Multi-criteria Recommender System

Student:
Jeremiah Wangaruro - 16495506
(jeremiah.wangaruro@ucdconnect.ie)

Supervisor:
Michael O'Mahony (michael.omahony@ucd.ie)

UCD School of Computer Science
University College Dublin
November 15, 2019

Contents

List of Figures

List of Tables

List of Important Abbreviations Used Within

RS	*Recommender systems*
CB	*Content-based*
CF	*Collaborative Filtering*
UB	*User-based*
IB	*Item-based*
MF	*Matrix Factorization*
HR	*Hybrid Recommender*
MCRS	*Multi-criteria Recommender Systems*

Abstract

Recommender Systems (RS) are a subclass of information filtering systems which aim to predict the "rating" or "preference" a user would give to an item based on historic data. In the early days, the most prominent type was Content based (CB) filtering systems which tied similar items based on key text phrases. It proved to be very useful, however it had its limitations. Collaborative Filtering (CF) systems, on the other hand, calculated a prediction rating based on users' feedback on items. This method inherits "human" qualities and allows for *serendipity*. CF quickly became one of the most useful filtering tools.

In this report, I will explore the different implementations of CF and its evolution into Multi-criteria Recommender Systems (MCRS). I will also discuss CB systems and how CF can be used in combination with these techniques to create Hybrid Recommender (HR) systems which solve some of the limitations of the two approaches.

Link to GitHub: https://github.com/jeremiah-wa/FYP

Project Specification

The core objectives:

- Background reading
- Design and implementation of a baseline collaborative filtering recommender system
- Design and implementation of a multi-criteria recommender system
- Perform a comparison (offline evaluation) of recommendation performance

Additional advance goals:

- Implementation of a content-based recommender using review text; compare performance to baseline and multi-criteria recommenders.
- Incorporate sentiment mined from review text into the recommendation process; compare performance to baseline and multi-criteria recommenders.
- Enhancement(s) of the student's choice.

1 Introduction

In this report, I will discuss in detail the different implementations of recommender systems and examine the algorithmic frameworks used for each method. I will also take some time to highlight the flaws of each of the studied systems and illustrate some of the solutions used to mitigate these limitations. These constraints will help us understand the selection process and give rationale for the use of multi-criteria recommender systems.

The key to any successful business is to "know" your customers. This means knowing what your customers want without them asking or even before they know themselves. In the past, before the likes of Spotify and Apple music, a typical record store owner would only keep the most popular records in store. Due to the limited space, the owner could only afford to keep what they could sell and they knew that having a smaller collection of the most popular records would yield more sales than a large collection of diverse but obscure records [1]. However, there are many problems with this reactive model. For one, the owner is always lagging behind the current trends. The emphasis on a select few items also limits customers by a lack of choice. With the boom in *e*-commerce, the seller was no longer limited by store size. They could now hold all the songs at hand, but to maximize their sales they needed a way to get the supply to the demand. This led to the introduction of recommender systems (RS). RS are intelligent engines that collect information related to past user behaviour, with the aim of providing personalized suggestions on unobserved items that are likely to be of interest [2]. RS allows the seller to efficiently promote the right stock to the right user. This ultimately creates a better user experience because users do not have to sieve through large libraries to find their taste, which in turn maximizes the website sales and leads to more profit for the provider. In the beginning, RS had limited use, but thanks to advancements in data storage and processing, these techniques are now used industry wide. They are incorporated into just about every market from streaming sites such as Netflix, Spotify, etc., to social media e.g. friend suggests on Facebook and online stores like Amazon and even supermarkets like Tesco e.g. value card and personalized coupons. RS have become key players in the business model of these sites. 35% of Amazon purchases and 75% of Netflix views come from recommendations [3] and with consumers attention spans getting less and less, businesses need smarter and more proactive engines to drive these recommendations. RS can be grouped into three main categories:

Content-based (CB) filtering systems which analyse textual descriptions from items users have previously looked at and make recommendation to similar items. Most of the early RS used CB techniques to make recommendations on textual items such as blogs, magazines. CB systems are very useful when dealing with items containing textual information, however they have their limitations. CB systems cannot distinguish between a well written and a badly written piece of text i.e. they do not filter based on *quality*. Additionally, CB systems have no inherent method for generating serendipitous finds. This often leads to an *over-specialization* problem, which occurs when users are suggested items that have a high similarity to the observed items [2, 4]

Collaborative Filtering (CF) systems suggest items based on user preferences. These techniques are more recent but have proven to be very effective. There are many variations of this technique e.g. *user-based* (UB), *item-based* (IB), *matrix factorization* (MF). There are also numerous ways to find correlation between the users/items. CF systems have the innate ability to filter based on qualities that transcend CB systems, such as quality and taste. Furthermore, CF suggestions[1] are intrinsically *serendipitous*, i.e. their recommendations have more novelty or variation. This is because they do not derive from item-to-item correlation, but rather other users that share similar interest. Essentially, CF automates the word of mouth process, which has been proven to be an effective marketing tool [5]. It works on the basic premise that users typically adhere to recommendations made by like-minded people. However, CF systems have their limitations, namely sparsity and scalability which we will discuss later. [2, 6, 7]

Hybrid Recommender (HR) systems are a combination of two or more systems. They aim to mitigate some of the limitations of the individual systems. The are several different configurations. Each configuration concentrates on increasing an evaluation metric e.g. coverage, accuracy, etc.

Multi-criteria (MC) recommender systems extend CF methods to include ratings on the multiple criteria of each item (sub-ratings). The premise being that more information will help provide a more accurate recommendation. By providing more points of comparison the system can create a more reliable neighbourhood of similar users. MCRS inherits the same limitations from CF models, because of this there is a trade-off between increased accuracy and scalability.

In the following section, I will layout the various systems, discussing in detail their underlying methods and highlighting their advantages and limitations. This will give us the background understanding needed to conduct viable experiments.

[1] Recommendations and suggestions can be used interchangeably. Predictions are quantitative suggestions i.e. ratings (1-5 stars)

2 Related Work and Ideas

I have examined papers and experiments conducted in the domain. Below I have laid out my finds on the aforementioned systems.

2.1 Content Based (CB)

In cases where items can be represented by text e.g. news articles, blog post, tweets, etc. Content-based recommender systems can be used to analyse a collection of descriptive documents for items previously observed by users to build a profile of the user interests based on the features of the items observed by that user. The profile is a structured representation of user interests and is used to recommend interesting new items. A recommendation is made by matching up the features of the user profile against the features of an item. CB methods can be extended to use on instances where the items cannot be represented by text by using the product specs to construct the feature vectors e.g. electronic products, cars. This technique is known as case-based recommendation. CB systems use information retrieval methods to extract features from the text. A commonly used method is the *vector space model* (VSM) with *term-frequency-inverse document frequency* (TD-IDF) weighting [8].

Vector Space Model (VSM)
VSM creates a spatial representation of text documents, whereby each document is represented by a n-dimensional term (feature) vector. Each dimension corresponds to the weight of a term in the vocabulary of the overall document collection. The term is determined and weighted using TD-IDF weighting. This technique uses some basic assumptions to select and weigh the terms:

- Rare terms are not less relevant than frequent terms (IDF assumption).
- Multiple occurrences of a term in a document are not less relevant than single occurrences (TF assumption).
- Long documents are not preferred to short documents (normalization assumption).

In short, this means that terms that appear frequently in one document but rarely in the rest of the collection are more likely to be relevant to the topic of the document. Additionally, normalizing the weight ensures that longer documents do not have a better chance of being retrieved [8].

Comparisons can then be made using several similarity measures each describing the proximity of the two vectors. The most widely used is the *cosine* algorithm (discussed later). CB systems employing the VSM approach represent both their user profiles and items as weighted term vectors. Recommendations are made by computing the similarity between the user profile and items.

2.2 Collaborative Filtering (CF)

CF systems use preference data collected from users' past behaviour to generate predictions (ratings) based on their similarity to other users/items. Because the

predictions are based The user preference data can either be *explicit* or *implicit* [7]. *Explicit* preference data are ratings supplied by the user, as such the rating is a precise representation of the user. However, gathering these ratings can be difficult. Users are reluctant to rate items and often need convincing. CF algorithms also require a lot of ratings to function accurately and users may find it intrusive and tiring to supply all these ratings. *Implicit* preference data is obtained indirectly from a user. The rating is inferred by user actions. For example, a user probably likes a song if they play it regularly therefore a rating scale can be generated by analysing the number of times a song is played. Implicit ratings can be gathered from a wide range of user interaction, removing the associated cost of collecting explicit ratings. However, implicit ratings can contain a lot of noise which decreases the quality of the rating. Collaborative Filtering techniques can be categorized in to three main groups:

User-Based (UB)

These systems maintain user profiles represented by n-dimensional vectors where each entry is a rating the user give to a specific item (if no rating was given its value is null). A given user profile is compared to other user profiles using a similarity measure to generate a neighbourhood of similar user profiles (I will discuss neighbourhood generation later)[9, 10]. The similarity measure used can drastically change the neighbourhood generated and therefore the predicted rating. Some of the common algorithms used to compute the similarity include [11, 12]:

Mean Squared Difference (MSD): This algorithm measures the degree of dissimilarity between two profiles by computing the mean squared difference between the common ratings of the two profiles. The similarity measure can then be calculated by getting one minus the dissimilarity score. The score is usually normalized using the min and max rating values before this calculation to ensure the similarity metric ranges from 0, if there is no similarity or no common ratings between the profiles, to 1 if there is a strong similarity between the profile ratings.

Pearson r: This algorithm uses Pearson correlation coefficient to measure similarity between user profiles. The coefficient ranges from -1, strong negative correlation, via 0, no correlation, to 1 strong positive correlation between two vectors. An advantage of this algorithm is it can account for negative correlation, meaning it can be used to predict dislikes as well as likes.

Cosine: This algorithm calculates dissimilarity by getting the cosine of the angle between two vectors (profiles). The algorithm essentially computes the dot product of two normalized vectors. The normalization is crucial to ensure an equal scale regardless of vector sizes. The similarity score is then calculated by getting the one minus the dissimilarity. The metric ranges from 0 (no similarity) to 1 (high similarity) for ratings greater than or equal to 0.

When evaluating these algorithms, there is a trade-off between *accuracy* of the predictions and the *coverage* (number of total recommendations made over the number of possible ratings). In the case where normalization is not used i.e. Pearson the similarity measure is heavily influenced by the number of co-rated

items. For example with regard to Pearson r, if two profiles only have two common ratings the line of best fit is a straight line through two points so the coefficient will yield either -1, 0 or 1. This does not accurately reflect the level of similarity between these two user profiles. To combat this issue, *significance weighting* can be used. This technique modifies the similarity weight based on the number of common ratings. A threshold (N) is given and if the number of common ratings (n) is less than this threshold the weight is multiplied by n/N to give a more reliable similarity weight. This is a useful technique, however the threshold depends on the number of overlaps in the dataset so testing is required to get the optimal threshold value. Another technique that is commonly used is the *Jaccard index.* This is a useful technique when the number of common ratings between profiles is very low. It works by multiplying the similarity weight by the intersection (number of co-rated items) over the union (number of ratings from both profiles).

Item-Based (IB)

Item-based is very similar to user-based, except the similarity calculation is between items. IB algorithms look at the set of items a given user has rated and computes how similar they are to the target item and then selects the most similar items [13]. Similarity between two items can be calculated using the same algorithms as UB, however this time it uses M-dimensional item vectors, where M is the number of users and the components of the vector are the ratings given by each user on that item. Amazon used an optimized IB algorithm in the past [14]. They built an item-to-item matrix by iterating through all the item pairs and computing a similarity metric for each pair. Additionally, they only computed similarity between a single item and all related items. The computation of this similar-items table was extremely time intensive and as such was performed offline. However, the resulting table yielded high quality recommendations based on highly correlated items. The similarity values could then be retrieved very quickly online.

Neighbourhood Generation

In traditional CF models a profile (e.g. user) is compared to all the other user profiles to find the most similar user (neighbours) to the given user. Each neighbour profile in the neighbourhood is weighted by its degree of similarity to the given profile. There are several configurations that can be made to optimize the system e.g. changing the similarity measure or algorithm used to select the neighbours. For instance, the most similar profiles can be described using the top k profiles with the highest similarity weight (where $k > 1$), this is known as the *k-nearest-neighbour* algorithm. An alternative would be to capture neighbours with a similarity weight of at least t (where $0 \leq t \leq 1$) this is known as the *threshold neighbour* algorithm.

Prediction algorithm

A profile (e.g. user) rating on a target item can be predicted by merging the neighbourhood ratings on the target item. One approach is to use the mean average of the neighbourhood ratings; however, this does not account for neighbours with low similarity, who may have opposing ratings on a target item.

An improvement might be to use the similarity score to weigh the ratings of each neighbour to reduce the influence of low similarity neighbours. Although, both these approaches suffer significantly when all the neighbours have opposing ratings on a target item. One way to mitigate this problem is to centre the prediction around the active user's mean rating. This is known as the deviation from mean approach or Resnick's algorithm [7, 13].

Matrix Factorization (MF)

Matrix factorization represents both items and users by factors (features) vectors inferred from the rating patterns [15]. Prediction ratings are then calculated by getting the dot product of a user and item vectors. If you imagine the traditional UB or IB models creates a large user-item matrix of size M x N where M is the number of users and N is the number of items. Matrix factorization allows us to drastically reduce the number of entries by extracting key features, forming two separate smaller matrices, one of size M x F, the other of N x F, where F is the number of features. Each vector of the M x F matrix corresponds to a user and their responds to the features. Similarly, the N x F matrix reflects a given items possession of these feature. The matrices can then be used to calculate each cell of a large M x N matrix by multiplying corresponding vectors on the two matrices (dot product). The major challenge is computing the mapping of each item and user to feature vectors. Once mapped the system can easily compute the prediction rating. This model is closely related to *singular value decomposition* (SVD), a well-established technique for identifying *latent semantic* factors in *information retrieval* [15, 16].

2.3 Pros and Cons

Advantages of Content-based [7, 8, 16]:

User independence: Content-based systems solely use reviews provided by an active[2] (given) user to build their own profile. As such they do not suffer from the scarcity problems.

Transparency: Recommendations made by CB systems can be easily traced and explained because they are explicitly based on item features. These features are indicators to consult to decide whether to trust a recommendation.

New items: CB systems can recommend items not yet rated by any user because recommendations are solely based on item descriptions and user reviews.

Disadvantages of Content-based:

Limited content analysis: The number and types of features that CB systems use to describe their items are inherently limited. Domain knowledge and ontologies can help but sometimes the information needed to distinguish user-item preferences is not captured in the textual documents, which can diminish the accuracy of the

[2] person accessing the application

recommendations. For example, human qualities such as taste are beyond the scope of CB systems and so if a user wrote a positive review on a particular type of horror film, the recommender might not capture the user's taste in horror film and just recommend generic horror films that contain similar key words.

Over-specialization: CB systems have no inherent method to finding unexpected items. Recommendations are solely based on the items that closely match the user profile. Therefore, they are very similar to items the user has already rated. Recommendations lack *serendipity,* i.e. the items recommended have a limited level of novelty.

New user: CB methods recommendations are solely based on the given user's past preference, hence when a new user enters the system, there is a severe lack of information to build a user profile in order to make a prediction. This problem is known as the *new user cold start* and is a subset of the *cold start problem.*

Advantages of Collaborative filtering [14, 15, 17, 18]:

Item independence: CF systems make recommendations based solely on user preferences, they do not rely on any item features that need to be pre-determined and extracted prior to filtering.

Complex domains: CF systems can be readily implemented in complex domains which contain items that are difficult to analyse by automated processes. CB systems has its limitations when it comes to distinguishing items especially where the class boundaries are ambiguous. CF leaves the "understanding" to the users, allowing the model to make recommendations over complex items that are not machine analysable.

Serendipity: CF systems make recommendations based on a pool of similar users/items (neighbours) and such recommendations are likely to vary from a given user's previously rated items.

Disadvantages of Collaborative filtering:

Sparsity: This is when the system lacks the information needed to make a good recommendation. It occurs when the many items are not rated by users. CF systems are based on similarity measures computed over common ratings therefore it is not possible to calculate similarity if there are no common ratings between two profiles. Additionally, similarity is likely to be unreliable if calculated over a small number of common ratings. One of the common causes of sparsity is the addition of new users or items to the system, this is also known as the *cold start* problem. When a new user/item enters there is no ratings to compare for similarity as such it is not possible to generate a recommendation.

Scalability: This refers to the algorithms capacity to withstand scaling (change in size of input). It is not enough that an algorithm works for a certain dataset, it must be universal and handle an arbitrary number of inputs especially with the current

rate of data growth. The neighbourhood generation required in traditional CF methods (user/item-based) involves computations that grow linearly with the number of users and items [18]. As such CF systems inherently suffer a lot from scalability due to their *time complexity*[3]. They have a complexity of O(MN)[4] in the worst case, where M is the number of users and N is the number of items, since it analyses M users for a max of N items. This complexity quickly becomes computationally expensive when scaled. Items tend to be less volatile than users, who often enter and leave the system. The rate of growth in the number of items also tends to be less than the users, hence item-item systems are sometimes more stable and suffer less from scaling than user-based systems. This limitation can also be mitigated using the dimension reduction techniques used in matrix factorisation. Once the large user-item matrix used in traditional CF models is decomposed into two dense feature matrices, the computation needed to evaluate a prediction rating is much faster. However, the construction of these feature matrices requires heavy computation to yield accurate prediction ratings.

2.4 Multi-criteria (MC)

In most RS, the system makes a recommendation based on a single-criterion value e.g. a rating for an item given by a user. Multi-criteria (MC) recommender systems attempt to improve the quality of the recommendations by using sub-ratings to represent more complex user preferences. Multi-criteria techniques extend the CF model from a 2D user-item matrix to a 3D user-item-criteria matrix where a given user has an overall user rating and additional ratings for each individual criterion. The MC approaches can be categorized into two groups:

Heuristic Approaches

These approaches extend the similarity calculation used in traditional CF models to reflect multi-criteria ratings.

Aggregated similarity: one approach is to aggregate the similarities based on individual criterion ratings. The similarity is computed between two profiles on each individual criterion of commonly rated profiles using the traditional similarity measures, mentioned above, then a final similarity value is obtained by aggregating the individual similarity values. The values can be aggregated by either calculating an average, computing the minimum (worst-case) or the weighted sum of the individual similarities where the weight corresponds to the importance or usefulness of the criterion [19].

Distance formula: another approach would be to calculate similarity using multi-dimensional distance metrics such as *Manhattan, Euclidean or Chebyshev* (maximal value) distances [19]. The distance between two users can be calculated by getting the average distance for all common items that both users rated. The distance is then inversed[5] to give the final similarity value.

[3] Degree of computational difficulty
[4] Due to *sparsity* within the ratings, it's closer to O(M+N).
[5] Inverse of 1+distance to avoid division by zero

Weighted similarity: other approaches consider preference similarity [20]. For example, one can weigh criteria according to relative importance (based how many times a profile rates a criterion) to create neighbourhoods of profiles that share similar ratings on important criteria.

All these approaches utilize the neighbourhood generation and prediction algorithms used the single-rating neighbourhood-based CF systems.

Model-based approaches

These approaches construct a predictive model to estimate unknown ratings by learning from the observed data. For example, *Multilinear Singular Value Decomposition* (MSVD) [21], based on *singular value decomposition* (SVD) which is a technique used in MF models. The decomposition used for the 2D user-item matrix can be extended to multi-dimensional data like the 3D user-item-criteria matrix. [21] used MSVD to reduce the dimensionality of a MCRS, where users rated a restaurant under 10 criteria. The result displayed an accuracy improvement of around 5% compared to the traditional single rating CF model.

2.5 Hybrid Recommender Systems

Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual system. For example, CF methods can be combined with other CB techniques to mitigate the cold start problem. There are several ways to implement a hybrid system, these include [22]:

Weighted Hybrid

Weighted hybrid RS are systems whereby the score of a recommended item is computed from the results of all the recommendation techniques in the system. The simplest implementation of this would be a linear combination of recommendation scores. For example, the *P-Tango* system [22] which initially gives CF and CB recommenders equal weight but gradually adjust the weighing as prediction about user ratings are confirmed or disconfirmed.

The benefit of a weighted hybrid approach is that the system uses all its capabilities to produce a recommendation. It is also easy to perform a post-hoc credit assignment and adjust the RS accordingly. The main drawback is the system assumes that the relative value of the different techniques is roughly uniform across the possible items, however we know this is not true because of the various limitations of the individual techniques.

Switching Hybrid

Switching hybrid RS use some criterion to switch between recommendation techniques. For example, the *DailyLearner* system [22] uses a combination of CB and CF methods whereby the CB technique is applied first. If the system does not yield an accurate recommendation, then a CF recommendation is attempted.

Systems like this employ a "fallback" approach where one model is used first and the other only comes into play when the first fails. Other switching hybrids use other parameters to dictate the switch [23]. Switching hybrids utilize their constituent recommenders more efficiently because they can account for their strengths and limitations, however parameterization also adds to the overall complexity of the system.

Mixed Hybrid
These systems make use of several methods simultaneously. An example would be the *Personalized Television* (PTV) system [24] which uses CB techniques based on textual descriptions of TV shows and the collaborative preferences of other users combined to give a program recommendation. Mixed hybrids like this can avoid the new items cold starts relying the CB component and can use implicit feedback like viewing time to build up a user profile. It can also use CF methods to yield serendipitous recommendations. When conflicts between the different recommenders occur, the system employs preference to decide which recommendation to display or presents multiple recommendations side-by-side.

In 2006, Netflix announced a contest to improve the performance of its recommender system [25]. They provided a training set of over 100 million ratings from over 480 thousand randomly-chosen, anonymous subscribers on nearly 18 thousand movie titles, each movie was rated on scale of 1 to 5 stars and a qualifying set of over 3 million of the most recent ratings from the same subscribers over the same set of movies was withheld. Contestants were required to make predictions for all 3 million test cases. The RMSE was computed immediately and automatically for a fixed but unknown half of the qualifying set (quiz set). The result was reported to the contestants and posted on a leader board. The other half of the qualifying set (test set) was used to determine the potential winners of a prize. In 2009, the Grand Prize of 1 million dollars was award to team BelKor's Pragmatic Chaos for achieving a winning RMSE of 0.8567 which represented a 10.06% improvement over Netflix's recommendation algorithm Cinematch [26] . The winning strategy utilized 100 different predictor sets [15].

3 Data Consideration

The evaluation of the different recommender systems will be conducted on a real-world TripAdvisor dataset. The data was collected from TripAdvisor during the period of June 2013 to August 2013 [27]. I was provided with static datasets stored in SQL files for the project. The data folder consists of 3 different files, one with the hotel information, one with the review information and the other with the user information.

The reviews file consists of a table of all the user reviews. Some of the relevant attributes include:

- *review_id/member_id/hotel_id* - the unique identifiers for the review and the corresponding member and hotel identifiers
- *city* - the hotel location

- *rating* - the overall user rating*
- *stay_time* - the time of stay
- *recommend_list* - the ratings* for the different aspect of the hotel (i.e. multi-criteria ratings) e.g. value, location, sleep quality, service etc.
- *review_text* - the review text

*The ratings use a 5-star rating system where above 3 is considered a good rating.

The files contain 150,961 members (users) and 2,370 hotels (items) giving a total of 227,125 reviews. However, in line with other studies [27, 28], I will be considering a dataset containing a subset of 3,008 members, 1,389 distinct hotels, amounting to a total of 18,182 reviews. Each member will have at least 5 reviews on 5 distinct hotels and each hotel will have been rated by at least 5 members to ensure there is enough correlation between the users/hotels. I have defined a review as feedback consisting of a written review, an overall rating and at least one sub rating on a criterion.

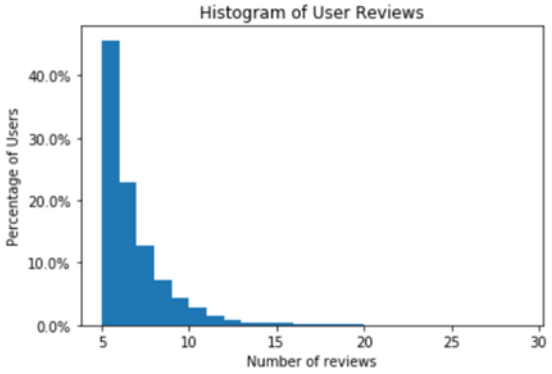

Figure 1: Histogram of Reviews per User

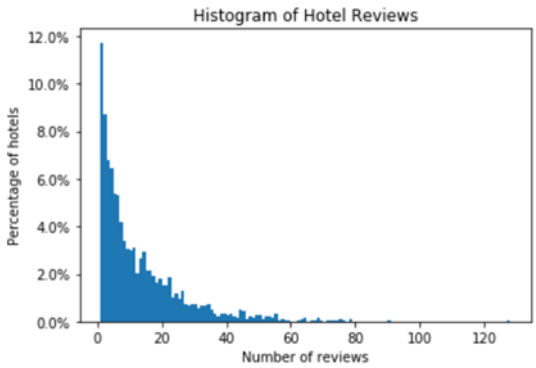

Figure 2: Histogram of Reviews per Hotel

As shown in *Figure 1* and *Figure 2*, the distribution for both hotel and user are skewed to the right. The majority of hotels have a low number of reviews (11.7% of the hotels having 1 review) and most users (45.7%) only provided the minimum threshold of 5 reviews. The number of reviews each hotel received ranged from 1 to 40, with exception of a few outliers. The majority of hotels received between 1 and 20 reviews with a mean of 13 reviews. Each member had between 5 to 25 reviews with a mean of 6 reviews.

This real-world supplied dataset has distinct characteristics that will prove to be difficult for the different RS to overcome e.g. sparsity. The same dataset needs to be use on all the different RS when evaluating for a controlled test of their performance. This means the dataset must contain enough information to cater for all the different RS including textual reviews to enable content-based methods. There also must be enough correlated ratings between users/hotels to facilitate effective collaborative filtering techniques, hence the requirement for at least 5 reviews per user/hotel. Each user/hotel must also have an addition layer of ratings on the different aspects of the hotel to implement a multi-criteria RS.

4 An Outline of My Approach

I will be using SQL to query the three files and extract an appropriate dataset. I propose only members with at least 3 reviews should be selected. This would create a large dataset with an adequate amount of correlation between users. From the preliminary data analysis, we can see that increasing the minimum review threshold would drastically reduce the size of the dataset.

The implementation will be done through Python. I will be importing libraries such as Pandas, SciPy and Scikit-learn to help with processing and matplotlib for illustrating the data analysis and evaluation. I will use the knowledge gain from the related works to implement the best strategy for the RS detailed in the project specification.

For the baseline collaborative filtering (CF) RS, I will be implementing a traditional User-based CF model. The model will create a user-item matrix using the filtered dataset, where each row (vector) will represent a user and each column will represent a hotel. Each entry on a row will correspond to the rating the user gave to an item. The value of the rating will come from the user's overall 5-star rating for the given hotel. The absence of a rating score indicates the user has not yet rated the item. I will experiment with different similarity measures and neighbourhood sizes to find an optimum model. Once the similarities are calculated the weighted average of the neighbours is used in addition to the mean rating of the active user to predict the rating the user would give to a given item. The weight in this case is the similarity between the active user and the neighbour.

The multi-criteria RS will essentially extend the baseline CF model by adding an additional layer of ratings, creating a 3D user-item-criteria matrix. A proposed model might utilize the similarity-per-priority technique [19]. The premise being that by prioritizing neighbours that share similar criteria preferences I can increase the quality of the neighbourhood. The similarity will be calculated by getting the average of the individual criterion similarities where the sub-ratings carry an equal weight (equal importance). An overall rating can then be predicted using the weighted average of neighbours' overall ratings.

Time permitting, I will also implement a content-based (CB) RS. The model will employ the vector space model (VSM) and the term frequency-inverse document frequency (TF-IDF) algorithm (described in 2.1). The proposed model will identify descriptive terms by analysing the review text using the TF-IDF method. Items can then be characterized by term vectors, where each value describes the relevance this term has to the item. Similarity can be computed by calculating the distance between the vectors using the cosine algorithm. The system can identify which hotels the member rated high and recommend a hotel similar to it. Additionally, I can combine the CF and MCRS models with this CB model to create a weighted hybrid RS. Similar to the *P-Tango* system [22], this approach will initially give the two recommenders equal weight but gradually adjust the weighing as prediction about user ratings are confirmed or disconfirmed. I can then perform a post-hoc credit assignment and record my findings.

Evaluation will be conducted using a repeated random sub-sampling (Monte Carlo cross-validation) technique [28] where the dataset is randomly split into a training set containing 80% of the data, a validation set containing 10% and a test set containing the remaining 10%. The training set is used to "train" or model the RS, to predict a set of unseen ratings from the test set. Each RS model will be trained with various similarity measures and neighbourhood sizes. An optimum model will be found by evaluating the models on the validation set. A final evaluation of each RS will then be performed using the test set. The model is evaluated by comparing the predicted ratings of the model and the actual test set ratings. To ensure the trained models generalize for data it was not trained on, the split will be performed 5 times and an average of the evaluation results across the splits will be recorded.

I will be evaluating the RS using two metrics, Root Mean Squared Error (RMSE) and Coverage [7]. RMSE is a standard method of measuring the error of a model. It works by calculating the mean squared difference in the predicted ratings and the actual ratings. The models will be trained to minimize the RMSE and hence increase prediction accuracy (proportion of correct predictions). Coverage refers to the proportion of the test cases the system can make a prediction for. I will be randomly selecting reviews by member id, i.e. 80% of reviews per member to train with. By ensuring the same users in the training set are in the test set, I remove the possibility the model has not been trained on that user. However, other factors such as sparsity in ratings can also negatively influence coverage [6].

5 Project Work Plan

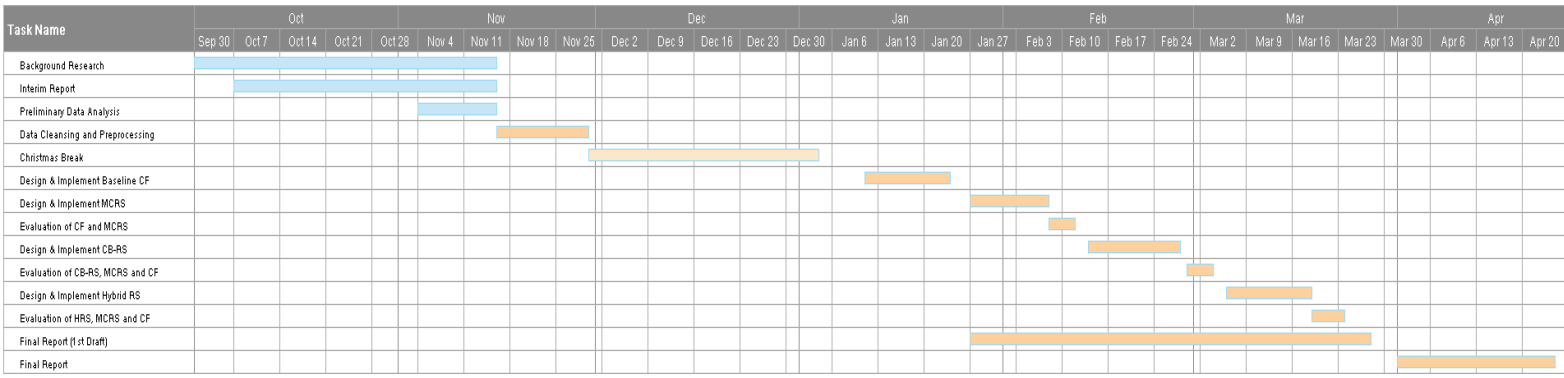

Figure 1: Gantt Chart

The Gantt chart above lays out my work plan for the project. It ranges from week 4 of the academic year (starting 30th Sept) to the final week 33 (ending 26th April). I allowed for a tolerance of 2 weeks in my plan. The main steps are:

- Background Research: Extensive research into the various recommender models and algorithms in order to propose strategies that meet the project specifications.
- Preliminary data analysis: Initial examination of the data to formulate strategies for the various RS.
- Interim Report: Centralizing the accumulated information into a concise report. Using the knowledge from the research and initial data analysis to propose concepts for data pre-processing, implementation and evaluation.

- Data Cleansing and Pre-processing: Further data analysis to rationalize the extraction of an appropriate dataset for training and testing the models. Once a dataset is selected, different aspect of the dataset is required for each of the RS, hence some pre-processing will be required.
- Design & Implementation: Extending the proposed strategies to practical execution. Designing the model to be in line with the proposed models (and project specs.) and implementing the design thereafter.
- Evaluation: Assessing the models under certain evaluation metrics (discussed above)
- Final report: Documenting the data cleansing and pre-processing. Recording the design and implementation for the various models. Reporting my findings from the evaluations.

The background work (blue) occurs in the first semester alongside the interim report. Most of the implementation occurs after the Christmas break. However, the data pre-processing will start before Christmas. It is important the dataset is the same throughout because it will form the foundation for training and testing the models. I have allowed for an additional few day for each process as an added precaution. I also allocated time in between each implementation to write up a report. At least one full draft will be completed by the 30th of March and allowing for a delay for feedback, the final report should be complete by 24th of April.

6 Summary and Conclusions

With the rise of e-commerce, recommender systems (RS) quickly became a crucial tool for extracting additional value for business, by helping user find items they want to consume. The first RS used content-based approaches to make recommendations based on textual data. They prove to be very useful however the recommendations they provided lacked serendipity and could not account for human qualities such as taste. These limitations necessitated the need for RS which utilize collaborative filtering techniques such as user based (UB) and item based (IB) approaches. These RS provide recommendations based on user preferences (ratings) enabling them to inherent the human qualities associated with the users. By providing recommendations based on similar users or items (neighbours), the recommendations gained a significant degree of novelty.

With the huge volume of data companies now have at their disposal new technologies are needed to effectively process the information into accurate recommendations. For example, multi-criteria (MC) RS which can operate over multiple dimensions, allowing them to use ratings on various aspects of an item for improved prediction accuracy [21]. By conducting experiments across the aforementioned RS, I will illustrate their utility and rationalize the use of MC RS.

However, I must mention the distinct characteristics of the dataset creates limitations for the experiment. Any system trained on this dataset has the drawback that it may not generalize to work effectively on other datasets. Ideally, I would conduct the experiment on various datasets, however freely available datasets containing both multi-criteria ratings and textual reviews to facilitate the required RS are extremely limited.

To date, I have done extensive research to aide in designing the models that meet the project specifications. I have also performed preliminary data analysis to plan how best to filter the data into an appropriate dataset to use for training and testing the various RS. The next step is to extract the dataset and perform the necessary pre-processes to prepare the data for implementing the numerous RS.

7 Data

Historical review data collected by TripAdvisor was used for an empirical analysis of the various recommender systems. A dataset was created by filtering static relational tables using a SQL query. After filtering, some further pre-processing was also required. This included the removal of duplicated reviews not seen during the preliminary analysis and the parsing of textual reviews. To correct the offset created after the removal of the duplicates the minimum number of reviews per user/hotel was set to 3. This was still enough reviews for overlap to occur. The textual reviews were written in html script. Parsing involved the removal of the html syntax e.g. tags. The final dataset consisted of 18,182 reviews given by 3,008 users to 1,389 hotels. Each user had one review per hotel (i.e. no duplicates) and had reviewed at least 3 distinct hotels. In addition, each hotel had been reviewed by at least 3 distinct users. A review consisted of an overall 5-star rating and at least one sub-rating with a 5-star rating and a textual review. The Tables 1 and 2 below show the distribution of reviews per user and hotel.

Table 1: Distribution of reviews per user

Reviews per User	Values
mean	6.04
std	1.88
min	3
25%	5
50%	5
75%	7
max	24

Table 2: Distribution of reviews per hotel

Reviews per Hotel	Values
mean	13.09
std	11.11
min	3
25%	5
50%	10
75%	17
max	103

In addition to the overall rating, hotels are also rated under 8 different criteria: Business service (e.g., internet access), Check in / front desk, Cleanliness, Location, Rooms, Service, Sleep Quality and Value. On average 5.8 criteria were rated per review. If we can consider the dataset as a 3D user-item-rating matrix, a slice of the overall ratings represents the user-item matrix used for traditional CF methods. Each entry in this matrix corresponds to the overall rating a user gave to a hotel (NANs where no rating was given). This matrix had a density of 0.44%, which is very low compared to other datasets used in similar experiments e.g. Movielens [29, 30], with most having ≈ 4%. The high sparsity meant there is very little chance of overlap between the user/item ratings. To further understand the overlap between overall user ratings, I created a binary user-user matrix. An entry of 1 indicated that there were co-rated items between a pair of users and 0

indicated there were no common ratings. By computing the density[6] of this matrix, I got the percentage of users that had co-rated items, 4.09%. Similarly, the item-item matrix gave a density of 4.5%. The average number of common ratings between users is very low at 1.06. It is similarly low for items at 1.18. This level of sparsity will have a negative effect on the quality and size of the neighbourhoods produced. Even though by using the criteria ratings, the multi-criteria approach will not improve the number of comparable pairs, it will hopefully increase the comparative strength of the algorithm which will lead to better neighbourhood generation and ultimately better recommendations.

8 Methodology

In line with other similar studies [31, 32], the dataset was randomly split into three disjoint sets. The split was performed over users and hotels, ensuring that every user and hotel in the validation and test sets appeared in the train set. A training set contained approx. 80% of the dataset and a validation and test set each contained approx. 10% of the dataset. Each model is trained using the training set and the final evaluation was conducted on the test set. The validation set was used to test the parameters of each system for the purpose of optimization. Once the optimal parameters for each system were established, the values were fixed for the final evaluation. For a more thorough evaluation, a 5-fold cross validation was conducted by randomly selecting different training, validation and test sets each time and taking an average of the evaluation metrics.

8.1 Baseline Collaborative Filtering approach

To compare the performance of a multi-criteria-based approach, a baseline model that employed traditional User-based (UB) collaborative filtering (CF) techniques [7] was created. The model only considered a given user's overall ratings on each hotel. The training set was entered into the CF model and the validation set was used to find the sequence of similarity and neighbourhood generation algorithms that deliver the highest quality predictions in terms of RMSE and Coverage. The following algorithms were considered:

Similarity Metrics

The similarity metric defines the relationship between two user profiles and is fundamental to the overall success of the algorithm. When building the recommenders, three similarity algorithms which have proven to be effective in similar experiments were considered [13, 31, 33].

Mean Square Difference measures the degree of dissimilarity between two profiles by computing the mean squared error between the common ratings of the two user profiles. Formally [33],

$$\mathrm{MSD}_{a,i} = \frac{\sum_{j \in I_a \cap I_i} (r_{a,j} - r_{i,j})^2}{|\{j : j \in I_a \cap I_i\}|}$$

[6] Excluding where row equals column i.e. pairs of same users/items

where

- a and i represents user profiles
- I_a represents the set of items rated by user a
- I_i represents the set of items rated by user i
- $r_{a,j}$ represents the rating user a gave to item j
- $r_{i\,j}$ represents the rating user i gave to item j

A similarity measure can then be calculated by getting one minus the dissimilarity score over the squared difference of the training set overall min and max ratings. Formally,

$$w_{a,i} = 1 - \frac{\text{MSD}_{a,i}}{(r_{max} - r_{min})^2}$$

where

- $w_{a,i}$ represents the similarity between user a and user i
- r_{max} represents the maximum rating score
- r_{min} represents the minimum rating score
- $MSD_{a,i}$ represents the MSD between user a and user i

This algorithm yields a similarity score that ranges from 0, if there is no similarity between the common ratings, to 1 if there is a perfect similarity between the ratings.

Pearson r measures similarity between user profiles by calculating the correlation coefficient of the common ratings. The correlation coefficient is given by the covariance of the common ratings over the product of the standard deviation of the common ratings. Formally [16, 33],

$$w_{a,i} = \frac{\sum_{j \in I_a \cap I_i} (r_{a,j} - \bar{r}_a)(r_{i,j} - \bar{r}_i)}{\sqrt{\sum_{j \in I_a \cap I_i} (r_{a,j} - \bar{r}_a)^2 \sum_{j \in I_a \cap I_i} (r_{i,j} - \bar{r}_i)^2}}$$

where

- $\bar{r}_a$ denotes the mean rating of user a
- $\bar{r}_i$ denotes the mean rating of user i

The coefficient ranges from -1, strong negative correlation, via 0, no correlation, to 1 strong positive correlation between the two rating vectors. In the interest of fair comparison, I will only consider a score ≥ 0. The Pearson algorithm can be enhanced using significance weighting [16]. Significance weighting allows Pearson to account for the number of common ratings producing a more reliable similarity score. Significance weighting reduces the similarity score when the number of co-rated item is less than some threshold value. The optimal value

$$w'_{a,i} = \begin{cases} w_{a,i} \times \frac{n}{N} & \text{if } n < N \\ w_{a,i} & \text{otherwise} \end{cases}$$

depends on the density of the dataset. This reduction is a ratio of the number of co-rated profiles and the threshold weight. Formally,
where

- $w'_{a,i}$ represents the new similarity between user *a* and user *i*
- *n* represents the number of co-rated items
- *N* represents the threshold value

Cosine calculates similarity by computing the cosine of the angle between two user rating vectors (profiles). The cosine is given by computing the dot product of the normalised vectors. Formally[13, 16] ,

$$w_{a,i} = \frac{\sum_{j \in I_a \cap I_i} r_{a,j} \; r_{i,j}}{\sqrt{\sum_{k \in I_a} r^2_{a,k}} \; \sqrt{\sum_{k \in I_i} r^2_{i,k}}}$$

where

- *k* represents an item in the set of rated items

By normalising the metric accounts for the number of co-rated items. The score ranges from 0 (no similarity) to 1 (high similarity).

Neighbourhood generation

Selecting the most similar users (neighbours) to a given user profile is crucial to making a good prediction. The following are two of the most common algorithms for defining similar neighbours:

K-nearest-neighbours [7, 16] considers the top-k profiles with the highest similarity score to a given user profile. This strategy does not consider the similarity score of the neighbours. If the k is too large the neighbourhood can include user profiles with low similarity to the given user profile. On the other hand, if the k is too small there might not be enough neighbours to generate a good prediction.

Threshold neighbours [7] considers only profiles with a similarity greater than some threshold, irrespective of neighbourhood size. Similarly, it is important to select a suitable threshold score. I tested the algorithms with different k and threshold values using the validation set.

Prediction algorithm

Once a neighbourhood is formed, an algorithm is needed to compute a predicted rating. I decided on the deviation from mean or Resnick's algorithm. It has proven to be very effective in other experiments [31]. It combines both a given user's mean rating and the neighbours ratings normalised by their similarity to create a prediction rating centred around a given users mean rating. Formally [16, 31],

$$p_{a,j} = \bar{r}_a + \frac{\sum_{i=1}^{n} w_{a,i} \ (r_{i,j} - \bar{r}_i)}{\sum_{i=1}^{n} | w_{a,i} |}$$

where

- $p_{a,j}$ represents user a's predicted rating for item j

For the purpose of the experiment, when there were no neighbours that rated a target item the algorithm returned -1, signifying no prediction was made.

Recommendation algorithm

Recommendations can be generated by looking at the items rated by the neighbours of a given user and computing a prediction rating for the items not rated by the user. If the item has a prediction rating >= 4, the item is considered relevant and is added to a recommendation list. The list is sorted according to the prediction rating and the top-N items are recommended.

Item-based (IB) CF

In addition to the UB-CF approach, a baseline IB-CF approach was also considered. This approach used the same algorithm used in the UB-CF approach with some key differences:

- Similarity was computed between item profiles instead of user profiles.
- The neighbourhood generation formed neighbourhoods of items
- The prediction algorithm predicted a rating a given item would receive from a user using the mean rating of the given item and the ratings of similar items

8.2 Multi-criteria (MC) approach

The MC approach extends the user-item matrix used in the baseline CF model to a 3D user-item-criteria matrix. A given user rated items under 8 different criteria. The premise being that with more granular information the system can yield higher quality predictions. The coverage values are expected to stay similar due to the number of co-rated items between users remaining the same. The user-based multi-criteria (UB-MC) recommender computes the similarity of two given user profiles, by calculating the similarities of co-rated items based on the common criteria ratings. This is done in a series of steps:

1. Identify the co-rated items between two users
2. Identify the co-rated criteria on a given co-rated item
3. Compute the similarity of the co-rated criteria ratings
4. Repeat step 2 and 3 for each co-rated item
5. Take the mean of the similarity scores as the overall similarity between the two user profiles

For this comparative study, only the criteria ratings were used, the overall rating was not considered when computing similarity. Similarity was computed using the same metric used in the CF approaches. The neighbourhood generation and prediction functions were identical to the baseline UB-CF approach.

Item-based multi-criteria

An IB-MC system was also implemented. In this approach similarity was computed between item profiles. To compute similarity, the same steps used in UB-MC were used with some key differences:

- The common users who rated two item profiles were identified
- The co-rated criteria ratings of the common users were used to compute the similarity scores.
- The mean of the similarity scores was used as the overall similarity between two item profiles

The similarity metrics, neighbourhood generation and prediction function were all identical to the baseline IB-CF approach.

8.3 Content-based (CB) approach

In addition to the baseline CF and MC approaches, I also compared the performance of a content-based algorithm. This approach employs a vector space model based on TF-IDF [8]. For the CB system to recommend items to a user, the system analyses the written reviews grouped by item and constructs item profiles based on the keywords (terms) retrieved using TF-IDF vectorisation [34]. Each profile is represented as a term vector with corresponding relevance scores for each term. Each vector consists of a total of 1,670,704 terms. For a given item profile, the relevance score for each term is defined using TF-IDF weighting [34].

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t)$$

$$\text{idf}(t) = \log \frac{1+n}{1+\text{df}(t)} + 1$$

where

- t denotes a term
- d denotes a document (i.e. the collection of written reviews for a given item)
- n denotes the number of documents in the document set (i.e. the number

of items)
- *tf(t,d)* denotes the number of times the term appeared in the document
- *df(t)* denotes the number of documents in the document set that contain the term t

Once the score for the term vectors are computed the vectors are normalized to have a Euclidean norm using the following formula [34]:

$$v_{norm} = \frac{v}{||v||_2} = \frac{v}{\sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_n}^2}}$$

where

- *v* denotes the term vector for a given item

Similarity between item term vector is computed using the cosine similarity metric (note the vectors were normalized so cosine is given by the dot product of the vectors).

Recommendations, for a given user, are generated by looking at the most similar items (neighbours) to the items observed by the given user then selecting the top-N most similar items according to their similarity score.

Item-based Content-based

An IB-CB approach was also implemented. This approach employed the same methods seen in the UB-CB with some key differences:

- Recommendations were made for a given item to target users
- Vectorisation was based on the written reviews grouped by user, creating corresponding user term vectors.

9 Experiments

9.1 Evaluation Metrics

Recommender systems research has used several types of measures for evaluating the quality of recommendations. I will be using the following metrics [13, 31]:

Statistical accuracy metrics evaluate the accuracy of a system by comparing the numerical recommendation scores (predictions) against the actual ratings for user-item pairs in the test set. *Root Mean Squared Error* (RMSE) is a widely used metric. RMSE measures the difference between the prediction rating and the actual rating. RMSE is computed by finding the squared difference between of the

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

predicted and corresponding observed rating and then averaging over the sample before finally taking the square root of the average. Formally,
where

- n represents the number of predictions made
- $\hat{y}_i$ represents the predicted rating
- y_i represents the actual rating

RMSE has a lower bound of 0 and an upper bound determined by the rating scale. RMSE is negatively oriented i.e. the lower the score the more accurately the system predicts ratings. It also gives relatively high weight to large errors. Other metric such as Mean Absolute Error (MAE) and Mean Squared Error (MSE) can also be used. These metrics tend to correlate with RMSE. RMSE was used to compare the quality of predictions made by the CF and MC recommender systems.

Precision and Recall is used to evaluate how effective a system is at recommending relevant profiles[7] [13]. Relevant profiles were defined as profiles with a rating >= 4 [31, 35]. Precision (Pr) is a measure of exactness or fidelity, it represents the probability that a recommendation is relevant. Recall (Re) is a measure of completeness, it represents the probability that a relevant recommendation is recommended. Formally, precision and recall are defined as:

$$Pr = \frac{|T \cap R|}{|R|} \qquad Re = \frac{|T \cap R|}{|T|}$$

where:

- T represents the relevant (i.e. rating >=4) profiles in the validation/test set
- R represents the recommendations made

Precision and Recall metrics allow us to make a comparison between numerical and non-numerical recommendations i.e. content-based and collaborative recommendations. Precision tells us the proportion of correct recommendations; however, it also is important to consider when these correct recommendations occur (rank). A good recommender should have high precision on the top-N recommendations. To account for this, I considered Precision and Recall at cut-off N. I looked at different subsets of the recommendations, increasing the size (k) each time. To get an overall score, I computed the average of the precision across the different k values. Formally, as defined in [36]:

$$Average\ Precision\ \ for\ N\ recommendations\ = AP@N = \frac{1}{|T|}\sum_{k=1}^{N} P(k) \cdot rel(k)$$

where

- P(k) represent the precision at a certain k
- rel(k) indicates if the k^{th} item was relevant (rel(k) = 1) or not (rel(k) = 0)

To gain an understanding of the recommendation precision across all the profiles, I calculated the mean average precision (MAP). Formally [36],

[7] Depending on the approach used (i.e. UB or IB) the profiles may be either user or item profiles

$$MAP@N = \frac{1}{|P|}\sum_{p=1}^{|} P|(AP@N)_p$$

where

- P represents all the profiles
- p represents an individual profile.

MAP was computed for different sizes of recommendations and the change in MAP was analysed as the number of recommendations was increased.

Coverage measures the percentage of profiles for which a system can provide a recommendation [31]. When comparing CF and MC approaches, coverage was defined as the percentage of profiles in the test set for which the system was able to make a prediction.

$$Coverage = \frac{number\ of\ profiles\ for\ which\ predictions\ can\ be\ made}{total\ number\ of\ users}$$

When comparing all three approaches, (CF, MC and CB), coverage was defined as the percentage of profiles in the test set with corresponding relevant profiles (e.g. users in the test set that had rated items with a score >=4) which the system was able to make a recommendation. Formally,

$$Coverage = \frac{number\ of\ profiles\ for\ which\ recommendation\ can\ be\ made}{total\ number\ of\ profiles\ with\ relevant\ profiles}$$

Additionally, it is important to understand the type of recommendations being made. If the recommendations are too similar to each other they will not be of benefit to the user. Recommendation should vary such that the top-N recommendations are not all the same type. To measure the degree of variation, the diversity measure was computed.
Diversity looks at the dissimilarity between pairs of recommendations and returns the average dissimilarity of N recommendations. Formally[7, 16],

$$Diversity(r_1, \ldots, r_N) = \frac{\sum_{i=1\ldots N}\sum_{j=1\ldots N \wedge i \neq j}\left(1 - sim(r_i, r_j)\right)}{N(N-1)}$$

where

- r represents a recommended profile
- N represents the number of recommendations made
- $sim(r_i, r_j)$ represents the similarity between recommended profile i and j

An average of the diversity of all profile recommendations can then be computed to gain an understanding of the algorithm's overall diversity.

9.2 Validation Testing

In order to select the best baseline CF and MC in terms of RMSE and Coverage, several variants of each model were tested on a validation set. The respective models were tested using different similarity metrics and neighbourhood functions. Below I will analyse the results of the validation tests and discuss the selection process for the algorithm.

9.2.1 Collaborative Filtering

User based

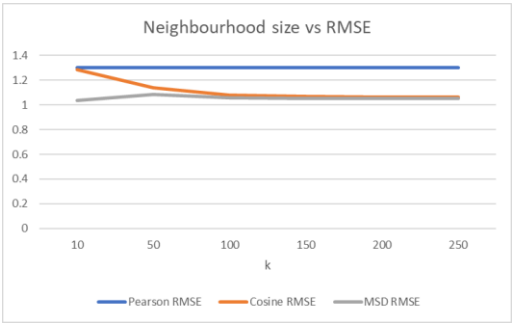

Figure 2: Neighbourhood size vs RMSE (UBCF)

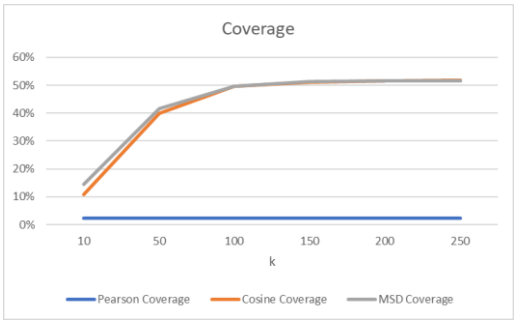

Figure 3: Neighbourhood size vs Coverage (UBCF)

Figure 6 illustrates the change in RMSE with respect to neighbourhood size (k) for the three different similarity metrics, while Figure 7 illustrates the change in coverage w.r.t.[8] to k. Pearson performed the worst in terms of both RMSE and coverage, followed by cosine and then MSD. Pearson appeared to have a constant RMSE of around 1.3 and coverage of about 2%. The coverage values indicate that there was no change in the small number of neighbours generated and hence the RMSE remain unchanged. Pearson does not perform well if the number of common ratings is low, which it commonly is in the dataset. Pearson is undefined if the number of common ratings is one. If there are only two common ratings the Pearson coefficient is either 1, 0 or -1. The algorithm only accepted similarity scores greater than 0, therefore there were few similar profiles found. Additionally, if the common rating vectors of a pair of users had zero variance, the resulting Pearson coefficient was undefined. This meant that users with very similar ratings are not included in the neighbourhood. These factors meant that Pearson was unable to make sufficient neighbourhoods which is evident from the low coverage values throughout. I did consider enhancing Pearson using significance weighting, however significance weighting only modifies the scores already computed and did not help with the lack of common ratings. It had no effect on the low coverage values. Cosine and MSD did not suffer from the same problems as Pearson and performed significantly better. Cosine and MSD had similar coverage curves however Cosine had slightly lower coverage values at the lower k values. In terms of RMSE, MSD outperformed cosine at the lower k values. Both Cosine and MSD's RMSE values converge as k increases. Cosine normalised for the number of ratings each user provided. This reduced the similarity score if the number of common ratings was less than the number of ratings provided by each user. This approach suffered due to the sparsity of this dataset. On average, a user rated 4-5 items. If two users were very similar, they shared common ratings on a lot of their respective ratings i.e. the set of co-rated items was close to, or the

[8] with respect to

same as the set of rated items. Therefore, a close neighbour was unlikely to have rated a target item not rated by the given user. This had a negative effect on both the RMSE and coverage. MSD on the other hand, does not account for the total number of rated items and only considers the co-rated item ratings. This would typically lead to a poorer performance in comparison to cosine [37]. However due to the sparsity in the dataset, MSD and cosine achieved similar results.

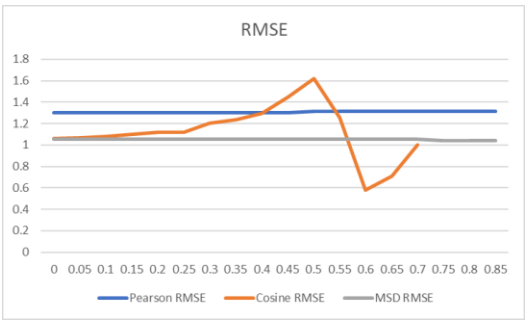

Figure 4: Threshold vs RMSE (UBCF)

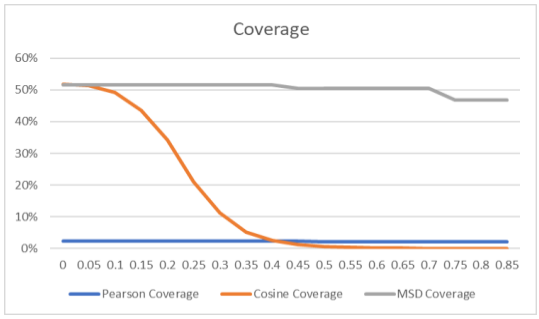

Figure 5: Threshold vs Coverage (UBCF)

Above are measurements taken with respect to the threshold value used in threshold neighbourhood formation. In Figure 8, we see the change in RMSE and in Figure 9, we see the change in coverage. When the threshold is 0, we get the maximum neighbourhood size. As the threshold approaches 1, the neighbourhood size typically decreases. Again, Pearson performed the worst returning a constant RMSE of 1.3 and 2% coverage. As previously discussed, Pearson yielded very few neighbours and generated high similarity scores. As such, there is very little change in the neighbourhood. As we can see, cosine performed similar to MSD at the lower threshold values, however as the threshold increases the coverage drops rapidly, this is a result of a drop in the number of neighbours. Cosine produced relatively low similarities scores. Most of the neighbours had a similarity less than 0.4. At the higher threshold values, the neighbourhood sizes are very small and RMSE values become less reliable which is evident from the sharp spike and drop in figure 8. By a threshold of 0.7, the coverage is 0% and there are no neighbours left to make a prediction indicated by the stop in the RMSE graph. MSD was the overall best performing metric. It produced high similarity scores which allowed the algorithm to retain a larger neighbourhood despite the high threshold.

Item-based

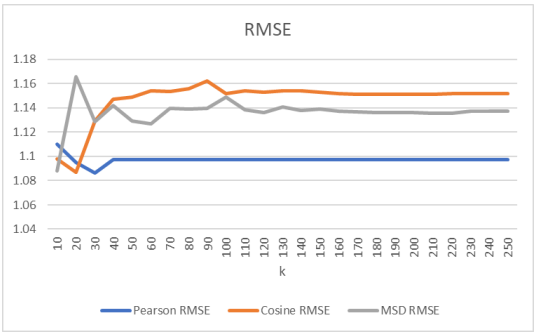

Figure 8: Neighbourhood vs RMSE

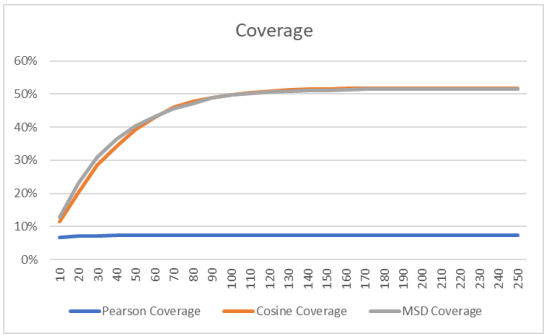

Figure 9: Neighbourhood vs Coverage

Figure 8 illustrates the change in RMSE w.r.t neighbourhood size (k) for the three different similarity metrics, while Figure 9 illustrates the change in coverage w.r.t. to k. The IB approaches performed slightly worse in terms of RMSE than the UB approaches. Out of the IB approaches, the Pearson metric performed the worst in terms of both RMSE and coverage. Like the UB approach, the RMSE and coverage were almost constant throughout with a small variation at the smaller neighbourhood sizes. MSD and cosine were closer in terms of RMSE and coverage. MSD performed slightly better than cosine throughout (except at the small neighbourhood sizes where cosine slightly outperformed MSD in terms of RMSE).

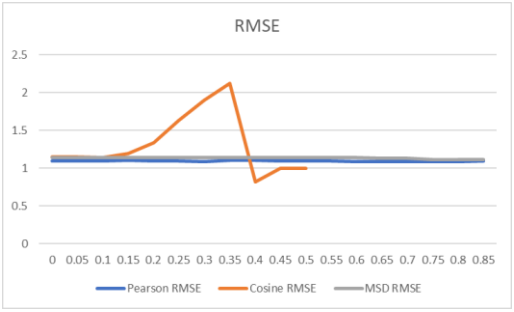

Figure 10: Threshold vs RMSE

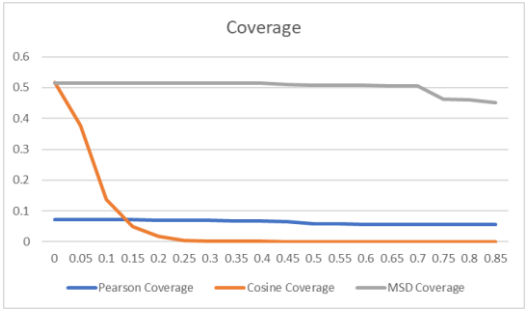

Figure 11: Threshold vs Coverage

Above are measurements taken with respect to the threshold value used in threshold neighbourhood formation. In Figure 8, we see the change in RMSE and in Figure 9, we see the change in coverage. Pearson performed poorly again. Cosine performed similar to MSD at a threshold of 0 but its coverage values drop rapidly with the increase in threshold. By a threshold of 0.5 cosine yielded no neighbours and was no longer able to make predictions. MSD retained a high coverage despite the change in threshold and similarly maintain a constant RMSE throughout. The mean number of ratings per hotel, 13, is significantly more than users at 6 (refer to Tables 1 and 2). The average number of common users per item was 1.18. This meant that similarity scores between neighbours decreased for the cosine metric as there was a larger set of ratings that were not shared. This is evident in figure 11, where the drop in coverage for cosine is more dramatic than the UB-CF approach and occurs at a lower threshold value. The coverage also reaches 0% earlier at a threshold of 0.55. In addition, the spike in the RMSE at the low coverage values is also more severe. The rest of the metrics follow a similar trend to UB when we look at RMSE and coverage with respect to threshold.

Pearson was the worst suited similarity metric. Its inability to generate similarity scores with a low number of common ratings coupled with the inability to define similarity when the common ratings were very similar made it an unsuitable metric. Cosine and MSD were the best performing metrics; however, MSD had a more stable performance across k and threshold values since it tended to produce relatively high similarity values for neighbours. The best performing algorithm was UB-CF using MSD similarity with 150-nearest neighbours (UB-CF w/MSD k@150). Ideally to reduce space complexity the smallest possible neighbourhood size is selected and increasing k pass 150 showed very little improvement in RMSE and coverage. It should be noted that sparsity was a significant issue in this dataset and negatively affected the performance of the similarity metrics.

9.2.2 Multi-criteria

User-based

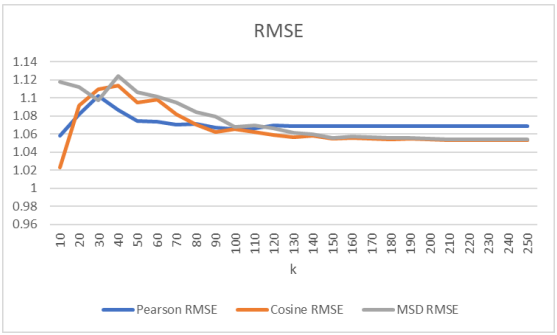

Figure 12: Neighbourhood vs RMSE (UBMC)

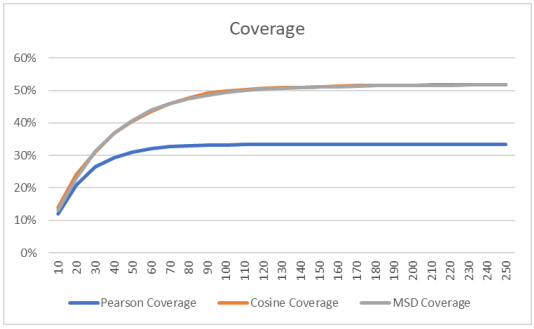

Figure 13: Neighbourhood vs Coverage (UBMC)

Figure 12 illustrates the change in RMSE w.r.t neighbourhood size (k) for the three different similarity metrics, while Figure 13 illustrates the change in coverage w.r.t. to k. Pearson performed the worse both in terms of RMSE and coverage throughout. Again, this is probably due to its inability to define similarity when the common ratings are very similar. Although, Pearson seen some improvement in coverage compared to UB-CF. The higher density in the criteria rating reduces the chances of having only one or two common ratings when computing similarity. MSD and cosine are closely matched in terms of RMSE and have identical coverage curves. However, cosine performed slightly better at the smaller k values.

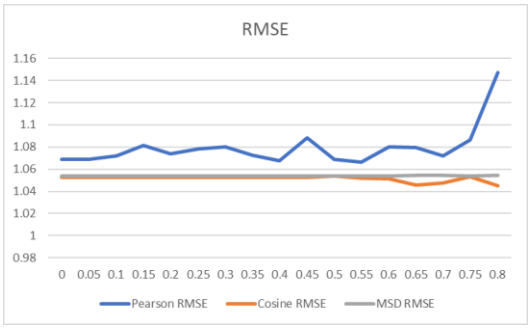

Figure 6: Threshold vs RMSE (UBMC)

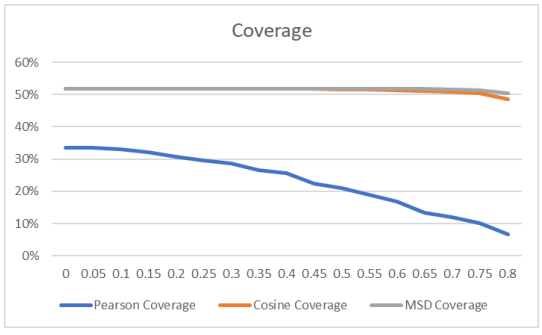

Figure 7: Threshold vs Coverage (UBMC)

Figure 14 shows RMSE w.r.t. threshold and Figure 15 shows coverage w.r.t. threshold. Both cosine and MSD retained their coverage until the very high threshold values indicating that a lot of neighbours had a high similarity score. The high density in the criteria ratings meant that the number of co-rated criteria was often the same as the number of the criteria rated. This resulted in the similarity scores between co-rated items and hence the similarity between the users being relatively high for cosine. Pearson on the other hand has a gradual sloping curve which indicates that the similarity scores were evenly spread across the neighbourhood. The gradual drop in coverage caused some fluctuation in the RMSE values, however the error stayed relatively the same until a threshold of 0.8. At this threshold, the neighbourhood sizes are much lower, resulting in the dramatic change in RMSE. For cosine and MSD, the steady coverage values

throughout meant very little change in the neighbourhoods and hence the RMSE values remain the same. Both cosine and MSD achieved similar results in this case.

Item-based

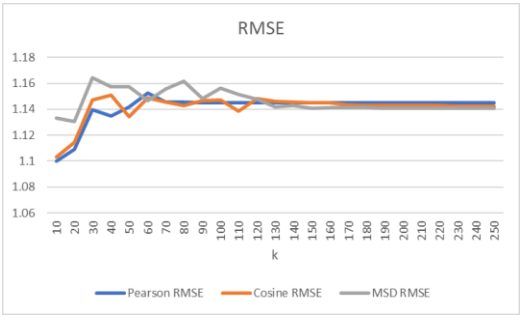

Figure 8: Neighbourhood size vs RMSE (IBMC)

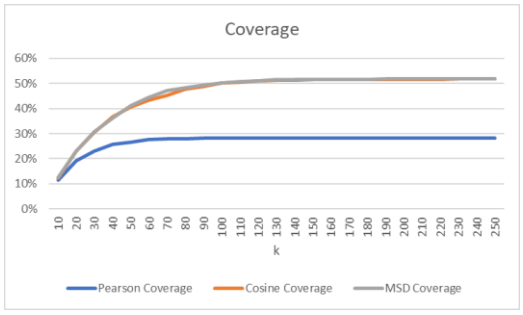

Figure 9: Neighbourhood size vs Coverage (IBMC)

Figure 16 illustrates the change in RMSE w.r.t neighbourhood size (k) for the three different similarity metrics, while Figure 17 illustrates the change in coverage w.r.t. to k. The metrics are evenly matched in terms of RMSE, cosine and Pearson are better at the lower k values. As the k increases the RMSE values become very similar. In terms of coverage, cosine and MSD both significantly outperform Pearson. MSD and cosine have almost identical curves. They both converge at around 50% coverage when k=100.

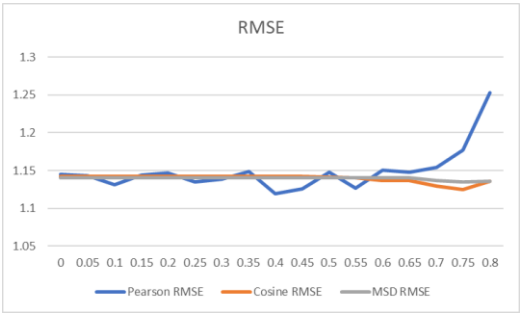

Figure 10: Threshold vs RMSE (IBMC)

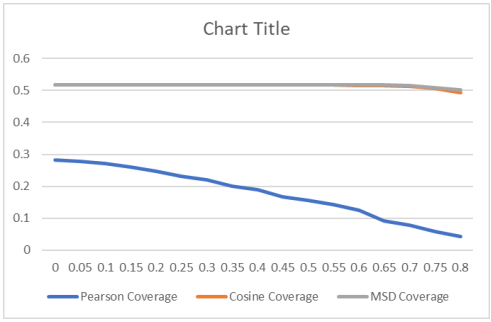

Figure 11: Threshold vs Coverage (IBMC)

We see similar trends in the RMSE and coverage with respect to threshold as seen in UB-MC. The MSD and cosine retained coverage right up until the very high threshold values indicated by the almost flat line. Pearson loses coverage gradually with the increase in threshold indicated by the shallow gradient. MSD and cosine are also very similar in terms of RMSE which is expected due to their almost identical coverage curves.

Both MSD and cosine performed well however, again MSD seemed to have steadier RMSE values. The best algorithm was UB-MC with MSD and 150-nearest neighbours (UB-MC w/ MSD k@150).

9.3 Experiment 1: MC vs CF – RMSE and Coverage

For the final evaluation, the best MC and CF algorithms were compared in terms of RMSE and coverage. The two candidates were UB-MC w/ MSD k@150 and UB-CF w/ MSD k@150. The results are shown in figures 22 and 23 below.

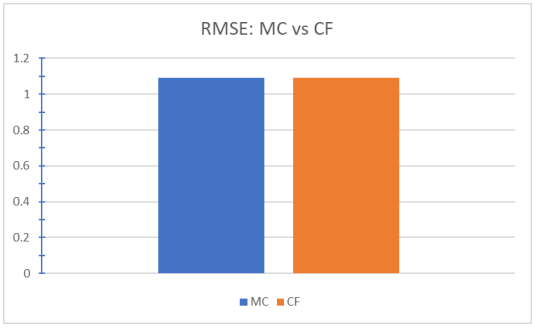

Figure 12: MC vs CF RMSE

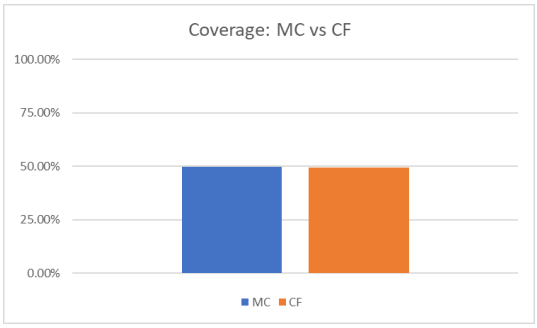

Figure 13: MC vs CF coverage

Both MC and CF had very similar RMSE and coverage values. The hypothesis was that additional information provided by multi-criteria ratings could help to improve the quality of recommendations because the system would be able to represent more complex preferences of each user [19]. It is clear in this instance, increasing dimensionality using the MC approach had little effect on the overall performance

9.4 Experiment 2: MC significance weighting

The MC algorithm produces an overall similarity for a pair of users by computing similarity scores between each co-rated profiles (e.g. items) based on their criteria ratings and then combining the scores using the average to give an overall similarity between two profiles (e.g. users). In this case, MC with cosine does not normalise for the number of rated items but instead the number of rated criteria. Consequently, I conducted an experiment to investigate if the similarity score can be improved by applying significance weighting on the overall similarity between profiles. However, there result showed no change in coverage and a marginal 2.22% improvement in RMSE. The average of common ratings between user was ≈1 (see 7.). Evidently, there was not enough overlap between ratings for the weighting to cause a considerable difference.

9.5 Experiment 3: CB vs CF vs MC

Below the recommendations made by both CF and MC approaches were compared against a content-based algorithm. For the following experiments, the same MC and CF used in Experiment 1 and a user-based content-based system was used.

9.5.1 Mean Average Precision (MAP)

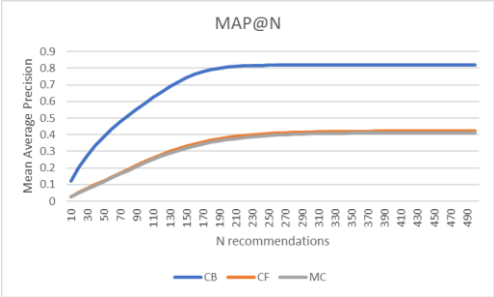

Figure 14: MAP of CF, MC, and CB

To investigate the accuracy of the recommendations the MAP for a different number of recommendations was computed. The results can be seen in Figure 24 above. Both CF and MC approaches had almost identical results. The CB approach significantly outperformed both CF and MC approaches. This could be attributed to the dense feature matrix produced by the CB approach. CF and MC approaches suffered from the sparsity of ratings in the dataset. The CB approach does not rely on user ratings and as such can draw similarity between pairs of users who do not have co-rated items. Therefore, the CB approach can make more recommendations. Both CF and MC approaches ranked recommendations based on their prediction ratings. Ranking recommendations according to predicted ratings created a lot of tied rankings in the CF and MC recommendations which had a negative effect on their MAP values.

9.5.2 Diversity

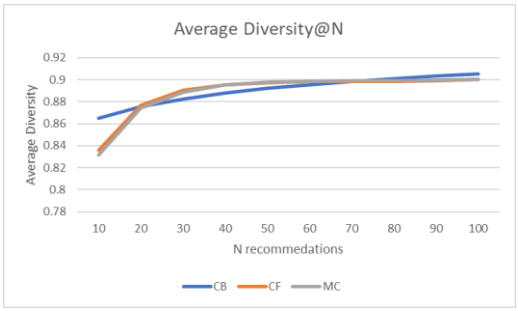

Figure 15: Average Diversity for N recommendations

The average diversity across N recommendations for all users in the test set was computed and the results for each approach can be seen in Figure 25. As we can see the CB approach performs better at the lower N values, however CF and MC approaches quickly outperform the CB approach in the intermediate N values. As the number of recommendations tends to the max recommended set, the diversity converges on the average diversity of this set. For the CB approach, this set is slightly larger and evidently had a higher average diversity.

9.5.3 Coverage

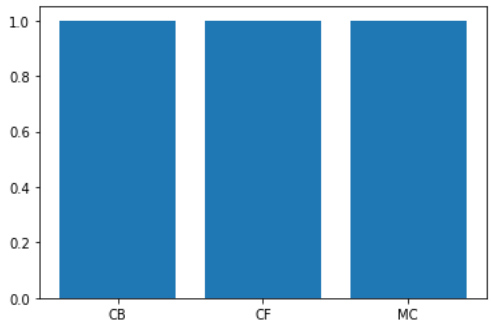

Figure 16: Coverage of CF, MC, and CB recommendations

From Figure 24, we can see that all the recommenders achieved a coverage of 100%. Coverage in this case is the proportion of users in the test set with corresponding relevant items (i.e. users that had rated items with a rating >=4) which the system was able to make a recommendation. In this dataset, the CB approach could always provide a recommendation as every item had some similarity with the other items in the dataset. The CF and MC approaches could always make a recommendation because every user had neighbouring users that rated items not rated by the user.

10 Conclusion

Recommender systems are powerful tools for extracting value for businesses from its user database. These systems allow businesses to find items suited for the users. This ultimately creates a better user experience because users do not have to sieve through large libraries to find item that suit their taste, which in turn maximizes the sales. As such these systems have become key players in the business model. However, with the expansion of e-commerce, new technologies are needed to improve the predictive capabilities of these systems.

In this paper, I conducted a comparative study on the multi-criteria (MC) approach which extends the collaborative filtering (CF) recommender system. The hypothesis was that additional information provided by multi-criteria ratings could help to improve the quality of recommendations because the system would be able to represent more complex preferences of each user [19]. By increasing dimensionality, I hoped to improve the reliability of the similarities computed and ultimately improve the performance of the algorithm. In addition, the performance of CF and MC approaches was compared to a content-based (CB) recommender. To reflect the reality of explicit user ratings a real-world TripAdvisor dataset was used in this study. From my experiments, I found there was no significant difference in results obtained using MC and CF approaches. This is likely due to the high sparsity of the dataset which negatively affected the

performance of the algorithms. As expected, given the sparsity, the CB approach was able to outperform both CF and MC approaches.

Please note that this dataset was especially sparse compared to other similar experiments in this domain. To further consolidate my findings similar comparisons can be conducted on datasets of varying sparsity. Additional experiments could be done on the configurations of the MC approach, for example the method used to combine the scores after the similarity is computed using the criteria ratings. One could also look at weighting the rating from each criterion according to some importance measure. Further experimentation and comparison could also be made with a hybrid recommender system which implements both CF and CB approaches.

SAcknowledgements

I would like to thank to Dr. Michael O'Mahony from University College Dublin for vital input and helpful corrections.

References

1. Anderson, C. and M.P. Andersson, *Long tail.* 2004.
2. Christodoulou, P., K. Christodoulou, and A.S. Andreou, *A real-Time targeted recommender system for supermarkets.* 2017.
3. MacKenzie, I., C. Meyer, and S. Noble, *How retailers can keep up with consumers.* McKinsey & Company, 2013.
4. Adomavicius, G. and A. Tuzhilin, *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.* IEEE Transactions on Knowledge & Data Engineering, 2005(6): p. 734-749.
5. Trusov, M., R.E. Bucklin, and K. Pauwels, *Effects of word-of-mouth versus traditional marketing: findings from an internet social networking site.* Journal of marketing, 2009. **73**(5): p. 90-102.
6. Ning, X., C. Desrosiers, and G. Karypis, *A comprehensive survey of neighborhood-based recommendation methods*, in *Recommender systems handbook*. 2015, Springer. p. 37-76.
7. Lü, L., et al., *Recommender systems.* Physics reports, 2012. **519**(1): p. 1-49.
8. Lops, P., M. De Gemmis, and G. Semeraro, *Content-based recommender systems: State of the art and trends*, in *Recommender systems handbook*. 2011, Springer. p. 73-105.
9. Deerwester, S., et al., *Indexing by latent semantic analysis.* Journal of the American society for information science, 1990. **41**(6): p. 391-407.

10. Zhu, Y., et al., *Addressing the item cold-start problem by attribute-driven active learning.* IEEE Transactions on Knowledge and Data Engineering, 2019.
11. Jia, Z., et al. *User-based collaborative filtering for tourist attraction recommendations*. in *2015 IEEE International Conference on Computational Intelligence & Communication Technology*. 2015. IEEE.
12. Shardanand, U. and P. Maes. *Social information filtering: algorithms for automating" word of mouth"*. in *Chi*. 1995. Citeseer.
13. Sarwar, B.M., et al., *Item-based collaborative filtering recommendation algorithms.* Www, 2001. **1**: p. 285-295.
14. Linden, G., B. Smith, and J. York, *Amazon. com recommendations: Item-to-item collaborative filtering.* IEEE Internet computing, 2003(1): p. 76-80.
15. Koren, Y., R. Bell, and C. Volinsky, *Matrix factorization techniques for recommender systems.* Computer, 2009(8): p. 30-37.
16. Aggarwal, C.C., *Recommender systems*. Vol. 1. 2016: Springer.
17. Desrosiers, C. and G. Karypis, *Solving the sparsity problem: collaborative filtering via indirect similarities.* Department of Computer Science and Engineering University of Minnesota, 2008.
18. Deshpande, M. and G. Karypis, *Item-based top-n recommendation algorithms.* ACM Transactions on Information Systems (TOIS), 2004. **22**(1): p. 143-177.
19. Adomavicius, G., N. Manouselis, and Y. Kwon, *Multi-criteria recommender systems*, in *Recommender systems handbook*. 2011, Springer. p. 769-803.
20. Manouselis, N. and C. Costopoulou, *Experimental analysis of design choices in multiattribute utility collaborative filtering.* International Journal of Pattern Recognition and Artificial Intelligence, 2007. **21**(02): p. 311-331.
21. Li, Q., C. Wang, and G. Geng. *Improving personalized services in mobile commerce by a novel multicriteria rating approach*. in *Proceedings of the 17th international conference on World Wide Web*. 2008. ACM.
22. Burke, R., *Hybrid recommender systems: Survey and experiments.* User modeling and user-adapted interaction, 2002. **12**(4): p. 331-370.
23. Tran, T. and R. Cohen. *Hybrid recommender systems for electronic commerce*. in *Proc. Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, Technical Report WS-00-04, AAAI Press*. 2000.
24. Cotter, P. and B. Smyth. *Ptv: Intelligent personalised tv guides*. in *AAAI/IAAI*. 2000.
25. Bennett, J. and S. Lanning. *The netflix prize*. in *Proceedings of KDD cup and workshop*. 2007. New York, NY, USA.
26. *Netflix Prize: Forum*. Available from: https://www.netflixprize.com/community/topic_1537.html.
27. Muhammad, K., A. Lawlor, and B. Smyth. *On the use of opinionated explanations to rank and justify recommendations*. in *The Twenty-Ninth International Flairs Conference*. 2016.
28. Baroni, M., et al., *Predictive ability of regression models. Part II: Selection of the best predictive PLS model.* Journal of chemometrics, 1992. **6**(6): p. 347-356.
29. Harper, F.M. and J.A. Konstan, *The movielens datasets: History and context.* Acm transactions on interactive intelligent systems (tiis), 2015. **5**(4): p. 1-19.

30. Roko, A., et al., *An Enhanced Data Sparsity Reduction Method for Effective Collaborative Filtering Recommendations.* International Journal of Education and Management Engineering, 2020. **10**(1): p. 27.
31. Herlocker, J.L., et al. *An algorithmic framework for performing collaborative filtering*. in *22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*. 1999. Association for Computing Machinery, Inc.
32. Harpale, A.S. and Y. Yang. *Personalized active learning for collaborative filtering*. in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 2008. ACM.
33. Patra, B.K., et al., *A new similarity measure using Bhattacharyya coefficient for collaborative filtering in sparse data.* Knowledge-Based Systems, 2015. **82**: p. 163-177.
34. .
35. Herlocker, J.L., et al., *Evaluating collaborative filtering recommender systems.* ACM Transactions on Information Systems (TOIS), 2004. **22**(1): p. 5-53.
36. http://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html#Precision-and-Recall-of-Recommender-Systems.
37. Feng, J., et al., *An improved collaborative filtering method based on similarity.* PloS one, 2018. **13**(9).