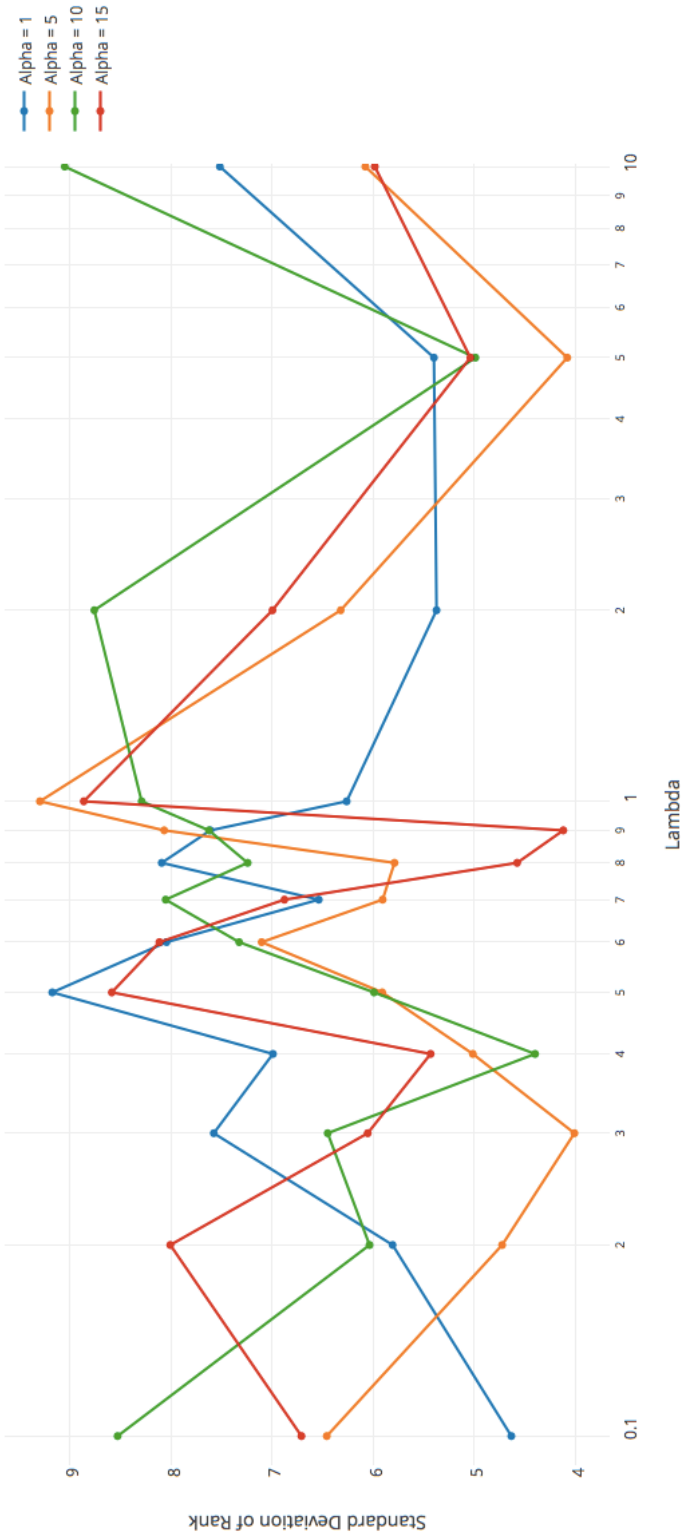


Figure 7.6: Cross Validation Results - Rank Standard Deviation (Alpha 1-15)

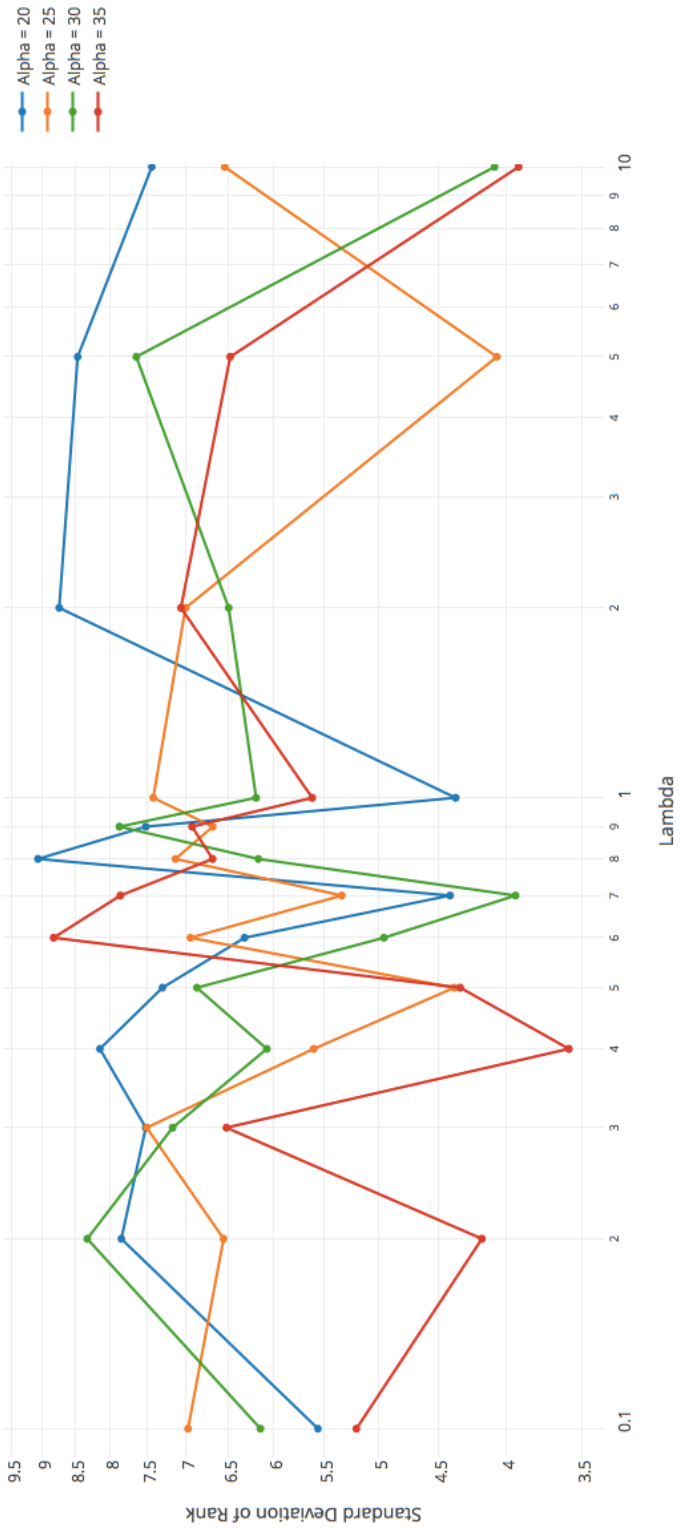


Figure 7.7: Cross Validation Results - Rank Standard Deviation (Alpha 20-35)

7.5.3 Recommendation Assessment

With a recommender system user feedback regarding the quality of recommendations is vital. To evaluate the three algorithms in the system an online survey was created in which users were instructed to select each algorithm in turn, generate 20 predictions in total ignoring duplicates within each algorithm, and rate the recommendation on a scale of 1-5. A rating of 1 implies the user strongly disliked the item, and a 5 in contrast represents a strong like. The survey was given to 7 users out of the 19 who were registered to the system. The small sample size is due to the time consuming and personal nature of data collection, with the remaining users not being close friends of the developer and therefore having no direct method of contact. To improve the validity of results users had no knowledge of which algorithm was which due to them being renamed algorithm 1 to 3 as opposed to containing the name. This aimed to also reduce bias in favour of the developer as some of the participants had knowledge of which algorithms the developer was responsible for. The results of this analysis are shown in Table 7.8. The results, despite the small sample size, suggest that the new algorithms in Revolvr perform better than their predecessor which was left unchanged for comparison purposes. Values in red are users that were not able to generate 20 unique items due to the way the semantic algorithm selects media as discussed in Section 6.3.3, which highlights further problems with this approach. This was also mentioned by a few users who said the recommendations from the existing semantic algorithm felt limited, with many praising the feature based recommendations. The social algorithm largely depends on whether a user has social media connections within the system and again the small user base may have been detrimental in this case.

Overall via a combination of successful testing, promising statistics regarding the performance of the most complex and abstract algorithm in the system, and generally favourable use feedback it becomes apparent Revolvr is a system with potential. Not only is it a project that has met its design goals, but a system that a user enjoys to use and is opening the doors to discovery.

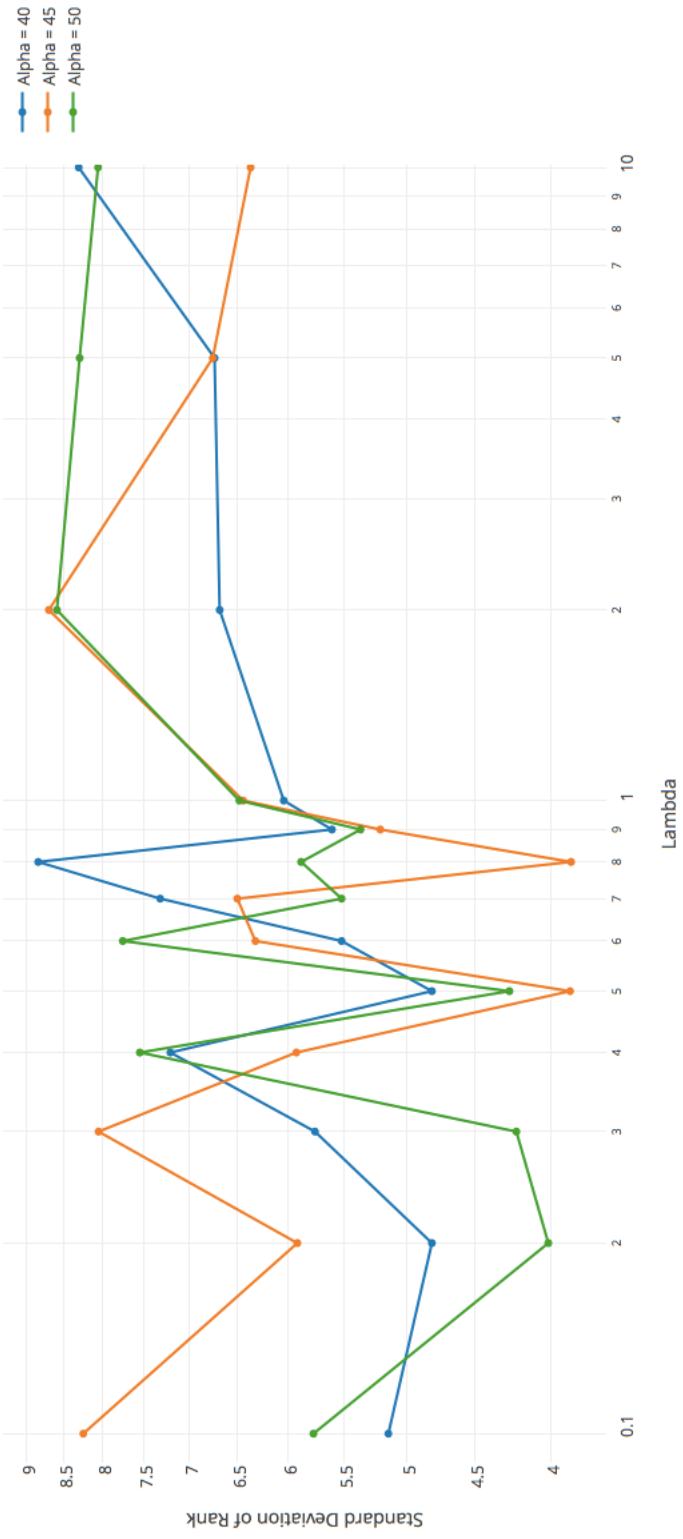


Figure 7.8: Cross Validation Results - Rank Standard Deviation (Alpha 35-50)

Table 7.8: User Evaluation

User	**Semantic Average**	**Feature Based Average**	**Social Average**	**Number of Items in System**	**Connected Services**	**Comments**
1	3.05	3.50	2.75	122	Spotify Soundcloud Youtube	Lots of repeats in semantic and social.
2	3.00	4.05	3.15	345	All	N/A
3	3.45	4.10	3.30	234	Spotify Soundcloud YouTube	N/A
4	2.80	3.70	3.50	37	Spotify	Good Recommendations
5	2.25	3.55	2.90	291	Spotify Soundcloud YouTube Twitter	N/A
6	2.40	3.04	3.00	77	Spotify Soundcloud	Lots of duplicates in the semantic algorithm. Feature based recommended items in my Soundcloud library without me having connected it
7	2.80	3.90	3.20	212	Spotify YouTube	N/A
Average(2DP)	2.82	3.69	3.11	N/A	N/A	N/A

CHAPTER 8

Project Management

With a project of this scale techniques to manage and achieve the proposed work are vital. To ensure development goals were met in a timely fashion in line with the those proposed in the projects progress report, planning and monitoring of the project was paramount to its success. The following subsections will discuss the design approach taken both at a development and management level, as well as outline the tools used to ensure the development and documentation of the project were appropriate.

8.1 Design Approach

By being provided with existing work to use as a basis, an agile methodology was adopted in regards to development. This was suitable as initial overheads of database design and a basic UI were avoided, allowing focus on polish and academic research into recommender systems. Such a design approach also allows flexibility in regards to proposed development plans, which changed greatly compared to those given in the initial project specification. However, despite a number of changes in the proposed system, the initial vision outlined in both the specification and the progress report has been realised. Unlike a lot of projects there was no need for a bottom up approach due to the aforementioned existing framework, and instead development was split into five separated stages: initial setup, research, enhanced aggregation, algorithmic development and finally UI improvements. Initial setup involves deploying the existing system and ensuring the developer understood the system inside out which is a timely process. Once comfortable with the existing environment research was then made into its functionality, attempting to dissect its flaws and weaknesses. During this period a number of recommendation algorithms and approaches were considered before arriving at the modified variant of implicit matrix factorisation to the data format we defined as *asymmetric feedback*. In terms of development of enhanced aggregation, recommendation algorithms and user interface design these were carried out sequentially and again

utilised the flexibility of an agile approach, with a relaxed development plan solidified by preliminary understanding and research. This approach allowed the ability of timely responses of suggestions via discussion with test users and the project supervisor, without having to be limited by existing plans and documentation.

8.2 Software Development Methodology

The project utilised a highly agile workflow in terms of development and minimal documentation was produced outside of the required hand ins. However, both the developer and the project supervisor felt it was important to maintain weekly targets and goals to ensure that work was not only moving forward, but it was relevant to the end goal of the project. By setting such goals it allows a constant drive and focus to be placed on development and making a large, loosely planned project manageable by partitioning the work into smaller chunks via micromanagement and continual feedback. The combination of this approach combined with the free reign of an agile workflow was suited to the developer, who often had implementation ideas that using a more stringent approach such as a waterfall model would have been difficult to include in the project outside of the initial specification. Examples include the *Spotlight* feature and also the streamlined training the recommender system, concepts which are now fundamental to the operation of Revolvr in its current state.

8.3 Project Timeline

Since the proposed project timeline given in the progress report, expected and actual progress have been largely in line with one another. Figure 8.1 shows the previous iteration of the gannt chart, and Figure 8.2 gives the final version. These figures show that the timeframe of the project remained largely the same, however the planned work on incorporating higher level social media constructs into the project was instead replaced with extra time into researching and development recommendation algorithms. The matrix factorisation algorithm required a lot of initial understanding, as well as looking into whether the dataset Revolvr collects would be suitable via both independent research and community advice via machine learning forums. The developer felt it was better to prioritise not only such a core element to the system, but one with little existing research allowing the exploration of a new and interesting problem domain. The new gannt chart also accounts for parameter tuning, which was left to run on the production server whilst other tasks were completed. Finally, the inclusion of a user testing and refinement period towards the end of the project allowed for a period of reflection, where minor tweaks and fixes could be made to the system without having to focus on the implementation of major features.

8.4 Management Tools and Techniques

The existing implementation meant that the developer did not have free reign regarding the technologies used without significant redevelopments with little return in terms of the project. Despite

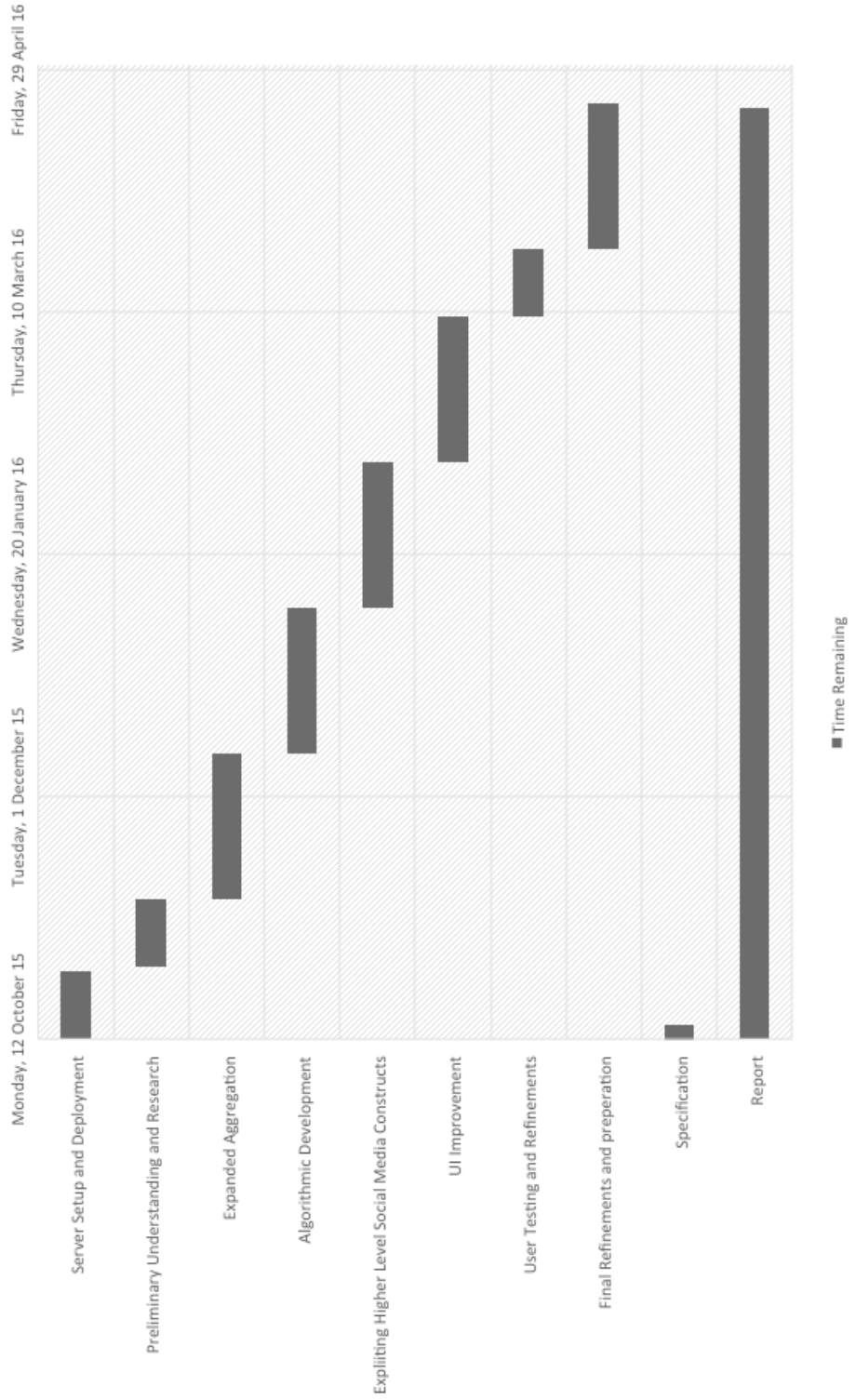


Figure 8.1: Gannt Chart From Progress Report

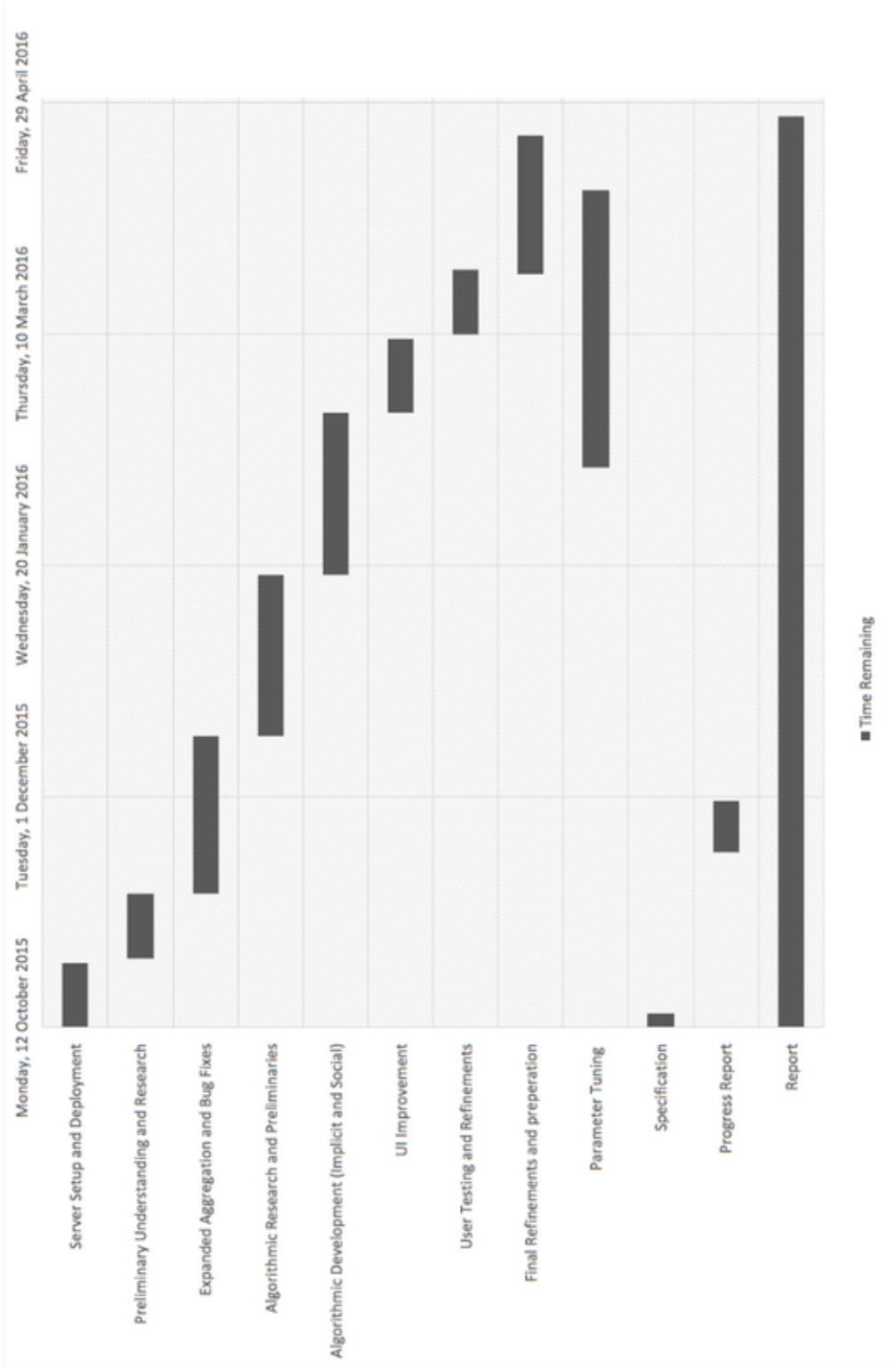


Figure 8.2: New Gannt Chart Showing Project Progression

this, Revolvr utilises standard frameworks and popular tools meaning that documentation and support for these technologies are readily available. Although the developer was not familiar with Ruby on Rails and MongoDB, two of the main technologies used in the system, vast documentation and learning resources allowed the unfamiliarity to open up doors to hands on learning process. The tools used throughout the project can be split into two discrete categories: development tools and management tools. Descriptions of the tools themselves and the motivation behind using them can be found in the following subsections, with a focus on the role each component played in the project as a whole.

8.4.0.1 Development Tools

Figure 8.3: Development Technologies Utilised

As outlined in Section 6.1, the key tools for the development of Revolvr were those utilised for data processing, the DBMS and finally the UI with the reasoning for doing so provided in the relevant section. A number of small complementing tools were used to assist development alongside these core technologies to provide a well rounded system and development process. Firstly, the distinction between a local development environment using a Rails Thin web server (VM) and an externally hosted Ubuntu server via DigitalOcean allowed two main development streams. For experimental work the local instance was used, allowing full control and experimentation of the system in a closed environment. Following the testing, validation and appropriate stability of new components these could then be pushed to the production environment with confidence that the components would simply plug and play due to internal review [52] [53]. Ruby on Rails itself incorporates a number of technologies within its framework including HTML5, JavaScipt, CSS, CoffeeScript, SASS and ERb technologies. The first three are standard web development technologies used for the mark up, styling, and automation of web pages. CoffeeScript allows JavaScript to be written in a concise and elegant fashion, ERb (Embedded Ruby) allows generation of HTML from templated, and SASS allows the use of variables and objects within CSS [54] [55]. The production server hosts an Apache2 web server to process requests, with Phusion Passenger being used to host a Rails app on the web server [56] [57]. External JavaScript libraries such as *bxSlider* were also used to provide instantaneous sought after functionality as opposed to spending time developing features with little contribution to the projects main goal of recommendation.

8.4.0.2 Management Tools

As well as a number of core development technologies, the organisational aspect of the project also utilised a number of tools. Firstly, Dropbox was used to synchronise documentation files between

Figure 8.4: Management Technologies Utilised

the developer and the project supervisor. GitHub was the chosen version control system, allowing the synchronisation and deployment of code among both the production and development server as well as acting as a code based log of changes [58]. Sublime text was the chosen code editor as opposed to a conventional IDE, as using one program for every language seemed more logical than using a Ruby IDE and a Python IDE separately [59]. Finally, to organise and log development Wunderlist and a Wordpress blog were used. Wunderlist is a simple mobile and desktop application that acts as a to-do list, which was used to monitor both pending and complete tasks. An informal Wordpress blog was also maintained during the earlier stages of the project to ensure the fast pace of development did not hinder the ability to recall early work in the final report [60] [61] [62].

8.4.1 Risk Management

With a project of this length, size, and the agile approach it promotes there are a number of risks that could have possibly hindered the success of the project. From the inception of the project to the finishing stages a number of risks have arose that need to be planned for appropriately.

- **Hardware Failure:** Software products often rely on many interacting components hosted across many different machines, exploiting redundancy in regards to failures and splitting up computation. However, in the project the developer was limited to the development machine and the production VPS. All code on the development machine was backed up to a physical external hard drive as well as GitHub, meaning a failure of the development machine would only affect the development pace without a loss of data. The production server from DigitalOcean also provided a *snapshot* feature to perform multiple backups of the entire system state, meaning a failure of the server would not result in lost data and only downtime which is not a large issue in a small scale system such as Revolvr.
- **Third Party Changes:** The reliance on third party sources to aggregate media into the system requires correct functionality on their behalf. If a service were to cease to exist (i.e company liquidation) or change their API significantly this would have to be accounted for. This was actually the case when deploying the existing system, which utilised an older version of the YouTube API and required redevelopment, motivating the consideration of such a risk. The modular design and independence of each service aggregator results in a system where if a third party service were to revoke access or experience issues, a number of other existing services can continue to be used despite these issues. A log of the daily aggregation process

is also emailed to the developer if an error occurs, meaning any issues with aggregation can be noted and resolved quickly.

- **Algorithm Applicability:** When proposing the project, one of the main goals was to simply offer 'better' recommendations with little notion as to how this would be achieved due to the agile approach of the work. Research into applicable algorithms proved initially difficult due to the asymmetric nature of Revolvr's data, and when choosing to use implicit matrix factorisation adapted to this data there was a great deal of uncertainty as to whether the data would adapt to the algorithm. In these case of this approach failing, other considerations would have to be made in the remaining time limit. A number of other collaborative filtering algorithms such as alternative latent factor models and neighbourhood models were made familiar to the developer, to offer possible alternative approaches if needed.

- **Scalability:** The implicit matrix factorisation algorithm adapted for asymmetric feedback has significant computation overheard and therefore scaling it to a large number of items and users is likely to be infeasible without the appropriate hardware. Due to the the project being for academic purposes the decision was made to keep the user base closed to direct and mutual friends of the developer, as well as keeping a whitelist of users who were allowed to sign up for the early stages in the project. This allowed controlled growth of the system appropriate with its capability in terms of performance. As the number of users increased the developer also used personal resources to increase the server hardware specification to ensure the system did not cause delays whilst serving both the web content as well as running the cross validation procedure. However, in future if the systems datastore and userbase were to grow in size alternative approaches would have to be made such as the Hadoop implementation used by Spotify [15].

  These are a few examples of the more interesting risks with standard considerations such as developer illness and user credential exposure being taken into account. Despite none of these risks posing a threat throughout the project, the consideration of possible risks results in the developer being prepared for such occurrences. This allows preparation and results in avoiding panicked uncertainty which could possibly hinder the development process if not accounted for. Reflection on the project does not only concern the processes undertaken but also a reflection of the end product, and therefore the next section will provide a system wide evaluation focussing on why the project can be deemed a success.

CHAPTER 9

Evaluation

Following the design, implementation, testing and evaluation of the complete system an comparison of the final product against the proposed requirements can now occur. Via comprehensive comparison against the requirements the project will not only be quantitatively assessed but also be subject to personal discussion from the developer and their opinions on whether it can be viewed a success.

9.1 Functional Evaluation

A comparison of the systems functionality to the requirements outlined in Section 4.2 provide a direct comparison between the proposed and implemented functionality, aiming as direct measure of success and satisfaction with regards to the final product. Tables 9.1 and 9.2 provide a detailed comparison of the requiremed and implemented functionality, highlighting the fact that the project has met all of its functional requirements successfully.

9.2 Non-Functional Evaluation

Functional performance does not dictate the success of a system, with usability, design and the general experience from a non technical perspective playing a big role. As a result it is also important to consider the non-functional requirements outlined in Section 4.3 and see whether these are reflected in the final system. Table 9.3 provides an evaluation of these requirements against the final solution. Here **NF4** has unfortunately been deemed inconclusive, current processing time on such a small dataset suggest the system will not scale well mainly in regards to the implicit matrix factorisation algorithm.

F#	Brief Description	Comment	Result
F1	Access and Storage of Third-Party Data	Six third party sources can be used as aggregation sources, each with a respective aggregator as part of a larger aggregation process allowing the collection of media items for authenticated users via appropriate API's	PASS
F2	Standardise and Validate Input Media	All media items within the system are abstracted and treated equally to provide a homogenous representation	PASS
F3	Media must be User Attributable	Media is attributed to its associated users upon entry into the database	PASS
F4	User Credentials must be Maintained	User credentials for services are authenticated through the user interface, and are refreshed upon use and / or expiry for longer lasting tokens	PASS
F5	Most popular media must be aggregated	An independent aggregation process collects the most popular items from each of the main media sources on a regular basis	PASS
F6	Inference Must Relate Media	Direct and Similar links are computed from aggregated media, providing a numerical similarity value relating any pair of given users	PASS
F7	Multiple models of inference	The user is able to choose from a semantic algorithm, a 'feature based' algorithm (latent factor model), and a social media focussed algorithm	PASS
F8	Suggestions should be personalised to user	Semantic recommendations are based on taste of titles, feature based recommendations are based on 'factors' or features personal to each user and item, and the social algorithm is based on taste in friends / connections	PASS

Table 9.1: Evaluation of Functional Requirements (1)

F#	Brief Description	Comment	Result
F9	The system should provide a way of recommending via the grouping of users	Both the social and semantic algorithms work by creating a graph of connected users	PASS
F10	Recommend to New and Unauthenticated Users	The system recommends a random subsection of the media contained within the system to new and unauthenticated users, with new user having limited access to the site	-
F11	Ability to train system via user feedback	Users are able to add new preferred media via simple like button, which creates new links / entries for each respective algorithm to process	PASS
F12	Media must be Viewable Within the System	All media being displayed is directly embedded within the site	PASS
F13	A user preferred media must be reconsumable within system	All preferred media (i.e entries in the *user_media table*) are viewable via the profile page	PASS
F14	Media delivery must be engaging and informative	All embedded media is provided in a dynamic interactive player with appropriate media information displayed	PASS
F15	Provide a functional UI	A user can log in and out, authenticate and revoke access to services, view their aggregated media, and have media suggested to them via the user interface	PASS

Table 9.2: Evaluation of Functional Requirements (2)

Requirement	Brief Description	Comment	Result
NF1	Extensibility	Components of the system have been designed with a *plug-and-play* approach, making the addition and removal of technical elements a relatively simple process. The distinction between aggregation, processing and the user interface has also been kept in mind within the design of the system meaning changes to one element can easily be reflected in later or previous stages	PASS
NF2	Documentation / Maintainability	Documentation for the project is vast, and covers a wide range of topics from functional, technical and theoretical viewpoints via the three submitted documents. The development blog and GitHub repository also provide logs of the design process to aid in understanding and code is commented appropriately. Standard coding practises have also been adhered to for familiarity and required technologies are kept up to date	PASS
NF3	Fluidity	The system utilises AJAX technologies to load media and process a majority of the user feedback. The logical flow of the site has also been reconsidered providing a minimalistic and streamlined process	PASS
NF4	Scalability	Despite an effort to minimise processing and delivery time, the inclusion of such a computationally expensive algorithm (IMF) has rendered the systems current implementation likely unscalable. This is not conclusive however as the system has not bene tested on a larger scale but current processing times suggest this rendering the requirement inconclusive	-
NF5	Optimality	The service provides an in-the-box, integrated experience of discovery and whilst utilising third party services does not deduct from the original viewing experience provided by them.	PASS
NF6	Security	Appropriate measures have been taken to protect user information. No direct user credentials are stored due to Facebook login, and sensitive information such as authentication tokens are stored in a password protected database. Parameter checks are also in place in various controllers to parse incoming data from requests	PASS
NF7	Usability	The system provides a consistent user interface throughout on all devices. The use of media staging also reduced the response time when recommending media, and the navigation of the website is clear	PASS

Table 9.3: Evaluation of Non-Functional Requirements

9.3 Legal, Social, Ethical and Professional Evaluation

As well as the key project requirements, a number of legal, social, ethical and professional issues were considered in Section 3. It is important that the final system adheres to these to respect its users, their data, and intellectual property of recommended media and has been a major consideration during development.

9.3.1 Legal Issues

The legality of intellectual property is a grey area on the internet, with piracy and peer to peer sharing growing faster than it can be stopped. Revolvr ensures intellectual property is adhered to by embedding content from trusted third party providers, and only under the condition the user has enabled the media to be embedded externally. YouTube, Vimeo, Spotify and Soundcloud all have respective privacy policies that contain information on content licensing and guidelines, and Revolvr simply relays information from these services. As a result of no direct media hosting issues regarding the storage and distribution of copyrighted content are avoided, and instead the appropriate service provider holds responsibility [63] [64] [65] [66]. Regarding user data, users are given explicit information when connecting a service as to what data it will access and requires them to confirm they allow such access. Users are also provided the option to disconnect a service at any time which results in Revolvr no longer being able to aggregate a users media. However, to fully disconnect Revolvr the user must visit the appropriate third party provider and revoke access to the Revolvr application which cannot be carried out within the system itself. Finally, the use of embedded players means that appropriate advertising, view counts and revenue streams belong to the respective service as opposed to being routed through Revolvr, as well as providing reference to where the original media came from through a stylised embedded player. The *Privacy Policy* also tells users how their information will be used and is easily accessible to ensure users are fully aware of the process.

9.3.2 Social Issues

The personal nature of media taste led to all users being part of an isolated experience, and although they receive media stemming from other users no user-to-user information can be gained. Suggestions are anonymous with no information on where the media came from that may lead to the identification of the user responsible for the aggregation of the item. Revolvr also incorporates algorithms that aim to avoid the social media bias issues described by Messing et al., with each algorithm not utilising cultural, religious or geographic information to make recommendations [18]. The social algorithm may suffer from bias simply due to the fact friends are generally located in the same place, however this is simply the nature of friendships expressed on social media networks. The system has also been designed in such a way that the UI is bold, strong and clear meaning a wide range of users will be able to access and benefit from the site.

9.3.3 Ethical Issues

Regarding privacy concerns, the system provides a full explanation of its motives and how data is used via the *About* and *Privacy Policy* pages, aiming to be transparent in its motives. Providing this information creates a level of trust and reduced uncertainty with users hopefully resulting in continued and loyal use to the system. Furthermore, when requesting authorisation to a users service account Revolvr always strives to use the lowest level of permission required to operate correctly, meaning it does not access a users data unless absolutely necessary.
Content of an explicit nature is not dealt with by Revolvr but is instead left to the established third party providers. SoundCloud and Spotify often contain warnings about explicit lyrics or themes in songs, whereas YouTube and Vimeo require sign in / incorporate an age gate to protect minors. Due to the use of these providers extreme or inappropriate content is avoided all together, and mature content is provided to the user but appropriately labelled.

9.3.4 Professional Issues

Professional conduct has been maintained throughout the project, with a focus on protecting the users of the system and remaining ethical in operation. All user-media relations in the system are done via unique DBMS keys as opposed to usernames to ensure that developers of the system cannot easily identify a user without making an active effort to, in which case appropriate reasoning should motivate the decision. The system also makes use of appropriate the established OAuth authentication protocol, ensuring the system itself never handles or stores valuable credentials. Information gathered on a user and their preferences is also never modified or modelled in ways other than discussed in this report, and there is no interest in providing data to other third parties for monetary gain.

9.4 Project Management Evaluation

As demonstrated in Section 8 the management of a project with such an agile nature is crucial for its success. The developer believes that via continual interaction with the project supervisor, an organised but flexible work schedule as well as a consistent work ethic the management of the project can be regarded as highly successful. Documentation of both the work carried out and that yet to be completed via the management tools discussed in Section 8 provided a structured work flow, contrasting yet complimentary to the free reign given in the agile development workflow. Furthermore, inviting users to test the system and as a result growing the media set found within the system provided data to work with throughout the project and making this decision early on was beneficial. Adjustments to the schedule of the project did occur in this document, as well as in the preceding progress report and specification however this did not hinder the end product and instead demonstrates the ability to look ahead and change plans accordingly without the need stall the creative process. Finally, the personal relationship between the supervisor and the developer allowed open and honest discussions to take place with no hesitation to express disagreement. This honest nature assisted in both parties expressing their best interests in the project and when combined helped move it in the right direction.

9.5 Author's Assessment of the Project

What is the (technical) contribution of this project?

The system in its current form combines technical content from a number of areas to provide a cohesive user focused experience, namely: data aggregation and processing, machine learning, web development, sever management and process automation. Via third party API's the system is able to aggregate preferred content for users to create a user-media profile, a collection of data modelling what items a user likes. Three varying algorithms are then applied to this data each with a unique operation and focus area. The first is a semantic analysis algorithm, the second is an experimental machine learning algorithm, and the final algorithm exploits the structure of social networks. Varying data and varying processes come together to provide a sleek, simplistic user experience in which a focus is on forming and analysing a users media footprint with the end goal of *discovering the undiscovered*. The use of existing work made this a multifaceted project with a focus on application deployment and understanding, the use of external API's, research into the area of recommender systems, working with unfamiliar frameworks, algorithmic design and evaluation and finally software engineering as whole.

Why should this contribution be considered relevant and important for the subject of your degree?

The project utilises a number of different aspects of a Computer Science degree. Firstly, an organisational element is present not only in the project planning and execution but also the deployment and preliminary understanding of the existing implementation. The ability to deploy and use an existing code base is a valuable skill both in academia and in industry and dominated a significant portion of the setup period. A research element also focusses heavily in the project as demonstrated in Section 5.3.2, with the hot areas of both machine learning and recommender systems being greatly explored before settling on a complicated recommendation algorithm. Major development work was also present which involved working with web technologies, multiple operating systems, a variety of differing technologies, server deployment and maintenance and also data collection and processing. The production of a system implementing established academic work which utilises data aggregation, processing and display via an adaptive web based user interface displays competence within a number of fields relevant to a bachelors degree in Computer Science. Finally, the project also displays awareness of the trending topics in Computer Science with big data and machine learning being fields that the project could utilise greatly given further work.

How can others make use of the work in this project?

The resultant system provides a starting point, or *playground*, for future developers to explore work into the parallelisation and scaling of recommender systems via big data processing paradigms such as Hadoop. Furthermore, recommender systems are never perfect and provide room for improvement which inspires future work on the project to be based on improving the existing algorithms. For users, the system provides a simple and user friendly interface to provide inference on a user's

media footprint, with the overall goal of providing recommendations that are tailored to what a user loves.

Why should this project be considered an achievement?

As mentioned previously the project contains a number of different elements making it well rounded and showing the developer is able to apply themselves across many areas. The system can also be considered a success meeting nearly all of its proposed functional and non-functional requirements. Furthermore, user evaluation and algorithmic testing has shown the developers contributions to the ongoing evolution of Revolvr have improved its ability to recommend media to users. Not only do the two new algorithms appear to perform better than the original, but the option to choose between them allows varying recommendations and contrasting processing to take place providing content a user would not normally consider. Asides from the research element, the refactoring of existing components and implementation of new features has allowed Revolvr to move from a proof of concept to a polished, well-rounded system capable of captivating both user interest and also market interest. Finally, the exploration and definition of *asymmetric feedback* was embraced as an opportunity to explore a problem with little existing research as opposed to being viewed as a limitation.

What are the limitations of this project?

Firstly as mentioned throughout the report the scalability of the system has been questioned, especially regarding the matrix factorisation algorithm. Solutions do exist such as generating and processing recommendations via Hadoop, however this was unfortunately out of the scope of the project [15]. Furthermore, the way media is aggregated currently relies on the user putting their preferred media in a specific area of each service. For example, a user must place their preferred YouTube videos in their *liked* playlist but often users prefer to maintain different playlists depending on the video content. The ability to import media within the system using links to playlists / collections of media would be beneficial, and provide the user with control and clarity over which information is being used to build their user profile.

The successful implementation, testing, management and positive evaluation of the project leads the developer and the supervisor to not only have confidence in the project, but also deem it a success. The hours of work put into the system and the challenges along the way may seem like a burden, but the system produced and the doors it has opened in terms of the developers knowledge and confidence are invaluable.

CHAPTER 10

Conclusion

10.1 Summary

The final product provides a way of abstracting user-media preferences into homogenous representations, from which a number of inference procedures can be used to produce recommendations based on a user's tastes (i.e their media footprint). This process is encapsulated in a multi-device user interface, utilising its form as a web app so users are not limited by platform. The potential for expansion still exists and will be discussed in the following subsection but this does not deduct from the wide scope and achievements presented in this project.

10.2 Future Work

Limited development time as well as other academic commitments resulted in development being focussed on specific goals, and as a result numerous ideas that the developer felt would benefit the system did not come to light. A few of these ideas are discussed below, opening up the possibility of further work on Revolvr in future.

10.2.1 Exploiting Higher Level Constructs

An original goal which was put aside to allow more time to be committed to the recommendation algorithms was the exploitation of higher level social media constructs. By this, the developer is referring to service, media, and social specific constructs that could be used to further infer relations to aid recommendation. In terms of social networks using information such as a user's liked Facebook pages could be utilised to provide a more concrete taste profile. For media, features such as genre, tempo, key and length could be exploited to provide further details about the item.

These would provide less of an abstracted approach like the current system and instead focus on unique traits of a service to tailor recommendations to an even better extent.

10.2.2 Scalability

As mentioned throughout the report the system in its current state is not likely to scale appropriately. Future iterations would benefit from exploiting concepts such as cluster computing and Hadoop, to provide a scalable performant solution to recommending media to a mass audience.

10.2.3 Integrated Media Management and Library

Towards the end of the project the developer realised Revolvr's potential to not only act as a discovery system, but also an integrated media management tool. The ability to both discover and manage media from a number of different sources to provide a centralised library is invaluable and although solutions exist none of them have perfected this concept yet. Adding media from external playlists / collections, creating and sharing playlists, and the ability to search across many external providers at once could be considered game changers in the digital media landscape and is a concept that excites the developer greatly.

10.2.4 Larger Dataset

A major criticism of the project was the small sample size, which stemmed from having to control user numbers due to the scalability concerns. Given an appropriate setup, reevaluation of how the system performs both computationally and qualitatively to backup the claims made in this report would be beneficial. More media and users is also likely to provide more performant and varied recommendation results increasing the appeal of the system due to a wide range of available media.

Bibliography

[1] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM.

[2] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[3] Hridya Sobhanam and AK Mariappan. Addressing cold start problem in recommender systems using association rules and clustering technique. In *2013 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–5. IEEE, 2013.

[4] Chris Chamberlain. Revolvr: A crowd-sourced media content aggregation and suggestion system, 2014.

[5] Pierre Carbonnelle. Pypl popularity of programming language. `http://pypl.github.io/PYPL.html`, 2016.

[6] Simon Kemp. Digital, social & mobile, 2015.

[7] Sandvine. Global internet phenomena report. africa, middle east and north america. `https://www.sandvine.com/trends/global-internet-phenomena/`, September 2015.

[8] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[9] IFPI. Ifpi digital music report 2015: Charting the path to sustainable growth, 2015.

[10] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize, 2007.

[11] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13:1–13:19, December 2015.

[12] Chris Johnson and Edward Newett. From idea to execution: Spotify's discover weekly. http://www.slideshare.net/MrChrisJohnson/from-idea-to-execution-spotifys-discover-weekly, November 2015.

[13] Xinxi Wang, Yi Wang, David Hsu, and Ye Wang. Exploration in interactive personalized music recommendation: A reinforcement learning approach. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1):7:1–7:22, September 2014.

[14] MN Carleton College, Northfield. Recommender systems. http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/collaborativefiltering.html.

[15] Chris Johnson. Algorithmic music discovery at spotify. http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify, January 2014.

[16] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[17] British Computing Society. Ethics of computing. chapter British Computer Society Code of Practice, pages 102–110. Chapman & Hall, Ltd., London, UK, UK, 1996.

[18] Solomon Messing and Sean J Westwood. Friends that matter: How social influence affects selection in social media, 2013.

[19] Jason Mander. Global web index: Social summary q1 2016. http://www.globalwebindex.net/hubfs/Reports/GWI_Social_-_Q1_2016_Summary.pdf, March 2016.

[20] Facebook. Facebook login. https://developers.facebook.com/docs/facebook-login.

[21] Dick Hardt. The oauth 2.0 authorization framework. 2012.

[22] Intridea Inc. Omniauth. https://github.com/intridea/omniauth/wiki, 2015.

[23] GitHub. Omniauth-spotify: Token refresh. https://github.com/icoretech/omniauth-spotify/issues/4, 2014.

[24] Bee-Chung Chen, Deepak Agarwal, Pradheep Elango, and Raghu Ramal. Latent factor models for web recommender systems. http://www.ideal.ece.utexas.edu/seminar/LatentFactorModels.pdf.

[25] S. Funk. Netflix update: Try this at home. http://sifter.org/~simon/journal/20061211.html, 2006.

[26] Lei Guo. Matrix factorization techniques for recommender systems. http://www.slideshare.net/studentalei/matrix-factorization-techniques-for-recommender-systems, 2012.

[27] Adam Lella and Andrew Lipsman. The u.s. mobile app report. http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report, August 2014.

[28] Josh Dehlinger and Jeremy Dixon. Mobile application software engineering: Challenges and research directions.

[29] Bootstrap. http://www.getbootstrap.com, 2015.

[30] Python. http://www.python.org, 2015.

[31] Python Software Foundation. Python package index. https://pypi.python.org/pypi.

[32] MongoDB Inc. http://www.mongodb.com, 2015.

[33] David Heinemeier Hansson. http://www.rubyonrails.org, 2015.

[34] Daniel Kehoe. Railsapp project: What is ruby on rails? http://railsapps.github.io/what-is-ruby-rails.html, October 2013.

[35] RubyGems. Rubygems guides. http://guides.rubygems.org/what-is-a-gem/.

[36] Leonard Richardson. Beautifulsoup. https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[37] Chris Johnson. Github: Mrchrisjohnson / implicit-mf. https://github.com/MrChrisJohnson/implicit-mf, January 2015.

[38] NumPy Developers. Numpy. http://www.numpy.org, 2016.

[39] pandas. http://pandas.pydata.org.

[40] Scipy Community. Scipy.org: Sparse matrices (scipy.sparse). http://docs.scipy.org/doc/scipy/reference/sparse.html.

[41] Christopher C Johnson. Logistic matrix factorization for implicit feedback data. 2014.

[42] Scipy Community. Scipy: scipy.sparse.linalg.spsolve. http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.linalg.spsolve.html.

[43] Twitter Inc. Twitter rest apis: Get friends/list. https://dev.twitter.com/rest/reference/get/friends/list, 2016.

[44] Facebook Inc. Facebook: Graph api. https://developers.facebook.com/docs/graph-api/reference/v2.5/user/friends, 2016.

[45] BlackTie Themes. http://www.blacktie.co/2013/12/flatty-app-landing-page/, March 2014.

[46] Font Awesome. http://fortawesome.github.io/Font-Awesome/, March 2014.

[47] Lato Google Fonts. https://www.google.com/fonts/specimen/Lato, March 2014.

[48] Steven Wanderski. bxslider: The responsive jquery content slider. http://bxslider.com, 2016.

[49] Gitte Lindgaard, Gary Fernandes, Cathy Dudek, and Judith Brown. Attention web designers: You have 50 milliseconds to make a good first impression! *Behaviour & information technology*, 25(2):115–126, 2006.

[50] Alexandre N. Tuch, Eva E. Presslaber, Markus Stöcklin, Klaus Opwis, and Javier A. Bargas-Avila. The role of visual complexity and prototypicality regarding first impression of websites: Working towards understanding aesthetic judgments. *International Journal of Human-Computer Studies*, 70(11):794–811, November 2012.

[51] Joost de Wit. Evaluating recommender systems : an evaluation framework to predict user satisfaction for recommender systems in an electronic programme guide context, May 2008.

[52] Thin A fast and very simple Ruby web server. http://code.macournoyer.com/thin/, April 2014.

[53] Digital Ocean. http://www.digitalocean.com, 2015.

[54] Hampton Catlin, Natalie Weizenbaum, and Chris Eppstein. Sass: Syntactically awesome style sheets. http://sass-lang.com, 2015.

[55] Jeremy Ashkenas. Coffeescript. http://coffeescript.org, 2016.

[56] The Apache Software Foundation. Apache http server project, 2015.

[57] Phusion. Phusion passenger. https://www.phusionpassenger.com, 2015.

[58] GitHub. http://www.github.com, 2015.

[59] Jon Skinner. Sublime text. http://www.sublimetext.com, 2015.

[60] WordPress. http://www.wordpress.com, 2015.

[61] Jordan Wyatt. Cs310 blog. www.CS310blog.wordpress.com, 2015.

[62] Microsoft Corporation. Wunderlist. https://www.wunderlist.com, 2016.

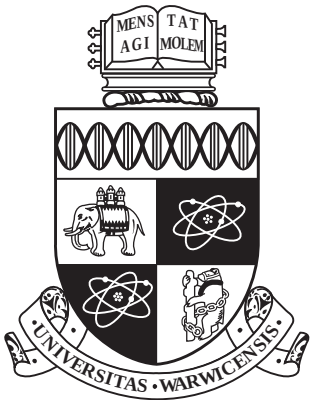
[63] Spotify AB. Spotify privacy policy. https://www.spotify.com/uk/legal/privacy-policy/, 2016.

[64] Google. Youtube privacy guidelines. https://www.youtube.com/static?template=privacy_guidelines&gl=GB, 2016.

[65] LLC Vimeo. Vimeo.com privacy policy. https://vimeo.com/privacy, 2016.

[66] SoundCloud Limited. Soundcloud privacy policy. https://soundcloud.com/pages/privacy, 2016.

Specification Report



Crowd Sourced Content Aggregation and Suggestion

CS310 Computer Science Project

Project Specification

Jordan Wyatt

Supervisor: Dr. Matthew Leeke

Department of Computer Science
University of Warwick

2015-2016

Contents

This project specification details the proposed improvement and ongoing development to Revolvr, a crowd-sourced media content aggregation and suggestion system developed by Christopher Chamberlain between 2013-2014 [6]. Revolvr is a web based software solution to identifying an individuals *media footprint* via the analysis and processing of homogeneous media items. The suggested improvements include algorithmic focused development, as well as various UI and usability improvements which are outlined in this specification.

1 Introduction

The modern world is vastly connected, with close to half of the population having access to the internet. As of January 2015 there are 3.010 billion active internet users, with 2.078 billion of these users having active social media accounts [15]. These figures are the product of a year-on-year growth, which saw a 21% increase in the number of active internet users, and a 12% increase in the number of active social media accounts in the space of 12 months. With such a rapidly evolving digital landscape also comes an influx in the quantity, accessibility and consumption of online media. However, with such easy access also comes the problem of over saturation. Given such vast quantities of data how can a user identify relevant and desirable content for their tastes?

1.1 Motivation

As discussed in Christopher Chamberlain's previous work on Revolvr, consumers of online media are concious of their *online footprint* [6]. With monumental developments in mobile technology in the past decade mobile and tablet devices now boast a 38% share of worldwide web traffic [15]. As a result, access to the web and its media is easier than ever before and the development of an online footprint is almost unavoidable. The sources of such content vary, from general purpose social networks such as Twitter and Facebook to format specific platforms such as Spotify and Netflix, a user has a wide range of options and discovering content they find relevant often feels like finding a needle in a haystack. Revolvr was developed as an attempt overcome this problem, with its documentation stating 'why settle for what you've already seen, when technology can steer you towards something you may love, perhaps before you know it?' [6].

However, as outlined in the existing project report for Revolvr, limited development time and ensuring the delivered product met the proposed requirements left the software more as a proof of concept, as opposed to an effective content aggregator. The existing work provides a good basis to allow not only improvements to the current algorithm and UI, but also research opportunities to evaluate variations of the implemented algorithm and explore areas of recommender systems, sentiment analysis, and machine learning.

1.2 Project Aims

The overarching aim of this project is to develop a content aggregation and recommendation based on the existing Revolvr framework, with focus on media aggregation, analysis and exploitation of platform specific features, algorithmic development and analysis and user interface design. These aims contain a mix of algorithmic research, scientific analysis, and development / design work which allow the project to have a good depth and breadth. Due to being continuation of work the overhead of setup is reduced allowing focused development points.

1.2.1 Expanded Aggregation

Although an instance of Revolvr has not been deployed on a server yet, inspection of the existing codebase and discussion with the project supervisor led the developer to believe that although Revolvr in its current form can connect to a number of services (Facebook, Twitter, YouTube, Vimeo, Soundcloud and Last.fm) it does not necessarily utilise these services to their full extent. Time will be devoted to ensuring the existing providers are suitable to use in the system, and in this case that the aggregation is effective and distributed between services. This aim will also consider any new services that surfaced since Revolvr's initial development, and aim to incorporate new providers where both possible and suitable.

1.2.2 Exploiting Higher Level Social Media Constructs

The system in its current state only performs aggregation on heterogeneous media source, ignoring meta data and any service by service discrepancies such as followers on Twitter or likes on Soundcloud. Identifying these unique features of each media platform would provide a further link within the system to help develop their media footprint. Examples of metadata that can be used include a users liked Facebook pages, their Soundcloud followers, and who they follow on Twitter. Genre based analysis of songs and videos is also a topic that is open to exploration. This area of improvement hopes to build on the foundations Revolvr has established allowing tailored suggestion for users, whilst at the same time taking an individual services features into account and abstracting these into generalised characteristics for aid in recommendation.

1.2.3 Algorithmic Development

Revolvr currently only uses a single algorithm to make suggestions to users, for the project it is beneficial for the developer both for research and the interests of the product to investigate and implement other recommender algorithms. Allowing a user to choose between a number of different algorithms is likely to provide fresh and perhaps unexpected content to be recommended when their interest is fading. Research will be made into state of the art recommender systems and the approaches they take in their suggestions, and from this a number of bespoke algorithms will be developed for the new iteration of the system. Another area of aggregation that can be improved includes enhanced sentiment analysis. Sentiment analysis is a feature in Revolvr which reduces a piece of medias title to a more meaningful subset of phrases allowing for simple analysis [6, Ch. 5.3.1, p. 40]. However this analytical process does not consider a scenario in which a user shares a

piece of media for negative reasons and therefore the associated phrases are negative. Development of this process will aim to analyse the *mood* of a sentence, identifying whether it is positive or negative. The developer also hopes to implement variations of the recommendation algorithm, the results of which will undergo comparative analysis.

1.2.4 Improved User Interface

The original vision for Revolvr was to incorporate jQuery and AJAX to create dynamic pages without the need for a refresh to receive new content, improving the fluidity of the project. The name of the software also comes from the idea of an infinitely scrollable wheel of media which *revolves* [6, Ch. 10.2.4, p. 118]. Implementation of this original design along with the aforementioned algorithmic improvements will create a great deal more of interactivity and immersion. General improvements of the interface will also take place to ensure the application is up to date regarding UI design trends.

1.3 Stakeholders

The main stakeholders that exist within this project are the developer and the project supervisor. The influence and opinion of the previous developer of Revolvr is also invaluable, and their opinion on development will be sought after where possible.

2 Related Work

Consumption of online multimedia is at an all time high, in 2014 the value of the digital music market increased 6.9% to $6.9 billion, representing 46% of global music sales and matching physical sales [12]. Streaming services such as Netflix have also overtaken live viewing this year according to Deloitte's 'Digital Democracy Survey', and content creators on YouTube are earning millions from advertising on their uploaded videos gathering view counts major TV networks dream of achieving [7] [3].

These services, their content providers, and users benefit from algorithms to suggest content to users based on what they like. A platform is only as good as the content it provides, and companies go to great lengths to ensure recommendation algorithms are as accurate as they can be. An example of this was the 2006 Netflix Prize, a *'machine learning and data mining competition for movie rating prediction'* [2]. Knowledge of real life application of recommender systems and the theory underlying it are essential for understanding the existing system and developing it further.

2.1 Recommender Systems

Recommender systems are defined in Resnick and Varian's seminal article as follows: *'In a typical recommender system people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the systems value lies in its ability to make good matches between the recommenders and those seeking recommendations.'* [20]. They have proven to be useful means for users to

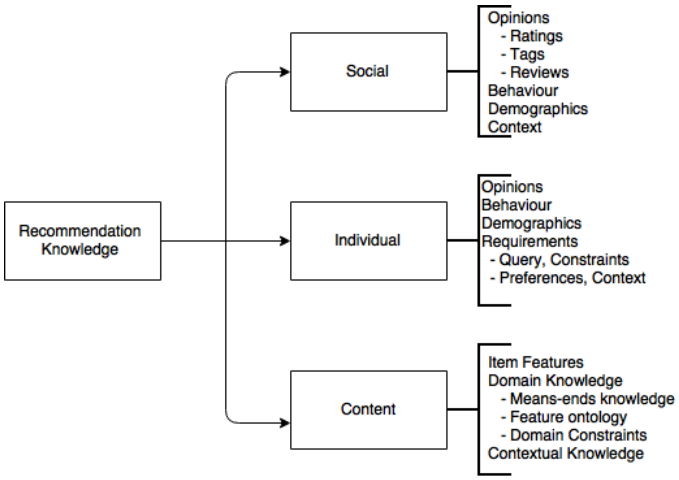


Figure 1: Taxonomy of knowledge sources in recommendation [9]

cope with overload of information and are used extensively in e-commerce. Recommender systems are able to be classified according to the knowledge sources they use, as seen in Figure 1. This taxonomy proposes three types of available knowledge to a recommender system: [9]

- **Collaborative (Social) based knowledge:** knowledge concerning other users.
- **User (Individual) based knowledge:** knowledge of the individual user
- **Content based knowledge:** Knowledge about the items being recommended

A recommender system may use a combination of these knowledge bases, or focus solely on a single one. This taxonomy also helps understand different recommendation techniques.

2.1.1 Recommendation Techniques

As defined in Felfernig and Burke's 2008 paper, a recommendation technique is *'a set of knowledge sources and an algorithmic approach to generating recommendations using those sources'* [9]. The most common recommendation techniques are collaborative recommendation, content-based recommendation and hybrid recommendation. Collaborative filtering is based on gathering information on a user's activities and preferences, and from this predicting what they may like based on similar users. This method benefits from not requiring prior knowledge on an item, so can generally be used without any prerequisite information. In contrast, content-based filtering uses a description of the item and a user's preferences and from this decides if the user will like the suggested item or not. In layman's terms collaborative filtering stems from the idea if two people agree they're likely to agree again in future, and content-based implies a user is likely to seek items similar to what they have liked before. Hybrid recommender systems combine collaborative filtering and content

Table I: Recommendation Techniques

Technique	Background	Input	Process
Collaborative	Ratings from **U** of items in **I**.	Ratings from **u** of items in **I**.	Identify users in **U** similar to **u**, and extrapolate from their ratings of **i**.
Content-based	Features of items in **I**	**u**'s ratings of items in **I**	Generate a classifier that fits **u**'s rating behavior and use it on **i**.
Demographic	Demographic information about **U** and their ratings of items in **I**.	Demographic information about **u**.	Identify users that are demographically similar to **u**, and extrapolate from their ratings of **i**.
Utility-based	Features of items in **I**.	A utility function over items in **I** that describes **u**'s preferences.	Apply the function to the items and determine **i**'s rank.
Knowledge-based	Features of items in **I**. Knowledge of how these items meet a user's needs.	A description of **u**'s needs or interests.	Infer a match between **i** and **u**'s need.

Figure 2: Recommender Techniques [5]

based filtering in a number of possible ways. One approach is to make predictions with each approach separately then combine the results, while in contrast you can design model which unifies the approach of both. Studies have suggested that combining collaborative and content-based filtering and be more effective in a recommender system. Netflix is a successful real world example of hybrid recommender system, collaborative filtering is implemented by comparing watching and searching habits of similar users and recommending content based on a user's previous ratings is content-based filtering.

The techniques discussed above are not exhaustive when talking about types of recommender systems, Robin Burke's 2002paper distinguishes the recommendation techniques shown in Figure 2 [5]. Here **I** is the set of items over which recommendations may be made, **U** is the set of users with known preferences, **u** is the user who requires recommendations, and **i** is some item for which we would like to predict **u**'s preference. Further into the project exploration of the more niche methods will be beneficial, but for specification purposes a high level understanding to plan the project is the key discussion.

2.2 Existing Systems

With social networks and online media platforms showing ever increasing growth the development of recommender systems continues to flourish. To help inspire the development of Revolvr and also see practical examples it is important to consider currently popular and successful applications to see the benefits of these systems. Although the source code and methodologies behind these

technologies are mostly private, it is still evident where concepts from recommender systems are being applied.

Social Networks - Social networks such as Twitter and Facebook offer recommendations in the form of suggested friends and followers, their respective names for users of the sites you can connect with. Although the details of how this works are not public, it is likely to be a matrix or graph algorithm which looks at common and different neighbours between nodes to make suggestions. Twitter has also recently implemented a 'While you were away' feature, which aims to *'recap some of the top Tweets you might have missed from accounts you follow'* based on engagement and other factors. Although a small feature in the grand scheme of things, it adds personalisation via recommendation to a user's overall experience [21].

Video Streaming - Online video streaming is arguably the most important main stream commercial use of recommender systems in the past few years. With online subscription based models such as Netflix boasting an impressive 65 million members as of July 2015 [17], it is important from a business perspective to ensure that new users can find content they want, and existing users have such a good experience that they become invested in the product. Netflix suggests movies using a number of advanced machine learning techniques, notably Restricted Boltzman Machines and a form of Matrix Factorization [1] [2]. Although the complexity of these systems are likely to be beyond the scope of this project, their effectiveness is asserted with the fact that *'75% of what people watch is from some sort of recommendation'* on Netflix and the importance of recommender systems within media oriented services becomes apparent [2]. YouTube also offers related and suggested videos based on engagement, promoting videos that have a high average viewing time to video length ratio [27]. These platforms show that a strong recommendation algorithm is crucial when it comes to a dedicated user base and viewer engagement.

Music Streaming - Similarly to the aforementioned video platforms, online music streaming services such as Spotify and Apple Music are often more sought after than physical media. By offering extensive libraries for a low monthly cost and portability via mobile applications they are quickly becoming the preferred method of music consumption for the average user. However with over 30 million songs on some platforms it is important to allow users to discover new content they will enjoy easily so they do not become bored due to lack of new material [24]. Spotify implements both a radio feature and a 'Discover Weekly' playlist. The former discovers songs and creates a shuffled playlist based on the song you choose to deploy the feature on, and the latter is a weekly playlist which provides 20 songs an algorithm believes the user may enjoy based on their listening habits. Spotify use an API provided by The Echo Nest which provides services such as dynamic music data and music discovery and personalisation [16].

These are just a few examples of widely used application incorporating concepts of machine learning and recommender systems at a large scale. Other areas of interest include e-commerce retailers such as Amazon and eBay, who offer both in site recommendations and also external advertisements based on your browsing habits.

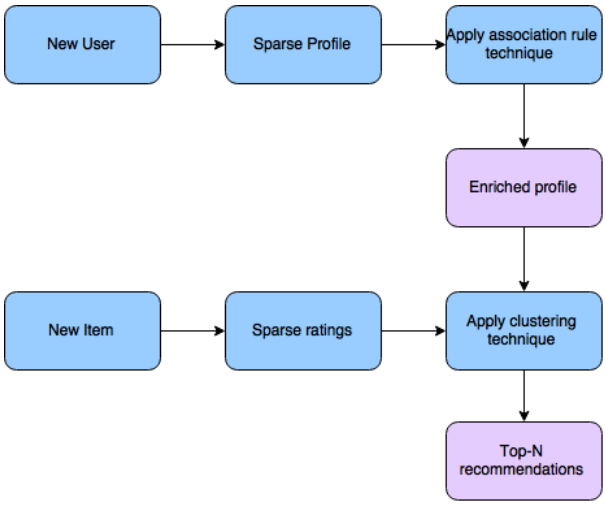


Figure 3: A possible cold-start problem solution [22]

2.3 Cold Start Problem

Recommendation systems often suffer from what is known as the *cold-start* problem. This is defined as when *'recommenders cannot draw inferences for users or items for which it does not have sufficient information'* [22]. This issue can effect both users and items. A new user will be given non personalised recommendations until they have rated enough items to build a profile on their tastes, and a new item in the system will not have any rating information and it is unlikely to be recommended a user. For a user the initial recommendations are crucial on the first impression the system leaves on them, and a poor performance could reduce the appeal of the system.

Two common ways to solve the cold start problem are *association rules* and *clustering techniques*, as discussed in Hridya Sobhanam's 2013 paper [22]. Association rules focus on expanding a user profile so it contains more ratings, and clustering groups of new and existing items based on similarity measures to make predictions for the item. Figure 3 shows one example of a flow of processes that can be performed to attempt to combat this problem. The intricacies of these processes are beyond the technical scope for this specification, but knowing about the problem and the solutions that exist allows peace of mind when designing aggregation systems. Revolvr currently aims to defeat this problem by simply aggregating the most popular content from its services and displaying them to users with no recommendations. Although this isn't fool proof it is sufficient until further work on the problem takes place.

3 Project Requirements

Clear project requirements are vital for tracking progress and providing clear development goals throughout the project. The following requirements outline the current vision for the system for April 2016, but as development occurs are subject to change.

3.1 Functional Requirements

F1: *The system must be able to connect to and retrieve data from a number of third-party sources*

F2: *The system must store and convert data from selected third parties in a common format*

F3: *Data in the system must belong to a group of individuals, or a single individual as a minimum*

F4: *Credentials for the media sources an individual wishes to use data from must be stored*

F5: *Inference must be made to enrich input data obtained from third-parties*

F6: *Multiple models of inference must be available to the user*

F7: *Suggestions should be personalised based on an individual's tastes*

F8: *The system should be able to group users, and provide suggestions based on the preferences of these similar grouped users*

F9: *The system should be able to recommend to new users whilst avoiding the cold-start problem*

F10: *User feedback must be fed into the system to refine the inference mechanism*

F11: *The media must be consumable within the application by the user, without deducing from the experience provided by the original source platform*

F12: *The inference mechanism must exploit platform specific features to enhance the recommendation experience*

F13: *The system must provide an 'infinite' revolving wheel of recommendations on an easily accessible home screen*

3.2 Non-functional Requirements

NF1: *The system must be modular, to allow extensibility*

NF2: *The system should be maintainable, commented and provide strong documentation for future work*

NF3: *The system should provide a seamless user experience, more so than the original implementation*

NF4: *The system should be designed so as to allow scalability*

NF5: *The system should be preferable for discovery as opposed to using the individual services separately*

3.3 Hardware and Software Constraints

As with any software application optimisation in regards to scalability is an unpredictable factor. For the whole duration of this project it is likely the server will be on a cheap, low capacity hosted server due to available resources. As a result this means ensuring the software can cope with extreme amounts of data and users will be a challenge. It must also be ensured that any additions to the existing system aim to impact performance positively and do not reduce the quality of the existing software solutions. The new revolving content wheel may be a challenge to incorporate on mobile devices, and in the case that it isn't feasible an alternative option per device must be developed.

3.4 Foreseeable Challenges

The large existing code base is yet to be deployed and explored, changes to the above requirements and the details listed in this specification are subject to change based on the pace of the project and unforeseen circumstances.

4 Legal, Ethical, Social and Professional Issues

As with all software products, especially ones of such a personal nature, it is important to ensure the product follows standards and can be trusted by users. This section of the specification will discuss the possible issues that could arise during the development of Revolvr in the legal, ethical, social and professional domains without careful planning and consideration. To assist the developer the British Computing Society Code of Practise was taken into account, the issues discussed are not exhaustive but instead are the most applicable [23]. The previous developer also ensured the work that was left adhered to these standards [6].

4.1 Legal Issues

Due to the nature of the system, Revolvr will be processing and displaying media from third party sources. A clear concern here is the topic of intellectual property and licensing. In regards to licensing, Revolvr will only be aggregating content from trusted and established sources that themselves adhere to the correct licensing laws and terms and conditions. As a result of this the system must also comply to the same standards that the third party providers do. Intellectual property will be preserved in a number of ways, firstly the API's for the aggregated services all provide watermarks on the embedded players to show the source of the content, and by following the link these watermarks provided the original uploader receives credit too. A software license agreement will also be developed which state where the products obligations lie.

Although legal issues are an important topic in all products, in this case the developer and the project supervisor are confident that by using trusted and reputable aggregation sources such problems can be avoided.

4.2 Ethical Issues

The key ethical concerns with user oriented sites is data privacy, and with media oriented products also comes the task of filtering explicit content. The former will be guaranteed by encrypting any sensitive user data such as passwords, and Revolvr's philosophy is concerned with respecting the user so their data will not be passed to third parties in any circumstances. Regarding suggested content of explicit nature the platforms Revolvr uses will themselves provide filtering of explicit content (i.e YouTube's age gate feature). This obviously is not fool proof, however this is again a situation where the system can rely on the third party content providers to do their jobs by being reliable and trusted platforms. Although this may come across as a lack of planning or carelessness, Revolvr will ensure it only aggregates from sources who the stakeholders believe fit the mould in regards to the provided content the provide and its suitability for the system.

4.3 Social Issues

Social issues are often the ones that can make the user feel victimised, as opposed to approaching from a business or financial perspectie. Revolvr will thrive for inclusivity, ensuring that perspectives such as cultural, social, gender and disability are taken into account into the refreshed design. This will be inforced by ensuring the technology is available to all, and does not cater to a specific race or gender. In terms of disability the system will take into account colour blind and disabled users, by ensuring multiple queues are available to promote the functions of the product and making the UI clear and intuitive. Although multiple languages are likely to not be supported due to time constraints and lack of resources the design of the UI will be approached such that both visual and language prompts are provided.

4.4 Professional Issues

Being a user oriented service it is important to value the user to create a sense of worth for the platform and a relationship of trust. As previously mentioned all data provided by a user will be kept private and only used for the service they receive. The previous developer also ensured all services included used secure channels and session tokens, this ensures that the connections to the providers cannot be exploited by malicious third parties if an attack were to occur. Finally, the developer will act according to the aforementioned BCS code of conduct by showing public interest, professional competence and integrity, duty to relevant authority and showing duty to the profession [23].

5 Project Management

Due to this project being a continuation of existing work, there is a significant amount of overhead with regards to understanding the product and the concepts it utilises before moving forward. As a result it is important to have a solid plan to ensure work continues to be efficient, whilst still allowing time for contingencies and set-up periods. However, situations can occur with plan driven development where deadlines can act as contingencies themselves and therefore flexibility will be provided by using an agile approach which provides lenience in proposals contained in this specification.

5.1 Design Approach

An agile approach will be taken during the development of this project, both in terms of the software development itself and also the project documentation and management. This allows structure and sensible deadlines, whilst still providing the ability to respond to unforeseen circumstances. The software methodology intricacies are further discussed in section 5.3 of this specification.
A weekly informal Wordpress blog will be updated weekly to map progress for the developer and supervisor, discussing the prior weeks achievements and the coming weeks aims to ensure a record of work is kept [26]. Alongside this the report will be compiled throughout the whole project to ensure the latest developments can be explained in good detail without long term memory recall. Finally, Git will also be used as a version control with appropriate commit messages to ensure the software itself is organised. This design approach ultimately aims to provide a structured and focused development process, whilst still allowing experimentation and change if deemed necessary or beneficial for the project.

5.2 Project Timeline

The project will aim to follow the deadlines proposed in the Gantt chart showing in Figure 4 and the accompanying table of dates in Table 1. Consideration for refinements and user feedback are taken into account towards the end of the project, during this period the developer hopes to be applying tweaks with the core functionality and additions tested and functioning correctly. This Gannt chart at this early point in the project can be subject to change due to the agile nature discussed previously, and will be revisited in the forthcoming progress report.

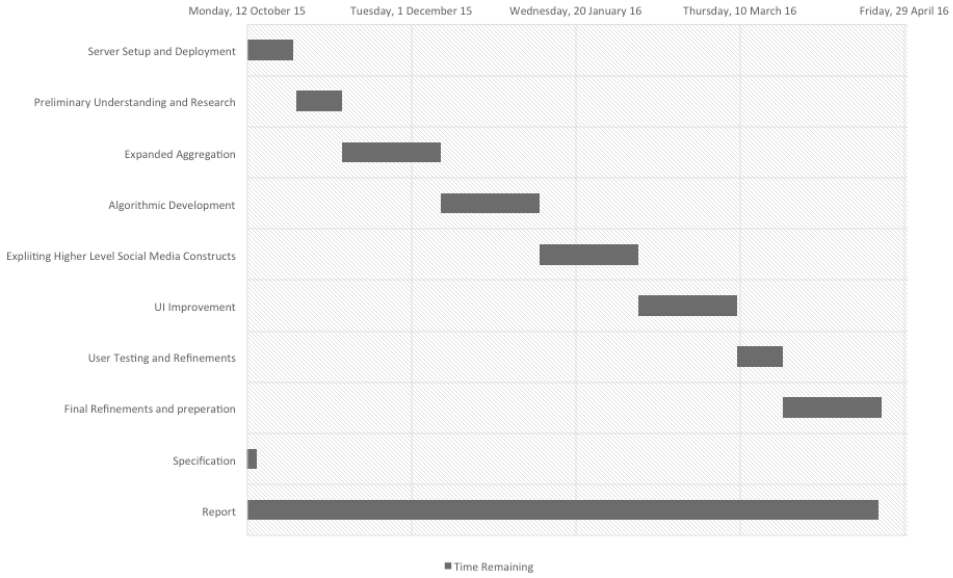


Figure 4: Gannt chart showing proposed project deadlines

Table 1: Proposed Deadlines

Process	**Proposed Start**	**Proposed End**
Server Setup and Deployment	12 October 2015	27 October 2015
Preliminary Understanding Research	27 October 2015	10 November 2015
Expanded Aggregation	10 November 2015	10 December 2015
Algorithmic Development	10 December 2015	9 January 2016
Exploiting Higher Level Social Media Constructs	9 January 2016	8 February 2016
UI Improvement	8 February 2016	9 March 2016
User Testing and Refinements	9 March 2016	23 March 2016
Final Refinements and Preperation	23 March 2016	22 April 2016
Specification	7 October 2015	15 October 2015
Report	12 October 2015	30 April 2016

5.3 Software Development Methodology

Similarly to the design ethos discussed in Section 5.1, flexibility with focus is the key to this project and therefore a hybrid agile development methodology will be undertaken. A concrete classification such as SCRUM or Extreme Programming will not be beneficial to this project as the developer is the only person working on the project and most agile methods are defined for a team. Instead, a flexible gannt chart subject to change where necessary (see Section 5.2) has been constructed, where each time period can be seen as a sprint with set yet open ended design goals.

5.4 Tools

Revolvr's existing documentation has provided a comprehensive list of tools used in its development [6]. At the current point in time the developer believes this list will remain mostly the same and the key aspects are given below, however if a proposed aim requires the addition of a new technology this can be adopted due to an agile approach and a modular design. Further details on this will be provided in the progress report when the developer has fully hosted and analysed the existing product.

5.4.1 Data Collection and Analysis

Python will be used to collect and analyse the data, this choice was made due to the wide variety of external plugins and modules available for working with data as well as the external platforms the system exploits [19].

5.4.2 Data Storage

The existing solution is built using MongoDB, an open source, document-oriented database. Mongo focuses on both scalability and development agility and therefore allows quicker work than conventional database solutions such as MySQL[13].

5.4.3 Data Visualisation

A combination of Ruby on Rails and Bootstrap will be used to develop the front end [11] [4]. Ruby on Rails will provide the content for the user interface, and Bootstrap will be used to provide sleek and consistent styling which performs well on both conventional and mobile devices.

5.4.4 Organisation and Communication

Organisation is key in a project of this size, and therefore exploiting tools to help direct and maintain the project is extremely beneficial. As mentioned previously a Wordpress blog has been created to track progress, to accompany this a Dropbox folder is being shared with the project supervisor to monitor all written work as well as code [25] [8]. Version control will rely on Git using a private GitHub repository to protect intellectual property, and all other communication will occur in person, email or instant messaging [10].

5.4.5 Other Considerations

As shown in Figure 4, the setup period and learning the required technologies is an ongoing process so implementations of the web server and smaller details are currently undecided. As of the time of writing, an Ubuntu server provided over the cloud by DigitalOcean is being setup to host the application on which is running nginx [18]. The server has 512MB memory with 20GB disk capacity, and is hosted via a SSD cloud server for optimal performance. The previous iteration ran on Apache but migrating to nginx should not pose any significant issues. Further details on this will be provided in later documents when the existing code base has been deployed successfully.

6 Testing and Success Measurement

Comprehensive testing using a number of established strategies will be used to ensure the project meets the specified requirements. The organisation and quality of the codebase will also be maintained throughout the project.

6.1 Testing Strategy

The testing strategies implemented are outlined below which include unit, integration, system and user testing.

6.1.1 Unit Testing

Unit testing is focused on verifying the functionality of a specific section of code such as individual functions or classes. All original and new code will have multiple tests to catch issues such as branching errors or corner cases. The relevant testing framework for the incorporated technologies being tested will be learnt and applied to reduce uncertainty.

6.1.2 Integration Testing

Once unit testing provides sufficient results, the tested modules will be integrated with one another to test functionality in a larger scale, moving closer to the real world execution conditions. This area of testing will aim to expose defects in the interaction between components as opposed to errors with the components themselves.

6.1.3 System Testing

Following integration testing system testing will be thoroughly applied, which involves testing the system as the whole software solution that the end users would interact with. This will ultimately be the testing stage which verifies the proposed requirements in Section 3 are achieved. However, the prior testing stages cannot be neglected as optimal results in unit and module level tests are necessary to ensure system testing is ready to occur.

6.1.4 User Testing

Live user testing will be invaluable to the final system due to being such a user oriented service. Once initial developer testing has been signed off peers of the developer and the supervisor will be given access to an alpha, hopefully providing useful feedback on the functionality and quality of the system.

6.2 Success Measurement

A lot of questionable metrics are suggested for measuring success within a software engineering project, a task which becomes even more of a challenge when implementing a nondeterministic agile development methodology. This project will measure the success of the project based on the

concepts delivered in The University of Warwick's CS261 Software Engineering module [14], which are the following:

- How does the software meet the original specification?
- How does the software meet the customer's expections?

Quantitative analysis will take place in the form of analysing the number of functional and non-functional requirements that are achieved, which will explore whether the project fulfils its original specification. The customer's expectations are qualitative, subjective metrics and although difficult to draw conclusions from will be taken into account by providing questionnaires or surveys to the users testing the software.

7 Conclusion

To summarise this specification, the project aims to build develop a system for content aggregation and recommendation based on the existing Revolvr framework. The emphasis on development will be focused on the aims outlined in Section 3 of this document, specifically these areas are media aggregation, analysis and exploitation of platform specific features, algorithmic development and analysis, and user interface design. By utilising the promising foundations of Revolvr, a project compromising of both a development and research element can be undertaken with the final goal of presenting a system which can not only assist the user, but take them on a personalised journey of content discovery

> 'A lot of times, people don't know what they want until you show it to them."
>
> Steve Jobs

References

[1] Xavier Amatriain. How does the netflix movie recommendation algorithm work? https://www.quora.com/How-does-the-Netflix-movie-recommendation-algorithm-work, 2015.

[2] Xavier Amatriain and Justin Basilico. Netflix recommendations: Beyond the 5 stars (part 1). http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html, 2012.

[3] Damon Beres. Youtube stars' huge earnings will make you question your life choices. http://www.huffingtonpost.com/2015/02/05/youtube-stars-money_n_6549906.html, 2015.

[4] Bootstrap. http://www.getbootstrap.com, 2015.

[5] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[6] Chris Chamberlain. Revolvr: A crowd-sourced media content aggregation and suggestion system, 2014.

[7] Deloitte. Digital democracy survey: A multi-generational view of consumer technology, media and telecom trends, 2015.

[8] Dropbox. http://www.dropbox.com, 2015.

[9] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM.

[10] GitHub. http://www.github.com, 2015.

[11] David Heinemeier Hansson. http://www.rubyonrails.org, 2015.

[12] IFPI. Ifpi digital music report 2015: Charting the path to sustainable growth, 2015.

[13] MongoDB Inc. http://www.mongodb.com, 2015.

[14] Stephen Jarvis. Cs261 software engineering - project management. https://www2.warwick.ac.uk/fac/sci/dcs/teaching/material/cs261/SE-L8.pdf, 2015.

[15] Simon Kemp. Digital, social & mobile, 2015.

[16] The Echo Nest. https://press.spotify.com/us/information/.

[17] Netflix. Q2 15 letter to shareholders. http://ir.netflix.com/common/download/download.cfm?companyid=NFLX&fileid=839404&filekey=C3CE9EE2-C8F3-40A1-AC9A-FFE0AFA20B21&filename=FINAL_Q2_15_Letter_to_Shareholders_With_Tables_.pdf, 2015.

[18] Digital Ocean. http://www.digitalocean.com.

[19] Python. http://www.python.org, 2015.

[20] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[21] Paul Rosania. While you were away... https://blog.twitter.com/2015/while-you-were-away-0, 2015.

[22] Hridya Sobhanam and AK Mariappan. Addressing cold start problem in recommender systems using association rules and clustering technique. In *Computer Communication and Informatics (ICCCI), 2013 International Conference on*, pages 1–5. IEEE, 2013.

[23] British Computing Society. Ethics of computing. chapter British Computer Society Code of Practice, pages 102–110. Chapman & Hall, Ltd., London, UK, UK, 1996.

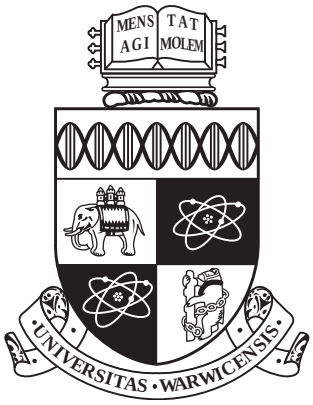
[24] Spotify. Information. https://press.spotify.com/us/information/, 2015.

[25] WordPress. http://www.wordpress.com, 2015.

[26] Jordan Wyatt. Cs310 blog. www.cs310blog.wordpress.com.

[27] Youtube. Changes to related and recommended videos. http://youtubecreator.blogspot.co.uk/2012/03/changes-to-related-and-recommended.html, 2012.

Progress Report



Crowd Sourced Content Aggregation and Suggestion

CS310 Computer Science Project

Progress Report

Jordan Wyatt

Supervisor: Dr. Matthew Leeke

Department of Computer Science
University of Warwick

2015-2016

Contents

List of Figures

This document provides details on the current state and progress made since submission of the project specification, which highlighted proposed work on a content aggregation and suggestion system named Revolvr. Throughout this report salient points regarding the project specification will be reiterated, and an overview of the current state of the system will be discussed including any change in the proposals given in the project specification.

1 Introduction

The modern world is vastly connected, with close to half of the population having access to the internet. As of January 2015 there are 3.010 billion active internet users, with 2.078 billion of these users having active social media accounts [27]. These figures are the product of a year-on-year growth, which saw a 21% increase in the number of active internet users, and a 12% increase in the number of active social media accounts in the space of 12 months. With such a rapidly evolving digital landscape also comes an influx in the quantity, accessibility and consumption of online media. However, with such easy access also comes the problem of over saturation. Given such vast quantities of data how can a user identify relevant and desirable content for their tastes?

1.1 Motivation

As discussed in Christopher Chamberlain's previous work on Revolvr, consumers of online media are concious of their *online footprint* [7]. With monumental developments in mobile technology in the past decade mobile and tablet devices now boast a 38% share of worldwide web traffic [27]. As a result, access to the web and its media is easier than ever before and the development of an online footprint is almost unavoidable. The sources of such content vary, from general purpose social networks such as Twitter and Facebook to format specific platforms such as Spotify and Netflix, a user has a wide range of options and discovering content they find relevant often feels like finding a needle in a haystack. Revolvr was developed as an attempt overcome this problem, with its documentation stating 'why settle for what you've already seen, when technology can steer you towards something you may love, perhaps before you know it?' [7].

However, as outlined in the existing project report for Revolvr, limited development time and ensuring the delivered product met the proposed requirements left the software more as a proof of concept, as opposed to an effective content aggregator. The existing work provides a good basis to allow not only improvements to the current algorithm and UI, but also research opportunities to evaluate variations of the implemented algorithm and explore areas of recommender systems, sentiment analysis, and machine learning.

1.2 Project Aims

The project aims remain unchanged, with 4 key objectives being the focus of development. However as time has passed the developer feels that the 2nd and 3rd aims listed below allow the most academic and technical discussion to be undertaken in the final report, and on reflection believes these can be considered the more substantial aims. The overarching aim of this project is to develop a content aggregation and recommendation system based on the existing Revolvr framework, with focus on media aggregation, analysis and exploitation of platform specific features, algorithmic development and analysis, and user interface design. These aims contain a mix of algorithmic research, scientific analysis, and development / design work which allow the project to have a good depth and breadth. Due to being continuation of work the overhead of setup is reduced allowing focused development points. The original project aims are shown in italics, with reconsiderations from the developers progress so far found beneath each respective aim.

1.2.1 Expanded Aggregation

"Although an instance of Revolvr has not been deployed on a server yet, inspection of the existing codebase and discussion with the project supervisor led the developer to believe that although Revolvr in its current form can connect to a number of services (Facebook, Twitter, YouTube, Vimeo, Soundcloud and Last.fm) it does not necessarily utilise these services to their full extent. Time will be devoted to ensuring the existing providers are suitable to use in the system, and in this case that the aggregation is effective and distributed between services. This aim will also consider any new services that surfaced since Revolvr's initial development, and aim to incorporate new providers where both possible and suitable."

The developers original suspicions about the selected aggregation sources and the extent to which they collect media were correct. In the systems original form Facebook and Twitter did not contribute a great deal to the system, and Last.fm is essentially obsolete compared to its competitors. Due to this Spotify has replaced Last.fm in the services the system uses, and throughout the project the aggregation will be updated where suitable to ensure a large amount of varied content can enter the system. Development of how Facebook and Twitter can be further exploited can be considered a sub-goal of 1.2.1, but due to the nature of these platforms (they are enablers of social media, not providers per say) work on this will be interleaved with goal 1.2.2.

1.2.2 Exploiting Higher Level Social Media Constructs

"The system in its current state only performs aggregation on heterogeneous media source, ignoring meta data and any service by service discrepancies such as followers on Twitter or likes on Soundcloud. Identifying these unique features of each media platform would provide a further link within the system to help develop their media footprint. Examples of metadata that can be used include a users liked Facebook pages, their Soundcloud followers, and who they follow on Twitter. Genre based analysis of songs and videos is also a topic that is open to exploration. This area of improvement hopes to build on the foundations Revolvr has established allowing tailored suggestion

for users, whilst at the same time taking an individual services features into account and abstracting these into generalised characteristics for aid in recommendation."

Development on this task has not begun yet, but as stated previously remains a key priority for the project.

1.2.3 Algorithmic Development

Revolvr currently only uses a single algorithm to make suggestions to users, for the project it is beneficial for the developer both for research and the interests of the product to investigate and implement other recommender algorithms. Allowing a user to choose between a number of different algorithms is likely to provide fresh and perhaps unexpected content to be recommended when their interest is fading. Research will be made into state of the art recommender systems and the approaches they take in their suggestions, and from this a number of bespoke algorithms will be developed for the new iteration of the system. Another area of aggregation that can be improved includes enhanced sentiment analysis. Sentiment analysis is a feature in Revolvr which reduces a piece of medias title to a more meaningful subset of phrases allowing for simple analysis [7, Ch. 5.3.1, p. 40]. However this analytical process does not consider a scenario in which a user shares a piece of media for negative reasons and therefore the associated phrases are negative. Development of this process will aim to analyse the mood of a sentence, identifying whether it is positive or negative. The developer also hopes to implement variations of the recommendation algorithm, the results of which will undergo comparative analysis.

See above.

1.2.4 Improved User Interface

"The original vision for Revolvr was to incorporate jQuery and AJAX to create dynamic pages without the need for a refresh to receive new content, improving the fluidity of the project. The name of the software also comes from the idea of an infinitely scrollable wheel of media which revolves [7, Ch. 10.2.4, p. 118]. Implementation of this original design along with the aforementioned algorithmic improvements will create a great deal more of interactivity and immersion. General improvements of the interface will also take place to ensure the application is up to date regarding UI design trends."

Via discussion with peers and the supervisor the developer has decided that the UI does not require as much work as first thought. Furthermore, revolving 3D / 'cover flow' style players do not follow current web design trends, and therefore an alternative generic way of displaying media will be developed, this is discussed further in the paper.

2 Research Direction

In the period of time since the initial specification the majority of development time has been focused on deploying the existing system, understanding its functionality, and working on the content aggregation. This part of the project requires little academic research and was tackled early to allow the developer to build familiarity with the system, but background reading into the implemented technologies and frameworks was vital. This section of the report reiterates previous research, research required for the setup period and development thus far, as well as discussing future areas to be studied in future that were discussed in the specification report.

2.1 Previous Research

Consumption of online multimedia is at an all time high, in 2014 the value of the digital music market increased 6.9% to $6.9 billion, representing 46% of global music sales and matching physical sales [21]. Streaming services such as Netflix have also overtaken live viewing this year according to Deloitte's 'Digital Democracy Survey', and content creators on YouTube are earning millions from advertising on their uploaded videos gathering view counts major TV networks dream of achieving [9] [4].

These services, their content providers, and users benefit from algorithms to suggest content to users based on what they like. A platform is only as good as the content it provides, and companies go to great lengths to ensure recommendation algorithms are as accurate as they can be. An example of this was the 2006 Netflix Prize, a *'machine learning and data mining competition for movie rating prediction'* [3]. Knowledge of real life application of recommender systems and the theory underlying it are essential for understanding the existing system and developing it further.

2.1.1 Recommender Systems

Recommender systems are defined in Resnick and Varian's seminal article as follows: *'In a typical recommender system people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the systems value lies in its ability to make good matches between the recommenders and those seeking recommendations.'* [33]. They have proven to be useful means for users to cope with overload of information and are used extensively in e-commerce. Recommender systems are able to be classified according to the knowledge sources they use, as seen in Figure 1. This taxonomy proposes three types of available knowledge to a recommender system: [13]

- **Collaborative (Social) based knowledge:** knowledge concerning other users.
- **User (Individual) based knowledge:** knowledge of the individual user
- **Content based knowledge:** Knowledge about the items being recommended

A recommender system may use a combination of these knowledge bases, or focus solely on a single one. This taxonomy also helps understand different recommendation techniques.

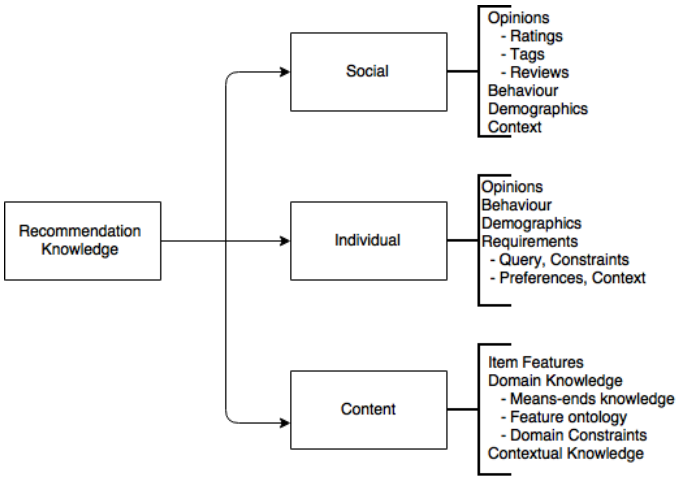


Figure 1: Taxonomy of knowledge sources in recommendation [13]

2.1.2 Recommendation Techniques

As defined in Felfernig and Burke's 2008 paper, a recommendation technique is *'a set of knowledge sources and an algorithmic approach to generating recommendations using those sources'* [13]. The most common recommendation techniques are collaborative recommendation, content-based recommendation and hybrid recommendation. Collaborative filtering is based on gathering information on a user's activities and preferences, and from this predicting what they may like based on similar users. This method benefits from not requiring prior knowledge on an item, so can generally be used without any prerequisite information. In contrast, content-based filtering uses a description of the item and a user's preferences and from this decides if the user will like the suggested item or not. In layman's terms collaborative filtering stems from the idea if two people agree they're likely to agree again in future, and content-based implies a user is likely to seek items similar to what they have liked before. Hybrid recommender systems combine collaborative filtering and content based filtering in a number of possible ways. One approach is to make predictions with each approach separately then combine the results, while in contrast you can design model which unifies the approach of both. Studies have suggested that combining collaborative and content-based filtering and be more effective in a recommender system. Netflix is a successful real world example of hybrid recommender system, collaborative filtering is implemented by comparing watching and searching habits of similar users and recommending content based on a user's previous ratings is content-based filtering.

The techniques discussed above are not exhaustive when talking about types of recommender systems, Robin Burke's 2002paper distinguishes the recommendation techniques shown in Figure 2 [6]. Here **I** is the set of items over which recommendations may be made, **U** is the set of users with known preferences, **u** is the user who requires recommendations, and **i** is some item for which we would like to predict **u**'s preference. Further into the project exploration of the more niche methods will be beneficial, but for specification purposes a high level understanding to plan the

Table I: Recommendation Techniques

Technique	Background	Input	Process
Collaborative	Ratings from **U** of items in **I**.	Ratings from **u** of items in **I**.	Identify users in **U** similar to **u**, and extrapolate from their ratings of **i**.
Content-based	Features of items in **I**	**u**'s ratings of items in **I**	Generate a classifier that fits **u**'s rating behavior and use it on **i**.
Demographic	Demographic information about **U** and their ratings of items in **I**.	Demographic information about **u**.	Identify users that are demographically similar to **u**, and extrapolate from their ratings of **i**.
Utility-based	Features of items in **I**.	A utility function over items in **I** that describes **u**'s preferences.	Apply the function to the items and determine **i**'s rank.
Knowledge-based	Features of items in **I**. Knowledge of how these items meet a user's needs.	A description of **u**'s needs or interests.	Infer a match between **i** and **u**'s need.

Figure 2: Recommender Techniques [6]

project is the key discussion.

2.1.3 Cold Start Problem

Recommendation systems often suffer from what is known as the *cold-start* problem. This is defined as when *'recommenders cannot draw inferences for users or items for which it does not have sufficient information'* [35]. This issue can effect both users and items. A new user will be given non personalised recommendations until they have rated enough items to build a profile on their tastes, and a new item in the system will not have any rating information and it is unlikely to be recommended a user. For a user the initial recommendations are crucial on the first impression the system leaves on them, and a poor performance could reduce the appeal of the system.

Two common ways to solve the cold start problem are *association rules* and *clustering techniques*, as discussed in Hridya Sobhanam's 2013 paper [35]. Association rules focus on expanding a user profile so it contains more ratings, and clustering groups of new and existing items based on similarity measures to make predictions for the item. Figure 3 shows one example of a flow of processes that can be performed to attempt to combat this problem. The intricacies of these processes are beyond the technical scope for this specification, but knowing about the problem and the solutions that exist allows peace of mind when designing aggregation systems. Revolvr currently aims to defeat this problem by simply aggregating the most popular content from its services and displaying them to users with no recommendations. Although this isn't fool proof it is sufficient until further work on the problem takes place. However, a problem does exist in that

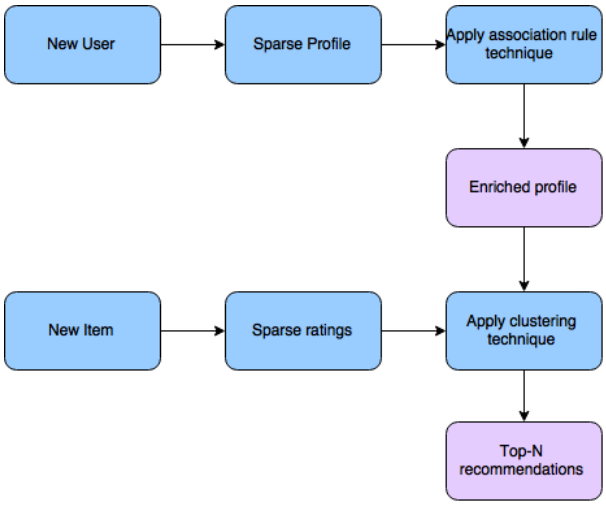


Figure 3: A possible cold-start problem solution [35]

if there is only a single user in the system no relationships can be made at all, and therefore no media is suggested to this user. This is to be fixed in future development.

2.2 New Research

To allow deployment of the existing codebase, a new web server capable of hosting a Ruby on Rails application was required. Furthermore, Revolvr utilises a number of concepts that were unfamiliar to the developer at the start of development. Some key research to allow work on the system to take place is outlined in the following subsections. More detailed research into the algorithmic section of the project is yet to occur, but remains a high priority for the developer.

2.2.1 Ruby and Ruby on Rails

Ruby is a programming language created by Yukihero Matsumoto that over the last decade has become one of the most popular programming languages of its time [1]. It is a general-purpose programming language but is mainly prominent in web programming, and features an important package manager known as RubyGems which provides external libraries also known as gems. Revolvr's front end processing is built using Ruby on Rails (RoR), a software library that extends the Rails language based on the Model View Controller (MVC) framework [19]. The full workings of RoR are beyond the scope of this report, but the key facts about are outlined below:

- Rails is a serverside framework for building websites
- Its conventions are focused on collaboration and maintenance.
- Rails combines HTML, CSS, and JavaScript to make web applications.

- A Rails app mainly consists of:
  - Model: A class in Ruby.
  - Views: What the user sees, displaying information given to them by a controller.
  - Controllers: A module that does the work of parsing requests, data submissions, and various browser related objects.

Familiarisation with this framework was essential to both understand the existing application and developing an improved version, both experimentation and online resources were used to understand Rails, as well as the languages and conceptual frameworks underpinning it. Useful papers include "Web-application development using the model/view/controller design pattern" [28] and a Ruby on Rails presentation by Craig, Mitchell, et al [8].

2.2.2 Sever Setup, Apache and Phusion Passenger

Although the developer had experience with web development, they had never been responsible for a full server side setup and deployment. The original project specification had proposed to use Nginx and Unicorn to host the application, however due to difficulties in setup the stack used in the original application was adopted. Key information on the deployment environment is outlined in Table 1. A number of educational / tutorial articles were accessed to assist with setup, for the server deployment a number of the community written DigitalOcean articles were extremely helpful [26]. Trial and error also complimented this, and experimenting with the server and the existing codebase allowed the developer to understand how each component of the system communicates.

2.2.3 OAuth and OmniAuth

Inspection of the systems previous documentation set and the initial codebase made it evident the system heavily relied on OAuth authentication to establish a user base and connection to external services. OAuth is an authorization framework that "enables a third-party application to obtain limited access to an HTTP service on a user by user basis, or as a client application as a whole" [20]. Essentially it allows authorisation of a user on one site (Revolvr) using authentication on another site. On initial setup of the existing system registration of both the production and development versions of the application on the 3rd party media providers developer sites was required. This provided a public and private client key used to identify the requesting application when connecting via OAuth.

Table 1: Server Specifications

Component	Capacity
OS	Ubuntu 14.04 LTS
Memory	512MB
Disk	20GB

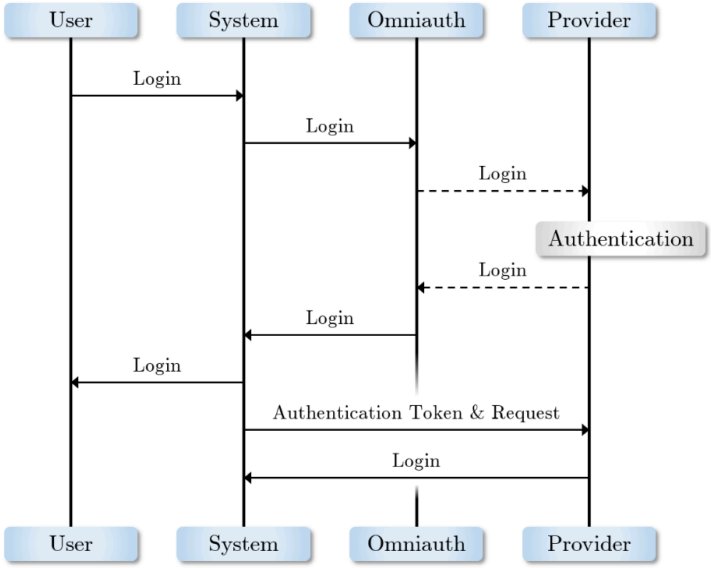


Figure 4: Omniauth Authentication Flow [7]

Omniauth is a library that 'standardizes multi-provider authentication for web applications', and makes use of 'strategies' per provider which contain functionality required to connect via the OAuth protocol [23]. Facebook, Twitter, and each of the aggregated media sources each have their own respective OmniAuth strategies for authentication via the OAuth protocol. All this functionality and the respective strategies are available in the aforementioned RubyGems package manager, allowing fast integration into the application. A simplified diagram in Figure 4 taken from the existing Revolvr documentation outlines this process.

2.2.4 Application keys and External API's

As discussed in the previous section regarding OAuth authentication connecting to each service provider makes use of a combination of the OmniAuth Ruby Gem and the OAuth authorisation protocol. As a result, complementing research into each respective OmniAuth strategy [22] and also referencing of the appropriate developer documentation packages was carried out [11] [41] [39] [18] [42] [36].

2.3 Future Research

The previously discussed research that has taken place since submission of the project specification has been largely focused on the understanding and deployment of the existing system, as opposed to the significant algorithmic changes proposed. A key task for the advancement of the project in December will be to look into the suggestion techniques covered in 2.1.2.

3 System Progress

Current progress on the system has been slower than expected but steady due to unforeseen challenges which will be discussed later in this report. Currently the existing system has been setup and analysed by the developer to allow an understanding of the processes and structures adopted. Furthermore, minor work on the user interface has been carried out as well as work on expanded aggregation to provide a richer data set to the system. One of the largest challenges overcome since the project specification was deploying the existing codebase, which proved to be more difficult than anticipated. This section of the report will describe the systems current state and the progress made so far.

The following key points will be described in more detail in the relevant sub sections, but act as a summary of the technical work completed up to this point in the project.

- Apache Server setup
- Application deployed on the server
- Deprecated API's accounted for, specifically YouTube API v2
- Last.FM removed from aggregation process, replaced with Spotify
- OAuth authorisation registration and implementation
- Richer dataset collection and standardising of storage
- Minor UI changes

3.1 System Deployment

Before the work described below could be completed, the system required an initial setup period. For the development purposes the local Ruby on Rails Thin web server was utilised, and for production an Apache & Phusion Passenger stack has been deployed for hosting the Ruby on Rails application. The domain name `http://revolvr.me` has been purchased to host the application on the web server described in Table 1, however the live version of the site is not representative of the development version. The research described in Section 2 of this report based on the MVC framework, web servers, OAuth authentication, and the workflow of a Ruby on Rails web application was all relevant and beneficial in this setup period.

3.2 Architectural Design

The architectural design of the system is currently identical to the initial implementation, using a three tier architecture with each layer providing its own role. It is believed that this structure will largely remain the same, just with a more complex data processing stage. The following description of the functionality of the system is a high level overview, and full explanations and algorithms will be provided in the final report. Data propagates from a Python based aggregator and

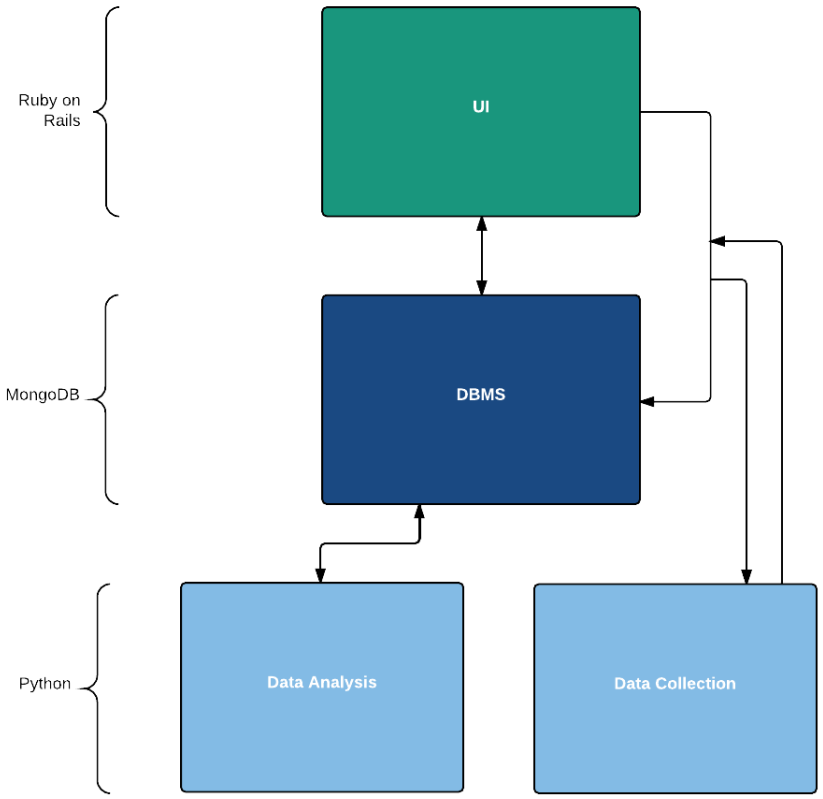


Figure 5: High level System Design.

processor, is stored in a MongoDB database, and finally output to the web application utilising the MVC framework Ruby on Rails. Figure 5 shows this process graphically. Data collection involves obtaining access to services for every user in the system, then retrieving information on media the user likes in the system and stores these items in the database, alongside a list of users who have also expressed a direct interest in this piece of media. Data processing currently runs a semantic analysis algorithm on the retrieved data, reducing media titles to more meaningful and concise forms known as tags for similarity identification. Similarity is calculated by identifying frequently coupled tags, which can be seen to imply increased likelihood of two items being related if they share common tags. Implicit links are formed between tags that are often associated with other tags, and direct links are formed when an identical piece of media is common between two users. Links act as a network of related users, which allows media to be recommended. These results are then passed to the UI and displayed to the users using the Ruby on Rails web interface. A full system diagram of the current current state of the system can be found in Figure 6.

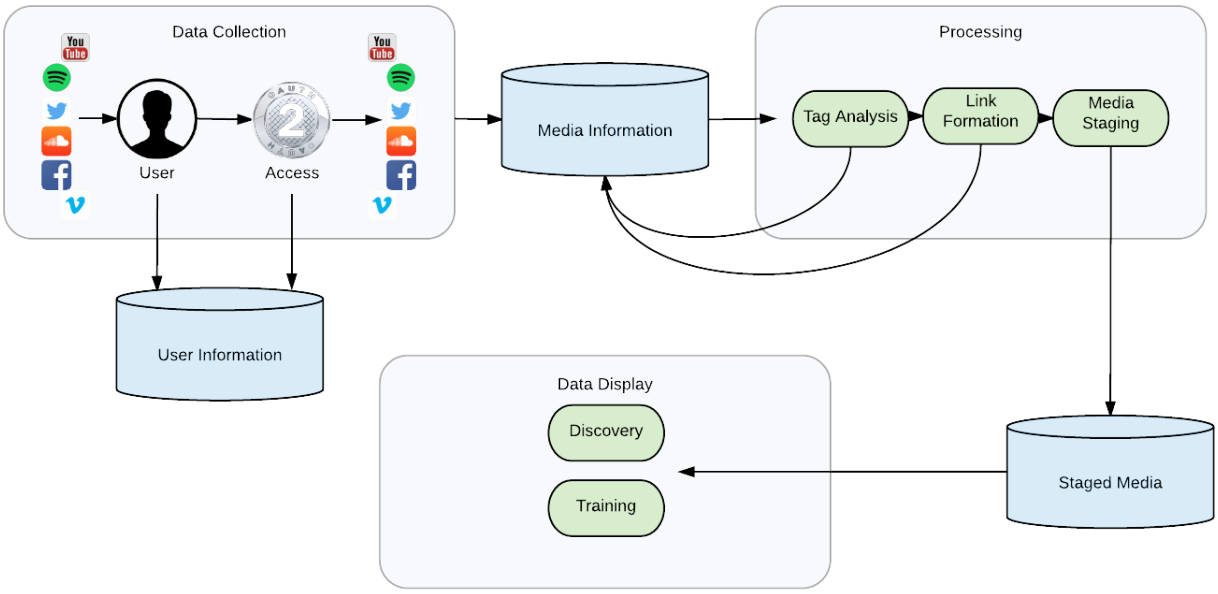


Figure 6: System Architecture.

3.3 Data Collection

Data collection is undoubtedly the most important part of a media aggregation and suggestion system, as a rich collection of content allows a varied source for recommendation. In the systems current state it is able to connect to Facebook, Twitter, Spotify, Soundcloud, Vimeo and YouTube. The developer has removed Last.fm in favour of Spotify, as Spotify has overtaken Last.fm in terms of mass popularity [38] and its API is more flexible. After the initial setup of the system issues were also discovered with changes in the 3rd party API's since Revolvr's initial inception, especially the deprecation of the XML based YouTube API v2 [17]. These errors and changes in the media providers were accounted for, and a revision of how each service retrieved its user media and top media was undertaken. User media refers to a users individual collection of media that is inserted into the system to build a profile, and top media is trending media on each provider. The current methods of data collection can be found in the Table 2.

Due to changes in the Soundcloud API the previous implementation for top tracks had stopped functioning correctly due to the most popular tracks no longer being in an explicit playlist. The new solution for this is likely to require HTML parsing, the developer was not aware of this during the time of specification and this flaw will have to be accounted for. It is also currently unclear what the original developers intentions for Twitter and Facebook were, and whether these services actually are used in aggregation or are just 'for show'. These overheads were not predicted for during specification, but are accounted for in the revised Gannt Chart in 5.2. The top Vimeo list was also reduced from 4 playlists (nature videos, HD music videos, travelling videos, and staff picks)

Table 2: Revolvr's Aggregation Sources

Service	Media Type	User Media Source	Top Media Source
Facebook	TBC	TBC	N/A
Twitter	TBC	TBC	N/A
Soundcloud	Audio	25 most recently 'favourite' songs	TBC
Spotify	Audio	25 most recently 'liked' songs	20 tracks from the official 'Today's top hits' playlist
Vimeo	Video	25 most recently 'liked' videos	20 videos from Staff Picks
YouTube	Video	25 most recently 'liked' videos	20 most popular videos of the day

to simply 'Staff Picks' as the developer felt the others were purely saturating the returned media list. The modifications to the data collection thus far and also the work to be completed (TBC in above table) summarises the goal of extended aggregation, with this subsection summarising the state of the system at this point in time.

3.3.1 Data Analysis

As previously mentioned aggregation and setup has been the main focus so far in the project, and due to this and other commitments the developer is yet to start work on improved data oriented algorithmic analysis. In its current state the system, as outlined in the existing documentation, performs simple semantic analysis on the stored media items from the collection stage. This processing decomposes media titles into a collection of tag-words and performs analysis to identify tags which frequently occur together, implying their respective media items may be similar. This, along with 'direct links' (items common between two users), builds up a related network from which suggestions are made. The developer was concerned that the systems abilities may have perhaps been less flexible than proposed by the previous developer, but as a result allows room for substantial and discussable improvement. Future work on this aims to build on research from Section 2 by implementing some of the discussed recommendation techniques in a number of algorithms, and although no specifics have yet been decided the developer believes this task is achievable.

3.3.2 User Interface

The user interface was one of the strongest elements of the work left for the current developer. Although only making use of an existing theme with a few modifications, the app looked clean and presentable which is what is needed for a proof of concept such has Revolvr. Establishing a strong UI leaves more time for impressive functionality to be developed, and therefore is not a focus of the project. The Bootstrap [5] framework also provides UI scalability for mobile applications, avoiding repetition of work where it is not needed. However, as discussed in the specification, Revolvr was named around the idea that content could be delivered fluidly and dynamically with the original vision of a revolving wheel providing content. Early implementations of this were tested in the last few weeks using a jQuery implementation known as Flipster, however this did not scale well for mobile devices, had issues with embedded media players, and also is not currently considered

'trendy' within current web design practises. As a result the design of a new, generic way to display content is being designed. The current proposal is to use Bootstraps carousel feature and placing the content and information about it within HTML div elements which can be moved through one by one. This will be developed alongside the other objectives as it is believed it will not be a significant task. The developer has also decided to slightly change the colour scheme as they felt the previous one was too bright. Screenshots of the current state of Revolvr on a non-mobile platform can be found below in Figure 7, and a draft of the proposed content delivery is shown in Figure 8

4 Development Technologies

Due to an implementation of Revolvr already existing time has largely been spent understanding development technologies as opposed to choosing them, as well as setting up development and production environments. The learning process and information about a number of the chosen technologies was discussed in Section 2.2 of this report, therefore this will be a brief overview.

4.1 Python

Python is a general purpose, high level language with a large focus on simplicity and readability [31]. The data processing and data collection elements of the system rely on Python, which makes fairly complex algorithms more about the process, as opposed to syntactical complexity. PyCharm by JetBrains [25] is being used as the development environment for this section of the code, as well as Sublime Text [34] when only a single file requires editing. Another benefit of using Python is that a large number of external libraries (known as modules) are available. Revolvr's current module usage and the role of each module can be found in Table 3. Notable modules include the requests package for complex HTTP requests [32], Pymongo for interacting with the MongoDB database [2], and API's for interacting with specific third party media providers [29] [12] [37].

Table 3: Notable Python Modules

facebook-sdk	Client library designed to support Facebook Graph API
nltk	Natural Language Toolkit, used for semantic analysis
oauthlib	A generic, spec-compliant, thorough implementation of the OAuth request signing logic
pymongo	Python driver for MongoDB
requests	Python HTTP
requests-oauthlib	OAuthlib authentication support for Requests.
simplejson	Simple, fast, extensible JSON encoder/decoder for Python
soundcloud	A friendly wrapper library for the Soundcloud API
spotipy	Simple client for the Spotify Web API

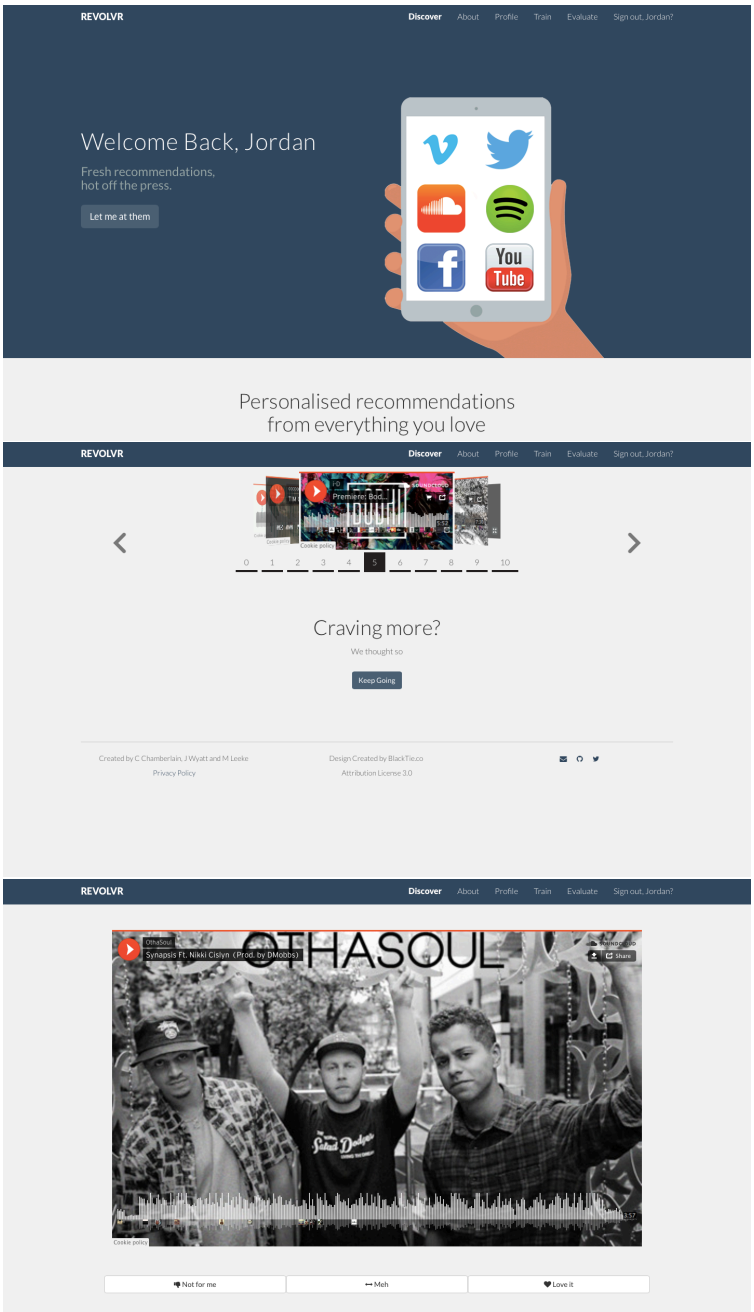


Figure 7: Current implemented UI. The revolving wheel in the second shot is to be swapped out for an alternative solution.

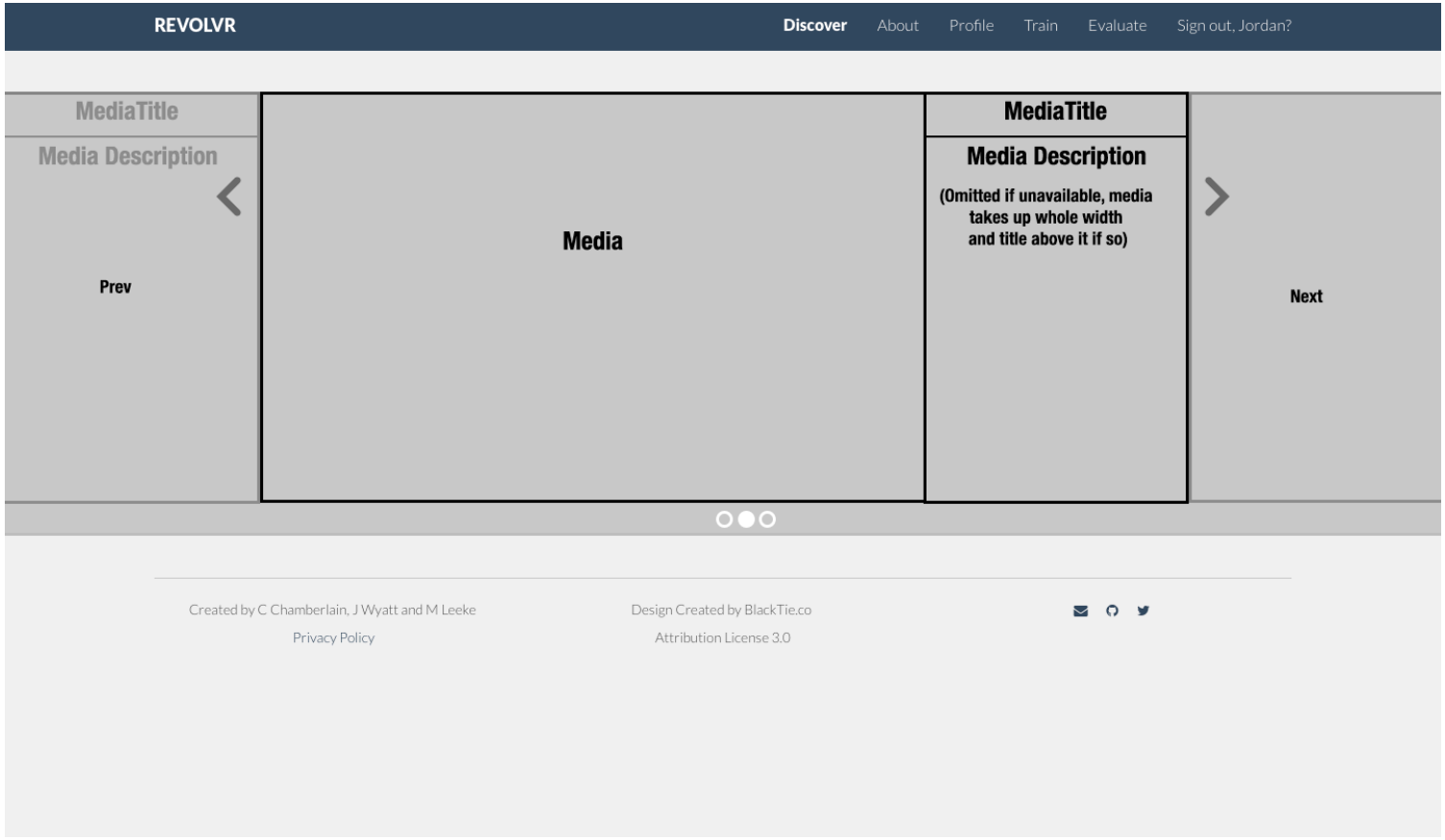


Figure 8: Proposed UI, based on Bootstrap carousel with generic embedded HTML divs

4.2 Ruby on Rails

See 2.2.1 for the main features of Ruby on Rails. Alongside these discussed core services the framework aims to provide there a number of useful specifics that assist development. Firstly, RoR provides a Thin web server which allows connections to be routed via a Virtual Machine for development purposes without the need to manage a production server. Rails uses a combination of HTML5, CSS and Javascript to provide rich web apps. Speciality features such ERb (a ruby markup language), Coffeescript (a JavaScript extension), and SASS (allows OOP features in CSS) technologies are also offered as part of the default install.

4.3 MongoDB

The previous developer decided to use MongoDB as opposed to a standard MySQL database. MongoDB is an alternative to the traditional relational database structure, and instead uses a document-oriented structure. It has been shown to be scalable and also more flexible that standard relational databases, and offers advanced data processing techniques such as MapReduce as standard [24]. Unlike a traditional MySQL database there are no strict rules or schemas for the data to adhere to, meaning data in a single collection can be more varied and have less constraints on it. Various models exist for various aspects of the system including users, tokens, and media items. The main benefit of continuing to use MongoDB is that the structure and existing code was already there, allowing focused development whilst avoiding further setup overheads. Full specifications of the MongoDB models are beyond the scope of this report but a substantial foundation has been left to work with.

4.3.1 Apache & Phusion Passenger

For a web application to function correctly it is essential to have a strong implementation of a HTTP web server. Apache is described by its creators as "a collaborative software development effort aimed at creating a robust, commerical-grade, featureful, and freely-available source code implementation of a HTTP (web) server" [14]. The developers knowledge of web servers is not extensive enough to comment on the pros and cons of each possible choice, but Apache was chosen because for two reasons. Firstly, there is a vast amount of existing documentation making troubleshooting issues faster, allowing more time for meaningful work. Secondly, the original developer had used an Apache and Phusion combination for hosting the application and the developer believed it would be safer and quicker to use the same stack as opposed to migrating to another with minor, subjective benefits. However, Apache cannot run Ruby like it does PHP, Python etc as there is no built-in Ruby module that works as well as it needs to. Due to this, Phusion Passenger is utilised in the production version, as on the local version testing and deployment is done via the Thin web server using the `rails s` command, launching a local VM to host the application. Passenger is a system that allows the preparation and deployment of Ruby for use with Rack-based applications such as Ruby on Rails [40] [30]. By using these technologies together it allows a live instance of Revolvr to be hosted on a reliable, robust web server.

5 Project Management

With such an agile, multi-focus project it remains important to make progress with flexbility terms of planning and design decisions, whilst still ensuring time is being effectively used. This section of the progress report reflects on the project management so far with regards to the progress discussed in Section 3, making adjustments to the project schedule where required.

5.1 Software Development Methodology

The start of this project proposed a highly agile workflow, devoting time and work on the four primary objectives defined in Section 1 of both this paper and the specification. Although the project has only really just began to make progress on these proposed points it has become apparent to the developer that despite an agile methodology it is still important to maintain weekly targets and goals. By setting these short lived goals it allows a constant drive and focus to be placed on the development, avoiding procrastination and making a large project manageable by partitioning work into smaller chunks via micromanagement.

5.2 Project Timeline

The progression of the system so far has been largely in line with the original timeline in terms of time frame, but the division of work has overlapped. During setup period a lot of fixes were made to accommodate for changes in the implemented API's as well as time spent troubleshooting initial issues. Currently we are in the third stage of development focused on expanding the aggregation of the system. Work on this has so far been successful, as discussed in Section 3 of this report.

One concern that the developer has discussed with the project supervisor is that too much time has been dedicated to UI improvements, due to the current state of the system already being quite impressive as mentioned in Section 3.3.2. Leading on from this the algorithmic work and exploitation of service by services features is beginning to look like a larger task than first anticipated. A revised Gannt chart is shown in Figure 10 as a result of these observations in which 2 weeks has been taken from UI improvements and distributed between the two algorithmic focused periods, as well as a table of these deadlines in Figure 5. Overall development is moving at a decent pace, with significant progress expected to be made over the christmas period.

5.3 Tools and Management Techniques

An agile approach as previously mentioned still requires monitoring and progression. A number of management techniques and tools have been applied to the project and are described in the section below, as well as their contribution to the project management.

5.3.1 Version Control

In software engineering version management and backup is essential, being able to roll back changes and compare progress during development is extremely important. Git is a free and open source

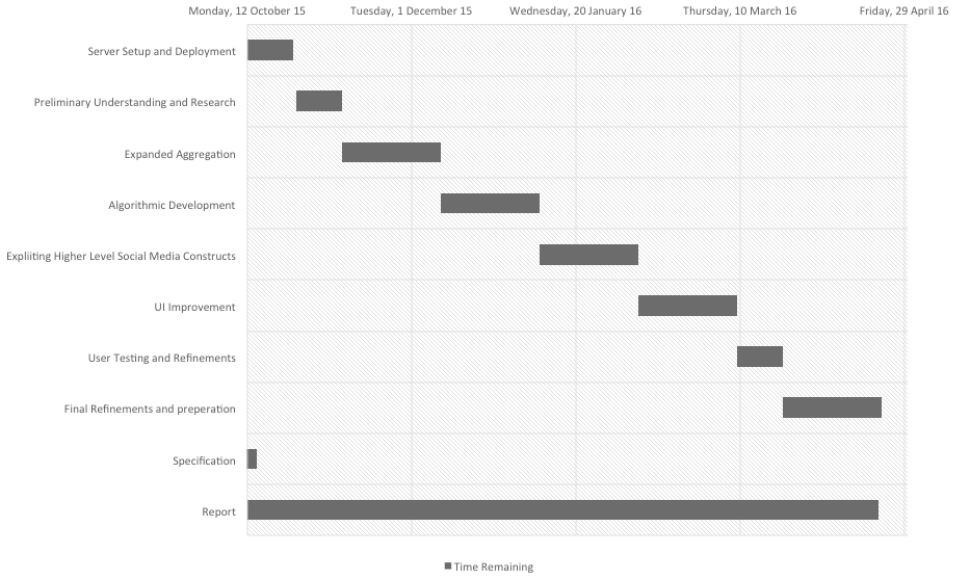


Figure 9: Gannt chart showing originally proposed project deadlines

distributed version control system used to manage source code history, and GitHub is an online repository used to host Git repositories [16] [15]. Both of these features promote code safety and also allow code based progress to be tracked easily. GitHub also has the benefit of allowing the code to be accessed and synchronised between multiple machines, in this case the local development environment and then the production server. The repository for the project can be found at `http://www.github.com/jordwyatt/revolvr`, please note that the most up to date developments are tested and written on the local environment before pushing and therefore this is not representative of the systems current state.

5.3.2 Document Sharing, Progress Monitoring, Communication

Monitoring of the developer's progress for their own sake, as well as sharing and updating the project supervisor is crucial in ensuring the pace and aims of the project remain on track. Dropbox, an online synchronous file sharing service, allows documentation associated with the project to be shared between both parties [10]. Outlook is being used for formal communications with the supervisor, and for less formal queries Facebook's chat option is being utilised to allow fast, mobile communication to take place so development is not stalled waiting for replies from either party. An informal blog has also been set up by the developer which is updated periodically to aid as a reminder of the current goals and progress that has taken place between posts. This will be useful towards the end of the project when early details may be hard to recall for documentation purposes, and can be found at `https://cs310blog.wordpress.com`.

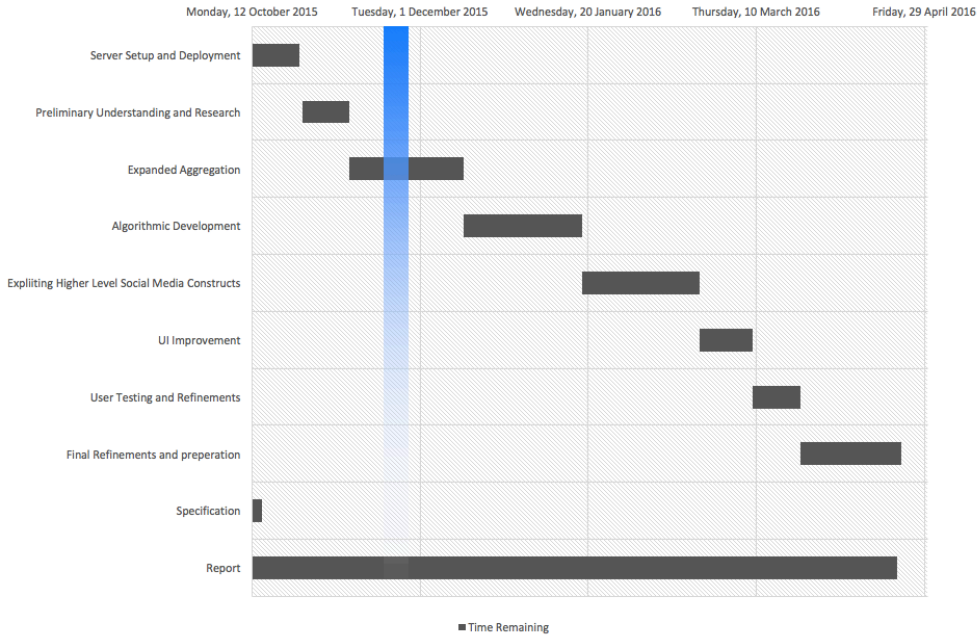


Figure 10: Revised Gannt Chart showing project deadlines

6 Further Work

Despite progress being made, a significant and arguably more important development period is yet to occur. Over the next few months the focus of the developer will be on:

- Development of multiple suggestion algorithms, incorporating and implementing the research into recommender systems previously mentioned in Section 2.1.3. The user will have the ability to choose different recommendation models.
- Exploiting service specific features and higher level social media constructs. For example, details such as Twitter followers, song genres, and video categories are all examples of features the developer hopes the systems will take account of to provide more personalised recommendations.
- Streamline the learning mechanisms (Evaluate and Training) to a single, centralised format.
- Implement AJAX to allow a dynamic experience without the need for page refreshes.
- Modify the UI so the focus on the system is mainly consuming content, reducing the number of pages to a minimum to further highlight the idea of simplicity and promote discovery.
- Development of a fluid and seamless way to present media to the user.

Table 4: Originally Proposed Deadlines

Process	**Proposed Start**	**Proposed End**
Server Setup and Deployment	12 October 2015	27 October 2015
Preliminary Understanding Research	27 October 2015	10 November 2015
Expanded Aggregation	10 November 2015	10 December 2015
Algorithmic Development	10 December 2015	9 January 2016
Exploiting Higher Level Social Media Constructs	9 January 2016	8 February 2016
UI Improvement	8 February 2016	9 March 2016
User Testing and Refinements	9 March 2016	23 March 2016
Final Refinements and Preperation	23 March 2016	22 April 2016
Specification	7 October 2015	15 October 2015
Report	12 October 2015	30 April 2016

Table 5: Revised Deadlines

Process	**Proposed Start**	**Proposed End**
Server Setup and Deployment	12 October 2015	27 October 2015
Preliminary Understanding Research	27 October 2015	10 November 2015
Expanded Aggregation	10 November 2015	14 December 2015
Algorithmic Development	14 December 2015	18 January 2016
Exploiting Higher Level Social Media Constructs	18 January 2016	22 February 2016
UI Improvement	22 February 2016	9 March 2016
User Testing and Refinements	9 March 2016	23 March 2016
Final Refinements and Preperation	23 March 2016	22 April 2016
Specification	7 October 2015	15 October 2015
Report	12 October 2015	30 April 2016

Focus on these development points, as well as monitoring and documenting development will be key to the success of this project. Due to the agile nature of the development process it is possible these remaining objectives may change as the project progresses.

7 Conclusion

To conclude this report, progress on the system has been steady and largely in line with the original plan outlined in the specification. Although the largest challenges are yet to be faced, a solid grounding in the existing environment and inner workings of the system have been beneficial. The system is on track to allow suggestion of aggregated content using a number of algorithms, and exploiting non-generic features between services aims to make these suggestions as accurate and as beneficial as possible. Finally, the developer is confident that as long as progress is made and targets are met, Revolvr will be a well rounded system with the ability to display technical, commerical, and academic strengths.

References

[1] https://www.ruby-lang.org/en/about/, 2015.

[2] Pymongo. https://api.mongodb.org/python/current/, 2015.

[3] Xavier Amatriain and Justin Basilico. Netflix recommendations: Beyond the 5 stars (part 1). http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html, 2012.

[4] Damon Beres. Youtube stars' huge earnings will make you question your life choices. http://www.huffingtonpost.com/2015/02/05/youtube-stars-money_n_6549906.html, 2015.

[5] Bootstrap. http://www.getbootstrap.com, 2015.

[6] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[7] Chris Chamberlain. Revolvr: A crowd-sourced media content aggregation and suggestion system, 2014.

[8] Mitchell Craig, Tam Nguyen, Becker Luu, Eliezer Mar Manarang, and Colin Williams. Ruby on rails. 2006.

[9] Deloitte. Digital democracy survey: A multi-generational view of consumer technology, media and telecom trends, 2015.

[10] Dropbox. http://www.dropbox.com, 2015.

[11] Facebook. Facebook developer documentation. https://developers.facebook.com/docs, 2015.

[12] Facebook. Facebook sdk. https://github.com/pythonforfacebook/facebook-sdk, 2015.

[13] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM.

[14] The Apache Software Foundation. Apache http server project, 2015.

[15] Git. git. https://git-scm.com, 2015.

[16] GitHub. http://www.github.com, 2015.

[17] Google. https://developers.google.com/youtube/2.0/developers_guide_protocol?hl=en, 2015.

[18] Google. Youtube developer documentation. https://developers.google.com/youtube/, 2015.

[19] David Heinemeier Hansson. http://www.rubyonrails.org, 2015.

[20] Dick Hardt. The oauth 2.0 authorization framework. 2012.

[21] IFPI. Ifpi digital music report 2015: Charting the path to sustainable growth, 2015.

[22] Intridea Inc. https://github.com/intridea/omniauth/wiki/List-of-Strategies, November 2015.

[23] Intridea Inc. Omniauth. https://github.com/intridea/omniauth/wiki, 2015.

[24] MongoDB Inc. http://www.mongodb.com, 2015.

[25] Inc JetBrains. Pycharm. https://www.jetbrains.com/pycharm/, 2015.

[26] jkostolansky Juraj Kostolanský. https://www.digitalocean.com/community/tutorials/how-to-deploy-a-rails-app-with-passenger-and-apache-on-ubuntu-14-04, November 2014.

[27] Simon Kemp. Digital, social & mobile, 2015.

[28] Avraham Leff and James T Rayfield. Web-application development using the model/view/-controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[29] Paul Lemere. Spotipy. https://github.com/plamere/spotipy, 2015.

[30] Phusion. Phusion passenger. https://www.phusionpassenger.com, 2015.

[31] Python. http://www.python.org, 2015.

[32] Kenneth Reitz. Requests. http://docs.python-requests.org/en/latest/, 2015.

[33] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[34] Jon Skinner. Sublime text. http://www.sublimetext.com, 2015.

[35] Hridya Sobhanam and AK Mariappan. Addressing cold start problem in recommender systems using association rules and clustering technique. In *Computer Communication and Informatics (ICCCI), 2013 International Conference on*, pages 1–5. IEEE, 2013.

[36] SoundCloud. Soundcloud developer documentation. https://developers.soundcloud.com, 2015.

[37] SoundCloud. Soundcloud-python. https://github.com/soundcloud/soundcloud-python, 2015.

[38] Spotify. Information. https://press.spotify.com/us/information/, 2015.

[39] Spotify. Spotify developer documentation. https://developer.spotify.com, 2015.

[40] tadman. Phusion passenger (for dummies!). http://stackoverflow.com/questions/6155399/phusion-passenger-for-dummies, May 2011.

[41] Twitter. Twitter developer documentation. https://dev.twitter.com/overview/documentation, 2015.

[42] Vimeo. Vimeo developer documentation. https://developer.vimeo.com, 2015.

Quick-Start Guide

This section gives a brief guideline on how to setup the system using the provided source code. A functional production server can be found at `http://revolvr.me/`.

Prerequisite Software

A number of prerequisite installations are required before being able to run the system. These are mainly the core technologies as well as additional packages used throughout the system as shown in Tables 1 and 2. This list may not be exhaustive due to environment specific dependencies, but provides the most vital packages for operation. If any packages required are not present the system will throw appropriate errors indicating which packages are required. Rails Gems are explicitly defined in the `Gemfile` and therefore do not require any downloads.

Software	Supported Version
Ruby on Rails	4.2.4
Ruby	2.2.1
Python	2.7.6
MongoDB	2.4.9

Table 1: Required Software for Deployment

Server Installation

To setup a Rails app on a server the following guide was used: `https://www.digitalocean.com/community/tutorials/how-to-deploy-a-rails-app-with-passenger-and-apache-on-ubuntu-14-04` which highlights the deployments of Rails apps on an Ubuntu environment

Python Module	Version
beautifulsoup4	4.4.1
facebook-sdk	0.4.0
nltk	3.1.0
numpy	1.10.4
oauthlib	0.6.0
pandas	0.17.1
pymongo	3.1.0
PyVimeo	0.3.2
requests	2.9.1
requests-oauthlib	0.5.0
scipy	0.13.3
simplejson	3.3.2
soundcloud	0.5.0
spotipy	2.3.7
tweepy	3.5.0
urllib3	1.7.1

Table 2: Required Python Modules

and provides a comprehensive overview. The following steps must be followed once the Rails environment is deployed:

- The database setting must be entered in `revolvr/Interface/config/database.yml`, pointing to the host instance of the MongoDB server.
- For third-party authentication API keys are required by registering a developer application via the appropriate third-party developer sites. These will provide public and secret keys required to authenticate Revolvr as a registered application. These keys are placed in `rails/Interface/config/initializers/omniauth.rb` as per instructions on each strategies documentation.
- The Ruby on Rails gems can be installed using the `bundle install` command, and the application is launched on local instanced using `rails s`. Launching the application on the production environment required Phusion Passenger to host the Rails app, and details can be found in the hyperlink given above.

Processing Initialisation

To allow the python processes to connect to user accounts the keys defined in `omniauth.rb` must also be provided via a number of hidden files. These files contain sensitive details of authentication codes for the application which can be acquired by the respective developer sites for each

service. These files are included with appropriate scripts and exclusion of them will result in incorrect functionality. CRON jobs must also be setup server side for both `full_process.py` and `stage_process.py`.

Hidden File	**Format**
twitter_auth.py	client_public_token, client_secret_token
mongo_env.py	current_env='prod' / current_env='development'
soundcloud_auth.py	client_public_token, client_secret_token
spotify_auth.py	client_public_token, client_secret_token
youtube_auth.py	client_public_token, client_secret_token, api_key
vimeo_auth.py	client_public_token, client_secret_token

Table 3: Necessary files for Processing