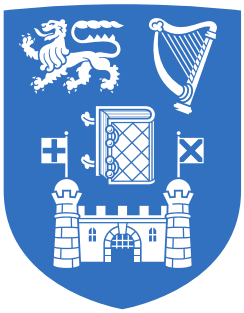
University of Dublin



TRINITY COLLEGE

Privacy Enhanced Recommendations Using Group Clustering Techniques

Yasir Mohamed

Master in Computer Science

Masters Dissertation Project May 2018

Supervisor: Dr. Douglas Leith

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Yasir Mohamed

Date

Permission to lend

I agree that the Library and other agents of the College may lend or copy this report upon request.

Yasir Mohamed	Date

Acknowledgements

First and foremost, my eternal gratitude to my parents Amira & Ahmed who pushed me to pursue this masters, and indeed for their overwhelming support in my education these past 5 years.

My sincere gratitude to my supervisor, Dr. Douglas Leith, for his enthusiasm, support, and guidance over the course of this dissertation.

Lastly, thank you to my sister Maye, my brothers Mohamed and Mazin, and my closest friends who each supported me in their own way.

Summary

Online privacy has become an increasingly important topic of discussion in recent years in light of leaks by Edward Snowden and privacy scandals such as that of Facebook and Cambridge Analytica. Websites are collecting user data to provide content personalisation to users, but this data collection comes at the cost of individual user privacy. Users who try to circumvent this data collection by deleting their browser cookies end up with a severely degraded online experience as content is no longer tailored to their interests.

OpenNym provides a middle-ground to this all or nothing approach, where users can browse as part of a group with similar interests. Using the BLC Clustering algorithm (an unsupervised machine learning algorithm), we cluster users with similar interests into groups called Nyms, then build item ratings for the Nyms based on their user's ratings. Then we propose a system to allow users to browse as part of a group composed of a browser extension for Firefox and a web server. The browser extension is responsible for substituting a user's cookies for a selected Nym's cookies, and intercepting all individual user rating requests and substituting them with a group's average rating. This allows the user to appear as part of a Nym to websites that track users. Additionally, by sending a group average rating to the website, the group's interest is effectively represented to the website without disclosing any personal rating information to the website. The web server is responsible for storing all information pertaining to Nyms, their ratings, and a wealth of metadata required for the browser extension to function. Additionally, the web server has a responsibility to be as fast as possible so the impact of the browser extension to a browser's performance is minimal.

To demonstrate the functionality of the OpenNym system, we cluster 40,000 users from a publicly available music dataset called the Million Song Dataset. This resulted in 14 Nyms which were then loaded into the OpenNym web server. The resulting OpenNym system was successful in allowing users to browse as part of a Nym and intercepting user ratings for YouTube.com. To evaluate the performance of the OpenNym system, we evaluated the accuracy of Nym recommendations, the browser extension, and the web server separately.

To evaluate the performance of each Nym, we measured the accuracy of recommendations

from Spotify for Nyms and generate wealth F-measure. This was to demonstrate that users browsing as part of a Nym do not experience a loss in content personalisation when browsing online. The ground truth for this evaluation was a sample of users taken from each Nym and subjected to the same evaluation process as each Nym. Results showed that of 14 Nyms, 10 scored higher than the average F-measure of it's users. 3 of the 4 Nyms that scored lower than it's users only scored marginally lower, within a ≈ 0.05 range. The Nym that scored the worst performed exceedingly poorly due to only 12 users being assigned to the Nym.

To evaluate the performance of the browser extension, we measure the average page load times of Spotify when substituting user cookies for Nym cookies, and the average increase in rating request latency for videos on YouTube. Results showed that page load times on Spotify only increased by $\approx 8\%$ over browsing without OpenNym in the best case scenario. There was less than a 1% increase in rating request latency for videos on YouTube in the best case scenario.

To evaluate the performance of the web server, we measure request latency to the server for a number of concurrent users and the overall server throughput. Results showed most of the APIs on the server performed exceedingly well, with some APIs when serving 1,000 concurrent users attaining a mean request latency below 400 milliseconds and throughput of over 600 requests per second on a relatively low powered server.

In conclusion, the proposed OpenNym system is effective in affording users a privacy enhanced browsing experience by allowing them to browse anonymously as part of a larger group, while retaining the ability to meaningfully rate items and receive a largely personalised experience.

Abstract

OpenNym: Privacy Enhanced Recommendations Using Group Clustering Techniques

Yasir Mohamed

This project is an investigation into the feasiblility of a system that affords users privacy enhanced browsing. The system, known as OpenNym, would allow users to 'hide in a crowd' while browsing online such that they can subvert individual user tracking by websites while still receiving a personalized online experience.

The premise for this project is that users with similar interests can be grouped together such that recommendations generated for the group are still relevant to the group's users. To achieve this, an unsupervised machine learning algorithm is used to cluster the users. A system for allowing users to browse as a group is also implemented using a browser extension and an accompanying OpenNym server.

The effectiveness of this proposed system is demonstrated by clustering users in a publicly available music dataset and showing recommendations from Spotify for the resulting groups are still accurate.

Contents

List of Tables

List of Figures

Abbreviations

Acronym	Meaning
API	Application Programming Interface
CA	Certificate Authority
HTML	Hypertext Markup Language
HTTPS	Hypertext Transport Protocol Secure
IMDB	Internet Movie Database
JSON	JavaScript Object Notation
KB	Kilobytes
MSD	Million Song Dataset
MVC	Model-View-Controller
NO-SQL	Not Only SQL
RAM	Random Access Memory
RMSE	Root Mean Square Error
RPS	Requests Per Second
SQL	Structured Query Language
SSO	Single Sign On
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Chapter 1

Introduction

This dissertation is an investigation into whether a system can be developed such that users can browse websites online as part of a group of users with similar interests. This is an extension of the proof of concept demonstrated by Checco et al. [14] who showed users of the MovieLens service could effectively be clustered into groups with similar interests, and then access MovieLens as part of a group without experiencing a loss in content recommendation accuracy. Group based access to service recommender systems provides individual users of the group an enhanced level of privacy by allowing them to 'hide in the crowd' while using the service.

Before discussing an approach to attaining an enhanced level of privacy, we must first define privacy. Indeed privacy is a context-sensitive concept, but at it's most fundamental level it is an individual's right to be free from interference or intrusion [53]. Privacy is a broad, open concept, but as this dissertation is concerned with an individual's privacy while browsing online, all references to privacy henceforth will refer to *Information Privacy*. *Information Privacy* refers to the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others [57].

This dissertation will examine content personalisation techniques online and the inherent privacy concerns of the collection of user information for use in content recommendation. We will then propose a novel design for a functioning system that will allow individuals to browse online as part of a group in such a manner that they still receive personalised content while also achieving an enhanced level of privacy.

In this chapter, the motivation behind the research is introduced, a brief overview of the Research Background is given, the Research Question is stated, and Research Objectives for the dissertation are outlined. Finally, the structure of the document is outlined in the last section of this chapter.

1.1 Motivation

Content personalisation has become an important aspect of user experience when browsing online. Users have come to expect a personalised experience when using a service, where a personalised experience is one where the service recommends content that is tailored to the user's personal interests. Content personalisation plays such an important role in user experience that companies such as Netflix [37] have come to rely on it heavily to ensure users remain interested in content served to them and return to their service.

Online content personalisation is powered by recommender systems. Recommender systems are sophisticated software tools developed to provide suggestions for items that would be of use to a user [47]. Recommender systems are required as often there are more items available than can be displayed to a user at once, and so they allow a service to show items to a user that are deemed to be of the most interest to the user. Recommender systems operate on information known about user interests and the set of items available to be recommended.

Recommender system accuracy improves when more user preference information is made available to it. For this reason it is important for web services to be able to collect user information to better understand their interests and be able to make better predictions. However, the gathering of user interest information for use by these web services raises some inherent privacy concerns. The collection of personal information by web services for use in recommender systems increases the risk of exposure of this information to third-parties [49], either willingly from the web service or forcibly obtained by a malicious third-party. Calandrino et al. were also able to develop a series of passive inference attacks to demonstrate that information about individual users is leaked by collaborative filtering recommender systems [12].

Recommender systems rely on user feedback to improve recommendation accuracy, where feedback refers to a measure of how positively or negatively a user has reacted to an item. Given user feedback for a set of items, a recommender system would be able to infer the interests of that user. Feedback mechanisms are methods incorporated by web services to gather user feedback on items consumed by the user. There are currently a number of feedback mechanisms in use by web services that are effective in gathering user information to improve recommender performance, but they also introduce a number of privacy concerns. One of the major concerns for privacy is the lack of user control over what data is collected, when it is collected, and what is done with the data. These concerns are discussed in greater detail in Section 2.2.

1.2 Research Background

The research conducted in this project will build off the proof of concept demonstrated by Checco et al. [14] in their OpenNym project. The basis of OpenNym is that users with similar interests can effectively be clustered into groups (henceforth referred to as *Nyms*) based on their ratings for items. Users can then browse as part of these Nyms and still receive personalised recommendations while achieving enhanced privacy through 'hiding in a crowd'. This proof of concept was carried out by clustering users in the MovieLens dataset [22] and implementing a prototype browser extension to intercept and replace cookies sent to the MovieLens service. By intercepting the session cookies sent to the MovieLens service, users of the browser extension were able to browse as part of a Nym without the need to authenticate themselves. Results of this work showed that users could browse as part of a Nym and still receive a high level of personalisation in recommendations from the MovieLens service.

1.3 Research Question

With the above research motivation and background in mind, the research question for this dissertation is:

> "In the context of the work performed by Checco et al. [14], can a robust, scalable system be implemented such that users with largely similar interests can browse as part of a group and still receive a largely personalised service online while subverting individual user tracking?"

1.4 Research Objectives

In order to achieve the stated research question, the following objectives must be fulfilled:

1. A browser extension capable of assigning users to Nyms based on their interests and allowing users to browse as part of a Nym for multiple recommender systems must be developed.

2. A web server that maintains a list of Nyms, their item ratings, and associated metadata must be developed such that the browser extension can use it to facilitate users browsing as part of a Nym.

3. The working system must be demonstrated by clustering users in a publicly available

dataset and using the top rated items of resulting Nyms to seed an appropriate recommender system.

- Performance metrics measuring the effectiveness of the clustering of users should also be generated using an appropriate recommender system.

1.5 Dissertation Outline

The remainder of this document is structured as follows:

- **Chapter 2** is a discussion of the current State of the Art of online content personalisation, user data collection, and existing privacy enhanced recommendation approaches.
- **Chapter 3** details the system architecture and design proposed to achieve the outlined research objectives, and contains an in depth review of all design choices made.
- **Chapter 4** details the implementation of the system design proposed in Chapter 3 and will contain a review of the technologies used in the implementation.
- **Chapter 5** details the process of clustering users in a publicly available dataset and generating ratings for each of the resulting Nyms.
- **Chapter 6** evaluates the performance of the implemented system and the accuracy of the generated Nyms.
- **Chapter 7** discusses the results obtained from Chapter 6, how they relate to the research objectives, and draws conclusions based on these results. Additionally, Chapter 7 discusses research limitations, and outlines possible future work.

Chapter 2

State of the Art

This chapter will discuss current methods of content personalisation through recommender systems and user data collection, and the privacy concerns they raise. This chapter will also discuss previous attempts at improving privacy in recommender systems.

2.1 Recommender Systems

Recommender systems are the driving force behind content personalisation. Recommender systems operate on existing knowledge of users and content items to recommend items that are deemed to be of the most interest to users. Two of the most popular approaches to developing recommender systems for content personalisation are:

- Content Based Recommender Systems
- Collaborative Filtering Recommender Systems

Both Collaborative Filtering and Content Based Recommendations make extensive use of user information. In this section we will examine how recommender systems work, how user information to seed recommender systems is obtained, and their associated privacy concerns.

2.1.1 Content Based Recommendations

A Content Based recommendation system is a system that recommends items to users based upon a description of the item and a profile of the user's interests [41]. Content based recommender systems analyze the features of available items and identify items that would be of particular interest to the user, based on predefined user interests. Each user for which recommendations are generated for is examined independently of other users.

When generating recommendations for a user, the user profile with predefined interests is examined and compared with the set of available items to be recommended to the user.

The items which align the closest with the user's interests are recommended. Table 2.1 shows how users are represented in a content based recommender system for a music service. Note how each user has a set of scores for the predefined metrics valence and arousal. These metrics were obtained from the valence-arousal interface to measure perceived emotion of music developed by Eerola et. al [18]. Valence is a measure of how positive or negative a song is, and arousal is a measure of how exciting or calm a song is. Each item in the set of available songs also has a score for the same metrics, as each score defines the song's properties. This can be seen in table 2.2. The properties used to describe items are usually domain specific, and would be expected to be different in a non-music recommender system.

User	Username	Valence	Arousal
00	Alice	0.4	0.7
01	Bob	0.7	0.8
02	Collin	0.3	0.5
03	Dean	0.9	0.4

Table 2.1: Content Based Recommender System Users for a Music Service

Item	Song Name	Artist	Valence	Arousal
00	King Kunta	Kendrick Lamar	0.5	0.3
01	Black Skinhead	Kanye West	0.8	0.5
02	Hot Thoughts	Spoon	0.6	0.4
03	Rosa Parks	OutKast	0.5	0.4

Table 2.2: Content Based Recommender System Items for a Music Service

Given knowledge of both a users interests and details of the set of available items, items can then be recommended to the users based on their interests. A popular method of evaluating which items to recommend is to map the user interests and item properties in vector-space and use the Euclidean Distance formula [41, 43].

2.1.2 Collaborative Filtering

Collaborative filtering recommender systems try to predict the utility of items for a particular user based on items previously rated by other users [3]. Collaborative filtering relies on the underlying assumption that users with similar tastes will rate items similarly. It is a popular method of rating prediction, and is used by large organisations including GroupLens [26] and Netflix [7]. Data used in a collaborative filtering algorithm can be represented as a user-item matrix as seen in table 2.3.

	King Kunta	Black Skinhead	Hot Thoughts	Rosa Parks
Alice	4	3	5	3
Bob	2	5	1	4
Colin	3	4	2	4
Dean	5	3	4	3

Table 2.3: User-Item matrix, where each Item is a song rated between 1 and 5

Early collaborative filtering algorithms used association inferences between similar users to generate recommendations. Association inferences would calculate the rating prediction for a new item i by user u by examining the ratings given to i by users similar to u. The issue with association inferences, however, was that they had quite a high time complexity and didn't scale well [10]. As a result, matrix factorization techniques have become a popular approach to collaborative filtering. Matrix factorization allows users and items to both be characterized by vectors of factors inferred from item rating patterns [28]. Collaborative filtering algorithms based on matrix factorization techniques have been shown to be highly accurate, even when dealing with sparse matrices. This was best demonstrated by Koren et al. [28] whose team won the Netflix grand prize in 2009 using matrix factorisation techniques [27].

2.1.3 Privacy Concerns

Recommender systems raise concern about individual privacy. As discussed by Resnick et al. [46], recommender system recommendations improve the more personal information they have. However, not all users want to have all their habits known. Indeed, Ackerman et al. [2] surveyed a number of users and found they could be split into three groups in regards to their view of privacy:

- **Privacy Fundamentalists** (17%) - This group was extremely concerned about use of their personal data and were generally unwilling to provide their data to websites, even

those with privacy protection measures in place.

- **Privacy Pragmatists** (56%) - This group was concerned about the use of their personal information, but to a lesser degree than the privacy fundamentalists. This groups' privacy concerns were greatly reduced by the presence of privacy protection measures on the website.

- **Marginally Concerned** (27%) - This group was willing to provide personal information under almost any condition, though they often expressed a mild general concern about privacy.

Furthermore, Ramakrishnan et al. [44] have discussed in detail the inherent risk of identity compromise, where an engineer managing a recommender system, or indeed third-party consultants carrying out routine auditing or data backup procedures, has access to the recommender system database. Research performed by Calandrino et al. [12] also showed that it is possible to design a series of inference attacks on collaborative filtering recommender systems, where information about individual users could be obtained given limited auxiliary information such as a user's public review for an item. Algorithms developed for these attacks aggregate publicly available data with no personally identifiable information to infer a user's non-public transactions with a website.

Given the above privacy risks and the clear concern the majority of users have about the use of their private information, it is clear the performance gain from collecting more individual user data is antithetical to the concerns of many users.

2.2 User Data Collection

As discussed in Section 2.1, information is needed on users to build user profiles for recommender systems. User information is gathered through the use of feedback, where feedback can be broken down into two main categories:

- Explicit Feedback

- Implicit Feedback

In both cases, feedback refers to a measure of how positively or negatively a user has reacted to an item. In section 2.2.3, methods of consolidating user profiles with active users of a system are discussed.

2.2.1 Explicit Feedback

Explicit Feedback is a process that involves users assigning either scores or ratings to evaluate the items recommended by the system [30]. It is different from implicit feedback techniques as it requires the user to actively provide feedback for an item. Surveys are a common explicit feedback mechanism, where a user is prompted to provide feedback about a particular item consumed by a user. The following are some of the most popular forms of surveys:

- **Rating or Score** - This feedback mechanism prompts a user to give a numeric rating for an item. This can be done in the form of a star rating between 1 and 5, where a 1 star rating denotes a very poor rating and a 5 star rating denotes a very good rating. An advantage of this feedback mechanism is that a user can express dislike for items, but a disadvantage of this mechanism is there is no guarantee a user will remain consistent with their ratings.

- **Like** - This binary feedback mechanism allows a user to express their interest in an item. A major advantage of this feedback mechanism is it's simplicity, but this comes at the cost of granularity and the inability to express dislike for an item.

- **Like/Dislike** - This feedback mechanism allows a user to express either like or dislike for an item. This feedback mechanism has the advantage of allowing users to express dislike for items, and unlike the Rating/Score mechanism there is no granularity to the ratings and so there is no concern regarding users remaining consistent with their ratings. However, this lack of granularity comes at the expense of users not having the ability to differentiate between items they like and items they strongly like. This mechanism also has an implicit feedback element to it, in that a user leaving no rating can be interpreted as a neutral reaction. However, it's important to note that a user can simply choose not to rate the item, and so any interpretation of a user not rating an item is likely to be noisy.

Explicit user feedback is valuable in building a user profile for a recommender system, but users generally don't provide enough explicit feedback to generate accurate recommendations for each user. This is a result of the intrusive nature of explicit feedback techniques, which rely on user input. Consequently, many systems adopt a hybrid approach to gathering user feedback by supplementing Explicit Feedback with Implicit Feedback.

2.2.2 Implicit Feedback

Implicit feedback mechanisms evaluate user ratings for objects without the intervention of users, often without the user being aware [39]. Implicit feedback mechanisms rely on assumptions that can be made about user interactions with the system that can then be evaluated by some metric. For example, given a music streaming service, the assumption that the more a user listens to a song the more the user likes the song would be valid. Using play count as a metric, a rating prediction can be generated for the user.

Implicit Feedback Mechanisms play a large role in current web systems where implicit feedback can be gathered without the need for user intervention. This has the added benefit of being able to gather much more information on a user and their interactions with the system than would be obtained through just explicit feedback mechanisms. It is also worth noting White et. al [58] also showed in their comparison of Explicit and Implicit feedback techniques for web retrieval that both feedback mechanisms can be used interchangeably with no real significant difference in performance.

Another major advantage of implicit feedback mechanisms is that a number of mechanisms can be put in place in the same system. For example, the same music streaming service monitoring user playback can also monitor how playtime is divided among different genre and artists, what artists or album pages a user visits, etc. Results from each feedback mechanism can be combined to build more advanced user interest profiles.

The issue with implicit user feedback mechanisms is that there is no way for a user to control what information about themselves is collected by these mechanisms. Since implicit feedback mechanisms require no user input, information is often collected without user knowledge, and additionally there are no controls in place for a user to decide what is done with this data. Web System providers have full autonomy over information collected about it's users through implicit feedback mechanisms. This collection and usage of user information poses a substantial privacy risk, as demonstrated by Calandrino et al. who successfully designed a series of passive inference attacks with limited auxiliary information to learn about the behaviour of individual users through information leaked by collaborative filtering recommender systems [12].

2.2.3 User Tracking

User interest profiles for generating recommendations are useless if a user can't be identified after they leave a website. There are a number of methods of tracking users, including client side information storage, user authentication, and browser fingerprinting. All three methods will be examined in detail in this section.

Client Side Information Storage

This method uses local browser storage mechanisms to store information about the user that will be communicated back to the server when a user revisits a website. Traditionally this has been done through the use of HTTP cookies, where a server would assign a cookie with a unique ID to each user that visits their website. These HTTP cookies have their expiry date set to expire later in the future so they don't expire when a user leaves the website. HTTP Cookies can store up to 4KB of information, and are communicated back to the server hosting a website as part of the user's browser's HTTP request headers. This allows the server to identify a user, match them with an existing user interest profile, and generate recommendations for the user. Use of HTTP Cookies to identify users is a common practice among advertisers and analytics companies, who set cookies on websites to allow the website owners to sell ads or view usage statistics for their websites [16, 20] . Cookies set by these third parties will henceforth be referred to as *third-party-cookies*.

A study conducted by Abraham et al found approximately 31% of U.S. users clear their first party cookies (cookies set by the website a user's visiting) within a month [1]. If HTTP cookies were the only method of user identification employed by the server, the server would no longer have any method of re-identifying the user and a new interest profile would need to be built under a new cookie. In an effort to overcome this, servers and third parties have begun to use other browser storage mechanisms to ensure assigned cookies persist despite deletion. Initially third parties used Flash cookies as the client side storage mechanism to track users as they are less prone to deletion than standard HTTP cookies and have the advantage of being able to store up to 100KB of information. Additionally, they could be used to 're-spawn' deleted HTTP cookies [50]. Though initially popular, Flash cookies depend on a Flash plugin being installed on the user's device, which is becoming less common as Flash becomes less popular. As a result, HTML 5 Web Storage has been adopted as a popular client side storage mechanism. Indeed HTML 5 Web Storage is supported by all major browsers, with some browsers supporting up to 10MB of storage for a single domain. This is considerably larger than both Flash and HTTP cookies, and HTML 5 Web Storage is more available than Flash cookies as it does not require users to install any third party plugins on their browser. Research by Ayenson et al. [6] shows the use of HTML 5 Web Storage to store cookies has already become quite popular in the top 100 websites online. The prevalence of these tracking methods is a major cause for concern, as website owners and third parties are explicitly taking action to circumvent users deleting their cookies (and consequently their individual user IDs) to ensure cookies persist on the user's browser.

User Authentication

User authentication refers to a user having a dedicated account with a website through which a user verifies their identity to use this account. These accounts are usually linked to an email address owned by the user. Users can then sign in to the service when they visit the website. This allows websites to consolidate user interest profiles with user accounts, as user authentication is a more reliable method of identifying users than tracking with just client side information storage.

When a user signs into a website, they are generally assigned session cookie(s) to allow them to revisit the website without logging in again. These cookies use the same client side browser storage mechanisms previously discussed, but do not suffer the same volatility as user tracking through cookies only. This is because when a user deletes their cookies they will also delete their session cookies, requiring them to sign into websites they visit again. When the user signs in, they'll be assigned new session cookies which are consolidated with their previous user interest profile.

User authentication is not limited to signing into independent websites. Many large services including Google and Facebook make use of 'Single Sign On' (SSO), where a user can authenticate themselves on a third party service using their Google or Facebook credentials. Hursti [23] defines a Single Sign On as a mapping from the physical world to the electronic and logical world, meaning a user's identity would not change between services. While a true single sign on system is not in practice, the availability of SSO APIs and the reputation of services such as Google and Facebook have resulted in widespread use of SSO services. The use of SSO authentication has resulted in large services gaining the ability to track its' users activities outside its' own network and gather additional user interest information through implicit feedback mechanisms.

Browser Fingerprinting

Browser Fingerprinting is considered a passive method of user tracking. It does not involve assigning users unique identifiers, but rather focuses on the individuality of user devices when visiting a site. Browser Fingerprinting tries to calculate a unique string to identify a user based on a number of properties of the user's browser and machine. These properties can include the User Agent sent with HTTP requests, the list of plugins installed on the browser, the user's timezone, the resolution of the user's display, the set of installed browser fonts, and so on [9]. The usability of browser fingerprinting was demonstrated by the Panopticlick experiment [17], where Eckersley showed that browsers could be successfully re-identified despite the relative volatility of browser fingerprints. Though browser fingerprints changed

over time, 99.1% of the time these browsers could still be identified with a false positive rate of only 0.86%. Furthermore, Eckersley concludes that despite the lack of stability of browser fingerprints, the volume of data that can be transmitted as part of a fingerprint is in itself a cause for concern for user privacy. The advantage of Browser Fingerprinting is that, unlike cookies, fingerprints cannot be deleted. They can be modified by changing a browsers properties, but as discussed, this does not guarantee tracking subversion. Though this dissertation does not concern itself with browser fingerprinting, it is important to portray the value of user data collection by discussing the various techniques that have been developed to enhance data collection.

2.3 BLC Clustering

A key objective of this dissertation is to effectively cluster users based on their interests such that recommendations made to the group are still relevant to the individual users. This objective will be achieved by using the clustering algorithm proposed by Checco et. al as part of their BLC: Private Matrix Factorization system [13]. This clustering algorithm allows unsupervised, automated inference of abstract groups based on user interests through matrix decomposition. Matrix decomposition is a method by which a matrix is reduced to it's matrix factors, similar to factorising the number 12 into $3 \cdot 4$. Similar to factorising numbers, a matrix A that has been decomposed into it's constituents B and C can be reconstructed by calculating the product of $B \cdot C$.

Checco et al. proposed a novel matrix factorisation method by which a matrix R of users u and their item ratings v (as seen in table 2.3) can be decomposed into $P^T\widetilde{U}^TV$, where the matrix P^T maps users to Nyms. The BLC Clustering algorithm calculates an appropriate assignment P of Nyms to users by clustering users with similar ratings together such that members of each resulting Nym have similar interests. Nyms can then be used as a proxy for users, with each user browsing as a part of their assigned Nym. Each Nym has it's own Nym-item rating pairs $\widetilde{R}$ calculated by using the ratings of the Nym's users for each item. Checco et. al [13] demonstrated that use of Nyms does not come at the cost of the recommendation accuracy, as will be further demonstrated in Section 6.

Additionally, the assignment of users to Nyms is also completed in a privacy-enhanced manner such that the assignment can be calculated on a user's machine. As a result there is no need for the user to submit their private rating information to any third party to be assigned to a Nym.

2.4 Private Recommendations

Privacy concerns surrounding the use of recommender systems are well known, and as such there has been a handful of previous attempts at making private recommendations through different methods.

The ALAMBIC hybrid recommender system proposed by Ameur et. al [4] attempted to make privacy-protecting recommendations by combining content based and collaborative filtering techniques. The system then split user data between the recommender system owner and a semi-trusted third party such that neither party would be able to derive sensitive information from their share of data alone. While in theory this model would relieve the major privacy concern of third parties knowing sensitive user information, it also has the downside of requiring a major rewrite to existing recommender systems in order to implement it. Additionally, the same amount of user information is still being collected for use by the recommender system, it's just partitioned across two entities. If someone was able to gain access to both entities then private user information could still be compromised.

Researchers at the University of Minnesota also developed a group recommender system called PolyLens [40]. While this dissertation is concerned with clustering a group of users with similar interests, PolyLens recommends content for groups of arbitrary users who would be consuming content together. For example, members of a family may not necessarily have the same taste but because of their familial relationship they would like a recommendation for a movie they can watch together. The PolyLens recommender system was successful in generating recommendations for groups, but it also came at the cost of group member privacy. While the goal of this dissertation is to ensure accurate recommendations for groups are made by recommender systems, it cannot come at the cost of individual privacy.

Another approach to private recommendations is a method called Content Recommendation System Based on Private Dynamic User Profile (CRESDUP) proposed by Chen et. al [15]. CRESDUP attempted to improve recommender system accuracy by increasing the amount of information available on a user in a privacy preserving manner. In a system using CRESDUP, all user information is collected on the client side. Instead of submitting user information to the server, a private Dynamic User Profile (DUP) that aligns with a user's interests is selected from the CRESDUP system and downloaded to the client's machine. This DUP already has a wealth of private user information from other users with similar interests, and this DUP is used to supplement the user's own information. The user's DUP can then be used by the recommender system to make more accurate recommendations, while also increasing the user's privacy. CRESDUP succeeds in improving content recommendations to the user in a privacy preserving manner, but does not address the issue of individual user

tracking. Additionally, CRESDUP requires each individual website to support the use of DUPs, otherwise the website would resort to traditional user information collection methods.

Shokri et al. [48] proposed a similar concept to CRESDUP, whereby each user would store their user profiles offline and would modify them by partly merging their profiles with the profiles of similar users through direct peer to peer communication with them. The user can then periodically upload their profile to a centralised server that is making recommendations. An individual's user-item ratings are hidden by establishing this peer to peer profile sharing mechanism as peers can transmit profiles directly without relying on an untrusted third party. Since the recommender system won't be able to distinguish user ratings from other users' ratings, users can still receive recommendations in a privacy preserving manner. While this system is effective in keeping user rating information private from third parties, it still requires a degree of trust from other peers in the system with whom users would be sharing rating information. Additionally, it relies on the user to select their peers to share rating information with so there is no guarantee a user will find a peer with similar interests to seed their profile with. This can negatively impact the user's ratings as their profile will be seeded with ratings that do not reflect their interests.

Chapter 3

Design

This chapter will outline the overall design of the OpenNym system and the design choices made for each component. The overall architecture of the system can be seen in Figure 3.1.

To facilitate enchanced privacy while users browse online, the OpenNym system must achieve two goals. The first is to intercept a user's session cookies as it is sent from the browser with appropriate session cookies for a Nym. The second is to intercept all user ratings for websites supported by OpenNym, and replace the ratings with an average Nym rating calculated using the user's intercepted rating. By accomplishing these two goals, users will be tracked as part of a Nym and individual user interests do not leave the browser.

3.1 OpenNym Web Service

The OpenNym Web Service will be a server that acts as the main point of interaction for users of the OpenNym Service. It will host several Application Programming Interfaces (APIs) that will facilitate communication between the OpenNym service and both the OpenNym Browser Extension and third-party recommender system services. There are five APIs that will be hosted on the web service:

- Nym API
- Rating API
- Session Cookies API
- Rules API
- Identity Interface API

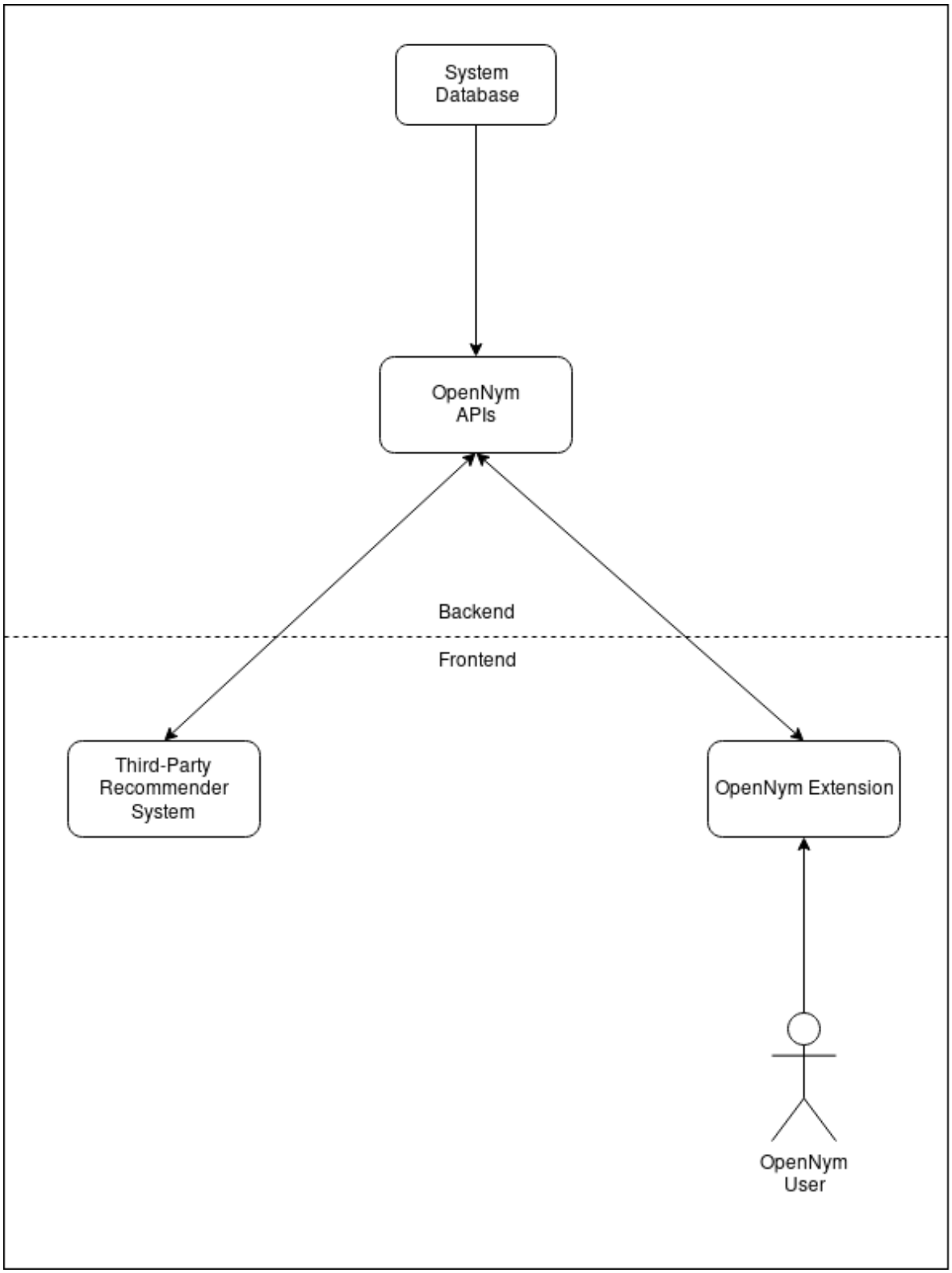


Figure 3.1: System Architecture Diagram

3.1.1 Nym API

The Nym API is responsible for communicating available Nyms and Nym details to the OpenNym Browser Extension. It provides a readily accessible list of Nyms, and information about the most rated items and most visited domains of each Nym.

```
{
    "Nym": {
        "id": 1,
        "topRatings": [
            { "rating" : {}},
            { "rating" : {}},
            { "rating" : {}},
            { "rating" : {}}
        ]
    }
}
```

Listing 1: Structure of Nym JSON Object

1. GET, /nym/, Returns List of Available Nyms
2. GET, /nym/:id, Returns details of a single Nym identified by an id

Both of the above endpoints will take an optional `lastUpdated` Uniform Resource Locator (URL) parameter that will contain the timestamp of the last time the endpoint was accessed by the client. This will allow the Nym API to only return Nyms that have changed since the timestamp provided. If there have been no changes then a 304-Not Modified response code is returned.

The **GET, /nym/** endpoint returns a JSON array of all Nyms, where a single Nym is represented by Listing 1. Each Nym returns a JSON array in the key `topRatings`, where each object in the JSON array is a JSON rating object as represented by listing 2. This endpoint can be called by anyone to retrieve a list of Nyms and details of the top ratings of each Nym. This endpoint is called by the OpenNym Browser extension to fetch and update the list of Nyms available to users.

Responses

Status Code	Response	Reason
200	JSON Array of Nym Objects	Request Successful
304	Empty	No Nyms Modified Since Last Request
404	Empty	No Nyms found

The **GET, /nym/:id** endpoint returns a single Nym in the form of a JSON Object, as represented by Listing 1. This endpoint takes a single URL Path Parameter, an Nym ID, which is used to retrieve a single Nym. As each Nym is assigned a unique ID on creation, there should only ever be one Nym assigned to an ID (if a Nym for the ID exists). This endpoint can be used by users and the OpenNym Browser Extension to retrieve a single Nym's details instead of having to retrieve the entire list of Nyms which can be an expensive operation.

Responses

Status Code	Response	Reason
200	Single Nym JSON Object	Request Successful
304	Empty	Nym Not Modified Since Last Request
404	Empty Response	Nym does not exist

3.1.2 Rating API

The Rating API is responsible for allowing the OpenNym Browser Extension to access a Nym's up to date ratings for items from a website, where these items could be songs on Spotify or movies on IMDB. The browser extension needs to be able to quickly access these ratings when intercepting user rating requests. For this reason, the API has an endpoint that will allow a subset of a Nym's most frequently accessed ratings for a web service to be downloaded at once, and then subsequent ratings can be retrieved by the browser extension by querying the API with the item ID. The Rating API endpoints are as follows:

1. GET, /ratings/:nym_id/:domain, Returns a list of ratings for the 10 most rated items for a given Nym for a given web service identified by it's domain name

2. GET, /ratings/:nym_id/:domain/:item, Returns a Nym's rating for a given item on a given web service identified by it's domain name

3. PUT, /ratings/update, Allows users to update a Nym's rating for an item.

For the rating API, each of the GET endpoints returns either a standalone JSON rating object or a JSON array of rating objects. The structure of a rating object can be seen in listing 2.

```
{
    "rating": {
        "domain": "imdb.com",
        "item": "tt0111161",
        "nymId": 0,
        "NymRating": {
            "score": 3,
            "numVotes": 12
        }
    }
}
```

Listing 2: Structure of Rating JSON Object

The item key in the JSON object displayed in Listing 2 represents the item identifier used by the domain to which this item belongs to. So given Listing 2, the rating object refers to unique item "tt0111161" in the IMDB database, which represents the movie "The Shawshank Redemption". Each item in the OpenNym database will have it's own unique ID, but for queries to the Rating API items will be referred to by their respective website's IDs. This is because services such as IMDB would already assign each of their items a unique ID. By pairing a website's domain and it's ID for an item, it should be possible to retrieve the rating for this item from the OpenNym database without any form of mapping function of the website's item ID to an OpenNym ID. The unique OpenNym ID given to each rating is purely for database foreign key referencing.

The sub-object `NymRating` contains the actual rating a Nym has given to an item and

the number of users of the Nym who have voted on the item. Storing the number of users who have voted and the actual rating for an item is important as it is necessary to calculate an updated average when a new user rates the item. It is important to note that the actual rating is only stored in the OpenNym database if the web service does not have an API available by which the browser extension can retrieve the ratings for the Nym it is currently using. If an API is available through which the Nym's ratings can be retrieved from the OpenNym Browser Extension, we opt to use it instead of storing the average rating on the OpenNym server. This is because it helps to preserve user privacy as they do not need to communicate their rating updates to the OpenNym server, but instead the average rating can be retrieved by the OpenNym server periodically.

The **GET /ratings/:nym_id/:domain** endpoint will be used by the OpenNym extension to get an array of JSON rating objects that it's most likely to use when it visits a website. The service determines which ratings are most likely to be used by simply returning the 10 most used ratings for a Nym for the given domain, as measured by each rating number of votes. This endpoint will be queried by the OpenNym extension when it detects the user is opening a website which is supported by the OpenNym web service.

Responses

Status Code	Response	Reason
200	JSON Array of Rating Objects	Request Successful
404	Empty Body	Requested Nym does not exist
404	Empty Body	Requested Domain not Supported

The **GET /ratings/:nym_id/:domain/:item** endpoint will be used to retrieve a rating object for a given Nym, domain and item. If the domain is supported by OpenNym but no rating for the item is available, a rating object will be created in the OpenNym database and a response returned with `score` 0 and `numVotes` 0. If a request to the endpoint is made for a domain which is not yet supported by OpenNym, a 404-'Not Found' Response code is returned. If the request is successful (The domain-item rating pair exists for the given Nym), a 200-OK response code is returned with a single JSON Rating object represented by Listing 2. Clients can include an optional `lastUpdated` URL parameter which contains the timestamp of the last time the client retrieved the rating for the specified item. This timestamp is used to determine if there has been any changes to the rating since it was last

retrieved by the client. If there hasn't then a 304-Not Modified response code is returned. Otherwise the rating is sent to the client as normal.

Responses

Status Code	Response	Reason
200	JSON Rating Object	Request Successful
404	Empty Body	Requested Nym does not exist
404	Empty Body	Requested Domain not Supported

The **PUT /ratings/update** endpoint will be used to update the rating details of an item for a Nym. The item is identified using the domain and item identifier from the web service itself, where the web service is identified by the domain. The body of this PUT request will contain the JSON rating object as shown in Listing 2, but will have the updated number of votes and, in some cases, the updated scores. Updated scores will only be included in the case where a web service has an API from which ratings can be retrieved from the browser extension. If this API is unavailable, then ratings must be saved in the OpenNym server too. All the information required by the OpenNym web service to update a rating can be found in the request body which contains the rating item's domain and ID.

Furthermore, the Web Service must ensure ratings are not overwritten. Take two users of the same Nym, Alice and Bob, for example. Alice and Bob both retrieve rating item C at the same time, but Alice updates C before Bob. When Bob tries to update C, his update would fail to take Alice's update into account. To ensure Alice's update is not lost, the Web Service must check the `numVotes` key in each update request and ensure it increments the current number of votes by 1. If the request fails to do so, a 409-Conflict response will be returned with the server's up to date score and number of votes for an item. The sender will need to recalculate the average score for the item using it's up to date information before re-sending the request. Conflict Resolution has been chosen over a preventative method such as Bob requesting his rating item again before sending an update request to try and keep the system as performant as possible. If a user had to retrieve the updated rating before sending an update request, this would effectively double the number of requests to the server. By allowing a user to submit an update request and only have them update their rating when a conflict occurs, the overall number of requests to the OpenNym Web Service is reduced.

Responses

Status Code	Response	Reason
200	Empty	Update successfully made
400	Empty	Request Body missing mandatory field
404	Empty	Document does not already exist
409	Up to Date JSON Vote count and rating	Update conflict

3.1.3 Session Cookie API

The Session Cookie API is responsible for allowing the OpenNym Browser extension to retrieve Session Cookies for a Nym to use on websites. This is important as these session cookies are how websites keep track of authenticated users, and so for OpenNym this means each Nym group account will have a number of Session Cookies associated with them. The OpenNym Browser Extension needs to be able to quickly access these session cookies, as it would be infeasible for the extension to download all session cookies for a Nym at once. Similar to the Ratings API, this API will allow for the session cookies for the 10 most popular domains of a Nym to be downloaded when a Nym is joined. The top domains for a Nym will be determined by the most popular domains accessed by members of a Nym. Subsequent requests for session cookies belonging to other domains can then be made. The Session Cookie API endpoints are as follows:

1. GET, /cookies/:nym_id, Return the session cookies for the 10 most popular domains of a given Nym
2. GET, /cookies/:nym_id/:domain, Return the Session Cookies given a Nym ID and a domain
3. GET, /cookies/issue/:nym_id/:domain, Returns the date-time for when a Nyms session cookies for a domain was last updated

For the Session Cookie API, the first 2 endpoints return an array of JSON Session Cookie Objects and a single JSON Session Cookie object respectively. The third endpoint returns a JSON Session Cookie date-time object. The structure of these JSON objects can be seen in Listings 4 & 3.

```
{
    "session": {
        "issued": 1521026899
    }
}
```

Listing 3: Structure of Session Date-time JSON Object

```
{
    "session": {
        "domain": "spotify.com",
        "issued": 1521026899,
        "cookies": {
            "cid": "abcdefghijklmnop",
            "val": "abc123"
        }
    }
}
```

Listing 4: Structure of Session Cookie JSON Object

The **GET /cookies/:nym_id** endpoint will be used by the OpenNym browser extension to retrieve the session cookies for the 10 most popular domains given Nym. As can be seen from the endpoint, a Nym ID is supplied as part of the URL. The extension will make a call to this endpoint when it first selects a Nym to join.

Responses

Status Code	Response	Reason
200	JSON Array of Session Cookie Objects	Request Successful
400	Empty	Nym ID Not supplied
404	Empty	Specified Nym does not exist
404	Empty	Specified Nym has no domains associated with it

The **GET, /cookies/:nym_id/:domain** endpoint is used by the OpenNym Extension to retrieve a session cookie object for a single domain, represented by listing 4. This is used by the Browser Extension when it doesn't already have the session cookies for a domain cached.

Responses

Status Code	Response	Reason
200	Session Cookie Object	Request Successful
404	Empty	Specified Nym does not exist
404	Empty	Specified Domain not supported

The **GET, /cookies/issued/:nym_id/:domain** endpoint is used by the OpenNym extension to retrieve the date-time a domain's session cookies was last updated. The relevant session cookies are identified by the Nym ID and a domain. This will allow the extension to check if a cached cookie has expired before using it.

Responses

Status Code	Response	Reason
200	Session Cookie Datetime Object	Request Successful
404	Empty	Specified Nym does not exist
404	Empty	Specified Domain not Supported

3.1.4 Rules API

The Rules API is responsible for allowing the OpenNym extension to retrieve up to date rules for intercepting rating requests on supported OpenNym websites. This API will allow the OpenNym extension to retrieve a list of supported OpenNym websites and retrieve rules in the form of Registered Expressions and URL endpoints to allow the extension to intercept rating requests to supported websites and parse the users rating out from the request body. This will allow the extension to calculate an average for the Nym the user is a part of before allowing the request to proceed. The endpoints are as follows:

1. GET, /rules/supported, Returns a list of domains OpenNym supports

2. GET, /rules/supported/version, Returns the current version of the OpenNym Support list

3. GET, /rules/top/:nym_id, Returns a list of rules for the top 10 domains for a given Nym

4. GET, /rules/:domain, Returns the intercept rule for a given domain

5. GET, /rules/issued/:domain, Returns the date-time the rule for a given domain was updated

```
{
    "domainRule": {
        "domain": "spotify.com",
        "timestamp": "2018-05-14T12:07:48.203697",
        "rule": {
            "method": "GET",
            "endpoint": "/track/1235",
            "regex": "~r{Some Regex}"
        }
    }
}
```

Listing 5: Structure of OpenNym JSON Rule Object

```
{
    "domainRule": {
        "timestamp": "2018-05-14T12:07:48.203697"
    }
}
```

Listing 6: Structure of OpenNym JSON Rule Version Object

The **GET, /rules/supported** endpoint returns a JSON object which contains an array of domains supported by OpenNym, as represented by listing 7. This endpoint will be used in

the OpenNym browser extension to determine which domains the extension should be active on.

Responses

Status Code	Response	Reason
200	JSON Object with array of domains	Request Successful

The **GET, /rules/supported/version** endpoint returns a JSON object which contains just the version number of the current list of websites OpenNym supports, as represented by listing 8. This endpoint will be used by the OpenNym extension periodically to check if there have been any updates to the list of supported websites.

Responses

Status Code	Response	Reason
200	JSON Object with current support list version	Request Successful

The **GET, /rules/top/:nym_id** endpoint returns a JSON array of JSON objects containing the interception rules for the top-n domains for a given Nym, where each individual JSON Object is in the form of listing 5. In listing 5, the key `endpoint` represents the endpoint to which ratings are submitted for the given domain. The key `regex` is a registered expression that will be used to retrieve the rating submitted by the user.

Responses

Status Code	Response	Reason
200	JSON Array of Rule Objects	Request Successful
404	Empty	Given Nym ID unrecognized

The **GET, /rules/:domain** endpoint returns a single JSON object containing the interception rule for a given domain, where this JSON object is represented by Listing 5. The endpoint will be used by the OpenNym extension to retrieve the interception rule for a domain where it's interception rule isn't already cached.

Responses

Status Code	Response	Reason
200	JSON Object representing Rule	Request Successful
400	Empty	Given Domain not supported by OpenNym

The **GET, /rules/issued/:domain** endpoint returns a single JSON object with the timestamp for when the rule of a given domain was last updated. This JSON object is represented by listing 6. This endpoint will be used by the OpenNym extension periodically to ensure cached rules are up to date, otherwise there may be issues in intercepting user ratings for given websites.

Responses

Status Code	Response	Reason
200	JSON Object with rule's timestamp	Request Successful
400	Empty	Given Domain not supported by OpenNym

```
{
    "supportList": [
        "youtube.com",
        "spotify.com",
        "movielens.org",
        "vimeo.com"
    ]
}
```

Listing 7: Structure of OpenNym supported website list JSON object

```
{
    "version": 1.23
}
```

Listing 8: Structure of OpenNym supported website list version JSON object

3.1.5 OpenNym Identity Interface API

The OpenNym Identity Interface is designed to allow third party services to access Nym ratings and update their own systems accordingly. This interface is necessary because for a Nym to receive accurate recommendations from a web service, the recommender system must have previous ratings for the Nym.

Ideally OpenNym wouldn't rely on web services to update ratings for Nyms themselves and would automatically update all ratings that changed for a Nym through an automated script. The issue with this approach however is that websites have taken considerable steps to prevent bot traffic on their networks through the use of captchas, and most notably, Google's ReCaptcha [21]. As a result, it would be infeasible to try and write a script to automatically update website ratings and circumvent state of the art bot-deterrents. As such, the decision has been made to design an interface that third party websites can interact with to manually retrieve ratings from the OpenNym service and update the ratings of OpenNym accounts on their own web services accordingly.

The OpenNym Identity Interface will not require authentication to query the API initially as all Nym rating information is already publicly available from the other APIs, but section 7.1.2 discusses how the API may need authentication in the future.

The endpoints for the OpenNym identity service are as follows:

- GET, /identity/:domain, Return a JSON array of all ratings for all Nyms for a domain
- GET, /identity/:domain/:timestamp, Return a JSON array of all ratings for all Nyms for a domain that have changed since the supplied timestamp
- GET, /identity/nym/:domain/:nym_id, Return a JSON array of all ratings for a specified Nym for a domain
- GET, /identity/nym/:domain/:nym_id/:timestamp, Return a JSON array of all ratings for a specified Nym for a specified domain that have changed since supplied timestamp

- GET, /identity/:domain/:username, Given a domain and username, return the Nym ID associated with the username

```
{
    "NymId": 1,
    "ratings": [
        {
            "item": "abc",
            "rating": 123
        },
        {
            "item": "def",
            "rating": 456
        }
    ]
}
```

Listing 9: Structure of JSON Domain Rating

```
{
    "NymId": 1
}
```

Listing 10: Structure of username to Nym mapping response object

The **GET, /identity/:domain** endpoint is responsible for retrieving all ratings associated with all Nyms for a given domain. A successful response to this request is a JSON array of domain rating objects represented by Listing 9. This will be used by third party recommender systems to retrieve ratings for Nym accounts registered with their service so they can update the Nym's ratings in their own system's database. This endpoint is intended to allow third party web services to populate each Nyms account with ratings to avoid the cold start problem in recommender systems. Ideally this endpoint would only be used once by each third party service to populate each Nym account with it's initial ratings.

Responses

Status Code	Response Body	Reason
200	Array of Ratings for each Nym	Query Successful
400	Empty	Specified Domain not supported

The **GET, /identity/:domain/:timestamp** endpoint is responsible for retrieving all ratings associated with all Nyms for a given domain that have been modified since a given timestamp. This endpoint should be used by third party services to view all rating changes since they last updated their ratings. The timestamp supplied must be in the same format (YYYY-MM-DDTHH:MM:SS) as is served by the Web Service. This endpoint should not be necessary as the OpenNym browser extension should update ratings on the third party service automatically, but has been provided to allow services to ensure ratings are being properly recorded. This endpoint also returns a JSON array represented by listing 9, but ratings are filtered to show only those that have been modified since the provided timestamp.

Responses

Status Code	Response Body	Reason
200	Array of Ratings for Each Nym	Query Successful
400	Empty Body	Specified Domain not supported
400	Empty Body	Specified Timestamp in incorrect format

The **GET, /identity/nym/:domain/:nym_id** endpoint is responsible for retrieving all ratings associated with a given Nym for a given domain. This endpoint is also to be used by third party services to allow them to update ratings for a given Nym. A successful response to this request will return a single JSON response represented by listing 9.

Responses

Status Code	Response Body	Reason
200	Nym Ratings for a Domain	Query Successful
400	Empty	Specified Domain is not supported
400	Empty	Specified Nym does not exist

The **GET, /identity/nym/:domain/:nym_id/:timestamp** endpoint is similar to the previous endpoint, but results are filtered to show only ratings that have been modified since the supplied timestamp. The supplied timestamp must be of the format (YYYY-MM-DDTHH:MM:SS). A successful response to this request will return a single JSON response represented by listing 9.

Responses

Status Code	Response Body	Reason
200	Nym Ratings for a Domain	Query Successful
400	Empty	Specified Domain is not supported
400	Empty	Specified Nym does not exist
400	Empty	Supplied Timestamp in incorrect format

The **GET, /identity/:domain/:username** endpoint is responsible for allowing a third party web service to map a Nym username to a Nym ID. This endpoint has been included because the Nym ID is needed for some endpoints and it is unreasonable to expect third parties to infer a Nym's ID from the username, which may not even be possible in all cases. The response to a successful query is represented by listing 10, where the response is simply the ID of the Nym associated with the username.

Responses

Status Code	Response Body	Reason
200	Single JSON Object representing Nym ID	Query Successful
400	Empty	Specified Domain not supported
404	Empty	Specified username does not exist for domain

3.2 OpenNym Database

The OpenNym Database will be responsible for storing all the information served by the APIs. A number of database tables will be used to store required information, as will be outlined in detail in this section.

3.2.1 Nym Database Table

The Nym table's responsibility will be to store key information about each Nym. This is the Nym's ID, the Nym's top rated items, and the Nym's most used services. Each Nym's top domains and top rated items will be stored in the database and periodically updated instead of being dynamically generated on each request to both improve API response times and avoid unnecessary computation. Each Nym's top domains and top ratings can be periodically updated through the use of a scheduled script.

Nym Table Fields

Name	Primary	Type	Not Null	Unique	Foreign Key
id	Yes	Integer	Yes	Yes	No
top_ratings	No	Array of Rating Foreign Keys	Yes	No	Yes
top_domains	No	Array of Domain Foreign Keys	Yes	No	Yes

3.2.2 Ratings Database Table

This table is responsible for storing all Nym's ratings for all domains. Each rating will be assigned a unique ID which will act as the primary key for this table, but the ID will only be used internally by the OpenNym service when referencing a rating as a foreign key. The OpenNym extension and API will still allow ratings to be searched by domain name & item ID. The reason a separate ID must be assigned as opposed to using a field such as domains is that it's expected there will be multiple entries in the table for the same domain, but different items.

Rating Table Fields

Name	Primary	Type	Not Null	Unique	Foreign Key
id	Yes	Integer	Yes	Yes	No
domain	No	String	Yes	No	Yes
item	No	String	Yes	No	No
score	No	Float	Yes	No	No
num_votes	No	Integer	Yes	No	No
inserted_at	No	Timestamp	Yes	No	No
updated_at	No	Timestamp	Yes	No	No

The `domain` field in this table references an existing domain in the Rules database table. This constraint ensures ratings for websites that are not yet suppported by OpenNym cannot be inserted into the database.

The only additional constraint to this Database Table will be that no two entries in the rating table can have an identical domain & item value. This is not expected to ever happen, but an alarm will be set in the case such a situation arises so it can be investigated.

3.2.3 Session Cookies Database Table

Only one table will be needed to properly store session cookie information. Session Cookies for domains will be stored such that they can be retrieved using the domain name only. Each entry in this table will have a unique ID to act as the primary key, but cookies will be retrieved using the domain's name and the cookies associated Nym ID.

Session Cookie Table Fields

Name	Primary	Type	Not Null	Unique	Foreign Key
id	Yes	Integer	Yes	Yes	No
domain	No	String	Yes	Yes	Yes
cookies	No	String	Yes	No	No
nym_id	No	Integer	Yes	No	Yes
inserted_at	No	Timestamp	Yes	No	No
updated_at	No	Timestamp	Yes	No	No

Session Cookies for each domain will be stored as a serialized JSON string instead of using multiple table entries for each cookie because it is expected there will be no need to retrieve cookies individually, and as such there is no reason to index them separately. The `domain` field references a domain name that exists in the Rules database table. The `nym_id` field references an existing nym in the Nym database table.

3.2.4 Rules Database Table

The rules database table will store a list of rules, one for each domain. Since there's only one rule for each domain, the domain field will be used as the primary key. For each domain the table will store the timestamp of when the rule was created and last updated, the endpoint the OpenNym extension should intercept requests to, the HTTP Verb used to make the request to intercept, and a registered expression to parse the user's rating from the request body.

Rule Table Fields

Name	Primary	Type	Not Null	Unique	Foreign Key
domain	Yes	String	Yes	Yes	No
method	No	String	Yes	No	No
endpoint	No	String	Yes	No	No
regex	No	String	Yes	No	No
inserted_at	No	Timestamp	Yes	No	No
updated_at	No	Timestamp	Yes	No	No

There is no need to have a separate table to store the list of supported domains because a supported domain list can be generated from the above table by returning only the domain names to the OpenNym extension.

3.2.5 Identity Interface Table

The Identity Interface as discussed in section 3.1.5 is primarily concerned with returning Nym ratings which are stored in the Nym Ratings table (section 3.2.2). As a consequence, the Identity Interface table will only store the mappings of Nym usernames for various services to Nym IDs. IDs for this table will be automatically generated integers which will act as the primary key. The `nym_id` key will reference the ID of an existing Nym in the Nym database table.

Identity Interface Table Fields

Name	Primary	Type	Not Null	Unique	Foreign Key
id	Yes	Integer	Yes	Yes	No
domain	No	String	Yes	No	Yes
username	No	String	Yes	No	No
nym_id	No	Integer	Yes	No	Yes
inserted_at	No	Timestamp	Yes	No	No
updated_at	No	Timestamp	Yes	No	No

3.2.6 Nym Metadata Table

The Nym Metadata Table is designed to store additional information about the OpenNym web service. It currently contains an ID to act as the primary key, and the field `support_list_version` which represents the version number of the list of domains supported by OpenNym. Users can retrieve the support list version to decide whether are not they need to update their list of supported domains. It is intended that additional metadata not currently used will be stored in this table.

Nym Metadata Table Fields

Name	Primary	Type	Not Null	Unique	Foreign Key
id	Yes	Integer	Yes	Yes	No
support_list_version	No	Float	Yes	No	No

3.3 OpenNym Browser Extension

The OpenNym Browser extension is responsible for facilitating users to browse as part of a Nym. To achieve this, the browser extension must:

- Allow a user to select a Nym to browse as part of.
- Intercept user session cookies and substitute them with a Nym's session cookies. This would result in websites tracking users as part of a Nym.
- Intercept user ratings and calculate a new Nym average rating using the user's ratings. Nym average ratings must then be sent in place of user ratings.

As a consequence, the OpenNym browser extension will be responsible for interacting with the OpenNym Web Service to retrieve Nym information, ratings, rules, and session cookies. The browser extension will also be the user's main point of interaction with the OpenNym service. This section will outline the functional and non-functional requirements of the browser extension, and how the browser extension should behave in a series of expected user scenarios to ensure the user successfully browses as part of a Nym. The description of the browser extension's behaviour in each scenario will be aided by the use of UML Sequence Diagrams.

3.3.1 Functional Requirements

- Keep an up to date list of available Nyms to browse as part of.
- Keep an up to date list of domains supported by OpenNym.
- Intercept all outgoing requests to supported domains, including rating requests. All outgoing requests must be intercepted as session cookies for each request must be substituted.
- Retrieve Nym ratings for items when the ratings are not cached.
- Allow a user to manually select a Nym.
- Keep an up to date list of the top 10 domains for a user's Nym.
- Keep a valid cache of cookies and rules for the Nym's top 10 domains.
- Substitute user session cookies in all outgoing requests with the Nym's session cookies.
- Cache the top ratings for each website a user visits.
- Calculate the Nym's average rating for each user rating that is intercepted.
- Update the average rating for items on the OpenNym web service.
- Store user ratings locally in the browser.
- Cache OpenNym session cookies for domains that are not in the Nym's top 10 domains when they are requested.
- Cache OpenNym interception rules for domains that are not in the Nym's top 10 domains when they are requested.
- Ensure cached information is valid.

3.3.2 Non-Functional Requirements

- Impact on website load times must be minimal.
- Usage should require no specialized knowledge from users.

3.3.3 Design

For the browser extension to function, it must be able to handle a number of browser events successfully to ensure private user rating information is not sent to third parties. This section will detail a number of sequence diagrams outlining the sequence of actions taken to ensure browser extension functionality. In this section, "YouTube.com" will be used as the example website.

Browser Start

The extension should have a listener defined such that it receives an event when a new browser window is opened. When the event is received, the extension must ensure it has an up to date list of Nyms, supported domains, rules, and session cookies for the selected Nym's top 10 domains. There are 2 possible sequences of events for when a new browser window is opened:

- **First Run** - This sequence of events is outlined in Figure 3.2. It outlines the process the extension must take to retrieve all necessary Nym information when no information is available in the cache. This sequence of events can occur when the user has just installed the extension or cleared their browser cache.
- **Subsequent Run** - This sequence of events is outlined in Figure 3.3. It is run when cached information is available to the extension. It is mainly concerned with ensuring cached information is still valid by querying the server with timestamps to check whether any information has expired.

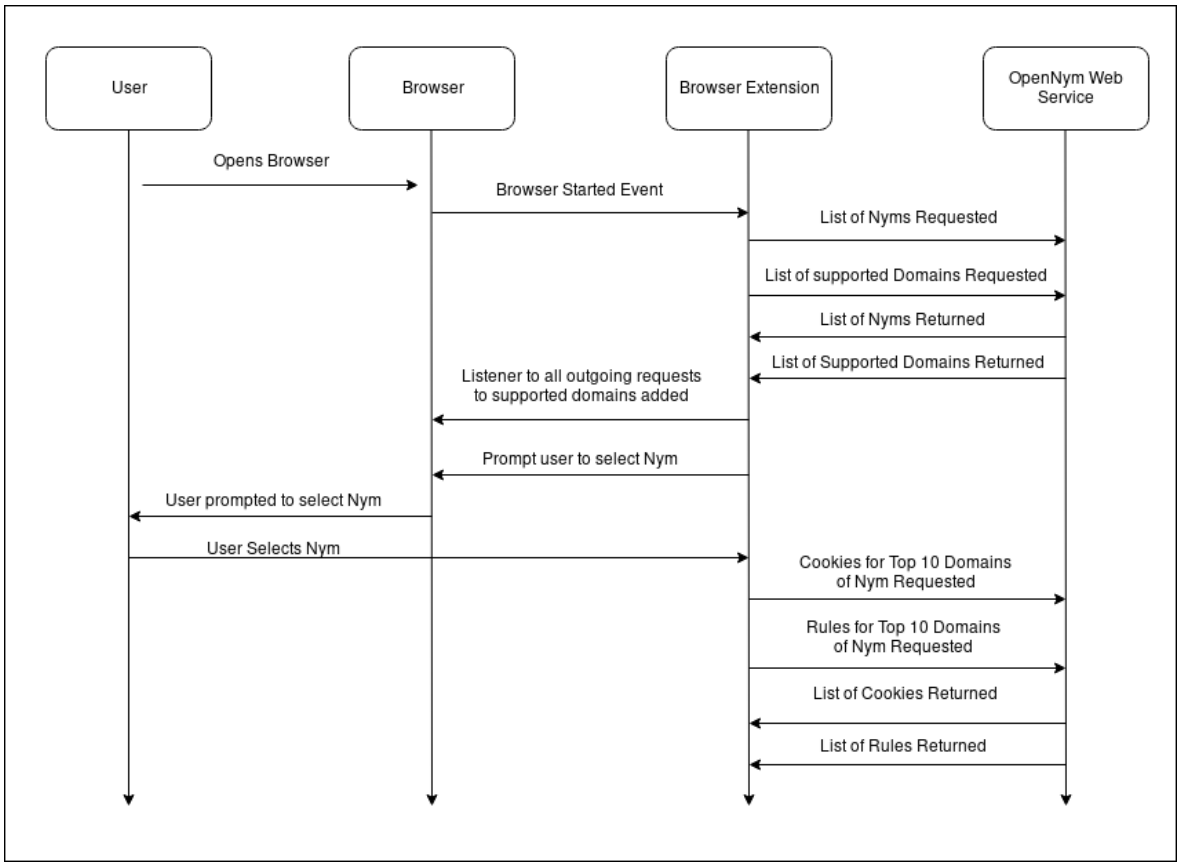


Figure 3.2: Browser Extension First Run Sequence Diagram

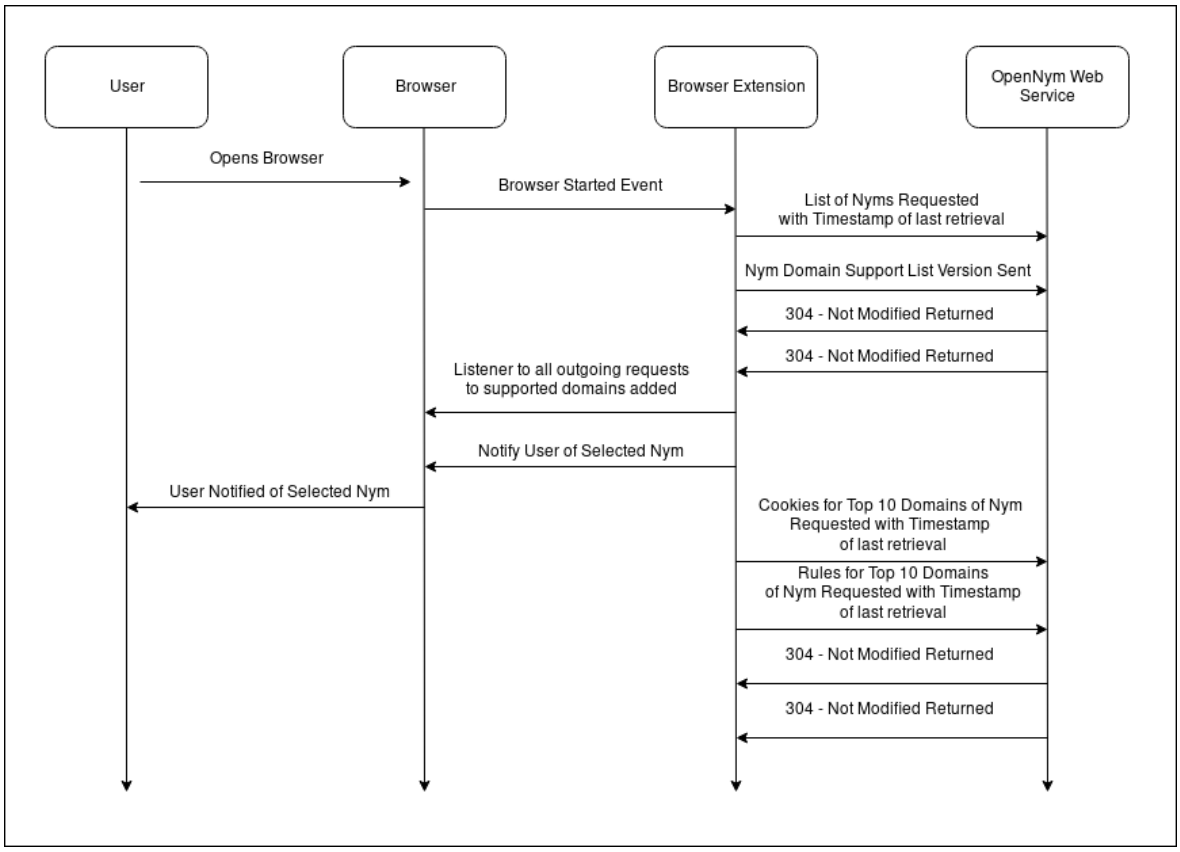


Figure 3.3: Browser Extension Subsequent Run Sequence Diagram

Website Request

This event is concerned with the user navigating to a website. When the browser extension retrieves the list of domains supported by OpenNym, it adds a listener for outgoing requests to each domain. When a user navigates to YouTube.com, the session cookies in the request must be substituted for the Nym's session cookies. There are two possible scenarios for retrieving the Nym's session cookies for YouTube.com:

- **Session Cookies Not In Cache** - If the Nym's session cookies for YouTube.com are not already cached, then the user's request is blocked while the session cookies are retrieved from the OpenNym Web Service. This is shown in Figure 3.4. It is important to note that the browser extension only waits to receive the session cookies for YouTube.com before substituting the session cookies and unblocking the user's request. The other requests are run asynchronously. This is to improve performance as the response to the other requests are not necessary for the user to load YouTube.com as part of a Nym.

- **Session Cookies In Cache** - If the Nym's session cookies for YouTube.com are already cached, then the user's request is blocked while the cached session cookies are

substituted into the user's request. This is shown in Figure 3.5. It is important to note here that all cached cookies are validated when the browser starts, ensuring the browser is not using expired cookies. It is also important to note that requests for the top ten ratings and the rule for YouTube.com is also sent asynchronously despite being cached. The timestamp of the when the cached information was retrieved from the OpenNym web service and if there is no change then the extension simply receives a 304 - Not Modified. If there has been a change, then the OpenNym web service would respond with the up to date information and the cache would be overwritten.

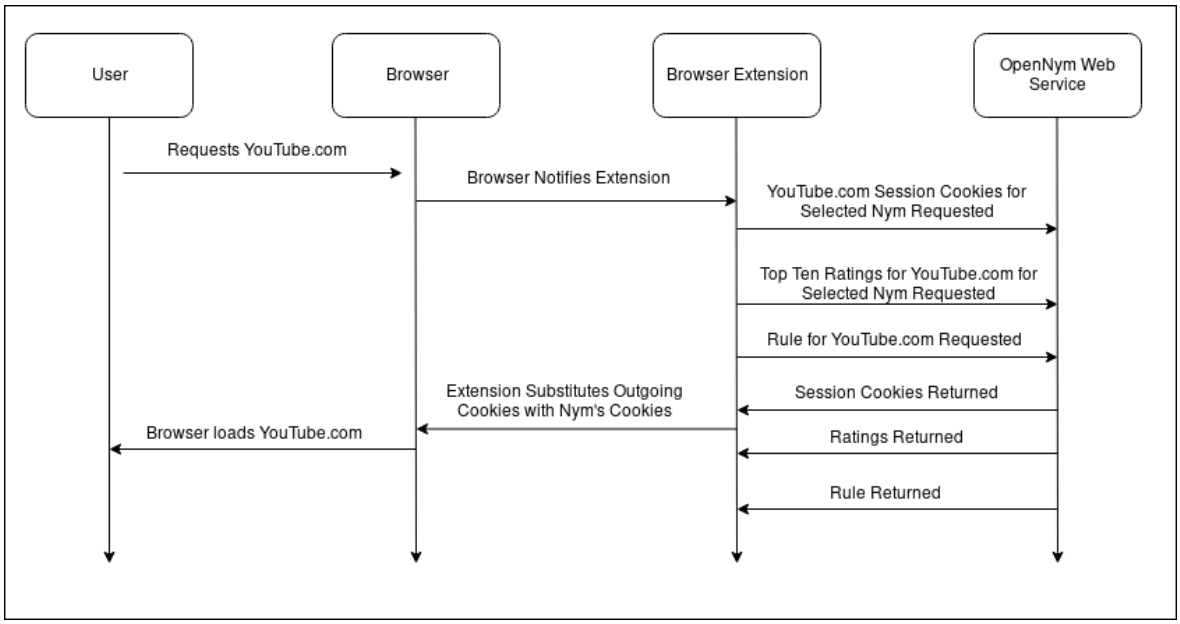


Figure 3.4: First Time Extension Loads Domain

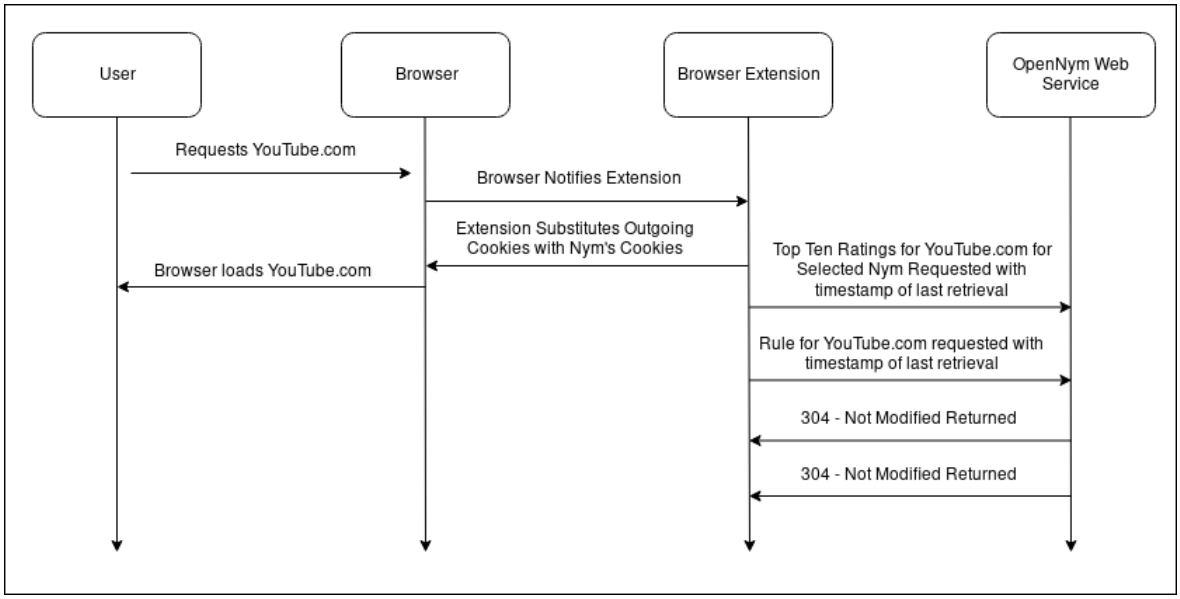


Figure 3.5: Extension loading Domain with cached Session Cookies

Rating Request

This event is concerned with intercepting user ratings to websites. It is important for the browser extension to intercept ratings as ratings submitted to YouTube.com must be representative of the entire Nym and private user rating information must not be leaked. As such, when a user rating is submitted to YouTube.com it is intercepted by the browser extension which blocks the request, and parses the users rating from the request body using YouTube.com's rule. An average rating for the Nym must then be calculated, and the average rating is first updated on the OpenNym server before being substituted into the rating request for YouTube.com. After substituting the rating in the rating request, the request is unblocked and continues as normal. There are three possible scenarios for this:

- **Uncached Rating** - When the rating request is intercepted, the rating item ID is retrieved and the cache is checked for an existing rating. If the rating is not already cached, then the rating request must be blocked while the rating is retrieved from the OpenNym Web service. The Nym's new average rating for the item is calculated and updated on the OpenNym web server. If this is successful, then the new rating is substituted into the user's rating request and the request is unblocked. The new average rating is cached. This is demonstrated in Figure 3.6.

- **Cached Rating with No Update Conflict** - Similar to with the uncached rating, when a rating request is intercepted the rating item ID is retrieved and the cache is checked for an existing rating for the item. If the rating is already cached, the new average Nym rating is calculated and updated on the OpenNym web service. If the rating updates successfully, the new average rating is then substituted into the user's rating request and the rating request unblocked. This is demonstrated in Figure 3.7.

- **Cached Rating with Update Conflict** - Similar to the previous example, the rating is successfully retrieved from the cache and the new average rating for the Nym is calculated. The new average rating is sent to the OpenNym Web Service but fails to update the server's rating as the cached rating was not the most up to date rating. As a result, the OpenNym Web Service returns a 409 - 'Conflict' with the up to date rating in the body. The extension must then recalculate the Nym's new average rating and send it to the OpenNym Web Service again. If this update is successful, the new average rating is cached and substituted into the user's rating request, and the user's rating request is unblocked. This is demonstrated in Figure 3.8.

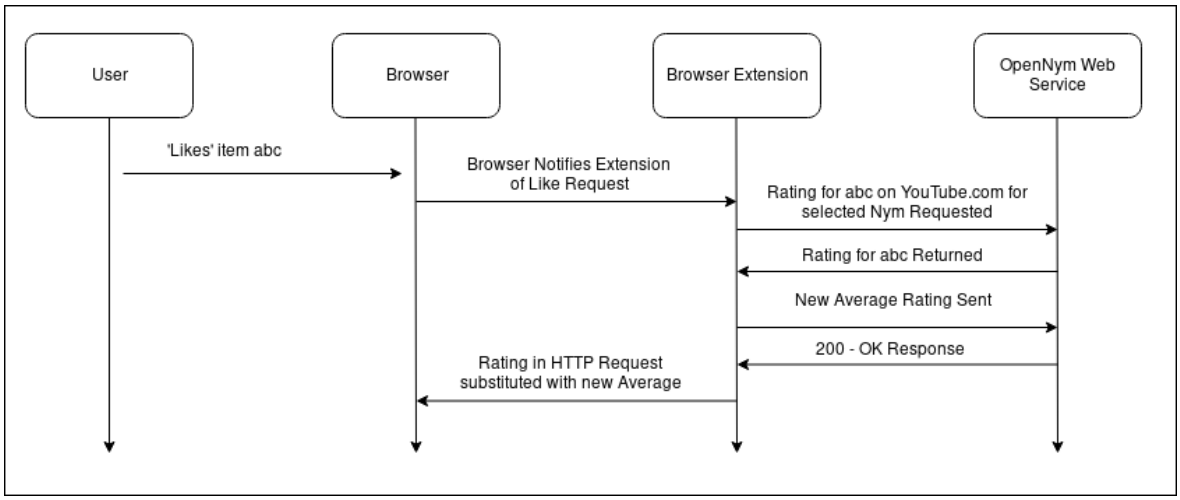


Figure 3.6: Extension intercepting uncached user rating

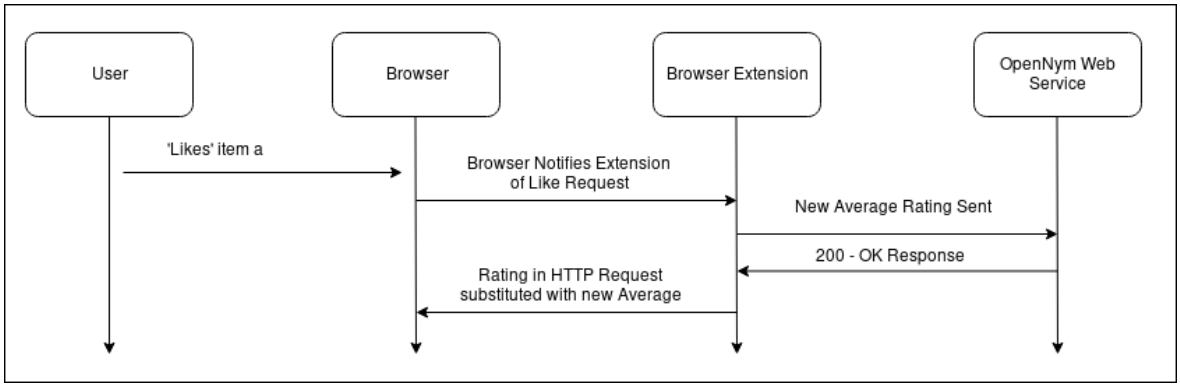


Figure 3.7: Extension intercepting cached user rating

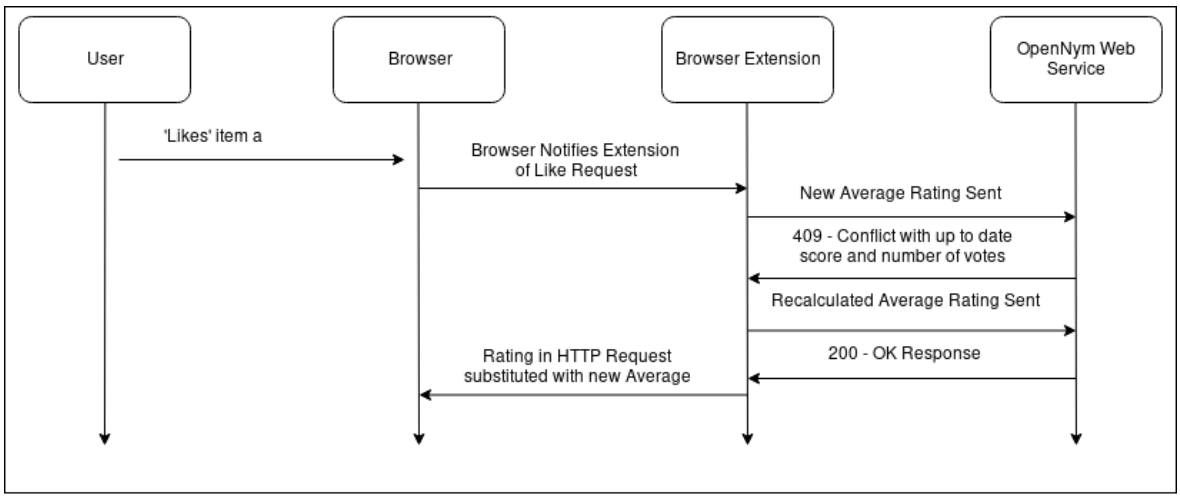


Figure 3.8: Extension intercepting cached user rating with an update conflict

Chapter 4

Implementation

This chapter will outline the technologies and processes used to implement both the web service and browser extension, according to their specifications outlined in Chapter 3. Additionally, section 4.3 will outline security and privacy issues related to the implementation of both the web service and the browser extension.

4.1 Web Service

The OpenNym Web Service was implemented according to the specification outlined in Section 3.1. The implementation can be split into two major components:

- The Web Framework
- The Database

Both these components will be discussed in detail.

4.1.1 Web Framework

The Web Framework used for the OpenNym Web Service was Phoenix v1.3 [33]. Phoenix is a relatively new web framework implemented in Elixir that uses the model-view-controller design pattern. Elixir is a dynamic, functional programming language that operates in the Erlang Virtual Machine [56]. Erlang is another functional programming language that was developed by Ericsson that is popular in the Telecommunications industry for it's ability to run low-latency distributed and fault tolerant systems [19]. By using the Erlang Virtual Machine, Elixir leverages Erlang's fault tolerance and scalability while providing additional features.

Phoenix & Elixir were selected because Phoenix benchmarks have shown it to be extremely high performant. This was demonstrated best by Gary Rennie, who managed to run a chat

room with 2 million concurrent users on a single server [45]. This benchmark was run on a server with 40 CPU cores and 128GB of RAM, but managed to broadcast all chatroom messages to all 2 million subscribers within 1 second. Phoenix was also selected because it is an open source web framework, allowing anyone to inspect the source code and ensure it is safe to use. This is a very compelling factor from a privacy standpoint.

Phoenix's use of the Model-View-Controller design pattern was also a compelling factor in it's selection for implementing the OpenNym Web Service. The Model View Controller design pattern abstracts the functionality of a web framework into the following components [25]:

- **Model** - A model is a representation of the data of the application. It is a structured representation with a number of field names and associated field types. A model can also have defined relations with other models. There can be many instances of the same model in the same web service. For example, if there is a Nym model, multiple Nyms can be represented in different instances of the model.

- **View** - A view is a template responsible for displaying information requested by the user. Views are usually populated with information from the Model.

- **Controller** - A controller is responsible for handling user requests. In Phoenix, controllers handle requests for resources by retrieving relevant information from models and populating views. As such, the controller can be seen as the middleman between models and views, where the main application logic is performed.

The model-view-controller design pattern was advantageous for the OpenNym Web Service as the web service is primarily concerned with retrieving information as quickly as possible for users. There is no real complex application logic as the response to most requests involves retrieving information from the database, formatting it as JSON, and returning it to the user.

Web Service Models

Each of the database tables outlined in Section 3.2 were implemented as models. This was done primarily to keep application logic simple by abstracting different components into their own models. As such, each of the following models have the same fields as the tables they represent:

- **Identity Interface** - This model represents the mappings of usernames to Nym IDs for given domains.

- **Nym** - This model is a representation of Nyms.
- **Nym Metadata** - This model is a representation of the Nym metadata table. There should only ever be one entry in the Nym metadata table, so this model should only ever have one instance.
- **Rating** - This model represents a single rating object.
- **Rule** - This model represents a single rule object.
- **Session Cookies** - This model is a representation of the Session Cookies table. There should be an instance of this model for each Nym for each website supported by OpenNym.

Web Service Views

Views in the OpenNym Web Service are responsible for populating JSON templates with formatted data received from the controller. In the OpenNym Web Service, each controller had it's own view with which it would format response data into predefined templates. Each view contained multiple templates, and the template to be used is selected by the controller. After the template is populated, Phoenix sends it to the user in response to their request.

Web Service Controllers

In the OpenNym Web Service implementation, there was a controller responsible for handling requests to each API outlined in section 3.1. Each controller was responsible for parsing any supplied URL parameters from the request, retrieving the necessary data from the database models, formatting the data, and sending on the formatted data to the appropriate view, specifying which template should be used. Core application logic is performed in the controllers.

4.1.2 Database

The Database for the OpenNym Web Service was implemented in PostgreSQL [55]. PostgreSQL is an object relational database system, meaning it's a database with object orientation. This is an important feature of the database as Phoenix abstracts the database with the use of models and each model would need to be represented as an object in the database. Additionally, PostgreSQL is open source. This means anyone can review the source code, which is an important attribute to have from a privacy standpoint as it allows us to ensure the database is safe to use.

PostgreSQL also makes use of Structured Query Language (SQL). This is advantageous as the OpenNym service will deal with large amounts of homogeneous data. All information stored in the OpenNym database is structured with clearly defined fields and relations. Thus, SQL was the best choice for the service as it allows for quick retrieval of structured information. The structured nature of the information is also why Not-Only-SQL (No-SQL) databases were not used.

4.2 Browser Extension

The OpenNym browser extension was developed for Mozilla Firefox [54] using the web extensions API [36]. Firefox is a free, open source web browser developed by the Mozilla Corporation. As with Phoenix and PostgreSQL, a major factor in selecting Firefox was the open source nature of the project allowing users to ensure the browser is safe to use. Additionally, Firefox's adherence to the Browser Extensions Standard means the browser extension should also work on Google Chrome, Microsoft Edge, and Opera Web Browser [11].

The Firefox extension was designed with simplicity and performance in mind. As such, most of the functionality of the extension is hidden in the background JavaScript file, requiring little user input. The only input required from the user is selection of a Nym, and so there is a very simple user interface (UI) presented to the user in this extension where the user can select a Nym to browse as from a drop-down menu.

The Firefox extension was designed to have as little impact on browser performance as possible. This was primarily achieved by using as little blocking OpenNym API requests as possible and by caching responses from the OpenNym Web Service. By using mostly non-blocking (asynchronous) requests to the OpenNym Web Service, information can be retrieved in the background from the OpenNym Web Service independently of other browser requests.

The Firefox extension makes extensive use of caching by caching session cookies, Nym information, user's selected Nym, rating information, and OpenNym supported domains in Firefox's local storage mechanism. This allows the Firefox extension to quickly load important information from local storage instead of querying the server for all information, improving extension performance and user experience. The extension currently performs sanitation checks when the browser starts to ensure cached information is still valid. These sanitation checks require less information to be sent from the server and less background processing than outright retrieving all Nym information each time Firefox starts.

Application logic for the Firefox extension is programmed in JavaScript [35]. JavaScript is an asynchronous programming language, meaning that by default it does not make blocking

function calls. This is advantageous as it means JavaScript is not waiting on a response to a function to continue working, and so overall performance is improved. However, in the case of the OpenNym Firefox extension, a number of calls must be made to the OpenNym Web Service to retrieve information. These requests are performed by using XMLHttpRequests, a built in function to allow JavaScript to make web requests. In most cases it is okay to allow these XMLHttpRequests to be made asynchronously as the user will not be depending on the result of the requests to browse as part of a Nym. However, there are still a number of requests which are made by the browser extension for which a response is required before the user can continue to browse anonymously as part of a Nym. These requests must be made synchronously, where the browser must wait for a response to the XMLHttpRequests before continuing. The following is a list of scenarios where a synchronous (or blocking) request must be made:

- When the user sends a request to a domain supported by OpenNym for which the Nym's session cookies for the domain have not already been cached. The request must be blocked until the session cookies for the Nym can be retrieved from the OpenNym Web Service.
- When the user rates an item for which the average rating has not already been cached. The user rating request must be blocked until the item rating can be retrieved from the OpenNym Web Service and an average rating for the Nym calculated.

4.3 Threat Model

This section is concerned with the security concerns associated with the OpenNym system as a whole. OpenNym was designed to improve user privacy, but the system must be secure and communications private to ensure user information remains private. As such, some of the major threats to OpenNym are:

- Transport security
- Server Security
- Non-Private Ratings
- Non-Private Nym Selection
- Lack of Anonymity
- Loss of Rating Integrity

4.3.1 Transport Security

Transport security is concerned with the privacy of information in transit. The OpenNym system relies heavily on communication between the Firefox extension and the OpenNym Web Service. If this communication occurs over HTTP then the connection is not private and user information can be intercepted in transit to and from the server.

To solve this problem, transport security should be employed in the form of Transport Layer Security (TLS) version 1.2. Use of TLSv1.2 would allow communications between the Firefox extension and the OpenNym Web Service to take place using HTTPS, ensuring all transmitted information is encrypted. Additionally, TLSv1.2 would allow message integrity to be verified. To make use of TLSv1.2, the server must obtain a Certificate Authority (CA) signed Certificate to verify that it is the OpenNym Web Service and allow users to verify the validity of the server's public key.

4.3.2 Server Security

The server is responsible for communicating OpenNym information to all users and storing most of the rating information. The server acts as a trusted entity in the service, meaning users can be communicating rating information to the server and must trust the server is secure and will not communicate this rating information to third parties. As such, it is important to ensure the server is secure and make it as difficult as possible to compromise.

This can be achieved by using the following security best practices:

- Use private key files to log in to the server, do not allow login using passwords.
- Restrict the IP addresses that can log in to the server.
- Use strong, secure, unique passwords with a combination of alphanumeric characters and symbols for the database and other password protected internal entities. No part of these passwords should appear in a dictionary.
- Change the CA Certificate for the server periodically.
- Use HTTPS for all communications.
- Do not support outdated cypher suites for TLS [5].
- Do not store any personally identifiable information, such as IP addresses. Ensure no IP addresses appear in server log files.
- Ensure software is up to date and the latest security patches are installed.
- Input sanitisation should also be performed to prevent against SQL Injection Attacks.

4.3.3 Private Ratings

A user's interests can be pieced together by monitoring outgoing rating requests from the Firefox extension. There are two possible solutions to this issue. The first is more concerned with privacy, and involves requesting multiple items at a time so even if browser requests are being monitored it is more difficult to establish which item exactly is being consumed by the user. Additionally, Nym item ratings are only requested when an item is being rated by the user. This reduces the number of rating requests to the OpenNym web service to essential requests only.

The second solution involves using transport security as discussed in section 4.3.1. By using TLSv1.2, both the Uniform Resource Locator (URL) path and URL parameters are encrypted and cannot be seen in transport. Additionally, HTTPS request bodies are also encrypted so PUT requests to update Nym ratings should also be secure.

The first solution is also useful if a user would like to obfuscate their interests from the OpenNym Web Service as well, but any ratings the user makes must be communicated back to the OpenNym Web Service. As such it must be treated as a trusted entity.

4.3.4 Private Nym Selection

Nym selection is carried out locally on the browser through manual user selection. When a Nym is selected, a number of requests are made by the Firefox extension to retrieve information pertaining to the selected Nym. This constitutes a privacy concern if the user wishes to keep their Nym selection private from even the OpenNym Web Service. While the OpenNym Web Service is supposed to be considered a trusted entity, an enhanced level of user privacy can still be obtained by requesting information pertaining to several Nyms (including the Nym selected by the user) at once. This makes it more difficult to determine which Nym was selected by the user, but would also require all subsequent requests to the OpenNym web service to be made for multiple Nyms.

4.3.5 Rating Anonymity

OpenNym provides users with an enhanced level of privacy by allowing them to 'hide in the crowd'. However, not every item has been consumed by enough users to allow this enhanced privacy through hiding in the crowd. This privacy concern is especially pronounced in the case of a user rating an item previously unknown to the Nym, a 'new' item. By being the first user to rate the item, the OpenNym Web Service knows exactly what the user's individual rating for the item is.

The issue with this concern is that for all new items there needs to be a first rating. Though not a solution to the problem, users can be made aware of this issue by having a popup notification appear, warning them that by rating an item they will not have an enhanced level of privacy. Users can then opt to not rate an item.

4.3.6 Rating Integrity

OpenNym does not require users to authenticate themselves in order to use the service. This was deemed necessary to keep the service as open and private as possible. The issue with this however is that it makes it possible for anyone to modify the ratings for any item stored on the OpenNym Web Service. A potential bad actor can flood the server with rating updates to manipulate the ratings of certain items according to their own agenda.

Since users cannot be authenticated before making requests, it is difficult to prevent users from tampering with ratings in this way. One method that could work is to rate limit IP addresses if they send too many requests to the OpenNym web service too quickly. However it is trivial for a motivated attacker to spoof the source IP address.

Another approach is to simply perform damage control. Regular backups of ratings should be made such that ratings that are suspected of being tampered with can be restored from a previous backup. A machine learning model could be developed in the future to aid with tamper detection.

4.3.7 Availability

OpenNym can only provide users with privacy enhanced browsing when the web service is available. As such, a number of precautions can be taken to make the service as available as possible:

- Take regular backups of database contents and store them off site.
- Set up multiple servers in different geographical locations in case of the event of a disaster occurring in one of the server locations.
- Try to mitigate denial of service by investing in Denial of Service protection from a provider such as Cloudflare.

Chapter 5

User Clustering

To achieve the final research objective of demonstrating the functioning OpenNym system, we cluster users in The Million Song Dataset (MSD) [8] and use the resulting Nyms to generate recommendations from Spotify [52].

The Million Song Dataset is a part of The Million Song Dataset Challenge, a challenge that was run in the beginning of 2012 to allow researchers to develop state of the art music recommender systems based on a massive collection of anonymised user's song histories and large amounts of relevant metadata. This dataset does not have explicit user ratings for each song a user has listened to. As a result, we use an implicit feedback mechanism to give each user-song pair a rating. To do this, we use the underlying assumption that the more the user likes a song, the more they will listen to it. Based on this assumption, we can extrapolate that the song with the most play counts from a user is the user's favourite song.

This chapter will outline how data from the Million Song Dataset was clustered using BLC Matrix Factorization and how ratings for each resulting Nym were generated. Section 6.3 contains an in depth analysis of the resulting Nyms, the distribution of users between Nyms, the top rated items for each Nym, and a comparative analysis of the accuracy of recommendations generated by the Spotify recommender system for both individual users and Nyms.

Python [42] was used for all work in this section. Python was chosen primarily because of it's NumPy Library [38], it's native Spotify API support [29], and it's ease of use.

5.1 Data Analysis

As mentioned in the previous section, the Million Song Dataset is a collection of user playback histories that is accompanied by large amounts of song metadata from various sources.

All users playback histories are available in the `train_triplets.txt` file[1], whereby the file contains a list of triplets in the format `<user, song, play count>`. An example of the formatting of this can be seen in listing 13 in the appendix. Analysis of this file shows there are:

- 1,019,318 users
- 384,546 unique MSD songs
- 48,373,586 triplets

Furthermore, each user in the file is represented by an anonymous alphanumeric ID and each song is represented by a unique MSD Song ID. Each MSD Song ID can be mapped to an actual song name and artist using the `unique_tracks.txt` file[2]. A small subset of this file can be seen in listing 14 in the appendix.

The following are some important figures regarding the data:

- Number of unique songs listened to by users:
  - Minimum: 3
  - Maximum: 4,316
  - Mean: 45
  - Median: 26
- Total play counts of users:
  - Minimum: 3
  - Maximum: 13074
  - Mean: 123
  - Median 69

5.2 Data Cleaning

The initial dataset provided in the MSD challenge was released with an error affecting how some unique song IDs were matched to the wrong track ID in the MSD [34]. Track IDs in the MSD are the base identifier by which everything else is associated with it. Each song in the MSD is associated with a track, and each track then has a wealth of metadata associated

[1] http://labrosa.ee.columbia.edu/millionsong/sites/default/files/challenge/train_triplets.txt.zip

[2] https://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/unique_tracks.txt

with it. An issue arose where it was found that songs were being incorrectly associated with tracks in the dataset.

As a result of this error, a list of affected song IDs were released to ensure there is no incorrect information in the dataset used for research[3]. This list was used in the Data Cleaning stage to filter out all triplets in `train_triplets.txt` that were incorrectly matched. This resulted in a loss of 2,578,486 triplets. This is the only Data Cleaning that was performed, and the code for data cleaning can be found in the `data_cleaner.py` script in the accompanying source code.

5.3 Data Preprocessing

This section is concerned with taking the sanitized `train_triplets.txt` file and formatting it such that it can be input into the BLC clustering algorithm. There are a number of steps involved in preprocessing the data:

1. Parsing `train_triplets.txt` such that each user's songs and play counts are easily accessible.
2. Normalizing play counts.
3. Generating a sparse matrix to input to the BLC clustering algorithm.
4. Filtering sparse songs from the matrix.

5.3.1 Data Parsing

As previously mentioned, the dataset was made available in the form of a list of `<user,song,play count>` triplets. Before any form of processing could be carried out on the dataset, first the data needed to be loaded into an appropriate data structure such that there would only be one reference to a user. Each user reference would then point to a list of the users songs and play counts.

To achieve this, a standard Python dictionary was used where the key was the unique user ID. The value for each user key in this dictionary was a list of tuples in the form `(song id, play count)`. A dictionary was used because keys in a dictionary have constant lookup time, allowing for much improved performance over the use of a data structure such as a Python list. An example of the resulting dictionary can be seen in Listing 11.

[3] https://labrosa.ee.columbia.edu/millionsong/sites/default/files/tasteprofile/sid_mismatches.txt

```
{
    "user_1": [("song_a", 1), ("song_b", 2), ("song_c", 3)],
    "user_2": [("song_d", 4), ("song_b", 5), ("song_f", 6)],
    "user_3": [("song_a", 7), ("song_e", 8), ("song_c", 9)]
}
```

Listing 11: Structure of parsed dataset dictionary

5.3.2 Data Normalization

Naturally users in the dataset don't all have the same number of play counts. The play counts of each user was normalized between 0 and 1 to ensure there was a common metric to define a user's like for a song, where 0 represents a song listened to by a user once and 1 represents a user's most listened to song. By normalizing a user's play counts, each song effectively scores between 0 and 1 with a higher score meaning the user liked the song more. Normalizing the play counts also means every user in the dataset is using the same scale to represent how much they like each song.

5.3.3 Sparse Matrix Generation

The BLC clustering algorithm takes a sparse matrix of user-item ratings as input, where a sparse matrix is a matrix in which most of the elements are 0. Since the BLC Clustering algorithm is performing matrix factorization, it is important to use as dense a matrix as possible to give the clustering algorithm as much rating data as possible. To do this, we take the users with the highest total play counts for the sparse matrix. An example of a sparse matrix can be seen in table 5.1.

Given there are over 1 million users in the dataset, not every user could be included in the sparse matrix. This is because the memory requirements to create a 1,000,000 x 384,000 matrix were simply too high. Thus, to find an appropriate sparse matrix to input to the BLC clustering algorithm we vary the number of users used in the dataset and the minimum number of values that must be in each column (song) of the sparse matrix. It is important to note, the number of users taken are the top-N users in regards to total play count. The results of this investigation can be seen in table 5.2.

Additionally, all the normalized ratings used in the sparse matrix are scaled to an integer between 1 and 5 inclusive, where 1 represents the worst rating and 5 represents the best rating. This means the only possible values for each rating are 1, 2, 3, 4, and 5. A rating

scale of 1-5 was chosen because it's one of the most popular rating scales, and allows for a distinction between having an opinion on a song (2 or 4), having a strong opinion on a song (1 or 5), and having a neutral opinion on a song (3).

	King Kunta	Black Skinhead	Hot Thoughts	Rosa Parks
Alice	1	0	3	0
Bob	0	5	0	3
Colin	0	4	2	0
Dean	3	0	0	1

Table 5.1: Sparse Matrix with a subset of users ratings

5.3.4 Song Filtering

As discussed in the previous section, a goal when generating the sparse matrix is attempting to make it as dense as possible to improve matrix factorization performance. As such, columns (songs) with too few users are removed after the sparse matrix is generated. This is carried out after the sparse matrix is generated, and the removal of some columns results in the loss of some users completely. A comparison of the number of users lost and the resulting sparse matrices can be seen in table 5.2.

Users	Min Users	Users Lost	Songs Lost	Songs	Density
20 000	100	17	196 842	3 889	0.0117
20 000	200	147	199 649	1 262	0.0219
20 000	300	413	200 114	617	0.0325
30 000	100	15	217 078	5 756	0.0083
30 000	200	108	220 706	2 128	0.0148
30 000	300	357	221 767	1 067	0.0217
40 000	100	10	230 128	7 363	0.0066
40 000	200	97	234 573	2 918	0.0114
40 000	300	315	235 982	1 509	0.0164

Table 5.2: Comparison of sparse matrix properties for differing numbers of users and minimum number of users in each column

5.4 BLC Clustering

This section is concerned with the actual clustering of users based on their song ratings in the supplied sparse matrix. Given a sparse matrix of user-item ratings, the BLC clustering algorithm will continue to cluster users until it converges. Convergence in BLC occurs when a predefined total error threshold has been reached. The total error threshold was set as 0.1, though it was never reached in any clustering of users.

The BLC clustering algorithm performs a number of iterations to cluster users. In the first iteration, all users are assigned to one Nym. In each subsequent iteration, the number of available Nyms are doubled and users reassigned to the Nym most suited to them. A predefined maximum number of Nyms is also used when running the algorithm, where the algorithm stops iterating if the maximum numbers of Nyms has been reached without reaching the predefined total error threshold.

To evaluate the performance of the clustering, root mean square error (RMSE) is used to calculate the factorisation RMSE and the prediction RMSE is used. RMSE is used to measure the predictive performance of the BLC recommender algorithm after clustering users, initially with a small subset of the training data (factorisation RMSE), and then with a test set (prediction RMSE). In each case of evaluating clustering performance, the test set for calculating the prediction RMSE represented 20% of the input sparse matrix. The remaining 80% was used to train the clustering algorithm. RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

where e is the error in the predicted rating for a user and n is the total number of users in the set being evaluated.

To decide which sparse matrix input to use, a comparison of all the inputs was made by running the BLC clustering algorithm on all the inputs listed in table 5.2 three times with a maximum number of 16 Nyms. Of each input's three iterations, the best factorisation RMSE and prediction RMSE were taken for comparison. These results can be seen in table 5.3.

Based on the results in table 5.3, there is a variation of ≈ 0.728 in the Factorisation RMSE, and a variation of ≈ 0.452 in the prediction RMSE. The best input based on factorisation RMSE is the sparse matrix with 20,000 users and a minimum of 300 users in each column. The best input based on prediction RMSE is the sparse matrix with 20,000 users and a minimum of 100 users in each column. Despite this, the sparse matrix selected for use in this research was the one with 40,000 users and a minimum of 100 songs. This sparse matrix was selected because despite it's lower density and poor factorisation RMSE, it's performance was

average in prediction RMSE despite having a much lower matrix density and a much higher number of columns (7,363). By using a sparse matrix with a larger number of columns, more user interests are represented through a larger selection of songs.

Users	Min Users	Max Nyms	Factorisation RMSE	Prediction RMSE
20 000	100	16	0.5645	0.7412
20 000	200	16	0.5605	0.7523
20 000	300	16	0.5449	0.7702
30 000	100	16	0.5989	0.7614
30 000	200	16	0.5879	0.7648
30 000	300	16	0.5723	0.7584
40 000	100	16	0.6177	0.7622
40 000	200	16	0.6083	0.7681
40 000	300	16	0.5974	0.7864

Table 5.3: Comparison of clustering predictive performances for different sparse matrices

5.5 Nym Construction

The result of the BLC clustering algorithm is the matrix P^T which maps users to Nyms. To construct each Nym, a list of every unique song listened to by the Nym's users must be made and the play counts for each song accumulated. This is represented in the same format as in listing 11, where Nyms are used in place of users and play counts for the song are the cumulative play counts for the song of all the users in the Nym.

5.6 Nym Rating Generation

Nym-song ratings then have to be generated for each song listened to by a Nym's users. Individual user-song ratings were calculated on the assumption that the more a user listens to a song the more they like it. While this assumption is useful in identifying individual user's tastes, an issue arises with song popularity when taking just play counts into account for Nym ratings.

When rating songs for Nyms based on play counts alone, it was found that the top 10 songs for each Nym had a substantial amount of overlap. This overlap was caused by songs that were hugely popular in the years preceding the release of the dataset, songs that would have been well known to almost all users in the dataset. While not every user in a Nym would have listened to these popular songs enough for them to be in their individual top 10

rankings, these songs were listened to by enough users for them to cumulatively make it into the top 10 for each Nym.

Since a Nym's top ratings should be representative of the interest of users in the Nym, a rating method that downplayed the influence of song popularity was developed. This rating method operates on the basis that songs that are rated consistently highly by the Nym's users are more Representative of the Nym's interests. To evaluate the consistency of ratings for songs in each Nym, the variance in the Nym's user ratings for the song is measured.

The initial approach for the rating method was to simply subtract the variance in a each song's rating from the Nym's mean rating to create an adjusted rating. However, it became evident that songs with more listeners tended to have a higher variance than songs with fewer listeners. This meant that songs with more listeners were more heavily penalised than songs with fewer listeners and would end up being rated relatively poorly. To alleviate this problem, the rating method was adjusted to take the number of listeners for each song in a Nym into account and applying a weighted penalty to every song in the Nym that has less listeners than the song with the most listeners in the Nym. The penalty was more severe the less listeners the song had. Each song's individual weighted penalty was then added to the song's rating variance. The pseudocode for the adjusted rating method is represented by listing 12.

```
    for each Nym in all_nyms {
        # Calculate the range in the variance for the Nym
        max_variance = get_largest_song_rating_variance()
        min_variance = get_lowest_song_rating_variance()
        variance_range = max_variance - min_variance

        # Get numbers of users who listened to each song
        max_listeners = get_maximum_song_listeners()
        min_listeners = get_minimum_song_listeners()

        for each song in Nym {
            num_listeners = normalize(song_num_listeners,
            ↪  min_listeners, max_listeners)

            penalty = variance_range * (1 - num_listeners)
            weighted_penalty = penalty * 0.1

            # mean_rating is the average of Nym users' ratings
            adjusted_rating = mean_rating - weighted_penalty
        }
    }
}
```

Listing 12: Nym adjusted rating method

With reference to listing 12:

- The rating variance range is used to calculate the penalty because in theory it will always provide a penalty factor that is relevant to the ratings in the Nym.
- Listeners in the listing refer to the number of users who have listened to a song.
- num_listeners is the number of listeners for each song normalized between 0 and 1 inclusive. Values are normalized only in the context of other songs in the Nym.
- num_listeners is subtracted from 1 to ensure that songs with less listeners incur a higher penalty.

- The penalty is then weighted using the factor 0.1 because using the penalty alone was too harsh. 0.1 was selected through trial and error by examining the resulting Nym song rankings using varying penalty values and selecting the value that saw the most even spread of songs with high and low play counts in the resulting rankings.

The final song ranking was generated based on each song's adjusted rating, with a higher rating being better. Nym-song ratings can then be generated on the assumption that songs that rank higher are more representative of the Nym's interests.

Chapter 6

Evaluation

This chapter will evaluate the performance of the OpenNym system implemented in chapter 4 and the Nyms created in chapter 5. It will discuss the results of load testing the OpenNym server, measuring the impact of the OpenNym browser extension on Firefox's performance, and an analysis of the Nyms generated in chapter 5 and the accuracy of recommendations made for the Nyms.

6.1 Server Performance

To evaluate the server performance, we will investigate two areas:

- Response Latency
- Server Throughput

The goal of investigating these two areas is to gain an understanding of user experience when interacting with the server and the servers current limitations. For the purpose of this test, the server was an Amazon Web Service instance with the following specifications:

- **Operating System** - Ubuntu 16.04
- **CPU** - 1 Virtual CPU, 2.5GHz Intel Xeon Processor
- **Memory** - 2 Gigabytes

All requests to the service were sent from a different machine using the open source load testing library Locust [32]. Locust allows the simulation of a number of concurrent users accessing an endpoint, and additionally allows the addition of randomness of time in between user requests. As such, Locust was used to simulate varying numbers of concurrent users querying the web service, with users randomly sending requests between 500 milliseconds

and 1 second after receiving their last response. To evaluate latency and throughput, one endpoint from each API will be selected to be queried by all concurrent users. The endpoint used will be the endpoint that has the most expensive Input/Output (I/O) operations, and in theory the endpoint that should have the worst performance. The selected endpoint for each Nym can be seen in table 6.1. Any parameters required by the endpoint are randomly selected. Additionally, by evaluating each endpoint individually we gain insight into the relative performance of each API.

API	Method	Endpoint
Nym	GET	/nym/
Rating	GET	/ratings/:nym_id/:domain
Session Cookies	GET	/cookies/:nym_id
Rules	GET	/rules/top/:nym_id
Identity	GET	/identity/:domain

Table 6.1: Selected endpoints for Each API based on expected number of I/O operations

Though the Identity API is listed in table 6.1, it's performance evaluation is not included in any of the comparison figures shown later in the section. The reason for this is it performed exceedingly poorly, with request failure rates in excess of 50% for the lowest numbers of concurrent users. On inspection of the performance of this endpoint it was found it was returning JSON payloads in excess of 291KB, which is significantly larger than the 15KB sent by the Nym API which is the next largest response. In it's current form, this API will not be very performant. Possible improvements to this API will be discussed in the conclusion for the web server in section 7.1.2.

It is important to note the response failure rate for each of the APIs are not graphed below because the failure rate never exceeded 0.005% for any of the APIs (excluding the Identity Interface API), where a failure is defined as the server returning any 5xx HTTP response code.

6.1.1 Response Latency

Latency is a measure of how fast a server responds to a client. It is calculated by measuring the time between when a client sends a request and when the client receives a response. Latency plays a major role in user experience, especially in the case of OpenNym where user requests can be blocked while waiting for responses from the server. If response latency is high, this will impact page load times and result in a poor user experience. It may even

discourage users from using the OpenNym service due to it adversely affecting their browsing experience.

Figure 6.1 shows a comparison of the mean response latency of the 4 assessed APIs. As can be seen from the figure, latency for the Nym API is substantially higher than any of the other APIs. This can be attributed to the queried endpoint returning the top ratings for each Nym available on the server, resulting in a large number of I/O operations to retrieve rating information for each Nym. While the Nym API is expected to be queried less than the other APIs, querying it is clearly quite a costly function and this must be addressed to improve overall performance and efficiency of the server. One possible way of improving this performance is to cache the response to the endpoint, as the response does not change for individual user requests. By caching the response, the number of I/O operations to build the response is drastically reduced. A more detailed view of the APIs excluding the Nym API can be seen in figure 6.2, where each API's mean latency is considerably better than the Nym API.

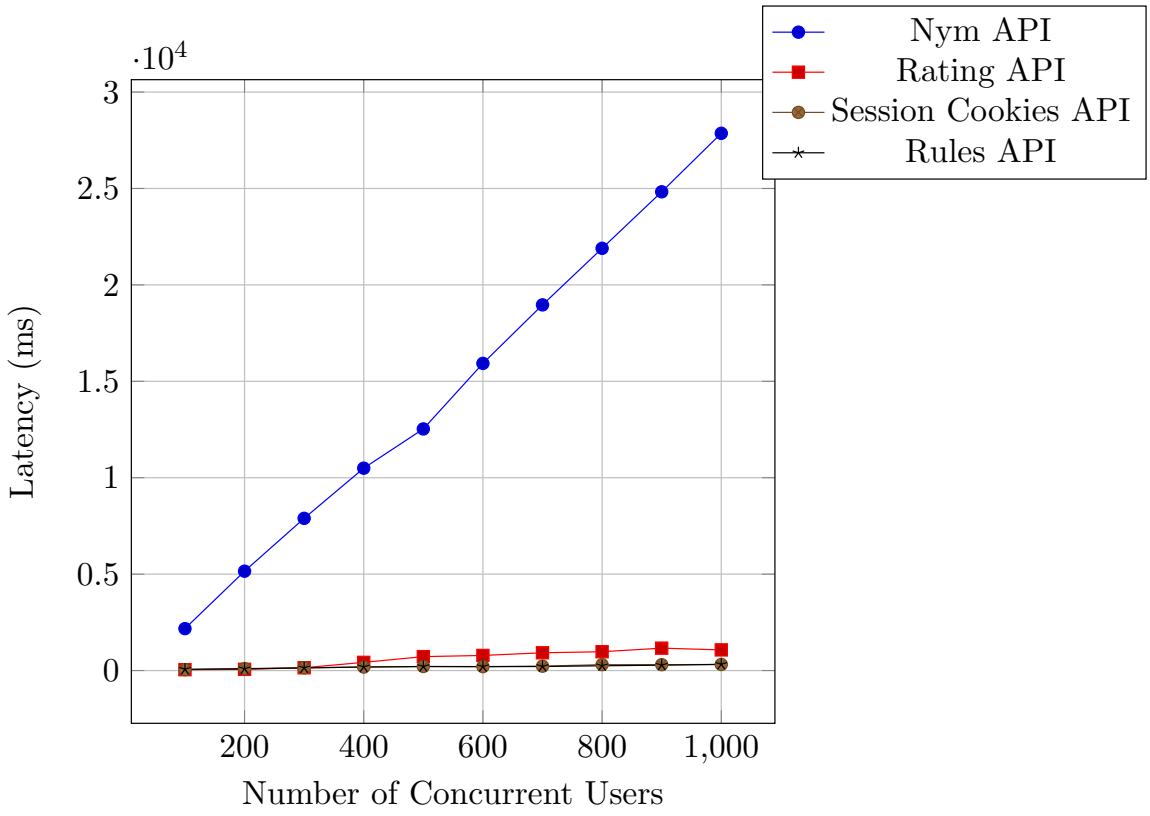


Figure 6.1: A comparison of mean latency between OpenNym APIs

In figure 6.2 we can see that both the Session Cookies and Rules APIs perform extremely well under load, with latencies below 400ms with 1,000 concurrent users. The Rating API does not perform as well as the other two APIs, but it does not perform poorly either. The endpoint used for load testing the Rating API will only be called asynchronously by the

browser extension, meaning responses to this endpoint are not time critical. Additionally, this endpoint requires the server to analyse every rating for a Nym for a domain and determine the 10 most popular ratings to return. Single item requests to this API by the Firefox extension will not have nearly as high an overhead. For these reasons, the Rating API's performance is acceptable.

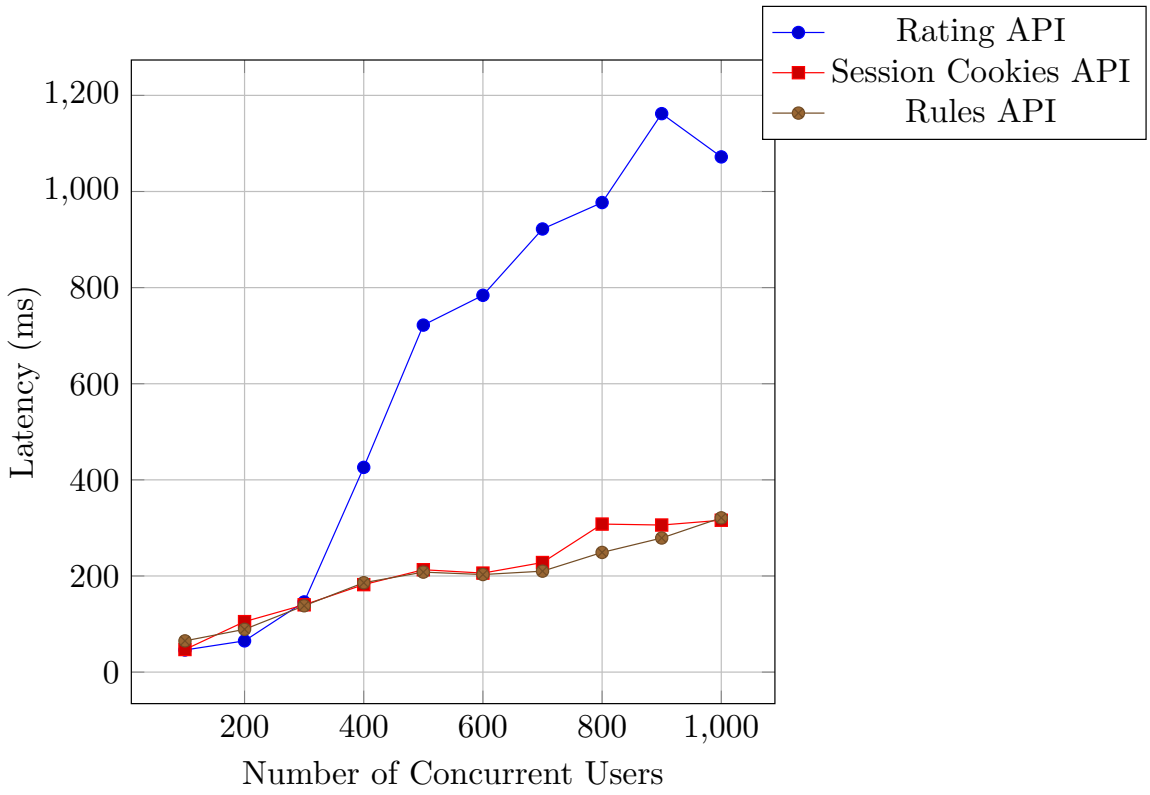


Figure 6.2: A detailed comparison of OpenNym API mean latencies excluding the Nym API

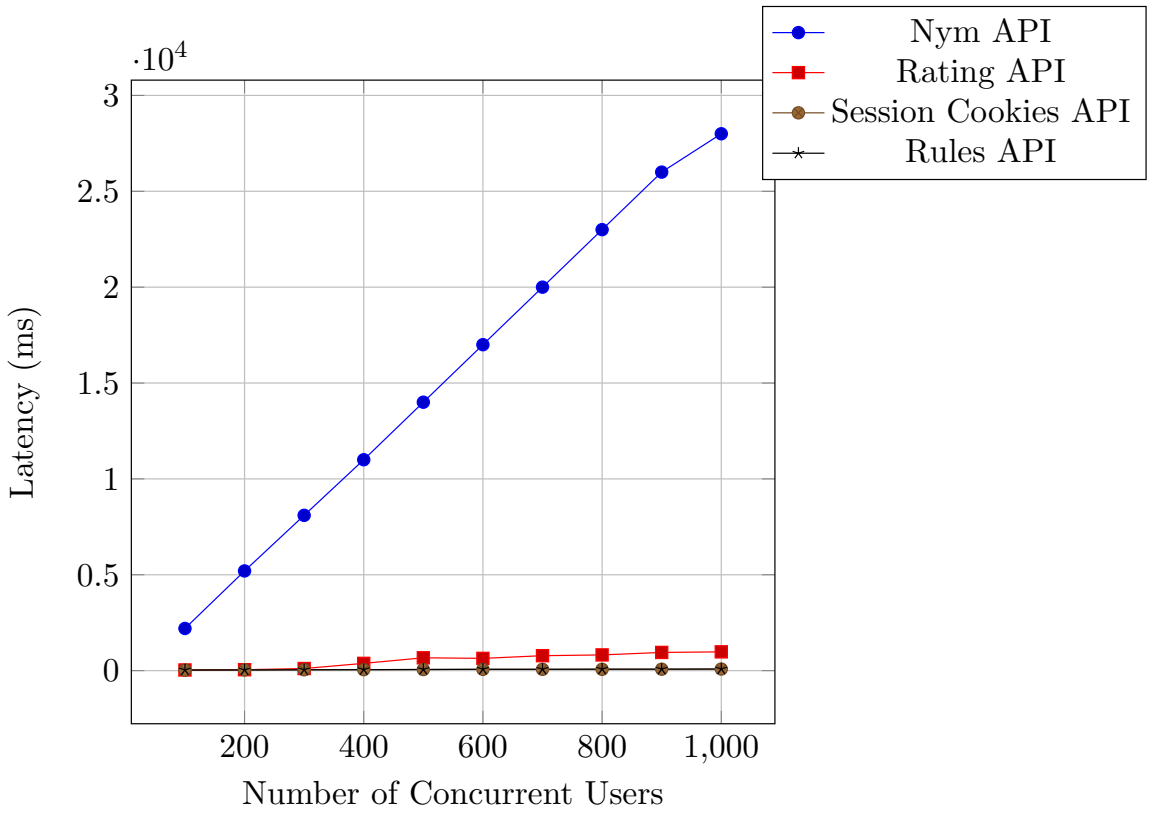


Figure 6.3: A comparison of median latency between OpenNym APIs

As discussed above, the poor performance of the Nym API in figure 6.3 can be attributed to the number of I/O operations it must perform to retrieve rating information for each Nym's top ratings. A more detailed comparison of the APIs excluding the Nym API can be seen in figure 6.4. This figure shows performance of the other three APIs are generally good, especially in the case of the Session Cookies API which needs to perform well due to the Firefox extension using blocking requests to retrieve session cookies. While the rating API performs worse with higher numbers of concurrent users than the other two APIs, it still returns valid responses within 1 second which is acceptable given the use of non-blocking requests by the Firefox extension for this rating API endpoint.

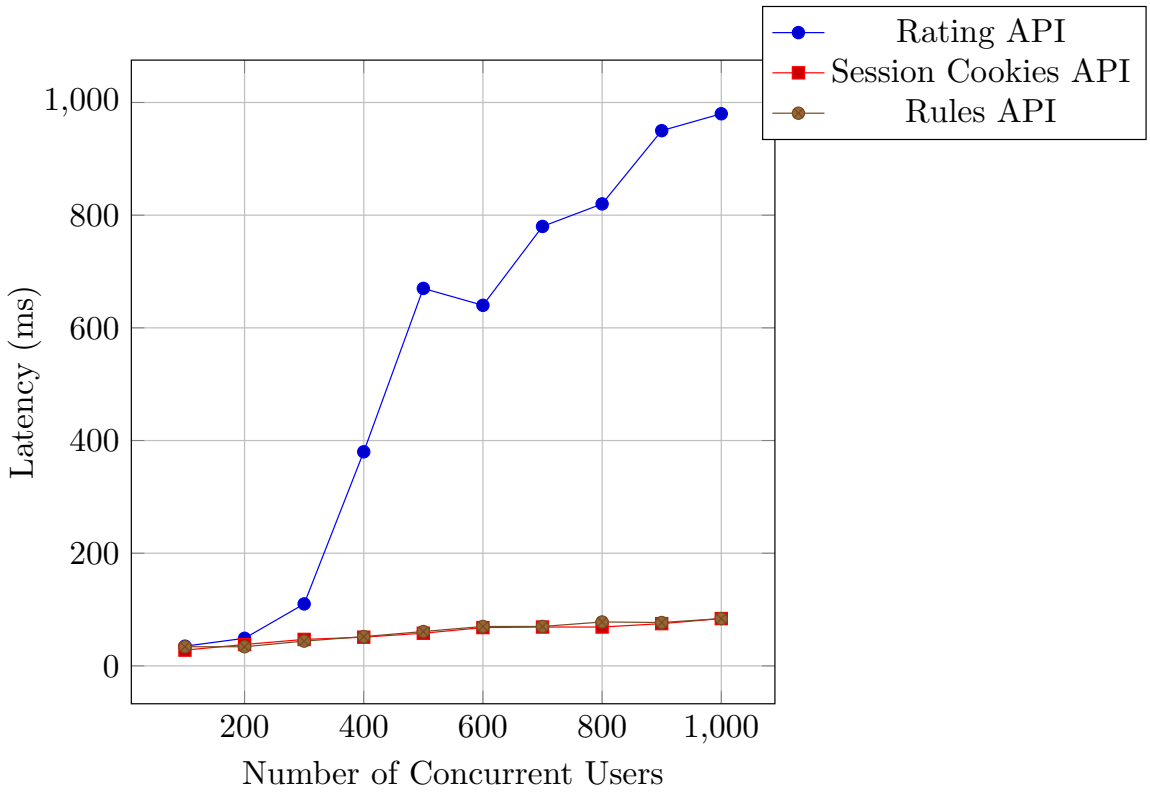


Figure 6.4: A detailed comparison of OpenNym API median latencies excluding the Nym API

6.1.2 Server Throughput

Server Throughput is a measure of how many requests a server can handle during a specific time interval. Requests per Second (RPS) will be used to evaluate server throughput. To evaluate the server throughput, varying numbers of concurrent users will be used to measure how many requests the server successfully responds to per second. Server throughput will be evaluated in the same way as latency, using the same endpoints to represent each API as outlined in table 6.1. The results of server throughput for each API can be seen in figure 6.5

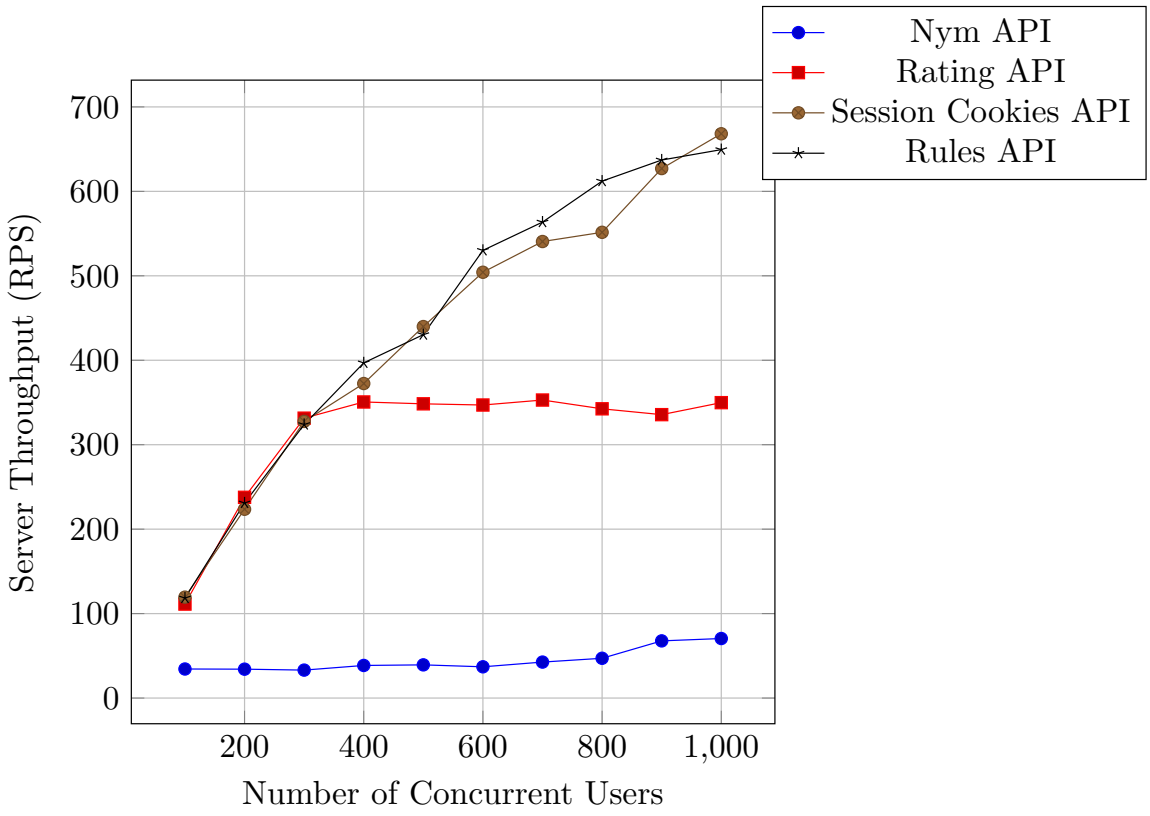


Figure 6.5: Comparison of server throughput for the different APIs

As can be seen in figure 6.5, the Session Cookies API and Rules API perform exceedingly well in server throughput with total RPS at 1,000 concurrent users $\approx$ 650 RPS. The Rating API performs moderately well with a score of $\approx$ 350 RPS, and the Nym API performs quite poorly with a score of $\approx$ 30 RPS.

6.2 Extension Performance

Extension performance is concerned with the impact of the Firefox extension on user load times, both for web pages and rating requests. As discussed in section 4.2, there are blocking requests made to the OpenNym web service where the Firefox extension temporarily stops a user's request to load a web page or rate an item while information is retrieved from the OpenNym web service. These blocking requests effectively increase the latency of the user's requests. While OpenNym web service response latency is investigated in Section 6.1.1, this section will investigate the added latency of the Firefox extension when a user browses to a website and rates items under three different conditions:

- User Browsing Individually (not in a Nym).
- User Browsing as part of a Nym with no cached information.

- User Browsing as part of a Nym with cached information.

As the blocking requests are the only requests that should have a real impact on user request latency, they will be the only requests evaluated. There are only two blocking requests made by the Firefox extension, and they will be evaluated as follows:

- **Session Cookies Request** - Spotify[1] will be loaded under each of the three conditions outlined above. This will be repeated five times for each condition, with the average page load time under each condition taken.
- **Rating Item Request** - Five different videos on YouTube.com will be rated by 'liking' the videos under each of the three conditions outlined above. This will be repeated five times for each condition, with the average rating request latency for all the videos taken. YouTube.com was used for this evaluation because Spotify had no clear explicit feedback mechanisms. As such, YouTube.com is used in lieu due to it's easily accessible like endpoint.

To measure the page load times and request latency, the built in Firefox network monitor will be used. The Firefox network monitor gives comprehensive information for individual requests and entire page loads. This will allow accurate results to be obtained. Additionally, there will be no extensions installed on Firefox when evaluating performance while browsing without a Nym. When evaluating performance of a user using a Nym, the OpenNym extension will be the only installed extension. Additionally, the Firefox browser cache will be disabled for all tests.

[1] https://open.spotify.com/browse/featured

6.2.1 Page Load Performance

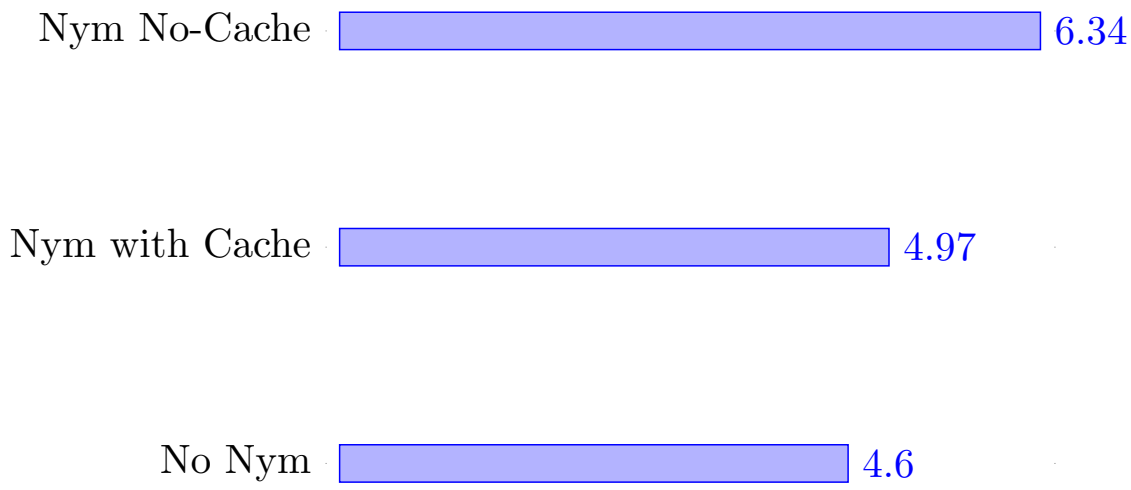


Figure 6.6: A Comparison of average page load times in seconds on Spotify

Figure 6.6 shows the impact of browsing as part of a Nym both with and without cached cookies. As can be seen in the figure, browsing as part of a Nym with no cookies cached results in an increased page load time of $\approx 40\%$ over page load times with no Nym. This overhead is much less pronounced when the session cookies are cached, with an increased page load time of $\approx 8\%$ over browsing with no Nym.

6.2.2 Rating Request Performance

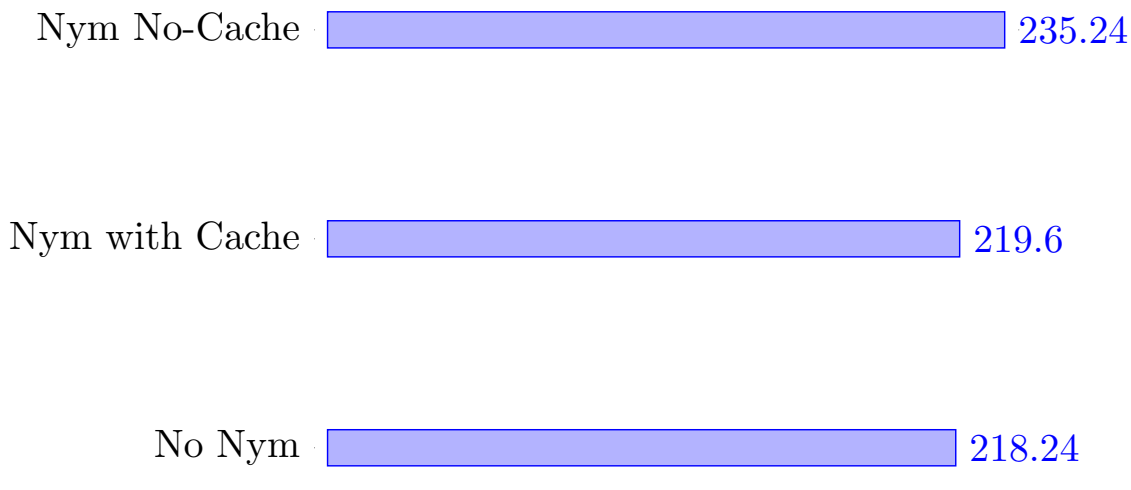


Figure 6.7: A Comparison of average rating request latency on YouTube in milliseconds

Figure 6.7 shows a comparison of average rating request latency in each of the predefined browser conditions. The impact caused by the OpenNym Firefox extension is minimal in both cases when compared to rating request latency when not using the Firefox extension. Additionally, the Firefox extension with cached ratings performs better than the Firefox extension with no cached ratings. This is in line with expectations as only one request is made to the OpenNym web service rather than two.

6.3 Nym Analysis

This section will cover a brief analysis of the properties of the Nyms produced in chapter 5. Figure 6.8 shows the number of users assigned to each Nym. As can be seen in this figure, user assignment is not uniform and Nym 0 in particular has substantially more users than other Nyms. This behaviour is expected as users are clustered based on interest and it's expected that some groups are more popular than others. Additionally, Nym 11 was assigned no users and Nym 13 assigned only 12 users. Nym 11 is not made available to users as a result, and Nym 13 is found to perform quite poorly when generating recommendations as discussed later in section 6.4.

In each of the below figures, the mean is represented by a solid line and the median is represented by a dashed line.

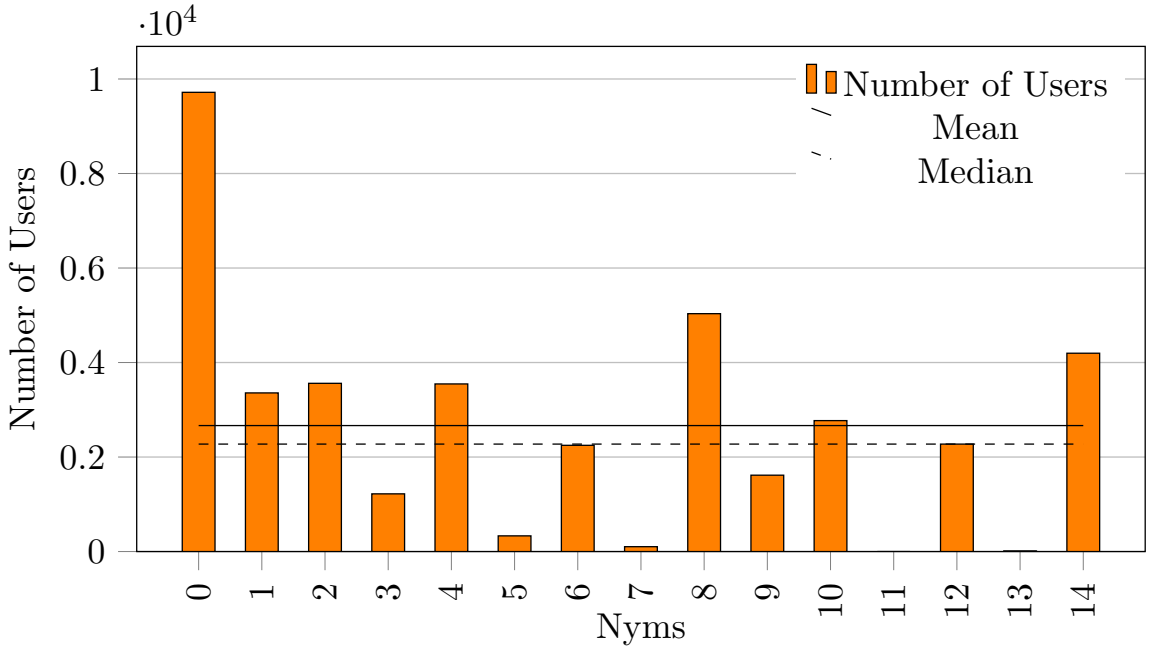


Figure 6.8: An analysis of the number of users assigned to each Nym

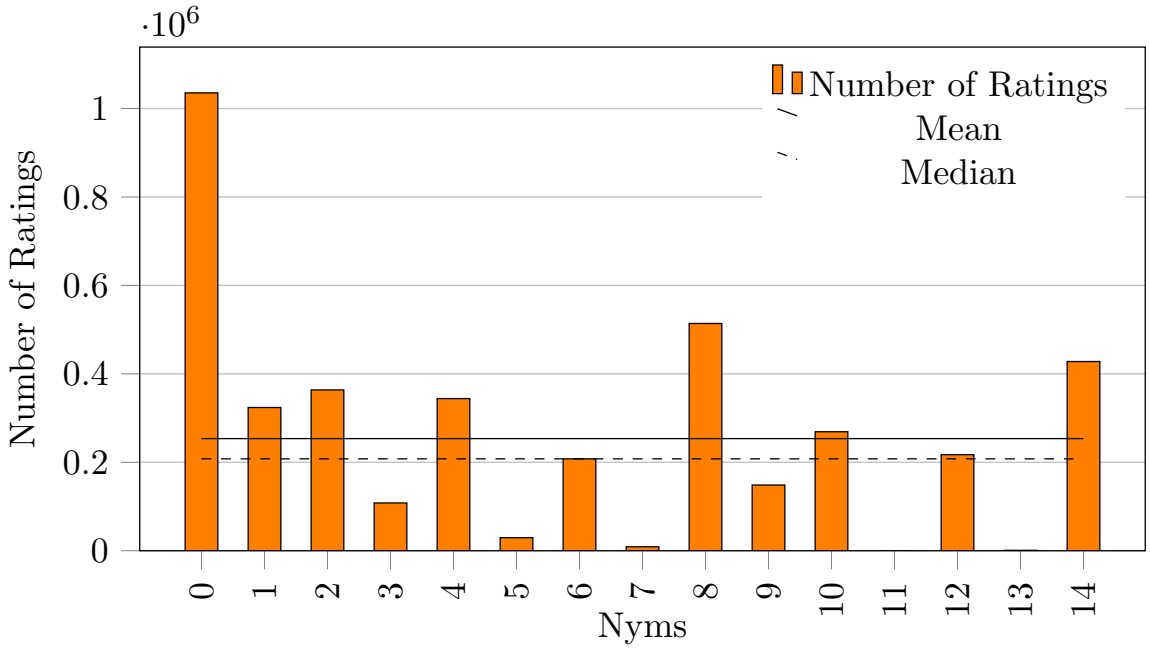


Figure 6.9: An analysis of the total ratings available for each Nym from it's users

Figure 6.9 shows the number of user ratings available to each Nym to generate Nym ratings. The total number of ratings available to each Nym generally correlates to the number of users in each Nym, as represented in figure 6.8.

Figure 6.10 shows the number of unique artists listened to by each Nym. It is important to note that despite quite large variations in the number of users in each Nym, there is much less variation in the number of artists in each Nym, where an artist listened to by a Nym can be an artist listened to by only one of the users clustered with the Nym. It is also important to note that this is not restricted to only the 7,363 songs used to generate the Nyms. The reasons for showing the number of artists as opposed to the number of songs listened to by each Nym are discussed in section 6.4.

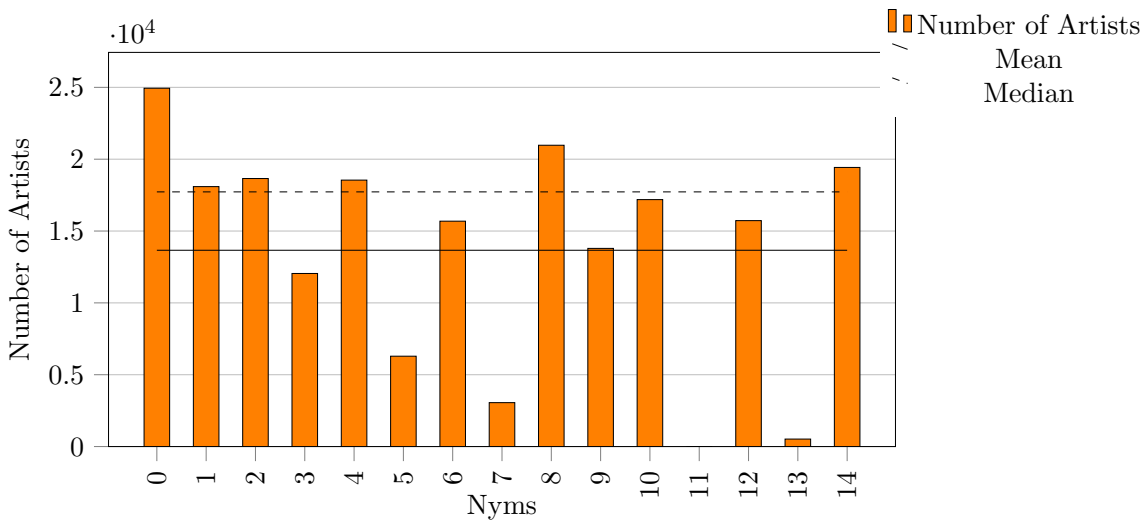


Figure 6.10: An Analysis of the Number of artists listened to by each Nym

6.4 Recommendation Performance

In chapter 5 we discussed how Nyms and their ratings were created using the Million Song Dataset. This section will describe the process by which music recommendations were obtained from the Spotify recommender system, and we will evaluate the accuracy of the recommendations for Nyms versus the recommendations for the top 5 users of each Nym. The top 5 users refers to the 5 users with the highest play counts in each Nym.

To evaluate recommendation accuracy, we will use the precision and recall metrics to generate an F-measure. These metrics were chosen because they accurately measure the relevance of recommendations generated by the Spotify recommender system. Additionally, they are very suitable metrics as recommendations returned by the Spotify recommender system are unranked, meaning no song recommendation is of a higher value than another.

6.4.1 Generating Recommendations

Spotify was selected to act as the music recommender system for three main reasons. The first is that it is a widely used service with 170 million monthly active users [51]. Secondly, as one of the largest music streaming services available it would have invested heavily in improving the performance of it's music recommender system. Finally, Spotify opened access to developers to access their service through extensive developer APIs. This made it much easier to generate recommendations from the Spotify service.

Spotify have made available an endpoint[2] on their developer API through which up to 100 song recommendations at a time can be retrieved. Requests to this endpoint can include

[2]GET https://api.spotify.com/v1/recommendations

up to 5 seeds, where a seed can be a track, artist, or genre. The resulting 100 song recommendations from the API are not ranked, and subsequent requests to the endpoint are performed independently of previous requests and can return songs that were already recommended in a previous request.

When generating recommendations for evaluation, there were two main challenges that needed to be overcome:

- Seeds for generating recommendations from the Spotify API must be in the form of valid Spotify Uniform Resource Identifiers (URIs).
- The Million Song Dataset was released in late 2011, but the Spotify API will recommend songs from post-2011. This makes it impossible to evaluate the relevance of recommendations for songs recommended post-2011 as we don't know if the song is relevant or not.

In order to tackle the issue of using Spotify URIs to generate recommendations, a Python script was created to automatically map song artists to an artist URI using the Spotify API's search [3] endpoint. Artists were used as the seeds for song recommendations because when mapping song names to URIs, there was no way to effectively distinguish a song from it's live version, a remastered version, a cover version, a remix credited to the same artist, and other variations of the same song. By mapping the artist instead of the song name, the accuracy of the mappings were much higher.

To work around the age of the dataset, the Python evaluation script would iterate through each song recommended by the Spotify API, retrieve a list of albums released by the song's artist[4], and check that at least one album was released by the artist before 2012. If the artist hadn't released an album before 2012, then the recommendation is considered invalid and is discarded. The reasoning behind this is that if the artist had released music when the dataset was released, then the artist would be a valid recommendation for the time. Using artists instead of songs was also beneficial for this step, as it reduced the scope for error when querying the API. This is because there was no longer the risk of songs being associated with the incorrect artist, resulting in incorrect data being retrieved from the Spotify API.

To try maximise artist to URI mapping accuracy, standard information retrieval techniques including stemming and removal of special characters were used to format search results and lists of artists from the dataset.

Using the above methods, recommendations from the Spotify API were generated as follows:

[3] GET https://api.spotify.com/v1/search

[4] GET https://api.spotify.com/v1/artists/id/albums

- Take the top 10 artists for each Nym and map them to Spotify artist URIs.
- Use the first 5 artists to generate 50 artist recommendations, then use the remaining 5 artists to generate another 50 artist recommendations. Since the Spotify API only recommends songs, to retrieve 50 artist recommendations the Spotify API is continuously requested to recommend songs using the same 5 seeds until 50 unique artists have been recommended. The same is done for the second set of seeds.
- Evaluate the precision of the recommendations, by checking how many of the artist recommendations appear in a Nym's top 500 artists.
- Evaluate the recall of the recommendations by checking how many of the top 500 artists of the Nym has been retrieved in 100 artist recommendations. The maximum Nym recall score is 0.2.
- Repeat the process 5 times for each Nym and calculate the average precision and recall scores. These will be used to calculate the F-measure score.

To evaluate the individual performance of the top users of each Nym, the 5 users with the highest play counts are selected from each Nym. Each individual user undergoes the same process as the Nyms outlined above, and the average precision and recall scores for the 5 users in each Nym is calculated.

Note that in each of the tables containing user performance results, there is an additional column called 'Num Artists'. This column is the average of the number of artists listened to by the 5 selected users of each Nym. It is included because not every user listened to 100 artists, so 100 recommendations could not be used to evaluate performance. As such, some table entries contain 'N/A' (Not Applicable) when there were insufficient user artists to calculate performance metrics.

Additionally, the top 10 artists selected for each Nym can be viewed in the appendix, in section 8.2.

6.4.2 Precision

In the context of our evaluation, precision is a measure of how many of the artists recommended by Spotify are relevant to the Nym. A relevant artist to a Nym is defined as an artist that is one of the Nym's top 500 artists. The higher the precision score the better, and the best possible score for precision is 1. A precision score of 1 means all recommended artists are relevant, where precision is defined as follows:

$$Precision = \frac{A \cap B}{|B|}$$

Where:

- A is the set of relevant artists for the Nym.
- B is the set of recommended artists for the Nym by Spotify.
- $A \cap B$ is the set of artists that are both relevant and have been recommended by Spotify.

Precision scores for each Nym and average precision scores for the users of each Nym can be seen in tables 8.1 & 8.2 in the appendix. The precision scores were taken at intervals of n, where n represents the number of recommended artists used to calculate precision. The intervals used for n are 10, 30, 50, 70, 100. These values for n are also used in calculating recall and f-measure scores. A comparison of the overall precision scores between Nyms and users can be seen in figure 6.11. It is clear from figure 6.11 that Nym recommendation precision performs better in nearly every case. The notable exception is in the case of Nym 13, where average user precision is higher by ≈ 0.069. This can be attributed to the lack of users in Nym 13. Indeed there are only 12 users in Nym 13, meaning there is a lack of information available to accurately select artists to properly represent the Nym and generate accurate recommendations. Conversely, precision scores for Nym 2 are vastly superior to the average of it's individual users. Nym 2's precision score is 0.497, whereas it's user's average precision score is 0.228. This results in a difference of ≈ 0.269.

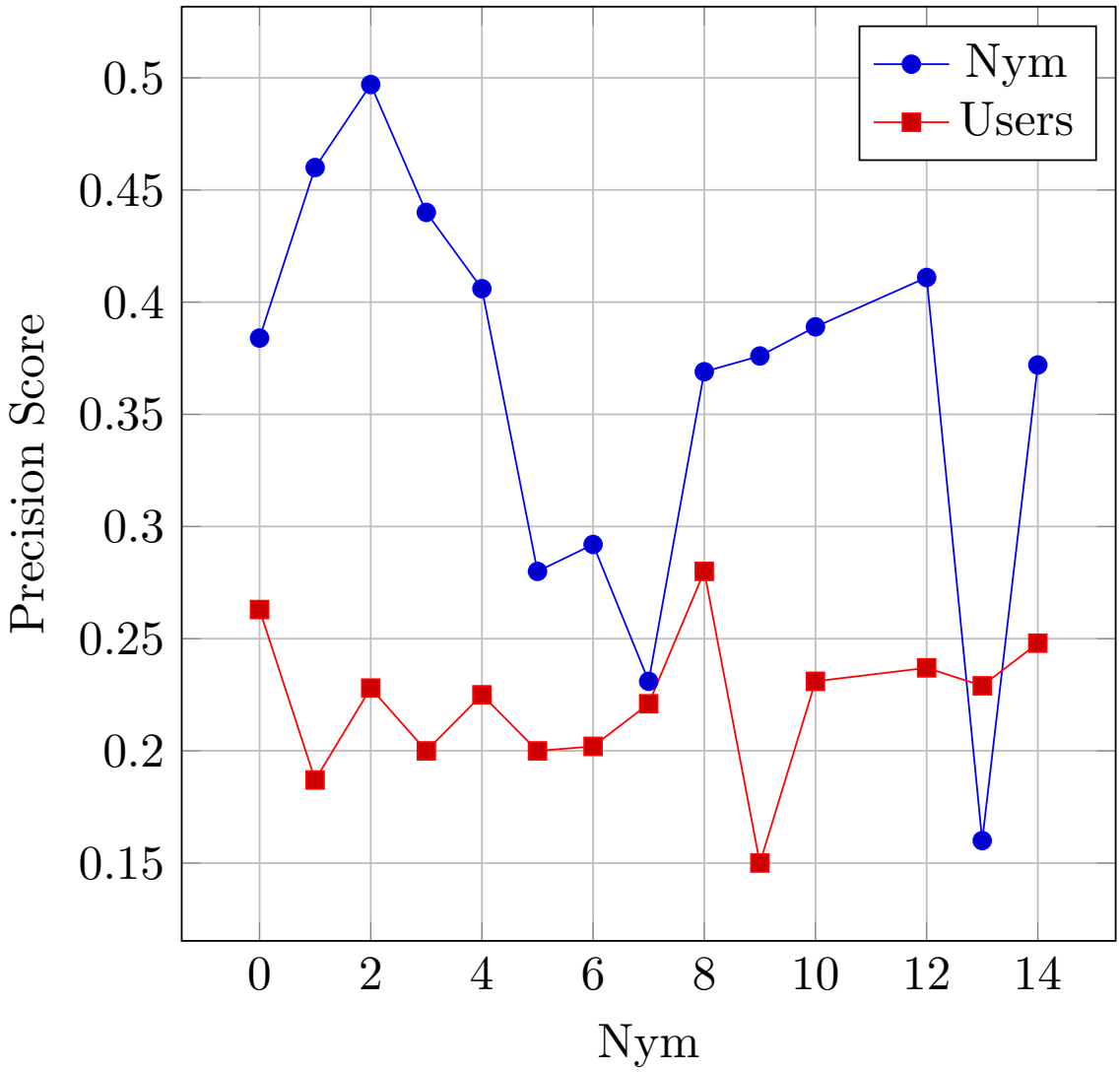


Figure 6.11: A comparison of Nym and average user precision scores taken at the highest common value of n

6.4.3 Recall

In the context of our evaluation, recall is a measure of how many of the relevant artists of a Nym are recommended by the Spotify recommender system, where relevant artists are again defined as artists who appear in the top 500 artists of a Nym. The higher the recall score the better, with the highest possible recall score changing depending on the number of recommended artists used to calculate recall. Recall is defined as follows:

$$Recall = \frac{|A \cap B|}{|A|}$$

Where:

- A is the set of relevant artists for the Nym.
- B is the set of recommended artists for the Nym by Spotify.

- $A \cap B$ is the set of artists that are both relevant to the Nym and have been recommended by Spotify.

Recall scores for each Nym and the average user recall scores for each Nym can be seen in tables 8.3 & 8.4 in the appendix. A comparison of the overall recall scores between Nyms and users at the highest common value for n can be seen in figure 6.12. As can be seen in figure 6.12, average individual user recall scores were consistently higher than their Nym counterparts. This, however, can be attributed to the smaller number of relevant artists (B) available for individual users. Nyms have the advantage of aggregating a number of user playback histories, resulting in a larger collection of relevant artists. As such, each Nym had 500 relevant artists from which to calculate recall. This is in stark contrast to the average number of artists available to calculate recall for each selection of users in each Nym, as can be seen in table 8.4.

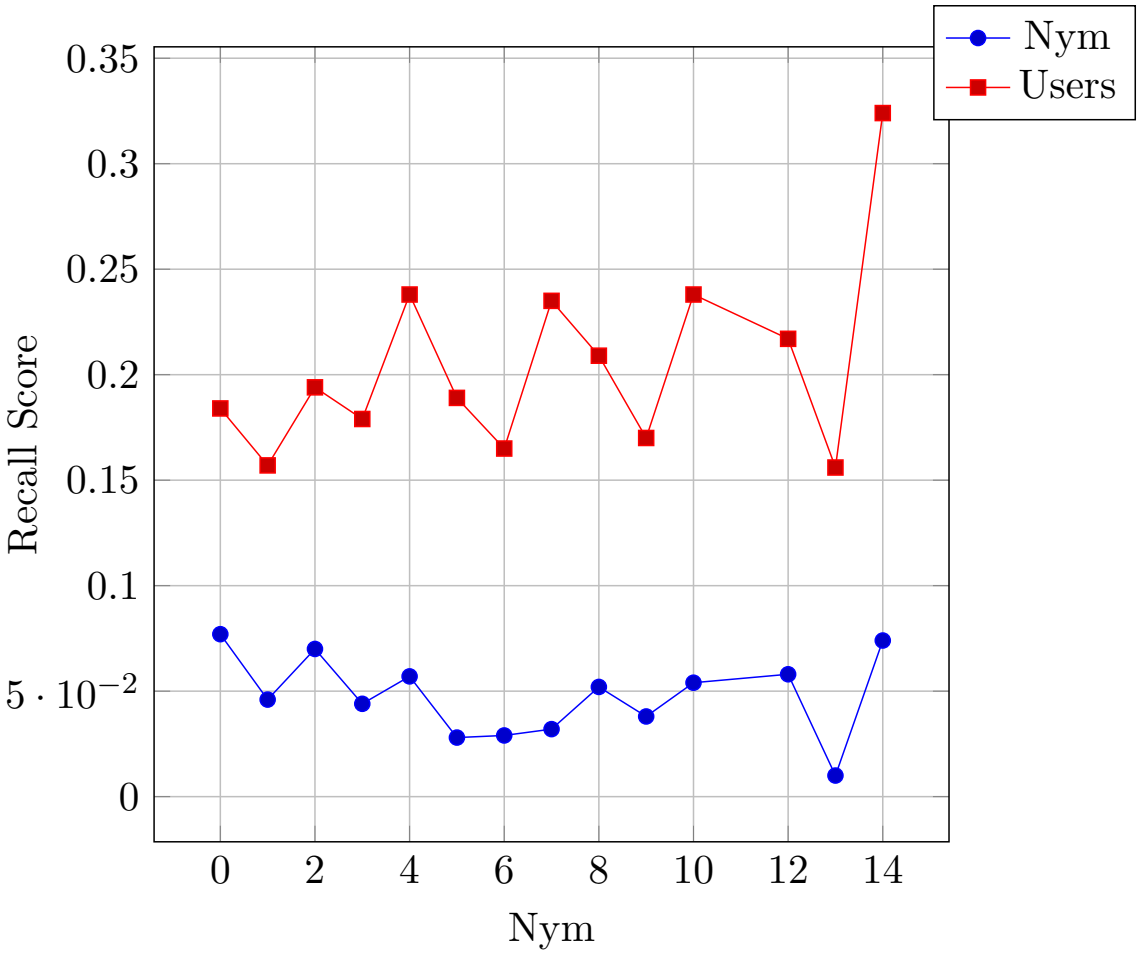


Figure 6.12: A comparison of Nym and average user recall scores taken at the highest common value of n

6.4.4 F-Measure

F-measure is generally considered the harmonic mean of recall and precision, and has the advantage of summarizing the precision and recall metric in one value. The higher the value of the F-measure the better, where F-measure is defined as:

$$F = (\beta^2 + 1) \cdot \frac{P \cdot R}{(\beta^2 \cdot P) + R}$$

Where:

- R is the Recall.
- P is the Precision.
- β emphasizes either precision or recall. $\beta = 0.25$ will be used to emphasize precision in our evaluation. This is because recommendation precision is more important than the overall recall, and also due to the differing number of relevant artists available for users, resulting in skewed recall scores.

F-measure scores for each Nym and the average user f-measure scores for each Nym can be seen in tables 8.5 & 8.6 in the appendix. A comparison of the f-measure scores between Nyms and users can be seen in figure 6.13. As can be seen in figure 6.13, Nyms generally outperform users with the exception of Nyms 5, 6, and 7. In the case of these Nyms, the f-measure scores of it's users generally beat those of the Nyms within a narrow range of ≈ 0.05. This can be attributed to the relatively poor precision performance of these Nyms, as outlined in table 8.2. The outlier in this figure is Nym 13, which performs exceedingly poorly both in relation to the performance of the other Nyms, and individual users of Nym 13. This again can be attributed to the low amount of users assigned to the Nym.

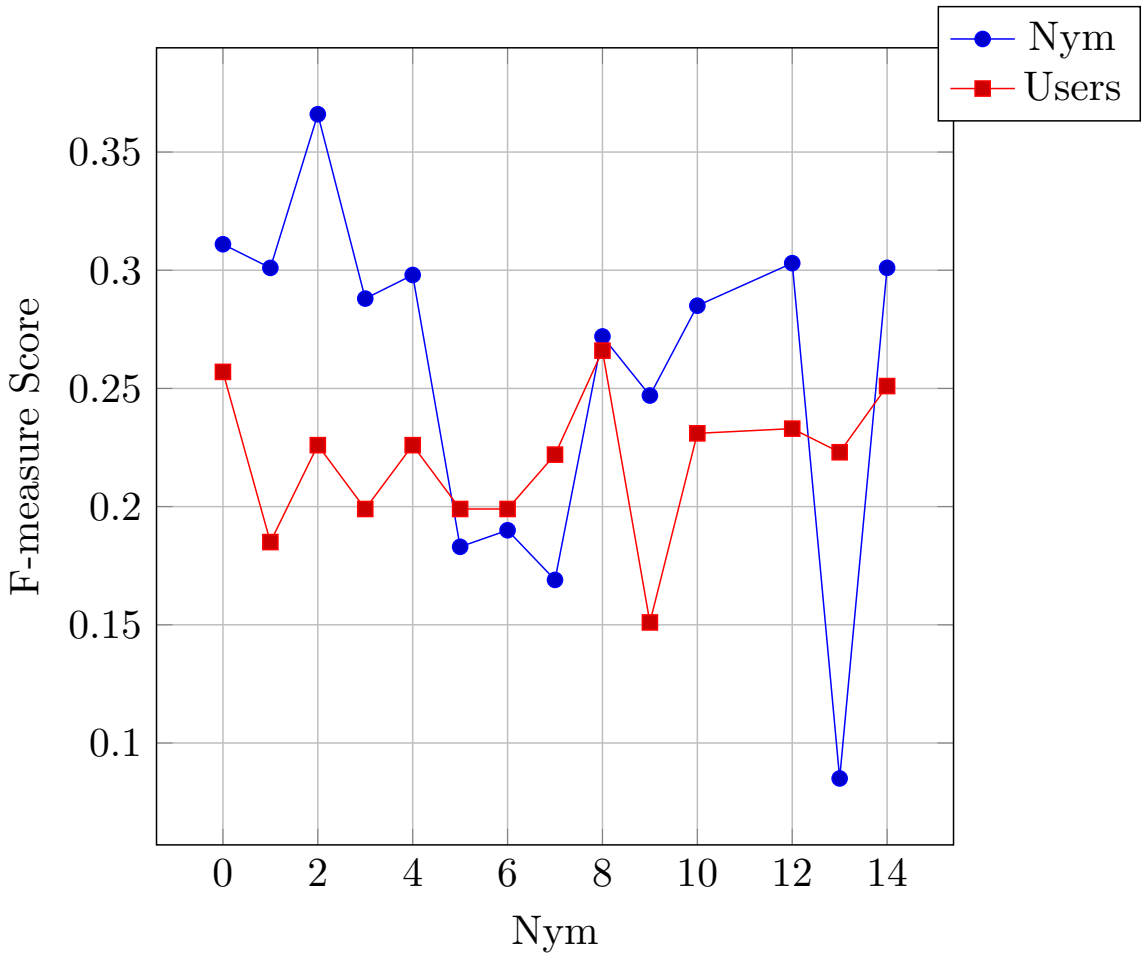


Figure 6.13: A comparison of Nym and average user f-measure scores taken at the highest common values of n where $\beta = 0.25$

Chapter 7

Conclusion

This chapter will reflect on work carried out in the dissertation, and discuss the outcome of each individual research objective outlined in section 1.4. Additionally, this chapter will detail the limitations of this research, and outline future work that may be completed.

7.1 Research Objectives

This section will discuss the results of each research objective, outline whether the objective has been completed, and detail any limitations of the results of each objective.

7.1.1 Browser Extension

As discussed in section 4.2, the browser extension was implemented in Firefox. The browser extension showed it was indeed possible to intercept user ratings where there is a defined rating endpoint (as was the case with YouTube.com), and it was possible to allow users to browse as part of a nym by substituting user cookies in real-time. Additionally, we evaluated the impact of the browser extension on page load times and rating request latency and found in most cases the overhead caused by the extension was almost negligible when caching was used. Indeed, better pre-emptive caching techniques could be implemented in the future to further minimise the impact of the extension.

To this extent, the research objective for the browser extension has been achieved. It is not, however, without it's limitations. The most glaring of which is it's dependence on third party websites for cooperation. In section 6.2, YouTube.com had to be used to measure the impact of intercepting rating requests because Spotify only used implicit feedback mechanisms, and it wasn't possible to intercept requests for individual songs on Spotify as the body of song requests were encrypted when intercepted by the browser extension. Additionally, websites can easily subvert rating request interception by sending ratings over WebSockets.

WebSocket traffic, at the time of writing, cannot be intercepted by browser extensions.

Additionally, the browser extension relies on goodwill from users. For users to be able to browse as part of a Nym, the browser extension relies on session cookies for OpenNym accounts on each supported website that are obtained by manually logging in to each individual website, and inserting these cookies in the OpenNym database. However, given Spotify for example, users of a Nym can easily browse to the account settings page and 'log out' of the website. This invalidates the cookies being used by the Nym and makes Spotify inaccessible as part of that Nym until an authorized user logs in to the Spotify service and updates the Nym's session cookies. This limitation can even be taken so far as to see users changing passwords to OpenNym accounts, locking OpenNym administrators out of Nym accounts.

7.1.2 Web Server

The OpenNym web server was to be developed such that it maintains a list of Nyms, their ratings and associated metadata so the browser extension could interact with it and allow users to browse as part of a Nym. The web server was designed in accordance with the specification outlined in section 3.1, and it's performance evaluated in section 6.1. Though good performance was not necessarily an objective for the web server, it is nonetheless important for user experience.

As shown in Section 6.1, the web server performs quite well when retrieving ratings, session cookies, and rules. Both the Nym API and the Identity Interface API performed relatively poorly. This can be attributed to the large amount of database read/write operations that must be carried out to respond to each request.

In the case of the Nym API, investigation into the code revealed the bottleneck was the retrieval of the rating information for each of the top ratings for all of the Nyms. A possible solution to this issue, as previously discussed, is to cache the responses from the Nym API as they should remain identical for each request until the top ratings for each Nym is updated. Any future optimisations to the web server should focus on improving performance of the Nym API.

In the case of the Identity Interface API, requests to this API should be made quite sparingly by website administrators intending to add Nym ratings to their own database. As a consequence, the API's relatively poor performance is acceptable as it is not expected to ever receive a high volume of traffic. Indeed, to ensure this API is not abused by malicious users taking advantage of it's poor performance, user authentication for queries to this API can be considered in the future. Authentication should not pose any privacy concerns for this API as users should not be using this API for privacy enhanced browsing.

Despite the limitations of the web server posed by the Nym and Identity Interface APIs, we still consider this research objective achieved. As demonstrated by the browser extension tests performed in section 6.2, communication between the Firefox extension and the web server is working and facilitates privacy enhanced browsing for users.

7.1.3 Proof of Concept

The final research objective was to demonstrate the viability of OpenNym by clustering users in a publicly available dataset and using the resulting Nyms to generate recommendations from a suitable recommender system. The steps taken to create the proof of concept were outlined in section 5, and evaluation of recommendation performance for Nyms are shown in section 6.4. The evaluation of the proof of concept recommendation performance on the Spotify web service showed that users can generally expect more precise recommendations when browsing as part of a Nym than when browsing individually. This can be attributed to the increased number of ratings available for each Nym, allowing for better seeds to be selected to represent a Nym's interests.

To this effect, the final research objective has also been achieved by demonstrating the viability of clustering users based on their ratings history. There are some limitations to this proof of concept however, with one of the most important being the cold start problem. The cold start problem is a common issue in Machine Learning and refers to the sparsity of information available for training a machine learning model [31], or clustering users in this case. OpenNym requires a relatively large dataset to perform initial user clustering so that it can create the initial Nyms which users can then join and browse as part of. These datasets need to have anonymised user information to preserve the integrity of OpenNym, and the data would also need to be relatively current to ensure new users have enough ratings to be assigned to an appropriate Nym.

Another major limitation to the approach taken in the proof of concept is the considerable background work needed to cluster users and generate Nym ratings. As discussed in Section 5, Nym ratings could not be generated by simply averaging user ratings due to the impact of song popularity. Popularity can be expected to affect items in other genres, meaning considerable work will also be required to generate nym ratings for items of other genres.

The final major limitation of this proof of concept is the association of items from the dataset to items in a website's recommender system. While standard information retrieval techniques were used to map artist names to artist URIs in the Spotify system, the mapping tool only had a success rate of $\approx 80\%$. This meant 1 in 5 artists in the dataset remained unidentified, and could not be used as recommendation seeds or for evaluating recommenda-

tion performance. No analysis was performed to determine the popularity of these artists.

7.2 Future Work

This section will detail work that can be carried out that may address some of the limitations of the work produced in this dissertation, or improve the overall privacy and performance of the OpenNym system.

- WebSockets can be used as the communication protocol between the browser extension and the web server instead of HTTP. WebSockets have the advantage of a much smaller communication overhead, as both parties can enjoy full duplex communication when a WebSockets connection is established. This is in contrast to HTTP, which requires an individual HTTP request be submitted for each individual resource. The reduced communication overhead would also result in reduced latency between the browser extension and the web server.

- Automatic Nym Selection on the browser extension could be implemented, where the browser extension recommends a Nym for the user to join based on the user's local rating history. This Nym recommendation can be calculated by using a standard distance function, such as Euclidean distance, to find the Nym which most aligns with the user's top ratings. Since the distance function would be used locally, no individual user ratings need to leave the browser to select a Nym. This feature would require that the Firefox extension also start caching user individual ratings locally.

- An OpenNym Single Sign On service could be developed, similar to OAuth [24]. This would be an API that allows users to authenticate themselves as part of a Nym when 'logging in' to a website. These OpenNym identities would have special properties on websites that support authentication via OpenNym, the most important being that users cannot modify any account settings for OpenNym identities. This approach would be beneficial for both OpenNym and websites, as OpenNym would no longer need to rely on substituting user's session cookies and websites would have a wealth of rating information to associate with new users.

7.3 Closing Remarks

This dissertation set out to prove the concept proposed by Checco et al. [14] using the MovieLens service could be implemented as a scalable solution to attaining privacy enhanced browsing. In this dissertation we designed and implemented an OpenNym web server and

companion browser extension, demonstrated the ability to browse as part of a group without changing exising web infrastructure, and proved Nyms could also receive accurate recommendations from recommender systems.

Through the process of achieving these research objectives, we also highlighted a number of limitations encountered during the course of our research. These are limitations that would need to be addressed before the OpenNym system could be made available to the public. Further research would need to be conducted to solve these limitations, preferably in cooperation with recommender system owners. However, as discussed in chapters 1 & 2, there is value in user data and it remains doubtful whether a privacy enhanced recommendation system would be embraced by recommender system owners.

Our concluding remark is that while OpenNym is very effective, there are a number of hurdles it must overcome before it can become a viable service that can be used by the public.

Bibliography

[1] Abraham, M., Meierhoefer, C., and Lipsman, A. (2007). The impact of cookie deletion on the accuracy of site-server and ad-server metrics: An empirical comscore study. *Retrieved October*, 14:2009.

[2] Ackerman, M. S., Cranor, L. F., and Reagle, J. (1999). Privacy in e-commerce: examining user scenarios and privacy preferences. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 1–8. ACM.

[3] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 17(6):734–749.

[4] Aïmeur, E., Brassard, G., Fernandez, J. M., and Onana, F. S. M. (2008). Alambic : a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Sec.*, 7(5):307–334.

[5] Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J. A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., and Shavitt, Y. (2016). Drown: Breaking tls using sslv2. In Holz, T. and Savage, S., editors, *USENIX Security Symposium*, pages 689–706. USENIX Association.

[6] Ayenson, M., Wambach, D., Soltani, A., Good, N., and Hoofnagle, C. (2011). Flash cookies and privacy ii: Now with html5 and etag respawning.

[7] Bell, R. M., Koren, Y., and Volinsky, C. (2007). The bellkor solution to the netflix prize. Technical report, AT&T Labs.

[8] Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011). The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

[9] Boda, K., Földes, Á. M., Gulyás, G. G., and Imre, S. (2012). *User Tracking on the Web via Cross-Browser Fingerprinting*. Springer.

[10] Bokde, D., Girase, S., and Mukhopadhyay, D. (2015). Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science*, 49(1):136–146.

[11] Browser Extension Community Group (2017). Browser extensions draft community group report 23 july 2017. `https://browserext.github.io/browserext/` [Online; accessed 07-May-2018].

[12] Calandrino, J. A., Kilzer, A., Narayanan, A., Felten, E. W., and Shmatikov, V. (2011). "you might also like: " privacy risks of collaborative filtering. In *IEEE Symposium on Security and Privacy*, pages 231–246. IEEE Computer Society.

[13] Checco, A., Bianchi, G., and Leith, D. (2017a). Blc: Private matrix factorization recommenders via automatic group learning.

[14] Checco, A., Bracciale, L., Bianchi, G., and Leith, D. (2017b). Opennym: Enabling privacy preserving recommending via pseudonymous group authentication.

[15] Chen, T., Han, W.-L., Wang, H.-D., Zhou, Y.-X., Xu, B., and Zang, B.-Y. (2007). Content recommendation system based on private dynamic user profile. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 4, pages 2112–2118. IEEE.

[16] Digital Advertising Alliance (2017). Understanding online advertising. `http://www.aboutads.info/consumers/#cookies` [Online; accessed 25-April-2018].

[17] Eckersley, P. (2010). *How Unique Is Your Web Browser?* Springer.

[18] Eerola, T. and Vuoskoski, J. K. (2011). A comparison of the discrete and dimensional models of emotion in music. *Psychology of Music*, 39(1):18–49.

[19] Erlang (2018). Erlang programming language. `https://www.erlang.org/` [Online; accessed 18-May-2018].

[20] Google (2018a). Cookies and user identification. `https://developers.google.com/analytics/devguides/collection/analyticsjs/cookies-user-id` [Online; accessed 25-April-2018].

[21] Google (2018b). recaptcha: Easy on humans, hard on bots. `https://www.google.com/recaptcha/intro/android.html` [Online; accessed 04-May-2018].

[22] Grouplens (2016). Movielens. `https://grouplens.org/datasets/movielens/` [Online; accessed 18-April-2018].

[23] Hursti, J. (1997). Single sign-on. In *Proc. Helsinki Uiniversity of Technology Seminar on Network Security*.

[24] IETF OAuth Working Group (2018). Oauth 2.0. `https://oauth.net/2/` [Online; accessed 18-May-2018].

[25] Kerr, D. (2013). An introduction to mvc frameworks. `http://blog.scottlogic.com/2013/12/06/JavaScript-MVC-frameworks.html` [Online; accessed 18-May-2018].

[26] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87.

[27] Koren, Y. (2009). The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81:1–10.

[28] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8).

[29] Lamere, P. (2014). Welcome to spotipy! `http://spotipy.readthedocs.io/en/latest/` [Online; accessed 18-May-2018].

[30] Lerato, M., Esan, O. A., Ebunoluwa, A. D., Ngwira, S., and Zuva, T. (2015). A survey of recommender system feedback techniques, comparison and evaluation metrics. In *2015 International Conference on Computing, Communication and Security (ICCCS)*, pages 1–4.

[31] Lika, B., Kolomvatsos, K., and Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073.

[32] Locustio (2018). Locust - a modern load testing framework. `https://locust.io/` [Online; accessed 18-May-2018].

[33] McCord, C. (2018). Phoenix framework. `http://phoenixframework.org/` [Online; accessed 04-May-2018].

[34] millionsong (2012). Fixing matching errors. `https://labrosa.ee.columbia.edu/millionsong/blog/12-2-12-fixing-matching-errors` [Online; accessed 08-May-2018].

[35] Mozilla Corporation (2018). Javascript. `https://developer.mozilla.org/en-US/docs/Web/JavaScript` [Online; accessed 18-May-2018].

[36] Mozilla Developer Network (2018). Browser extensions - mozilla — mdn. `https://developer.mozilla.org/en-US/Add-ons/WebExtensions` [Online; accessed 05-May-2018].

[37] Netflix (2012). Netflix recommendations: Beyond the 5 starts (part 2). `https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-2-d9b96aa399f5` [Online; accessed 25-April-2018].

[38] NumPy developers (2018). Numpy. `http://www.numpy.org/` [Online; accessed 18-May-2018].

[39] Núñez-Valdéz, E. R., Lovelle, J. M. C., Martínez, O. S., García-Díaz, V., de Pablos, P. O., and Marín, C. E. M. (2012). Implicit feedback techniques on recommender systems applied to electronic books. *Computers in Human Behavior*, 28(4):1186–1193.

[40] O'Connor, M., Cosley, D., Konstan, J. A., and Riedl, J. (2001). Polylens: A recommender system for groups of users. In *Proceedings of ECSCW 2001*, pages 199–218.

[41] Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer, Berlin / Heidelberg.

[42] Python Software Foundation (2018). Welcome to python.org. `https://www.python.org/` [Online; accessed 18-May-2018].

[43] Qian, G., Sural, S., Gu, Y., and Pramanik, S. (2004). Similarity between euclidean and cosine angle distance for nearest neighbor queries. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1232–1237, New York, NY, USA. ACM.

[44] Ramakrishnan, N., Keller, B. J., Mirza, B. J., Grama, A., and Karypis, G. (2001). Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62.

[45] Rennie, G. (2015). The road to 2 million websocket connections in phoenix. `http://phoenixframework.org/blog/the-road-to-2-million-websocket-connections` [Online; accessed 04-May-2018].

[46] Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communication ACM*, 40:56–58.

[47] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2011). *Recommender systems handbook*, chapter Introduction to Recommender Systems Handbook. Springer, New York; London.

[48] Shokri, R., Pedarsani, P., Theodorakopoulos, G., and Hubaux, J.-P. (2009). Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In Bergman, L. D., Tuzhilin, A., Burke, R. D., Felfernig, A., and Schmidt-Thieme, L., editors, *RecSys*, pages 157–164. ACM.

[49] Shyong, K., Lam, T., Frankowski, D., and Riedl, J. (2006). Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. *ETRICS 2006*, LNCS 3995:14–29.

[50] Soltani, A., Canty, S., Mayo, Q., Thomas, L., and Hoofnagle, C. J. (2010). Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*. AAAI.

[51] Spotify AB (2018a). More people than ever are streaming on spotify. `https://newsroom.spotify.com/2018-05-02/more-people-than-ever-are-streaming-on-spotify/` [Online; accessed 08-May-2018].

[52] Spotify AB (2018b). Spotify. `https://www.spotify.com/us/` [Online; accessed 08-May-2018].

[53] The International Association of Privacy Professionals (2018). What is privacy. `https://iapp.org/` [Online; accessed 25-April-2018].

[54] The Mozilla Corporation (2018). The new, fast browser for mac, pc and linux — firefox. `https://www.mozilla.org/en-US/firefox/` [Online; accessed 05-May-2018].

[55] The PostgreSQL Global Development Group (2018). Postgresql: The world's most advanced open source database. `https://www.postgresql.org/` [Online; accessed 05-May-2018].

[56] Valim, J. (2018). Elixir. `https://elixir-lang.org/` [Online; accessed 04-May-2018].

[57] Westin, A. F. (1967). *Privacy and Freedom*. Atheneum, New York.

[58] White, R., Jose, J. M., and Ruthven, I. (2001). Comparing explicit and implicit feedback techniques for web retrieval: Trec-10 interactive track report. In Voorhees, E. M. and

Harman, D. K., editors, *TREC*, volume Special Publication 500-250. National Institute of Standards and Technology (NIST).

Chapter 8

Appendix

8.1 Listings

```
user_id                                   song_id         play_count
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOAKIMP12A8C130995  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOAPDEY12A81C210A9  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBBMDR12A8C13253B  2
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBFNSP12AF72A0E22  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBFOVM12A58A7D494  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBNZDC12A6D4FC103  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBSUJE12A6D4F8CF5  2
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBVFZR12A6D4F8AE3  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBXALG12A8C13C108  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBXHDL12A81C204C0  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOBYHAJ12A6701BF1D  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SOCNMUH12A6D4F6E6D  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SODACBL12A8C13C273  1
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SODDNQT12A6D4F5F7E  5
b80344d063b5ccb3212f76538f3d9e43d87dca9e  SODXRTY12AB0180F3B  1
```

Listing 13: Format of `train_triplets.txt`

```
track_id <SEP> song_id <SEP> artist_name <SEP> song_name <SEP>
TRMMMYQ128F932D901<SEP>SOQMMHC12AB0180CB8<SEP>Faster Pussy cat<SEP>Silent
↪ Night
TRMMMKD128F425225D<SEP>SOVFVAK12A8C1350D9<SEP>Karkkiautomaatti <SEP>Tanssi
↪ vaan
TRMMMRX128F93187D9<SEP>SOGTUKN12AB017F4F1<SEP>Hudson Mohawke<SEP>No One
↪ Could Ever
TRMMMCH128F425532C<SEP>SOBNYVR12A8C13558C<SEP>Yerba Brava<SEP>Si Vos Quers
TRMMMWA128F426B589<SEP>SOHSBXH12A8C13B0DF<SEP>Der Mystic<SEP>Tangle Of
↪ Aspens
```

Listing 14: Format of `unique_tracks.txt`

8.2 Top Nym Artists

- **Nym 0**: Coldplay, Adam Lambert, B.o.B, Eagles, The Verve, John Mayer, Rihanna, Kings of Leon, 3 Doors Down, Muse
- **Nym 1**: Linkin Park, Coldplay, Eminem, The All-American Rejects, B.o.B, Jimmy Eat World, Death Cab for Cutie, 3 Doors Down, OneRepublic, Foo Fighters
- **Nym 2**: The All-American Rejects, John Mayer, Nirvana, Justin Timberlake, The White Stripes, Foo Fighters, Rihanna, Kid Cudi, OneRepublic, Coldplay
- **Nym 3**: Cage the Elephant, Paramore, The White Stripes, Radiohead, Plain White T's, Vampire Weekend, Muse, Bon Jovi, Justin Biever, OneRepublic
- **Nym 4**: Coldplay, Lady Gaga, Jack Johnson, Radiohead, Bon Jovi, The Verve, The White Stripes, Linkin Park, DJ Dizzy, Muse
- **Nym 5**: Florence + The Machine, Kings of Leon, Coldplay, Rise Against, Radiohead, M.I.A., Angels and Airwaves, The Killers, Edward Sharpe & The Magnetic Zeros, Three Drives
- **Nym 6**: Coldplay, Rise Against, The White Stripes, John Mayer, The Verve, Plain White T's, Beyonce, Usher, Cartola, The All-American Rejects
- **Nym 7**: Coldplay, Lil Wayne, La Roux, Train, Kid Cudi, Charttraxx Karaoke, Tub Ring, Justin Bieber, John Mayer, The Crests

- **Nym 8**: Coldplay, Usher, Drake, Black Eyed Peas, Nickelback, Justin Timberlake, The White Stripes, Cartola, Lady Gaga, B.o.B
- **Nym 9**: California Swag District, Daft Punk, Justin Timberlake, Rise Against, Kid Cudi, Cartola, Radiohead, Linkin Park, Kings of Leon, Jason Mraz
- **Nym 10**: Kid Cudi, Muse, Foo Fighters, Cartola, Jason Derulo, Modest Mouse, Coldplay, Bon Jovi, Jimmy Eat World, Radiohead
- **Nym 12**: Vampire Weekend, Justin Timberlake, Nickelback, Linkin Park, The White Stripes, The Killers, 3 Doors Down, Radiohead, Nirvana, Rise Against
- **Nym 13**: Amy Winehouse, Dwight Yoakam, Panic At The Disco, Keith Sweat, Sara Bareilles, Cannibal & The Headhunters, Creedence Clearwater Revival, Sarah McLachlan, Margot & The Nuclear So and So's, Me'Shell Ndegeocello
- **Nym 14**: Jack Johnson, Bon Jovi, Band Of Horses, Coldplay, The Verve, The All-American Rejects, Eminem, Foo Fighters, Cartola, Muse

8.3 Tables

Nym	@10	@30	@50	@70	@100
0	0.34	0.433	0.444	0.383	0.384
1	0.4	0.46	0.46	0.403	0.41
2	0.5	0.527	0.552	0.497	0.486
3	0.44	0.46	0.44	0.426	0.462
4	0.14	0.407	0.44	0.406	0.428
5	0.06	0.227	0.28	0.286	0.312
6	0.3	0.32	0.292	0.266	0.29
7	0.14	0.287	0.284	0.231	0.242
8	0.4	0.387	0.372	0.369	0.354
9	0.28	0.4	0.376	0.349	0.388
10	0.3	0.327	0.384	0.389	0.416
12	0.24	0.347	0.432	0.411	0.432
13	0.26	0.16	0.188	0.149	0.146
14	0.34	0.373	0.376	0.349	0.372

Table 8.1: Precision scores for each Nym

Nym	Num Artists	@10	@30	@50	@70	@100
0	200	0.331	0.284	0.287	0.275	0.263
1	66	0.169	0.207	0.187	N/A	N/A
2	77	0.147	0.181	0.204	0.228	N/A
3	66	0.179	0.215	0.2	N/A	N/A
4	72	0.285	0.254	0.233	0.225	N/A
5	67	0.196	0.219	0.2	N/A	N/A
6	67	0.135	0.198	0.202	N/A	N/A
7	95	0.104	0.205	0.216	0.221	N/A
8	166	0.249	0.275	0.28	0.28	N/A
9	58	0.133	0.158	0.15	N/A	N/A
10	74	0.22	0.224	0.221	0.231	N/A
12	98	0.174	0.24	0.239	0.234	0.237
13	49	0.237	0.229	N/A	N/A	N/A
14	123	0.317	0.234	0.241	0.255	0.248

Table 8.2: Average user precision scores for each Nym

Nym	@10	@30	@50	@70	@100
0	0.007	0.036	0.044	0.054	0.077
1	0.008	0.028	0.046	0.056	0.082
2	0.01	0.032	0.055	0.07	0.097
3	0.009	0.028	0.044	0.06	0.092
4	0.003	0.024	0.044	0.057	0.086
5	0.001	0.014	0.028	0.04	0.062
6	0.006	0.019	0.029	0.037	0.058
7	0.003	0.017	0.028	0.032	0.048
8	0.008	0.023	0.037	0.052	0.071
9	0.006	0.024	0.038	0.049	0.078
10	0.006	0.02	0.038	0.054	0.083
12	0.005	0.021	0.043	0.058	0.086
13	0.005	0.01	0.019	0.021	0.029
14	0.007	0.022	0.038	0.049	0.074

Table 8.3: Recall scores for each Nym

Nym	Num Artists	@10	@30	@50	@70	@100
0	200	0.022	0.06	0.098	0.132	0.184
1	66	0.029	0.107	0.157	N/A	N/A
2	77	0.017	0.063	0.122	0.194	N/A
3	66	0.03	0.107	0.179	N/A	N/A
4	72	0.041	0.116	0.178	0.238	N/A
5	67	0.029	0.119	0.189	N/A	N/A
6	67	0.021	0.088	0.165	N/A	N/A
7	95	0.013	0.079	0.156	0.235	N/A
8	166	0.0185	0.065	0.11	0.149	0.209
9	58	0.026	0.114	0.17	N/A	N/A
10	74	0.036	0.103	0.164	0.238	N/A
12	98	0.017	0.095	0.168	0.217	N/A
13	49	0.054	0.156	N/A	N/A	N/A
14	123	0.044	0.089	0.176	0.241	0.324

Table 8.4: Average user recall scores for each Nym

Nym	@10	@30	@50	@70	@100
0	0.090	0.263	0.289	0.282	0.311
1	0.103	0.241	0.301	0.295	0.332
2	0.129	0.276	0.360	0.366	0.393
3	0.115	0.241	0.288	0.314	0.374
4	0.038	0.210	0.288	0.298	0.347
5	0.013	0.120	0.183	0.210	0.252
6	0.077	0.166	0.190	0.195	0.235
7	0.038	0.148	0.185	0.169	0.196
8	0.103	0.200	0.243	0.272	0.287
9	0.076	0.208	0.247	0.257	0.314
10	0.077	0.172	0.250	0.285	0.337
12	0.064	0.181	0.282	0.303	0.349
13	0.065	0.085	0.123	0.110	0.118
14	0.090	0.192	0.247	0.257	0.301

Table 8.5: F-measure scores for each Nym where $\beta = 0.25$

Nym	Num Artists	@10	@30	@50	@70	@100
0	200	0.181	0.233	0.258	0.259	0.257
1	66	0.132	0.196	0.185	N/A	N/A
2	77	0.101	0.163	0.196	0.226	N/A
3	66	0.139	0.203	0.199	N/A	N/A
4	72	0.211	0.237	0.229	0.226	N/A
5	67	0.146	0.209	0.199	N/A	N/A
6	67	0.102	0.184	0.199	N/A	N/A
7	95	0.074	0.187	0.211	0.222	N/A
8	166	0.144	0.231	0.257	0.266	N/A
9	58	0.107	0.154	0.151	N/A	N/A
10	74	0.169	0.210	0.217	0.231	N/A
12	98	0.113	0.220	0.233	0.233	N/A
13	49	0.198	0.223	N/A	N/A	N/A
14	123	0.232	0.214	0.236	0.254	0.251

Table 8.6: Average user F-measure scores for each Nym where $\beta = 0.25$