University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

BSc Computer Science 2014

Music Recommendation through Collaborative, Temporal and Location Filtering

Student Name: James Murphy
Student Number: 109359842

Supervisor: Dr. Derek Bridge

Second Reader: Dr. Ken Brown

Acknowledgements

I'd like to kindly thank Soundwave for use of their data. It was fundamental to the whole project, and was very useful in testing all of the recommender systems. Their feedback on which recommenders suited them, and which didn't also helped greatly.

I'd like to thank Derek for his constant help and assistance with everything, without whom, this project wouldn't exist.

Finally I'd like to thank everyone who came around and viewed my project on the open day. All the complimentary remarks and warm appraisals really justified putting so much time into this project.

Declaration of Originality

In signing this declaration, you are confirming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:-

- With respect to my own work: none of it has been submitted at any educational institution contributing in any way towards an educational award;
- With respect to anothers work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other students work, whether published or unpublished, electronically or in print.
- This is all my own work, unless clearly indicated otherwise, with full and proper accreditation;

Name: James Murphy

Signed:

Date: 04/04/2014

Abstract

Soundwave, a new music application, needs a music recommendation service in their application. In this project, numerous music recommenders were implemented to satisfy their needs. Music recommendation was done mainly through collaborative filtering, with the aid of location services or temporal evaluation in some recommender systems. In some cases we used profile based re-ranking, to change the order in which songs appeared, which turned out to improve results and make recommender systems more accurate on the whole.

Contents

1 Introduction

Gracenote, the industry-recognised largest source of music metadata, has stated it has over 180 million tracks[1] in its database. It stands to reason with that number of songs, that finding a song to listen to at any given point in time is quite an ordeal. Deciding which song to listen to is becoming overwhelming. As a consequence of this, people have started to look for a narrower selection of songs to choose from: songs recommended just for their specific tastes.

It's from this observation that music recommenders have started emerging everywhere, from YouTube, to Spotify, to your very own iPod being able to recommend songs to you from your library through a service called iTunes Genius.

This project relates to just one application. Soundwave, a new mobile music application (partly founded by former UCC student Aidan Sliney) has had tremendous popularity since its release in June 2009. Soundwave, however, have no intelligent music recommender in their application. Feedback from their user base has indicated that this is a feature highly sought after in the application.

In this report, the various techniques music recommenders currently implement will be analysed for Soundwave, along with pioneering some new techniques that may have significant relevance to their market area.

[1]http://www.gracenote.com/company

1.1 Music - An Ever Expanding Industry

Ever since the birth of man, music has been ever-present. In 1877, when Thomas Edison invented the phonograph, he gave people the first opportunity to both record and listen to music. Such an important milestone cannot be overstated. People could now record themselves, distribute their music, and maybe as a by-product, become known publicly as an 'Artist'. Jump forward over a century to the present, and the music industry generated over $16.5 billion in 2012 [1].

The multiple ways in which music can be obtained now are greater than ever. Online music stores have now become the biggest avenue to obtain music. But even within that one avenue, there is a vast amount of choice. Wikipedia's article on Online Music stores compares 42 major online stores alone. Just one of these, iTunes, recorded sales of over $4.3 billion in one quarter. [2] Not restricted to having physical copies of the records they sell, online stores can store much more music than regular stores. iTunes, as an example, has over 37 million tracks. [3]

In recent years, massive growth in online subscription services had led to more and more people availing of this cheap new way to listen to music. The number of people paying to use subscription services grew 44 per cent in 2012 to 20 million. Subscription revenues are expected to account for more than 10 per cent of digital revenues for the first time in 2012 [1]. With all these services popping up, people now have more choice over which music to listen to, and it is easier than ever to listen to it.

The only drawback is, this exacerbates a problem which already resides in regular mortar and brick record stores - What should I buy? This, is where music recommendation comes in.

[2] http://appleinsider.com/articles/13/07/23/apples-itunes-brings-in-43b-in-q3-up-25-from-2012

[3] http://www.apple.com/itunes/itunes-match/

1.2 What Makes Music Special

Recommending items to users at a high level, can be seen as comparing users and items. However, music is somewhat special. And with this, care needs to be taken when recommending music. An article by Paul Lamere from MusicMachinery.com labelled "What makes music so special" [11] outlines some great points as to why music is so different to other areas of recommendation.

Outlined below are some of the reasons outlined in Lamere's report as to why music is so special:

- Huge Item space - Unlike the Netflix challenge, which worked on 17,770 movies, most industrial sized music collections range in upwards of 10 million songs
- Low Cost Per Item - When the cost of an item is low, the cost of recommending a bad item is also reduced
- Low Consumption Time -Unlike books or films, a song takes merely minutes to digest. This means more recommendations are needed more often, unlike books or films
- Consumed in sequences - Unlike movies or books, the order in which you recommend songs may have a massive influence on how good this recommendation is. The user may not be in the right mood, or right location for that track

As Paul Lamere summarises, there is no one fit-for-all recommendation algorithm. Algorithms need to be tailor-fit to their application, and this takes time and dedication. This is something I've endeavoured to do throughout this project.

1.3 Soundwave

Soundwave is a mobile application founded by past UCC student Aidan Sliney, and UCD student Brendan O Driscoll. It is a music discovery application for iOS and Andriod. Its key functionality is scrobbling (tracking) music played on phone locally, as well as from numerous music streaming services. All this data is recorded and stored on Soundwave servers. The application allows you see all your music history, thus never losing a song you listened to. With a growing number of tracks this is becoming a more and more valuable feature. A key feature is that Soundwave tracks location of all plays and depicts them on world map. It has over 900,000 downloads since its release in June 2013.

1.4 Project Objectives

The aim of this project was to provide Soundwave with a means of recommending music to their users. From meetings with Soundwave, they had gathered two main use cases for recommenders in their application.

The first use case was that someone could get recommendations based on their whole playing history. This, I've labelled as a 'Profile Based Recommender'. It uses the persons profile (playing history) to recommend them a song. Shown in figure 1 is the profile page of the application, where the recommender might reside.

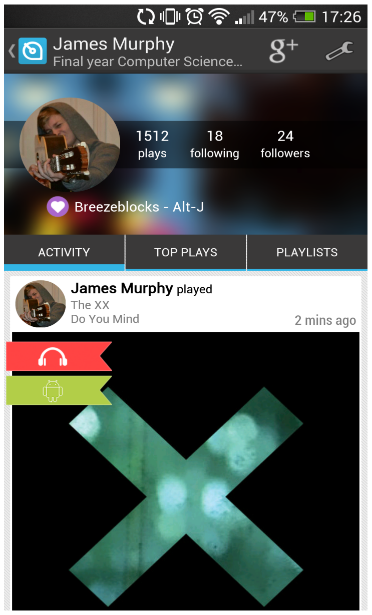


Figure 1: Profile Page

The second use case that Soundwave presented us was that if someone had just played a song they really liked, that they could get recommended items based on this song. This use case, I've called a 'Song Based Recommender'. It used a seed song, to recommend items to the user. The song page of the application is shown in figure 2, where this recommender might reside.

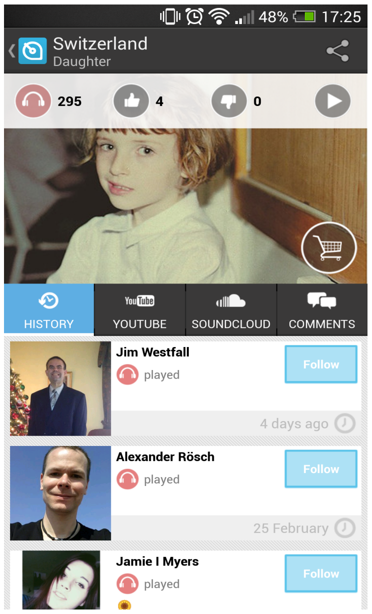


Figure 2: Song Page

Along with these two initial use cases that Soundwave had collected from users, an additional two use cases were formulated. The first was one that was envisaged due to a unique feature of the Soundwave data. Soundwave, unlike many other music applications, tracks the location of all plays. This location could be very useful in recommending people music. So our third use case is the 'Location Based Recommender'.

The second new use case that we came up with, was one aligned to another feature of Soundwave . Soundwave tracks your whole music playing history; It temporally records every song you play. An interesting feature would be to get recommended songs from a specific time period.

If one selected this week as the time period, this week's most popular songs would get recommended. If last summer was selected as the time period for example, you would get a snapshot of the most popular songs for those three months, and effectively get a blast from the past. This recommender is labelled then, the 'Snapshot Recommender'.

Both of these extra use case recommenders would be add-ons to the Soundwave application, and so would probably reside in the application menu, as opposed to the main pages. The menu is shown below in figure 3

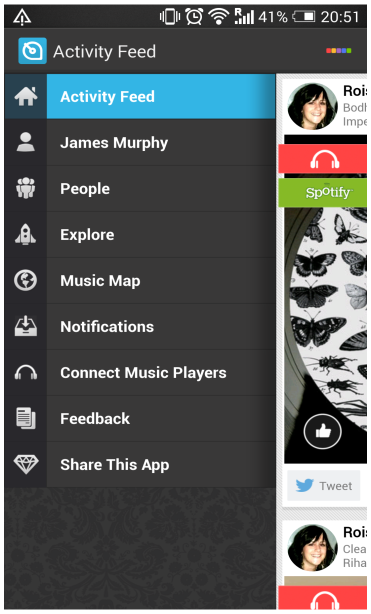


Figure 3: Menu Page

2 Datasets

2.1 Summary

Soundwave provided us with two datasets, to
test our recommender systems on. These were named Dataset A, and Dataset B. Below is table 1 summarising the characteristics of both datasets.

	DatasetA	*DatasetB*
Actions	161,770	1,340,280
Users	41,873	73,073
Songs	115,184	619,651
% of Users over 20 plays	1.9% (796)	24.6% (18,017)
Sparsity of Matrix	0.003	0.0002
Average plays per user	3.86	18.34
Standard Deviation of plays per user	4.98	33.17

Table 1: Table summarising Datasets

These datasets were held in a MySQL database, hosted locally on my PC. They were originally ported from a MongoDB database, as that was the implementation Soundwave had chosen to use. The data was stored in three tables. An Actions table, a Users table, and a Songs table.

Shown on the next page, in figure 4, is a a graph of the number of plays per user for dataset A. It is clear to see that the most active users are in the first 20,000 users in the dataset. After that not many have played over 20 songs. It is this low play count for a large number of the users, that will contribute to problems that will be outlined later in the report. Data sparsity is a big issue in recommender systems and these datasets are extremely sparse.

It is worth noting that these datasets are from the first three months of the application launch, so user profiles are not well established. This means that recommending to people is extremely tough, and accuracy scores may again be low.

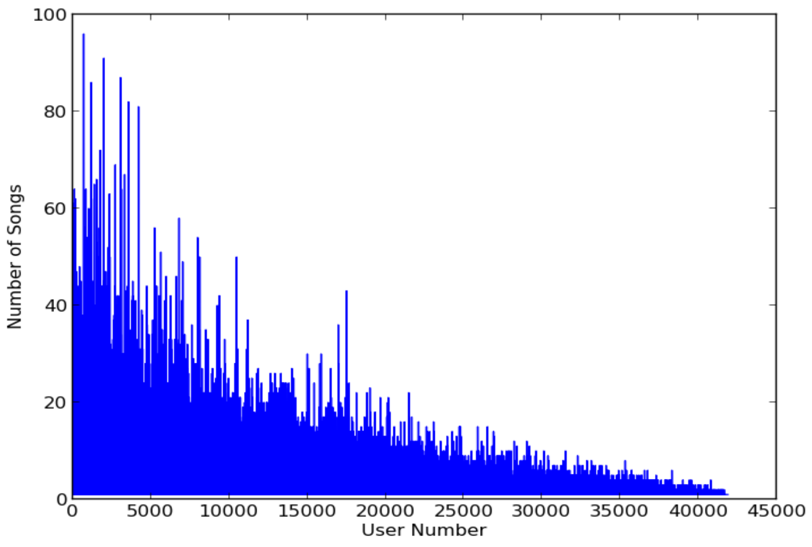


Figure 4: User Play Counts

2.2 Pre-computation

It was found out very quickly, that running these experiments for up to 73,000 users, or 619,000 items, was extremely slow. Tests were starting to take days to execute, so something had to be done. One positive aspect of working with dead data, is that it doesn't change. This could be used to our advantage.

To combat slow running times, all user similarities, and item similarities were pre-computed. This was done through the NumPy 'save()' method, storing a matrix of similarities. We could import this back into our running code every time with its accompanying function, 'load()'.

This had no negative side effects, only the positive of a much faster running times. In reality, this is what would happen in real life applications. A job would be constantly running in the background updating similarities, so this was a logical choice, and made running experiments much quicker.

3 Recommendation Techniques

3.1 Collaborative Filtering

Collaborative Filtering is widely used throughout our society. In a world where everything is getting more and more tailor-fit to the customer, collaborative filtering is one of the main approaches used to achieve this custom-fitting of goods. It is most simply described, by its founders, Goldberg, Nochols, Oki and Terry[7]:

> *"Collaborative filtering simply means that people collaborate to help one another perform filtering by recording their reactions to documents they read."*

This quote was in reference to the model that Goldberg et al used Collaborative Filtering for first - a mailing system. They described that the main theory of Tapestry[7], their mailing system, was that more effective filtering can be done by involving humans in the process. Indeed, using the reviews and thoughts of the mass, to decide on one individual, is an area being thoroughly harvested in all facets of life now.

Collaborative filtering comes in two main classes - Memory Based and Model Based [4]. Memory based operates over the entire user base to make predictions, whereas Model based uses the user base to train a model, and then uses this to make predictions. As noted by [14], memory based approaches work well in practice, and are more easily adapted - new data can be added incrementally.

Model based approaches can be more descriptive, in that they can give explanations for their recommendations [14]. However compiling the data into the model may be prohibitive, and adding new data may need a full recompilation. For this project, it was decided that memory based would be the better choice as they are more easily modified, and would allow for more experiments to be run.

Collaborative filtering may use explicit, or implicit data. Anything from a A single bit (recommended or not), to unstructured textual annotations[15] can be used as a basis for recommendations. To narrow down the scope of these applications relative to this report, it is necessary to state the uses made of collaborative filtering. Collaborative filtering in this report, is used in the context of unary data. This unary data is whether a given user played a given track. This method of using CF (Collaborative Filtering) makes evaluations quick, and provides simple manipulation of data.

One alternative approach could have been to use how many times users had played a specific song. We decided to use the unary approach, as it allows for easier and quicker use of data. More experiments can be run, which was an initial goal - to run as many experiments as possible.

There are two main types of collaborative filters, user-user filters, and item-item filters[6]. These are both discussed in detail, and implemented in this project. Item-Item filtering has been used extensively in our project, which was highly promoted in Amazon's recommendation systems, using methods documented in [12].

Collaborative filtering through Item-Item filtering, User-User filtering, or indeed both, is used in all recommenders listed below, except for the Snapshot Recommender, and Location Based Recommender. Collaborative filtering adaptations of these were implemented through user-user re-ranking of the recommendations.

The other main recommending technique, content filtering [2], has not been adopted in this project. Content filtering uses the descriptive aspects of the songs (artist, album, tags) to recommend to users. In future iterations of these recommenders, it is hoped that some aspects of content filtering can be integrated, as it can help greatly in areas where collaborative filters struggle, like recommendations for new users (new user problem) [3].

3.1.1 Limitations

Dead Data

When we are testing a recommenders output, the best case scenario is that we have the actual current user we are testing present, to confirm whether a given recommendation is good or not. However in our tests, since we were always working with data dumps from Soundwave, we call this 'dead data' [10]. This data is a history of past users' activities. It is because of this, that we had to assert from that person's past playing history if a recommendation was good or not. This is called offline evaluation. Whilst working with this retrospective look at the data is easier in the fact that it is static and can be reused, it has some quite important limitations.

The main issue is that to test these recommenders we have to hide some data, and try predict the data we have hidden. The problem with this, is that these hidden data evaluations actually punish algorithms that find new data [9]. The recommender could recommend a perfectly good song, one the user would love to have played, but since they haven't played it previously, the recommender experiment will punish the system as it thinks it is a bad recommendation. It will not be in the user's hidden data set, and therefore will evaluate to a false positive recommendation.The best metric to use, is to gauge how people reacted to their recommendation, and score the recommender accordingly.

These good recommendations, that the system does not know about for that specific user, that lower system accuracies, are part of a issue called push-down [9]. Push down are the good recommendations, that 'push down' other recommendations, and lower the system accuracy score. If we had user testing, we would know these are good recommendations, and they would improve the system score. However, since we are dealing with dead data, they negate the system score.

The last issue with offline evaluations and dead data, is the sparsity of data limiting the set of items that can be evaluated [8]. We cannot evaluate the appropriateness of a recommended item for a user if we don't have a rating from that user for that item in the dataset.

Play Data Noisy

As stated before, we are working with unary data - played or not played. The reason this is called unary, and not binary, is because we don't know if the user actually liked this song. We don't know if they played it and enjoyed it, or played it and didn't like it [8].

Just because a user played something, does not mean they like it, or want to hear it again. They could have been too lazy to change the next song, or just have been in a particular mood in which they tolerated that song. That means for all future evaluations, this implicit rating will have some impact, when in reality, the user doesnt even like this song. Because we are just dealing with plays in these datasets, and not likes also, we can never be fully sure that a user likes the song they are playing, or ever want to hear it again. Play data is extremely noisy in that respect.

Gray Sheep/Black Sheep

Two user groups that make recommendation very tough are labelled gray sheep and black sheep. Gray sheep are that group of people, whose playing habits do not agree any other group in the system, but also do not disagree with any group in the system, and so collaborative filtering does not benefit them [5].

Black sheep are those people who tastes are so idiosyncratic that making good recommendations is nearly impossible [13].

Both these types of people impact recommendation systems, and are ongoing issues in industry .

User Cold Start

Because user-user collaborative filtering requires matching you with other similar profiles to make recommendations, if you do not have many items in your profile, recommendations are usually poor, as there are very few people to match you with accurately. This issue, is called cold start. User-user collaborative filtering struggles intently with cold start. On the other hand, content filtering, which uses the descriptions of songs in your playing history, copes really well with cold starts, but struggles to find serendipitous recommendations.

3.2 Location Filtering

Location filtering is implemented simply by using the latitude and longitude fields in the play data. The Haversine formula, described in figure 8, is used for measuring distances between two geographical points. This is implemented in both Location Based recommenders.

3.2.1 Limitations

Retrieving details from Echonest

Retrieving the tempo of tracks from Echonest was a problem with the justification step of these recommenders (described later). To get the tempo of every song, we needed to use the song name, and artist name to programatically query Echonest . Since most of the users plays were from their local storage, they may have have input the song and artist name themselves. Any discrepancies in putting in either, would mean not being able to match that track in Echonest. On average per user, our systems could not match 7.2% of songs with Echonest. This figure is high, but this is an issue all large scale music recommenders struggle with, and is an issue for another report.

3.3 Temporal Filtering

Temporal filtering is implemented through the date field in the play data. This is used in both Snapshot recommenders, where a given time period can be specified to be recommended from.

3.3.1 Limitations

Time frame

Since this recommender runs on all previous user play data, we can only recommend from when the application was live. This, in our case, was June 2009. This is a major drawback for this recommender. A very nice feature would have been to have been able to recommend from the start of time. As we use previous Soundwave data for this however, this is not possible. As the application ages, it will have more and more data to recommend from, and this feature will get more and more useful.

3.4 Calculating Similarity

When choosing measures of similarity, a first attempt was to use Jaccard Similarity, as this is the most basic measure of comparing any two items in computer science. Further research [6] indicated however, that Pearson Correlation, and Cosine Similarity would be the best measures to use. For our re-ranking algorithms, Spearman Coefficient was also deemed the most appropriate to evaluate the amount of re-ranking [6].

3.4.1 Jaccard similarity

The simplest way of measuring how close two sets of data are, is seeing how many things they have in common. This is what Jaccard similarity does. It takes the size of the set intersection of the two sets and divides this by the size of the union. These are then ranked by the sets which have the highest set intersection, the most similar data sets according to this similarity measure. It is described below in figure 5, which is originally from Wikipedia[4]

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Figure 5: Jaccard Similarity Formula

Of the three measures, this is the most primitive, and thus is the first measure we used when conducting our tests.

[4]http://en.wikipedia.org/wiki/Jaccard_index

3.4.2 Pearson Correlation

Pearson Correlation is a slightly more complicated formula and measure of similarity. It is defined as the covariance (how random variables change together) of two sets of data, divided by their standard deviations, and usually represented by the letter 'r'. The formula is shown below in figure 6, originally from Wikipedia [5]

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

Figure 6: Pearson Correlation Formula for a sample

3.4.3 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors, that measures the cosine of the angle between them. Cosine similarity turned out to be the most accurate similarity for most of our experiments, which is not surprising as it is the most widely used in industry. Cosine similarity did have one drawback however, in that it was slower to evaluate than Pearson and Jaccard similarity. Cosine similarity also, does not factor in negative votes, but this is fine in our case because we don't have any negative data. The formula for Cosine Similarity is shown below in figure 7 which is originally from Filo Technologia [6]

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum\limits_{i=1}^{n} A_i \times B_i}{\sqrt{\sum\limits_{i=1}^{n} (A_i)^2} \times \sqrt{\sum\limits_{i=1}^{n} (B_i)^2}}$$

Figure 7: Cosine Similarity Formula

[5]http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient
[6]http://filotechnologia.blogspot.ie/2013/09/implementation-of-cosine-similarity.html

3.4.4 Haversine Distance

Haversine distance measures how far away two geographical points are from each other. Unlike Euclidean distance, it takes in the curvature of the earth into account, which makes it much more accurate, especially over large distances. It is denoted by the following formula in figure 8

$$\operatorname{haversine}\left(\frac{d}{r}\right) = \operatorname{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\operatorname{haversin}(\lambda_2 - \lambda_1)$$

Figure 8: Haversine Distance Formula

where haversin is the following formula, figure 9

$$\operatorname{haversin}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

Figure 9: Haversin Formula

These are both originally from Wikipedia [7]

[7] en.wikipedia.org/wiki/Haversine_formula

3.5 Recommender Configuration Choices

3.5.1 Novelty

Soundwaves slogan is 'Music Discovery', and is an application is for finding new music. It is because of this reason, that the systems described below were designed to be highly novelty based. Novelty denotes how seldom a system recommends you items you already know about.

Novelty can be defined in relation to whether you've played a certain song already, or at a more detailed level, whether you've played anything by a certain artist/genre etc. We shall only deal with Novelty in relation to the former, if a person has played a certain song before, as its not possible from our data to ascertain the latter.

If a user wants to hear a particular song they've played before, they can play it locally if they have it in their library, or indeed use any of the music streaming services Soundwave supports.

It should be noted that the system only knows about the songs you and others have played. There is the possibility that a recommender could recommend you songs that you have in library but just have not played yet. It this sense, the novelty is limited to the data the application has access to.

3.6 Technologies Used

3.6.1 Python

Python [8] was used in all aspects of this project. All the recommenders were coded in Python, queries to Echonest were done through Python, and database queries were done in SQL through Python.

Python is a high-level programming language that emphasises code readability. Its syntax allows for concise code, that it still legible. Its dynamic type system, and memory management mean that it is extremely useful for working with different types and sizes, of data. This is the reason we chose python, it can handle big data effectively.

[8]https://www.python.org/

It also has an extremely useful machine learning library called Sci-Kit Learn detailed below.

3.6.2 Sci-Kit learn

Sci-Kit learn [9] is a machine learning library for Python for data mining and data analysis. It has implementations of most of the main algorithms used in data mining, and saves programmers reinventing the wheel. We used sci-kit learn for clustering, cosine similarity, Pearson correlation and many other pre-built functionalities.

3.6.3 SciPy

Scipy is another Python extension, which proved invaluable to this project. It's subset, NumPy, allowed us to pre-compute data through its function 'save()'. Passing a matrix into this function would save it to disk, and allowed us circumvent upwards of 30 days processing time. Its sister function, 'load()' allowed us to import that matrix back into our running program.

Matplotlib, another subsection of SciPy, allowed us to plot graphs easily.

3.6.4 MySQL

SQL(Structured Query Language) allows you to interface with a RDBMS(Relational Database Management System). MySQL[10] is an open-source implementation of a database, and the implementation we decided to use for this project, as I had previous experience of it, and its competitors do not offer much in the way of advantages for this particular project.

Soundwaves data was stored in three tables in a MySQL database. This was accessed to compute similarities between users and items. This database was stored locally on my machine.

[9]http://scikit-learn.org/stable/

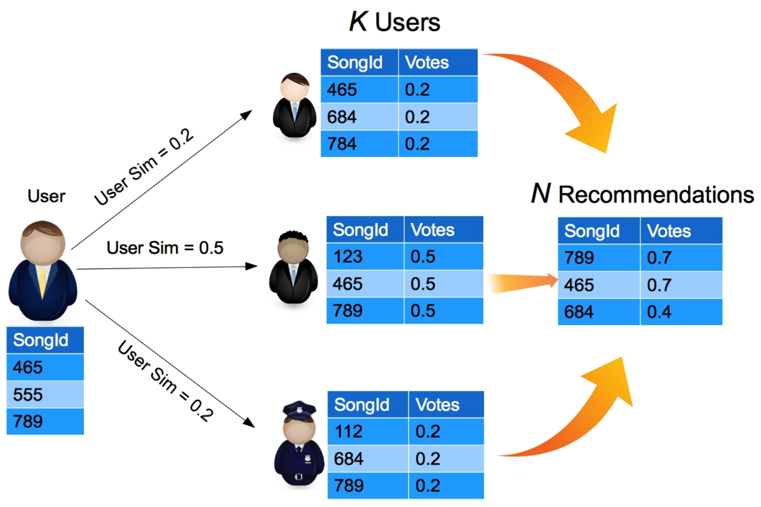
[10]www.mysql.com

4 Profile Based Recommenders

4.1 User-User Recommender

A user-user recommender operates using user profiles as a means of filtering. To make a recommendation for a user, the system take the songs that the active user has played, and see which users in the dataset are most similar to this user.

Firstly, the system stores the top-K most similar users. Candidate songs are ones that these users have played but which the active user hasn't played. Each candidate gets votes from the users who have played the song, the votes being how similar that user was to active user. So if a candidate song is 0.6 similar, this would mean the cumulative similarity of the users who have played that song would have been 0.6. The top-N number of songs are recommended, ranked by votes. A diagram describing the flow chart of this recommender is depicted below in figure 10



SongId
465
555
789

SongId	Votes
465	0.2
684	0.2
784	0.2

SongId	Votes
123	0.5
465	0.5
789	0.5

SongId	Votes
112	0.2
684	0.2
789	0.2

SongId	Votes
789	0.7
465	0.7
684	0.4

Figure 10: User User Recommender Flow Chart

4.1.1 Experimental Methodology

To test the accuracy of this recommender, it needed to be run on all users. Every user's ratings were split randomly into train and test data. The values used for this split were 70% train, and 30% test.

The recommender was built from the train data. Its recommendations were scored on their overlap with the test data. This recommender had three different variable settings. Firstly, was the similarity measured used to find similar users. Three different similarity measures were used: Jaccard Coefficient, Pearson Correlation and Cosine Similarity.

Second, was the number of nearest neighbours we stored in our top-K[11] users. Three different values were used: 15, 20 and 25.

Lastly was the number of recommendations the system would output. Again for this we used three different values: 5, 10 and 15.

Shown below is table 2 summarising all of these results, with winning values for each dataset in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		K=15	K=20	K=25	K=15	K=20	K=25
Jaccard Coefficient	N=5	0.021	0.031	0.043	0.029	0.033	0.035
	N=10	0.023	0.030	0.051	0.034	0.037	0.037
	N=15	0.031	0.035	0.061	0.023	0.039	0.043
Pearson Correlation	N=5	0.043	0.081	0.102	0.055	0.057	0.079
	N=10	0.077	0.087	0.124	0.063	0.069	0.081
	N=15	0.080	0.096	0.101	0.082	0.088	0.098
Cosine Similarity	N=5	0.068	0.071	0.123	0.072	0.094	**0.131**
	N=10	0.048	0.056	0.114	0.084	0.087	0.096
	N=15	0.063	0.093	**0.127**	0.064	0.101	0.116

Table 2: Table of User User Recommender Results

[11]It should be noted that any use of 'K' from here on in denotes the number of nearest neighbours a system uses. Any use of 'N' denotes the number of recommendations that the system gives.

4.2 Item-Item Recommender

An item-item recommender is the opposite of a User-User recommender. Instead of getting similar users, the system get similar items and recommends those. Items can be classed as similar in terms of the users that have played said items. To recommend to one user, the system takes each of their tracks individually, and gets the K most similar tracks to every one. Each of the top K most similar tracks gets a weighted vote, based on how similar it is to the original tracks. This process is repeated for each song the user has played.

Tracks are stored in a dictionary, with repeated tracks votes cumulated. The system then takes the top-N tracks, and recommends them to the user, ranked by votes.

A diagram describing the flow chart of this recommender is depicted below in figure 11

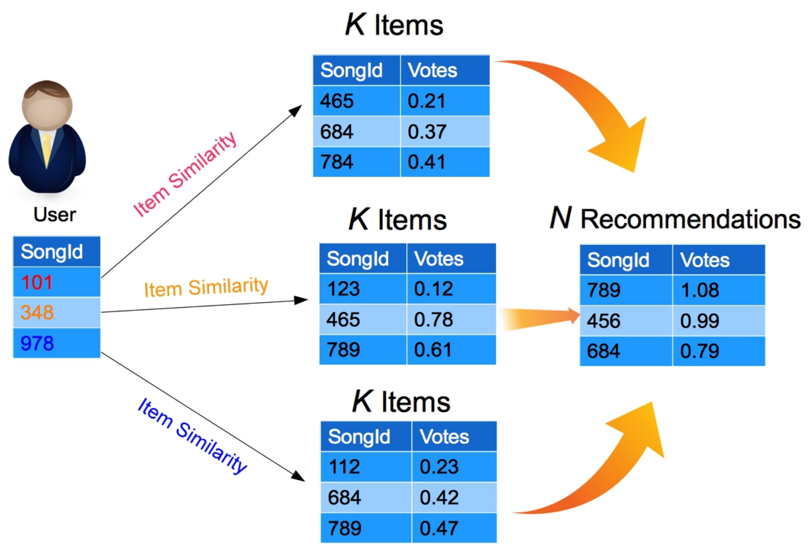


Figure 11: Item Item Recommender Flow Chart

4.2.1 Experimental Methodology

To test the accuracy of this recommender, it needed again be run on all users. To do this, every users data was split randomly into train and test data. The values used for this split were 70% train, and 30% test. Since this recommender is for the same use case as the previous recommender, the experimental methodology is the exact same. In this way, we can compare the recommenders.

The recommender was built on the train data. Its recommendations were scored on their overlap of the test data. This recommender had three different variable settings. Firstly, was the similarity measured used to find similar users. Three different similarity measures were used. Jaccard Coefficient, Pearson Correlation and Cosine Similarity.

Second, was the number of nearest neighbours we stored in our top-K items. Three different values were used: 15, 20 and 25.

Lastly was the number of recommendations the system would output. Again for this we used three different values: 5, 10 and 15.

Shown below is table 3 summarising all of these results, with winning values in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		K=15	K=20	K=25	K=15	K=20	K=25
Jaccard Coefficient	N=5	0.022	0.024	0.031	0.025	0.042	0.051
	N=10	0.048	0.051	0.037	0.047	0.046	0.042
	N=15	0.027	0.031	0.036	0.033	0.028	0.034
Pearson Correlation	N=5	0.038	0.051	0.056	0.058	0.054	0.065
	N=10	0.049	0.047	0.069	0.048	0.064	0.066
	N=15	0.030	0.042	0.057	0.036	0.048	0.048
Cosine Similarity	N=5	0.096	0.091	0.118	0.122	0.134	0.125
	N=10	0.122	0.128	0.132	0.084	0.087	0.112
	N=15	0.149	0.148	**0.156**	0.169	0.162	**0.176**

Table 3: Table of User User Recommender Results

4.3 Evaluation

By analysing the accuracy statistics from both recommenders, we can see that the Item-Item recommender is the more accurate implementation of the profile based recommender on the whole part. The winning values are higher for the Item-Item recommender, and the results on the whole are higher, bar one set. That one set, Pearson correlation does not perform as well in the Item-Item recommender. A reason for this is hinted at in [16], where Pearson Correlation does not fare as well as Cosine similarity in tests on Item-Item recommenders.

Empirically shown in [16] also, is how item-item recommenders outperform user-user recommenders at at all data sparsity levels, but especially low sparsity data. This is why I believe the item-item recommender mainly won in our tests - our data is extremely sparse.

Cosine similarity is shown to be the most accurate way of deeming similarity in both recommenders. In the user-user recommender, Pearson correlation is also quite accurate, but its accuracy scores are not as accurate as the cosine similarity scores.

K equal to 25 is deemed to be the best number of nearest neighbours for this algorithm. It performs better in both dataset A, and dataset B, on both recommenders, so there is no ambiguity.

For the number of recommendations, there are some discrepancies. In the user-user recommender, N = 5 is the most accurate result for dataset B. For dataset A however, N = 15 is the most accurate result. As dataset B is a better representation of Soundwaves data (it's less sparse, and has more data), I would recommend that N = 15 be used for the user-user recommender. For the item-item recommender, we have no ambiguity. N = 15 is shown to be the most accurate setting for the number of recommendations for this system.

As a result of the Item-Item recommender being more accurate in our tests on the Soundwave data, and the fact the Item-Item recommender outperforms the user-user recommender in [16] also, I would suggest Soundwave implement an item-item recommender as a profile based recommender, with cosine similarity as a similarity measure, 25 nearest neighbours as a value of K, and 15 recommendations as a value of N.

5 Song Based Recommenders

5.1 Item Recommender

The second use case that Soundwave gave was that if someone had just played a song you liked, that they could get recommendations, based on that song. This recommender does just that. It is a simplification of the Item-Item recommender. Instead of using all of a user's tracks to find similar tracks, it just uses the one track, and finds the most similar songs to that. Again, tracks are similar if the same user has played them.

The system will recommender the top-N most similar items to the seed song, ranked by similarity. In this system, K=N, because the number of nearest neighbours is the number of recommendations given. For results K will be used, as it is the number of nearest neighbours we are really configuring. It just so happens in this recommender, that this number is also the number of recommendations given.

A diagram describing the flow chart of this recommender is depicted below in figure 12.

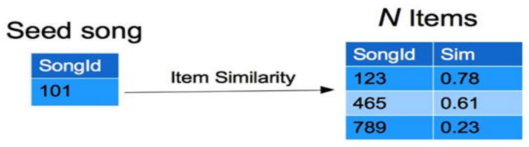


Figure 12: Item Recommender Flow Chart

5.1.1 Experimental Methodology

To test the accuracy of this recommender, it needed again be run on all users. To do this, every user's data was split randomly into train and test data, 70% train, and 30% test.

This experiment differs from the previous two recommenders from now on. Instead of the recommender being run on all the train data, it is instead just run on one song per user, picked randomly from the user's train data. The recommender was then scored on the overlap of the output of the results and the test data. This recommender has two different variable settings.

Firstly, was the similarity measured used to find similar users. Three different similarity measures were used. Jaccard Coefficient, Pearson Correlation and Cosine Similarity. Lastly, was the number of nearest items we stored in our top-K items. Three different values were used: 5, 10 and 15.

As an extra test in these recommendations, we measure precision at three values. Later, in the next recommender, we want to see if the re-ranking has made an improvement to the results. Because of user re-rank, better results might get pushed up, and improve precision at 5, or precision at 10, because more suited songs to the user will now be higher in the recommendation list.

Shown below is table 4 summarising all of these results, with winning values in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		K=5	K=10	K=15	K=5	K=10	K=15
Jaccard Coefficient	Precision at 5	0.016	0.019	0.019	0.012	0.018	0.021
	Precision at 10	-	0.022	0.027	-	0.014	0.017
	Precision at 15	-	-	0.016	-	-	0.017
Pearson Correlation	Precision at 5	0.021	0.024	0.027	0.021	0.019	0.027
	Precision at 10	-	0.028	0.031	-	0.018	0.027
	Precision at 15	-	-	0.043	-	-	0.031
Cosine Similarity	Precision at 5	0.036	0.049	**0.077**	0.071	0.078	**0.113**
	Precision at 10	-	0.067	0.065	-	0.053	0.072
	Precision at 15	-	-	0.057	-	-	0.064

Table 4: Table of Item Recommender Results

5.2 Item Recommender with User-User Re-rank

This recommender is an enhanced version of the Item recommender. Like it, the system stores the K most similar neighbours to the seed song, ranked by similarity. The difference is that these K songs are then used as candidate songs for a user-user re-rank . The songs will be re-ranked, in terms of the profile of the active user.

For each of these K items, the similarities between the user using the system, and the top K users who have played that song are recorded, and cumulated. This gives each song some votes, based on how similar you are to your closest neighbours that have played that song.

Again, in this system, K=N, because the number of nearest neighbours is the number of recommendations given. We reuse this same value of K for the number of nearest neighbours we use for the User-User rerank. As these are both nearest neighbour operations, we used the same value.

This is repeated for each of the K items, and the list is then re-ranked according to these user-scores. So the end result, is a personalised recommendation list on a given input song. A diagram describing the flow chart of this recommender is depicted below in figure 13

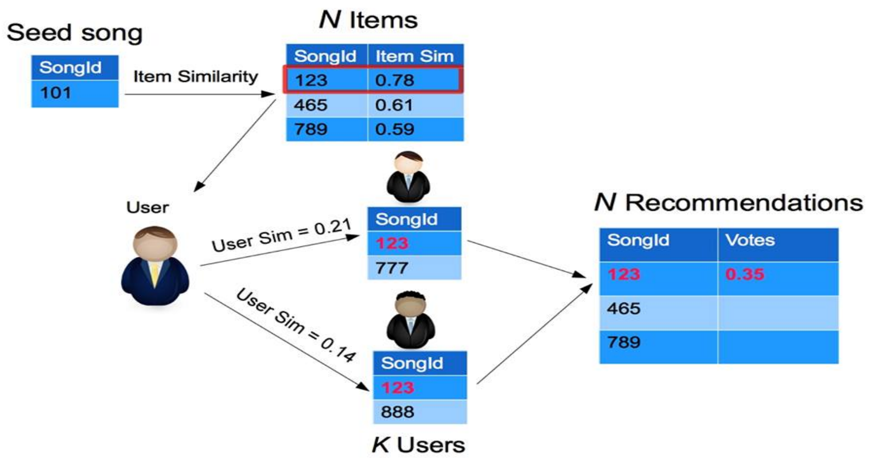


Figure 13: Item Recommender with User-User re-rank Flow Chart

5.2.1 Experimental Methodology

The experimental methodology for this recommender is the exact same as the one for the Item recommender.

We do perform one extra measurement on this recommender. We measure the Spearman Coefficient. The Spearman Coefficient measures how much the list was re-ranked. Figure 14 gives the formula for the Spearman Coefficient.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

Figure 14: Spearman Coefficient

We computed this to see if it was worth re-ranking the list, as if the re-ranking did not make much of a difference to the list, it was not worth the overhead of the User re-rank. The results of these tests are shown below in table 5, with lowest and highest values in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		K=5	K=10	K=15	K=5	K=10	K=15
Jaccard Coefficient	Spearman Coefficient	0.22	0.29	0.38	0.36	0.38	0.31
Pearson Correlation	Spearman Coefficient	0.34	0.32	0.24	0.28	0.33	**0.39**
Cosine Similarity	Spearman Coefficient	**0.12**	0.29	0.36	0.24	0.35	0.27

Table 5: Table of Spearman Coefficient Results

The three same similarity measures were used as before: Jaccard Similarity, Pearson Correlation and Cosine Similarity. We used three values of K, the number of nearest neighbours: 5, 10 and 15.

We again, measure precision at three values, because this time we want to actually check has the re-ranking made the results more accurate.

Because these are the exact same songs as before, we will not see differences in accuracy at Precision at 15 recommendations. If the recommender has worked effectively however, we may see improvements in accuracy at precision at 5, and precision at 10, as good items may have been pushed up from 10th-15th place in the original ranking, to 5th-10th place in the new re-ordering.

Shown below is table 6 summarising all of recommender accuracy results, with winning values in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		K=5	K=10	K=15	K=5	K=10	K=15
Jaccard Coefficient	Precision at 5	0.033	0.027	0.023	0.014	0.021	0.019
	Precision at 10	-	0.022	0.029	-	0.017	0.018
	Precision at 15	-	-	0.016	-	-	0.017
Pearson Correlation	Precision at 5	0.016	0.033	0.035	0.028	0.025	0.047
	Precision at 10	-	0.034	0.027	-	0.027	0.031
	Precision at 15	-	-	0.043	-	-	0.031
Cosine Similarity	Precision at 5	0.043	0.074	**0.091**	0.069	0.080	**0.128**
	Precision at 10	-	0.066	0.068	-	0.048	0.076
	Precision at 15	-	-	0.057	-	-	0.064

Table 6: Table of Item Recommender With User User re-rank Results

5.3 Evaluation

Lets first deal with the Spearman Coefficient, as this is the basis behind developing the re-ranked version of this recommender. Results from the Spearman Coefficient experiment show that it augments lists between 0.12 to 0.39 from their original ranking. These are significant changes from the original lists. So it's fair to say that the re-ranking is changing the lists considerably. Now what we need to verify is that it changed the lists for good, and not arbitrarily changed them.

As would be expected, precision at N for N is the same in both recommenders. It is only precision at 5 for N = 10, and precision at 5 and 10 at N = 15 that would show differences in the re-rank system. Good songs may have got pushed up, and improved the recommendation list.

From analysing the results, we can see that the re-rank version of the recommender outperformed the non-user-ranked version on almost all tests. Most importantly, the highest accuracy of the re-ranked system is better (0.128 as opposed to 0.113). This shows, on the right setting, the re-rank system is over 1% better. Which might not sound significant, but we are dealing with low accuracies.

Along side this, the overall scores of the re-rank system are also higher on inspection. Only precision at 10 for Pearson Correlation on dataset A, and precision at 5 for Jaacard Coefficient on dataset B are higher in the original system. This shows that the re-ranking is effective at improving accuracy scores.

To decide which value of K to use (the number of nearest neighbours/the number of recommendations), we can analyse the results. The results indicate that K = 15 is the best value to use. The highest precision values of both datasets, and both recommenders is K = 15, so we can conclude this is the most accurate setting.

So I suggest that Soundwave implement the Item Recommender with User-User Re-rank as their Song Based recommender, with K equal to 15.

6 Location Based Filtering

6.1 Justification

To be able to justify recommending by location, we first needed to verify that people played different types of music, in different locations. To do this we used clustering of user's data.

6.1.1 Clustering

To see if people played different types of music in different locations, we used clustering on users data by location. For every user, their plays were clustered by the location where they were played.

This is shown below in figure 15, which is an example of the output for a example user.

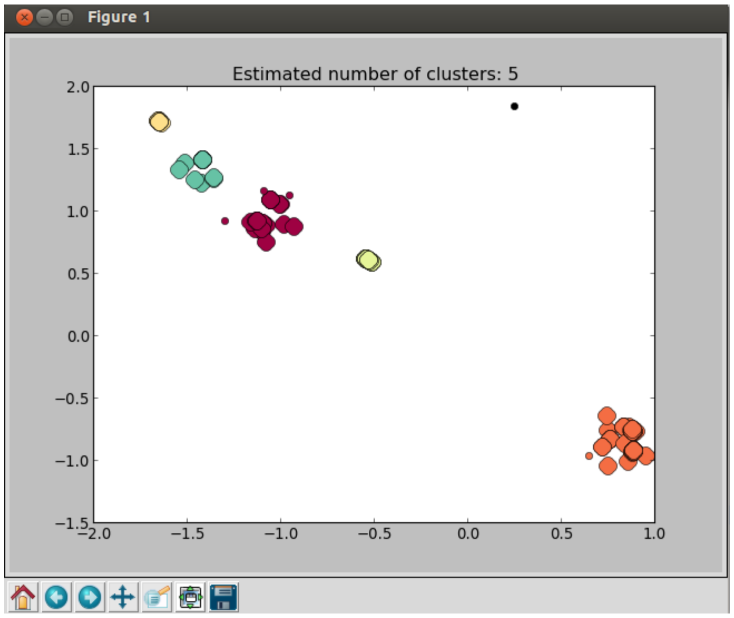


Figure 15: Output of a clustering example

Each cluster was then analysed to find out its composition, in relation to tempo. Tempo gives a good indication whether the music is of a specific genre. In future work, it would be nice to consider more aspects of the tracks to deem similarity. However, time did not allow for this.

Clusters were analysed in relation to average cluster tempo, and standard deviation of tempo. That way, it was achievable to say whether a specific type of music was played in that cluster. Standard deviation was computed for every user, and an average standard deviation for all users was attained.

The overall average for all users is shown below in table 7

	$DatasetA$	$DatasetB$
Average Standard Deviation	31.12	26.17

Table 7: Table of Standard Deviation Average

It could nowhere be verifiable how much a music genre ranged in BPM (Beats Per Minute), but online resources [12] were indicating that a normal genre ranges from anything from 10 BPM, to 40 BPM. Obviously there are some genres that range more than others (Techno is a good example), but on average 10-40 BPM is a normal range.

The values we derived so, were inside what an average music genre ranges. Thus, we had justification for building our Location Based Recommender.

6.1.2 Algorithms

DBSCAN

The clustering algorithm that was used, was called DBSCAN. This is a density based clustering algorithm which is good for this application, but also this clustering algorithm does not need to be told the number of clusters before running, unlike K-means. This is an obvious advantage. One thing this algorithm does work better with however, is its definition of near-ness. This value, is called Epsilon

[12] http://music.stackexchange.com/questions/4525/list-of-average-genre-tempo-bpm-levels

Choosing of Epsilon value

One aspect of DBSCAN that enhances its clustering ability is a good value for epsilon. This defines the distance between two points for them to be considered in the same cluster. Tests were run over all users, to see which value of Epsilon yielded the highest average. We used Silhouette Score to analyse cluster quality for different values of epsilon. This score determines how well clustered a set of given points are. The formula for Silhouette Score is shown below in figure 16. It can be explained as the following - a(i) is the average dissimilarity of points within a cluster, b(i) is the lowest average dissimilarity of of i to any cluster it is not not in. So a(i) really measures how well i is assigned to its cluster, b(i) measures how well separated and succinct all clusters are.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Figure 16: Silhouette Coefficient Formula

Depicted below in table 8 is our results from tests over all users on silhouette scores, with winning values in bold.

Epsilon Value	Dataset A Silhouette Value	Dataset B Silhouette Value
0.1	0.453	0.552
0.3	**0.653**	**0.756**
0.5	0.650	0.747
0.7	0.654	0.721
0.9	0.648	0.665
1.1	0.498	0.623
1.3	0.472	0.622
1.5	0.431	0.498

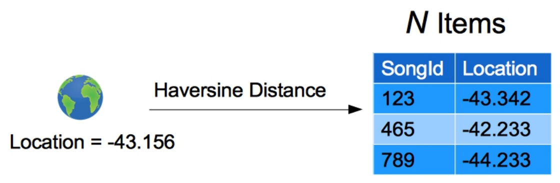
Table 8: Table of Epsilon Value tests

From these results it was deduced that 0.3 was the best epsilon value, so this was used for the recommender. 0.7 was slightly more accurate on the smaller dataset, but overall 0.3 was the best value. Aside from these results which attest to a smaller value being more accurate, a small value suited us better also, because the smaller our clusters were, the more accurate our recommender would be. If clusters were too large, for example, our 'gym' cluster, could merge into an area beside it, and classifying that cluster as an up-tempo cluster could be more difficult.

6.2 Location based Recommender

After we had established justification for implementing our third use case, a location based recommender, we went about building it. The way this recommender worked was that it took your current location, and recommended you the most popular songs that were within 1 kilometre of your location. Popularity was defined as the most played songs, within that 1 kilometre geofence. The system would recommend you N items, ranked by the popularity of the songs.

Shown below in figure 17, is a flow chart of the Location Based Recommender



SongId	Location
123	-43.342
465	-42.233
789	-44.233

Figure 17: Location Based Recommender Flow Diagram

6.2.1 Experimental Methodology

To test how accurate this recommender was, we ran it for all users. For each user, the system picked one track randomly. It would use the location of this track as the seed location for the recommender. The recommender ran on that location, and was scored on the overlap between the output of the recommender, and any other songs the user had also played within that 1 kilometre geo-fence.

Three values of N were used in this experiment for the number of recommendations.

Show below is table 9 which shows the results of the Location based recommender, with winning values in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		N=5	N=10	N=15	N=5	N=10	N=15
Haversine Distance	Precision at 5	0.045	0.039	**0.046**	0.067	0.069	**0.076**
	Precision at 10	-	0.031	0.036	-	0.061	0.062
	Precision at 15	-	-	0.032	-	-	0.051

Table 9: Table of Location Based Recommender Results

6.3 Location based Recommender with User-User Re-rank

This recommender works the exact same as the last Location based recommender. It will recommender the N most popular songs, within a 1 kilometre range of your current location.

The only difference is, we implement a User-User re-rank, as we have done before. This will re-rank the list, with the most suited songs to the user being recommended first. The end result is a personalised list to the user of the most popular songs in that 1 kilometre geo-fence.

Shown below in figure 18, is a flow diagram of the Location based recommender with User-User re-rank.

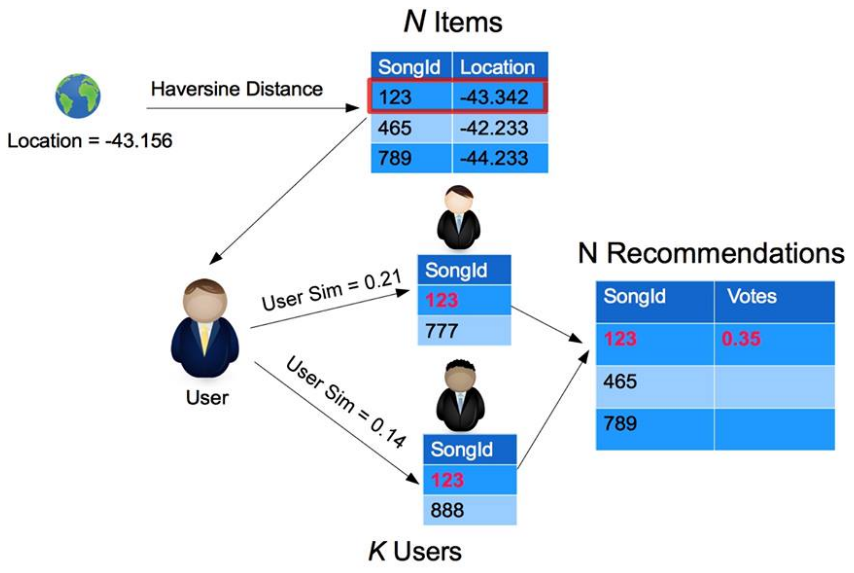


Figure 18: Location Based Recommender with User-User re-rank Flow Diagram

6.3.1 Experimental Methodology

To test this recommender, we perform the exact same experiment as we used in the last recommender. For all users, we pick one song randomly and use this as our seed location. We then run the recommender on this seed song, and score it on the overlap between this and the other songs that user has played in that 1 kilometre geo-fence.

As before we use three values of N for the number of recommendations - 5, 10 and 15. We also use K in this system, as there is a User-User re-rank. As these are both nearest neighbours operations we use the same value for N and K.

The thing we are looking for here, is the precision at 5 values, and the precision at 10 values. Like before, we want to see if any 'good' songs have been pushed up by the re-ranking, and thus improved the recommender accuracy.

Shown below in table 10, are the results from the experiments on this recommender, with winning values in bold.

Similarity Measure	Metric	Dataset A			Dataset B		
		N=5	N=10	N=15	N=5	N=10	N=15
Haversine Distance	Precision at 5	0.045	0.043	**0.072**	0.067	0.075	**0.084**
	Precision at 10	-	0.031	0.042	-	0.061	0.058
	Precision at 15	-	-	0.032	-	-	0.051

Table 10: Table of Item Recommender With User User re-rank Results

6.4 Evaluation

These results are extremely low because of the sparsity of plays geographically in this dataset. The main reason for this, is as they are just from the first three months of the application. To see how good these recommenders really are, we need to run them on a full dataset.

On our data, the results don't differ massively, but by analysing the results, we can see:

For dataset A, when N was equal to 15, precision at 10 is better in the re-rank version of the recommender(0.042, as opposed to 0.036). Precision at 5 for N equal to 15 is higher for the re-rank recommender (0.072 as opposed to 0.046 in the normal version). Precision at 5 for N equal to 10 is also higher in the re-rank version of this recommender (0.043 as opposed to 0.039)

For dataset B this is mostly also the case. Precision at 5 for N equal to 15 is in the re-rank version of the Location Based Recommender (0.084 as opposed to 0.072). Precision at 5 for N equal to 10 is also higher in the re-rank recommender (0.075 as opposed to 0.069 in the normal version). The only exception is precision at 10 for N equal to 15, this is lower in the re-rank recommender (0.058 as opposed to 0.062). This is a minimal difference however, and on the whole, the re-rank system is more accurate than the non re-rank version of this recommender.

To find which value of N is most accurate, we can again analyse the results. It's clear to see that N = 15 is a winner. 15 recommendations is most accurate in both datasets on the original recommender, and is also the most accurate on both sets on the re-ranked recommender.

We can say therefore, that the User-User re-rank version of the Location Based Recommender is the more accurate, and better implementation of this recommender, and the version that should be implemented by Soundwave with N = 15.

7 Temporal Based Filtering

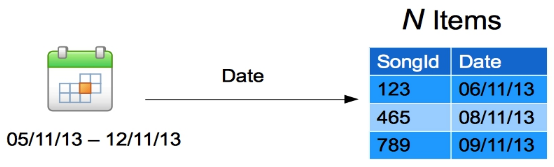
7.1 Snapshot Recommender

The last use case, to build a recommender that was based around time, was called a Snapshot recommender. This recommender worked in the following manner. You would supply this recommender with a start date, and an end date. This system would then recommend you the N most popular songs that were played in this time window, ordered by popularity.

There are a few draw backs to this recommender that have already been outlined. It can only recommend from the time the application was live. It is fair to say however though, that as this application ages that this feature will become more and more useful.

For example, for someone who was abroad last summer and wanted to make a video to document it, this feature would come in very handy for getting the most popular songs for those three months.

Shown below in figure 19, is a flow diagram of the Snapshot recommender.



SongId	Date
123	06/11/13
465	08/11/13
789	09/11/13

Figure 19: Snapshot Recommender Flow Diagram

7.1.1 Experimental Methodology

To test this recommender, we need to as always test it over all users to get the most accurate precision rating. The experiment used to test this recommender was as follows. For each user, we take the first song they have played chronologically. We use this as the start date for the recommender period.

From here we test the recommender with time periods from this start date, to D days later. We have used three values of D for precision of accuracy - 5 days, 10 days and 15 days.

When we run the recommender on the start date, plus D, we then score the recommender on the overlap of its output, and the songs the user had also played in that time frame.

Again, three values of N are used, for the number of recommendations.

Shown below is table 11 summarising the results, with winning values in bold.

Similarity Measure	D Value	Metric	Dataset A			Dataset B		
			N=5	N=10	N=15	N=5	N=10	N=15
Date	D=5	Precision at 5	0.031	0.037	0.039	0.044	0.052	0.061
		Precision at 10	-	0.023	0.027	-	0.047	0.059
		Precision at 15	-	-	0.033	-	-	0.045
	D=10	Precision at 5	0.037	0.038	0.038	0.053	0.057	0.079
		Precision at 10	-	0.025	0.033	-	0.058	0.072
		Precision at 15	-	-	0.036	-	-	0.067
	D=15	Precision at 5	0.039	0.043	**0.045**	0.049	0.066	**0.084**
		Precision at 10	-	0.028	0.044	-	0.051	0.068
		Precision at 15	-	-	0.030	-	-	0.059

Table 11: Table of Snapshot Recommender With User-User Re-rank Results

7.2 Snapshot Recommender with user-user rerank

This recommender runs the exact same as the same as the previous Snapshot recommender. It recommends the N most popular songs from a given time period.

In this system however, as before, we used collaborative filtering in tandem with Location filtering to refine the results. The list is re-ranked in order of the similarity to the user using the system.

The result is a personalised list of songs based on a given time period.

Shown below in figure 20, is a flow diagram of the Snapshot recommender.

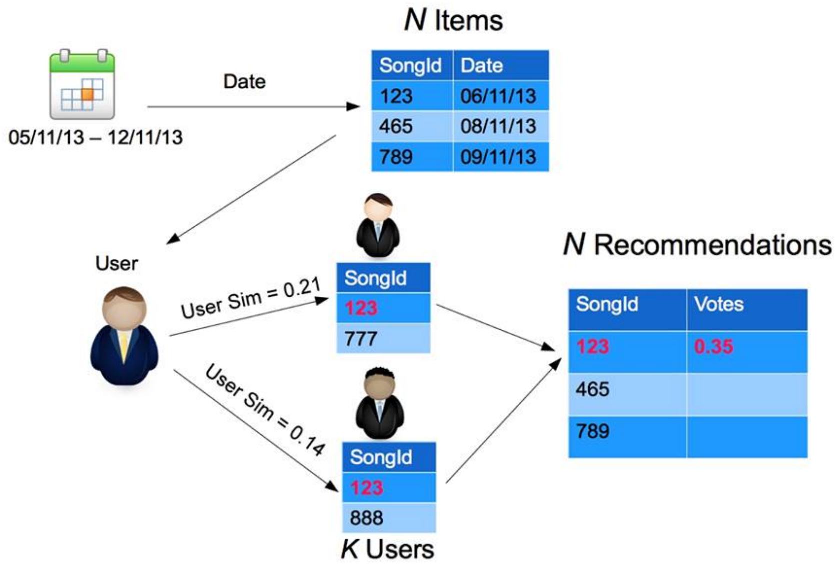


Figure 20: Snapshot Recommender with User-User re-rank Flow Diagram

7.2.1 Experimental Methodology

The experiment used to test this recommender was the same as the previous experiments.

Like before, what we are looking for here is to see if the User-User re-rank has pushed up some songs, and made the recommender more accurate.

Again, three values of N are used, for the number of recommendations. Three values of K are also used here, for the number of nearest neighbours we use for the User-User re-rank. As these are both nearest neighbours operations we use the same value for N and K. The three values we use are 5, 10, and 15.

Shown below is table 12 summarising the results from these experiments, with winning values in bold.

Similarity Measure	D Value	Metric	Dataset A			Dataset B		
			N=5	N=10	N=15	N=5	N=10	N=15
Date	D=5	Precision at 5	0.031	0.034	0.039	0.044	0.068	0.075
		Precision at 10	-	0.023	0.035	-	0.047	0.066
		Precision at 15	-	-	0.033	-	-	0.045
	D=10	Precision at 5	0.037	0.055	0.043	0.053	0.071	0.092
		Precision at 10	-	0.025	0.026	-	0.058	0.078
		Precision at 15	-	-	0.036	-	-	0.067
	D=15	Precision at 5	0.039	0.093	**0.087**	0.049	0.074	**0.102**
		Precision at 10	-	0.028	0.059	-	0.051	0.76
		Precision at 15	-	-	0.030	-	-	0.059

Table 12: Table of Snapshot Recommender With User User re-rank Results

7.3 Evaluation

We can observe that nearly all values were better as a result of the re-ranking. The biggest increase in results was for when D was equal to 15, which makes sense as this would have the biggest selection of songs, so a re-ranking would prove most effective in this category. We see all values had a significant increase in this category, for example: Precision at 5 for N equal to 15 in dataset B increased from 0.084 to 0.102.

Nearly all results from when D was equal to 10 were also better- all but one result was better. The only result that was worse, and only by a small margin, was precision at 10 for N equal to 15 (0.026 as opposed to 0.033 in the original).

For D equal to 5, the story is almost the exact same. All values are better or equal, except one. Precision at 5 for N equal to 15 is unchanged in the re-rank (0.039). Precision at 10 for N equal to 10 is the only value that is less accurate in the re-rank, being marginally lower (0.034 as opposed to 0.037). This is so minimal however that it could be discounted.

To find the most accurate setting of N to use, we can analyse the results. It's again clear that N = 15 is the most accurate choice. It performs better on both datasets in both recommenders.

On the whole so, it is clear to see that again, the re-rank has improved this recommenders accuracy, and is a better implementation of the Snapshot recommender and the version that Soundwave should implement, with N = 15.

8 Conclusion

8.1 Project Evaluation

At the start of the project we were given two use cases to fulfil. We managed to develop two implementations of each use case, and outline the better implementation in each use case to Soundwave.

Alongside this, we came up with two more use cases, and implemented two systems for each of those use cases, and outlined the better implementation.

Overall I feel like we have provided Soundwave with a host of systems and results to study, and have really succeeded in fulfilling Soundwaves goals.

8.2 Personal Conclusion

Overall I felt like this project went really well. All the goals made out at the beginning of the project were fulfilled, and we even managed to build some extra systems, that were really fun to do.

I feel like I learnt so much throughout the duration of this project. The most important one for me, that I learnt early on, was an issue with large-scale projects. It was that extensive research is key. Only when you know everything that has been developed previously, can you start to embark on leaving your mark in your particular field. Only when you have thoroughly planned out what you hope to achieve, can you start a project properly, knowing what lies ahead.

From studying Soundwave's data, I managed to ascertain that a time based recommender would be of great use to them. When I presented it to the CTO and co-founder, Aidan Sliney, he loved the idea and thought it was an idea definitely worth implementing at some stage.

I managed to encapsulate so many different aspects of recommending into a small number of systems. From that, I feel like I have a really strong knowledge of recommending systems, and feel ready to bring that knowledge into professional life.

9 Future Work

9.1 User Testing

The best metric to use is to see how people reacted to their recommendation [9]. Unfortunately, as we are dealing with dead data, we do not have this facility. For future testing, it would be preferable to have user testing to evaluate the systems, and get a more accurate portrayal of the accuracy of these recommendation systems.

9.2 More Cluster Evaluation Metrics

When we were evaluating our clusters, we used tempo standard deviation to evaluate how correlated a cluster was. If there was more time, we could have used other metrics to evaluate the clusters, like song energy variation, and song valence variation. These are both available from the Echonest library, and are things that could be used in future iterations.

9.3 Improved Location Based Recommender

One of the most interesting enhancements I'd like to do, would be to configure the Location Based Recommender to check if it is in a 'good' cluster first, before recommender songs from that area. If it's in a cluster with a low standard deviation, so we can say that the same type of music is playing there, recommend more of that music. Otherwise just recommend music as per normal.

References

[1] Ifpi digital music report. `http://www.ifpi.org/downloads/Digital-Music-Report-2014.pdf`.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[3] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Jesús Bernal. A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems*, 26:225–238, 2012.

[4] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 3–4. Morgan Kaufmann Publishers Inc., 1998.

[5] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems*, volume 60. Citeseer, 1999.

[6] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. *Collaborative filtering recommender systems*. Now Publishers Inc, 2011.

[7] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[8] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[9] Joseph Konstan. Introduction to recommender systems. [Online, accessed December 2013], `https://www.coursera.org/course/recsys`.

[10] Joseph A Konstan, John Riedl, A Borchers, and Jonathan L Herlocker. Recommender systems: A grouplens perspective. In *Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08)*, pages 60–64, 1998.

[11] Paul Lamere. What makes music so special. `http://musicmachinery.com/2011/10/23/what-is-so-special-about-music/`, 2011. [Online; accessed 25th March 2014].

[12] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[13] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM, 2007.

[14] David M Pennock, Eric Horvitz, Steve Lawrence, and C Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 473–480. Morgan Kaufmann Publishers Inc., 2000.

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

[16] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.